Machine Learning Techniques for the Preservation of Data Privacy

By

William R. Briguglio

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy (Ph.D.)

in the Department of Electrical and Computer Engineering

Machine Learning Techniques for the Preservation of Data Privacy

by

William R. Briguglio
B.C.S., University of Windsor – 2019
M.Sc., University of Windsor – 2020

Supervisory Committee:

Dr. Issa Traoré, Co-Supervisor
(Department of Electrical and Computer Engineering)

Dr. Waleed Yousef, Co-supervisor
(Department of Electrical and Computer Engineering)

Dr. Sherif Saad, Departmental Member
(Department of Electrical and Computer Engineering)

Dr. Hausi Muller, Outside Member
(Department of Computer Science)

Abstract

Machine learning has been successfully applied in various domains in recent years. Still, its use is limited since training or testing data may contain sensitive information which cannot be shared with model owners due to privacy concerns. For example, healthcare providers may be bound by patient privacy laws. Without large publicly available datasets, useful models become impossible to train. Therefore, ML methods that preserve the privacy of private training data are required. One solution is to use homomorphic encryption to carry out mathematical operations on encrypted data without compromising the privacy of said data. However, sometimes a large dataset that is difficult for a single institution to obtain is needed for complex learning tasks. In such a case, federated learning can be used to learn from private data distributed across multiple owners without compromising the privacy of each owner's data.

However, federated learning carries its own risks. For example, exchanging even the minimum information needed for training can compromise privacy, and rogue participants in a federated learning network may attempt to sabotage model performance. Further, data that is not independently and identically distributed hampers the convergence of federated learning techniques. Additionally, once training is complete, regardless of the means, extra steps must be taken to ensure model privacy during the inference phase. Such steps are needed to ensure the model owner(s) can retain sole proprietorship of the global model. Further, if a model's parameters are leaked, then an adversary may be able to reverse engineer them to compromise the privacy of the training data. Keeping a model private eliminates this risk.

In this dissertation, we provide an in-depth background to the problems of machine learning with encryption and federated learning. We propose novel techniques for private inference that maintain the privacy of both the model and the data the model performs inferences on. We propose a federated learning framework which, in addition to maintaining the privacy of the data used during training, is, to the best of our knowledge, the only approach that enables just a single participant to obtain the jointly trained model. We also present a secure method for distributed dimensionality reduction, which can be used as a preprocessing step to enhance the performance of the proposed federated learning framework. Finally, we combine these approaches and propose an end-to-end federated learning and private inference framework which maintains data privacy during the federated learning and private inference phase, as well as ensures the privacy of the trained model's parameters during each phase.

Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

1-NN	1-nearest-neighbour	74
AUC	area under curve	45
BSR	backdoor success rate	45
BFV	Brakerski/Fan-Vercauteren	25
CAF	cooperative activation function	82
CKKS	Cheon-Kim-Kim-Song	25
CNN	convolutional neural network	10
CRF	completely random forest	12
CV	cross-validation	20
DDoS	distributed denial of service	19
FDA	Fisher's discriminant analysis	11
FFNN	feed-forward neural network	35
FHE	fully homomorphic encryption	9
FL	federated learning	1
FeS-PCA	federated supervised principal component analysis	60
FeSK-PCA	federated supervised kernel principal component analysis	63
FLAMED	federated learning via aggregating multivariate estimated densities	29
FLAMED-PICAFE	federated learning via aggregating multivariate estimated densities with private inference via cooperative activation functions and encryption	82
GAN	generative adversarial network	14
GDRP	General Data Protection Regulation	37
GWAS	genome-wide association studies	17
HE	homomorphic encryption	1
non-IID	non-independent and identically distributed	2
KDE	kernel density estimation	32
KSPCA	kernel supervised principal component analysis	18

List of Appendices

ACKNOWLEDGMENT

Thank you to my supervisors Dr. Issa Traoré and Dr. Waleed A. Yousef for their thoughtful guidance throughout my doctorate. Their expertise was invaluable in shaping and refining my research.

DEDICATION

I dedicate this dissertation, and the work taken to complete it, to my father and mother, to whom I am gratefully indebted, and Casandra Malynowskyj, with whom I hope to pay that debt forward.

CHAPTER 1

Introduction

1.1 Problem Statement

Recent decades have seen significant benefits from the successful application of machine learning (ML) algorithms to large amounts of data. Typically, this data is gathered by a single institution onto a central server where it can be processed and used without restrictions. However, this implies an important limitation since useful data may contain sensitive information meaning owners cannot share their data due to privacy concerns. For example, cell phone users may not wish to share their typing data, or healthcare providers may be bound by patient privacy laws. In such cases, useful models may be impossible to train since sufficient datasets are not publicly available.

Therefore, ML methods that do not require direct access to the raw training data, thus preserving its privacy, are needed. One such method is private ML with homomorphic encryption (HE), an encryption method that allows mathematical operations to be carried out on encrypted data without compromising the privacy of said data. Such approaches can keep training data, the trained model, or testing data private. We elaborate on these different use cases in Section 2.1.2.

However, sometimes learning tasks may be very complex and require a large dataset that is difficult or impossible for a single institution to obtain for centralized training. Thus, the need arises for a ML method that is capable of learning from data distributed across multiple owners without compromising the privacy of each owner's data. One such method is federated learning (FL). In the standard FL approach, each data owner or client, starting from a common initial model, trains a local ML model. Then an aggregating server combines the local models to obtain the global shared model. This process repeats for several rounds, starting from the previous round's global model until convergence is reached.

Typically, the result of local model training is sent to the aggregating server in the form of weight or gradient updates. Initially, it was assumed that privacy would not be

compromised if the gradient updates were shared with the aggregating server, but [1] and [2] showed it is possible to leak training samples from the gradient updates alone in what is called a gradient leakage attack. In light of this, various solutions have been employed, and briefly surveyed in the next chapter, to protect the privacy of clients' data. These solutions leverage familiar security techniques, such as differential privacy, HE, and secure multiparty computation (SMC). However, as discussed in detail in Section 4.1.1, these solutions work counter to securing FL against model poisoning attacks that target the performance of the global model rather than the privacy of the training process.

Another obstacle FL faces in practice is non-independent and identically distributed (non-IID) data. In a standard ML setting with stochastic gradient descent (SGD), each batch is randomly sampled from the entire dataset independently and from the same distribution. However, during FL local training each batch is sampled only from the data available at a given client. This introduces multiple sources of non-IIDness such as clients having different dataset distributions or each client not being included in every round of training with the same probability. Non-IID data hampers or altogether prevents the convergence of the FL training process. In [3], Kairouz et al. present an excellent taxonomy of the different types of non-IIDness found in the FL setting. Many approaches have been proposed in the literature to overcome this hurdle in practice, some of which are discussed in Section 2.2, but none to our knowledge claim to address all types of non-IIDness, and many focus only on this problem while ignoring privacy and security considerations altogether.

Privacy preserving preprocessing is also a necessity for FL. Many datasets contain redundant or noisy features that are typically removed using some dimensionality reduction technique (discussed in Chapter 5). This step is crucial for improving the efficiency and performance of the resulting model, but most dimensionality reduction techniques assume that all training data will be available during the preprocessing step. Therefore, it is necessary to extend dimensionality reduction techniques to the distributed setting. These techniques should ideally return the same transformation as their centralized counterparts without the need for participating data owners to share their raw data. Further, these techniques should maximize the utility of the reduced data by taking advantage of the label information from supervised learning tasks.

Another important consideration is the privacy of a trained model deployed for real-world use. In many cases, a model owner may wish to keep their model private in order to retain a competitive advantage. Maintaining model privacy also impedes the ability of malicious users to reverse engineer an ML model and compromise the privacy of its training data. For example, membership inference attacks can be used to determine if a sample was present during training and model inversion attacks can recreate training samples. The model owner

could host their model on a private server and have users send their live data (gathered in production) to that server for inference, but data owners may be understandably hesitant to do so because of privacy concerns. Private ML provides a solution by enabling inferences on live data without the need for the owner to share their model. This is typically achieved by using HE, SMC, or differential privacy.

It is not a trivial matter to combine FL and private inference approaches to enable data privacy during training on distributed data while also preserving the privacy of the jointly trained model. Regardless of the approach used, the FL process ensures that all participants are given an updated copy of the jointly trained model, meaning no one participant can retain sole ownership of the model. Participants are also treated as trustworthy enough to not attack the model in an attempt to learn about other clients' training data. However, even if participants are trustworthy and accept shared ownership, the distribution of copies of the jointly trained model introduces multiple points of failure and increases the risk of the jointly trained model being leaked. That could compromise the privacy of both the model and the training data.

In this dissertation, we present an FL approach that does not require a copy of the trained or partially trained global model to be distributed to each participant, meaning the aggregating server can retain sole proprietorship of the global model (Chapter 4). We combine this approach with federated preprocessing and private inference approaches which we introduce in chapters 5 and 6 in order to obtain an end-to-end FL and private inference framework which both removes disincentives due to privacy risks and strengthens monetary incentives by simplifying model ownership. By providing stronger protection against privacy attacks at all stages of a jointly trained model's life cycle, and enabling the owner of the jointly trained model to be more secure in the propriety of said model, this approach motivates further innovation.

1.2 Research Contributions

Our research has provided the following contributions to the literature:

Contribution 1: *A framework that considers the requirements and constraints of private ML and facilitates a method for private inference using ML and HE.*

Contribution 2: *A FL framework that bypasses all forms of non-IIDness, makes gradient leakage impossible, and mitigates model poisoning attacks.*

Contribution 3: *A method for applying different variants of supervised principal component analysis in the federated setting.*

Contribution 4: *A FL and private inference framework that enables training on non-IID data while protecting against privacy and model security attacks and allowing the aggregating server to retain sole ownership of the global model. The framework also enables the aggregating server to make predictions on private data while keeping their model private. The framework is further enhanced with federated supervised dimensionality reduction techniques.*

Contribution 5: *Enable private inference with conventionally or jointly trained neural networks using the sigmoid or rectified linear unit (ReLU) activation functions with minimal loss in model accuracy.*

1.3 List of Publications

Contribution 1 was completed and published as *Machine Learning in Precision Medicine to Preserve Privacy via Encryption* in Pattern Recognition Letters [4].

Contribution 2 was completed and submitted as *FLAMED: Federated Learning via Aggregating Multivariate Estimated Densities* to the 29$^{\text{th}}$ European Symposium on Research in Computer Security.

Contribution 3 was completed and published as *Federated Supervised Principal Component Analysis* in IEEE Transactions on Information Forensics and Security [5].

Contribution 4 and 5 were completed and submitted as *FLAMED-PICAFE: a Federated Learning and Private Inference Framework* to IEEE Transactions on Information Forensics and Security.

1.4 Report Outline

In Chapter 2, we present the background and preliminaries for the research contained in this dissertation. Specifically, Section 2.1 provides an overview of private ML using HE,

followed by a review of FL techniques and distributed dimensionality reduction techniques in Section 2.2. Chapter 3 introduces Contribution 1, our HE-based private ML scheme that supports inference on encrypted data using a logistic regression (LR) model. Chapter 4 presents Contribution 2, our work on a general framework for FL, which simultaneously address non-IIDness, privacy, and model security concerns. Chapter 5 introduces Contribution 3, a federated version of supervised dimensionality reduction approaches. Chapter 6 combines contributions 1 through 3 together into an end-to-end FL and private inference framework to fulfil Contribution 4. In the same chapter, we present cooperative activation functions, our 5^{th} contribution. In Chapter 7, we give a summary of the contributions made in this dissertation and discuss directions for future work.

CHAPTER 2

Background and Related Work

ML with private data can be broadly separated into two settings. In both settings, the training data (and sometimes testing data) are considered sensitive and must be kept private. That is, the ML algorithm cannot directly access the raw, sensitive information. In the private ML setting, we assume all the training data is available at a single location. In the FL setting, we assume the data is held across multiple locations and cannot be brought to a single location. The reader may be familiar with distributed ML, in which a single data owner trains an ML model across multiple devices in an attempt to speed up training by taking advantage of more compute resources. In this setting, data can be moved between machines freely, so all training is done using IID data. FL thus faces challenges that are distinct from distributed ML, which does not concern itself with privacy considerations or convergence issues arising from non-IID data. Therefore, distributed FL is not relevant to the present work.

2.0.1 Data Flow and Threat Model

We illustrate the data flow in the private ML and FL settings in Fig. 1. We have two types of participants across both settings, the model owner/aggregating server and the data owners. In the standard threat model, the data owners wish to keep their data, during training and testing, private from all other data owners and the model owner/aggregating server. Further, in the private ML setting, the model owner/aggregating server wishes to keep their model weights private from the data owner. In the FL setting, the aggregating server typically wishes to keep their weights private from data owners not involved during training, but are, to the best of our knowledge, always required to share the model weights with the data owners who participated in training. However, sharing model weights with all data owners involved during training increases the risk that the model weights are learnt by untrusted individuals not involved in training since more copies of the trained model

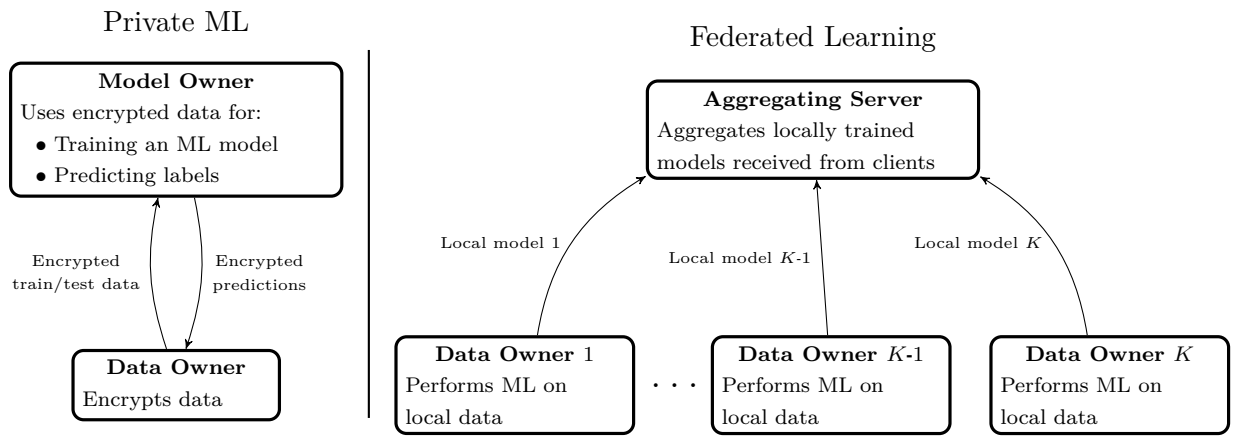

Figure 1: A diagram of the data flow in private ML and FL.

exist. Moreover, this risks untrusted individuals as well as the participants involved in the FL process leaking information about private training data using membership inference or model inversion attacks (see Section 2.2). Therefore, we assume a stricter threat model that only permits the aggregating server to know the model weights, allowing them to be the sole model owner. Hence, we may also refer to the aggregating server as the model owner in this instance. Using this threat model, because the owner of the jointly trained model can be more secure in their ownership of such asset, they can more confidently monetize their efforts. This approach motivates the model owner and can also benefit data providers in the FL network through agreements similar to any other scenario in which producers are compensated but do not own what they produce. Such incentives motivate further innovation. For example, the ability to monetize a jointly trained model may motivate an FL network in the healthcare setting to produce a model used for positively affecting patient outcomes by producing more accurate diagnoses. It may also be used to identify important biomarkers that can be singled out for further research, resulting in novel treatments [6].

2.1 Private Machine Learning

In the private ML setting, any privacy-preserving ML model builds on two main components: (1) the security and privacy features required to protect interactions with the data by stakeholders and (2) ML predictive models for this data. Each is reviewed briefly below.

2.1.1 Homomorphic Encryption

Encryption is the process of converting data from something intelligible into something unintelligible by sealing data in a metaphorical vault that can only be opened by somebody holding the secret decryption key to prevent unauthorized personnel access. A special scheme of encryption is HE, which was originally proposed by Rivest et al. [7] as a way to encrypt data such that certain operations could be performed on them without possessing that secret key (i.e. without decryption). The term "homomorphic encryption" describes a class of encryption algorithms that satisfies the homomorphic property; that is, certain operations, such as addition, can be carried out on ciphertext (i.e. encrypted data) directly so that upon decryption the same result is obtained as operating on the original messages. Therefore, HE allows other parties (e.g. cloud and service providers) to calculate certain mathematical functions expressed only in terms of these operations on the encrypted data while preserving the function and format of the encrypted data. For brevity, these types of functions are referred to as "HE-friendly". Formally, this can be expressed as

$$Dec[k_s,\ Enc(k_p, m_1) \diamond Enc(k_p, m_2)] = m_1 \circ m_2, \tag{1}$$

where k_s, k_p are the secret and public keys, respectively (since they are not equal, this is called "asymmetric encryption"), $m_1, m_2 \in M$ are two values on which we wish to perform encrypted operations on, M is the message space of the HE scheme (i.e. the set of all possible values acceptable by the scheme), and $\diamond, \circ$ are operations in encrypted and plain-text space, respectively. Like other types of encryption schemes, HE has three main functions: key generation, message encoding, and decoding.[1]

The remarkable properties of HE schemes are not without limitations. First, the set of functions that can be computed in ciphertext space is very restricted. Second, the computational complexity of the HE scheme depends primarily on the level of multiplication (i.e. the degree of the polynomial being evaluated) carried out on the encrypted data. Third, the ciphertext size increases considerably after encryption. Fourth, random noise is added to encrypted values for security reasons that varies with the type of operation (e.g. multiplication increases noise much more than addition). If this noise grows too large, then decryption yields incorrect results. There are three basic approaches to implement HE [see, e.g. 8]:

Partially homomorphic encryption (PHE), which allows only one type of operation to be executed on encrypted values an unlimited number of times.

[1] The online supplementary material provides some details on these operations at https://doi.org/10.1016/j.patrec.2021.07.004

Table 1: The eight possible scenarios of encrypting the three components: the training dataset **tr**, the ML model parameters $\mathcal{M}$, and the testing dataset **ts**.

#	**tr**	$\mathcal{M}$	**ts**	Literature	Dataset	ML	Enc. Library
0	0	0	0	Ordinary ML			
1	0	0	1	Our Approach in Chapter 3	MSK	many	SEAL
				Dowlin et al. [10]	MNIST	NN	SEAL
				Hesamifard et al. [11]	MNIST, CIFAR-10	DNN	Helib
2	0	1	0	Bost et al. [12]	Multiple	NB, HP, DT	self-implementation
3	0	1	1	——			
4	1	0	0	Graepel et al. [13]	Wisconsin	FDA	Magma
				Aslett et al. [14]	Multiple	NB, RF	EncryptedStats
				Nandakumar et al. [15]	MNIST6	DNN7	HElib
5	1	0	1	—— Not possible under the current theory			
6	1	1	0	—— Not practical: training on encrypted data already produces encrypted model			
7	1	1	1				

Somewhat homomorphic encryption (SWHE), where the size of the ciphertext grows with each homomorphic operation and hence the maximum number of allowed homomorphic operations is limited.

Fully homomorphic encryption (FHE), which supports an unlimited number of additions and multiplications [9]. This property makes FHE the most sophisticated HE scheme and the "holy grail" of modern cryptography. The FHE scheme supports basic arithmetic computations on encrypted data. Despite being a potential cryptographic technique, however, some FHE schemes remain impractical for real-world applications due to their computational overhead.

2.1.2 Machine Learning with Encryption

Any ML algorithm trains on some training dataset **tr**, fits a model's parameters $\mathcal{M}$, and finally tests on a testing dataset **ts**. Therefore, in principle, there are eight possible combinations or scenarios to introduce privacy via encryption to the learning process by encrypting (or leaving unencrypted) each of these three components. Table 1 summarizes those eight scenarios; below, we provide more details on them and their connections to the solution proposed in Chapter 3. We use 0 to denote an unencrypted component, where it still can only be encrypted using the public key k_p of another component without having access to

its private key k_s, and we use 1 to denote an encrypted component, where its private key k_s is not available for the other two components.

Scenario 0 is denoted by the binary combination 000, when **tr**, $\mathcal{M}$, and **ts** are all not encrypted; this is the typical ML paradigm, where no privacy is of concern.

Scenario 1 is denoted by 001, where only **ts** needs to be encrypted; this is the scenario of private inference and of the solution proposed in Chapter 3, as Sections 3.1 and 3.2 describe. In such a scenario, since the model has access to an unencrypted training set, such as a public dataset, only the test data **ts**, which could be patients' genome records, are sensitive. Although the standard homomorphic property as defined in (1) would imply that $\mathcal{M}$ must be encrypted with k_p to predict on an encrypted **ts**, this is not the case for our work, which leverages special techniques implemented in SEAL [16], that allow encryption with plaintext multiplication with the caveat that the results themselves are encrypted and can only be decrypted with k_s. Research exists in this category but in areas of application other than precision medicine.

Dowlin et al. [10] showed that a cloud service is capable of applying a neural network (NN) to encrypted **ts** to make encrypted predictions and return them in encrypted forms. They constructed a convolutional neural network (CNN) model from the unencrypted MNIST dataset and then produced a simpler FHE-friendly version of the CNN constructed only from addition and multiplication operations so that the parameters could be encrypted using the public key of the private testing dataset **ts**. In [11], Hesamifard et al. developed new techniques to allow testing CNN on encrypted **ts**. First, they designed methods to approximate the activation functions commonly used in CNNs with low-degree, FHE-friendly polynomials. Then, they trained a CNN on unencrypted **tr** with the approximation polynomials instead of the original activation functions. Finally, they converted the trained CNN to make predictions on encrypted **ts**.The work in [17] allowed private inference with NNs by using HE to compute the product between each layer's input and weight matrix. The result was then returned to the client, who unencrypted it for use as input to nonlinear activation functions before sending the re-encrypted activation values back to the server to be used as input for the next layer. In later work [18], they attempted to improve the security of the model by randomly permuting the returned products so that malicious clients cannot reconstruct the NN by submitting vectors with one non-zero element. However, as shown in the Appendix, this method does not actually ensure the privacy of model weights when using their sigmoid activation function. In [19], private predictions were enabled using multiplication triplets to privately compute the dot products between the client's input and the server's weights. Then, garbled circuits or splines enabled ReLU or an approximate sigmoid, respectively, to be privately computed. The process was repeated for each layer without the

client learning the model's weights or either party learning intermediate results.

Scenario 2 is denoted by 010, where the model is trained on an unencrypted training dataset **tr**. However, the model parameters themselves are then encrypted, which may imply privacy in **tr**, as well if the training is pursued locally where **tr** resides. Although the testing data **ts** is denoted by 0, it must be sent to $\mathcal{M}$ encrypted with its public key, as it is not possible, according to the theory of FHE, to pursue binary operations on encrypted numbers (parameters of $\mathcal{M}$) and unencrypted numbers (**ts**), without the results being encrypted and only decryptable with the same k_s that can decrypt $\mathcal{M}$. The virtue of scenario 2 is that it entails more freedom in choosing the model $\mathcal{M}$ as opposed to scenarios 4–7, where **tr** is encrypted and a stringent limitation is incurred for choosing the model $\mathcal{M}$ that can train on encrypted data. We are not aware of any literature that applies scenario 2 explicitly; however, Bost et al. [12] provided a very nested, layered model that could be classified as 010 scenario, but without relying solely on HE. They implemented a decision tree, naive Bayes and hyperplane decision that could test (not train) on encrypted data and built their models using cryptographic "building blocks" that emphasized protecting the model parameters and test data. They also used garbled circuits to compare encrypted data, which allowed a construction of argmax with alterations to ensure the ordering was not leaked. However, this introduced $k-1$ round trips between the party performing argmax and the party that can perform decryption. These building blocks allowed the implementation of decision tree, naive Bayes, and hyperplane decision with some minor changes. The building blocks also allowed the construction of other ML methods and a combination of methods using AdaBoost, which the authors demonstrated.

Scenario 3 is like 2 in that the model is trained on an unencrypted **tr**, and $\mathcal{M}$'s parameters are then encrypted; however, the test dataset **ts** is also encrypted with a different k_s than $\mathcal{M}$. Since there is no known method in the literature that allows the use of binary operations on two numbers encrypted with different k_s, scenario 3 (011) is not theoretically feasible under the current theory of cryptography.

Scenario 4 is denoted by 100, where a ML model is trained on encrypted **tr** (as in scenarios 5–7, as well). Hence, the model $\mathcal{M}$ will have encrypted weights by product, and the testing data must be sent to $\mathcal{M}$ encrypted with the same public key of **tr**, as explained above. Therefore, the reason scenario 5 (101) is not theoretically possible is the same as scenario 3. Furthermore, scenarios 6 and 7 (11x) are not of any practical interest, since the produced encrypted $\mathcal{M}$ does not need further encryption; this is possibly the reason for the absence of literature on these two scenarios. Under scenario 4, Graepel et al. [20] defined a fully confidential version of linear means and Fisher's discriminant analysis (FDA), which can train and test on encrypted data. Linear means are rewritten to avoid division when

learning the weights. The resulting decision function returns a multiple of the original decision function with the same sign. However, FDA requires the inverse of the covariance matrix to obtain the feature weights. This is found using gradient descent, the r^{th} iteration of which is shown to be a d-degree polynomial, where $d = 2(r-1)+1$. Aslett et al. [14] provided a completely random forest (CRF) implementation that could train and test on encrypted data. Among other alterations to the algorithm, the key difference was encoding feature values using one hot encoding after quantile partitioning. CRFs have important benefits, especially learning incrementally. The authors also provided a naive Bayes classifier that could train and test on encrypted data. It avoided parametric Gaussian modeling of predictors by directly modeling the decision boundary $x_j m_j + b_j$ for each predictor X_j. This required a homomorphic implementation of regression, which they also provided. Hesamifard et al. [11] discussed the computational complexity that makes NN training on encrypted data impractical despite being theoretically possible. Other authors have targeted simpler ML algorithms to avoid heavy computations of encrypted NNs. However, Dowlin et al. [10] show that training CNNs on encrypted data is possible. If all the activation functions and the loss function are polynomials, back-propagation can be computed using only addition and multiplication. However, high-degree polynomials used during back-propagation make it computationally prohibitive. Nandakumar et al. [15] evaluated the feasibility of training NNs on encrypted data completely non-interactively. Their proposed system used the FHE toolkit HElib to implement SGD for training. They used "ciphertext packing" to minimize the number of required bootstrapping operations and to enable the parallelization of computations at each neuron, thereby significantly reducing the computational complexity. This, in combination with simplifying the network architecture, allowed them to practically train NNs over encrypted data despite the computational hurdles.

2.2 Federated Learning

FL is but one approach which involves learning from data held by multiple owners, generally referred to as multi-institutional learning. FL distinguishes itself from other such approaches in that learning takes places at all participants simultaneously. Sheller et al. [21] compared multiple methods for multi-institutional learning, namely FL, institutional incremental learning, and cyclic institutional incremental learning. The latter two train the model at each participant sequentially by passing the model from client to client instead of aggregating multiple locally trained models in one place. Using the standard FedAvg approach applied to U-Net, a CNN architecture designed specifically for image segmentation tasks using fewer training images, they found FL to obtain the best score with the highest consistency across

a variety of dataset configurations.

FedAvg [22], by McMahan et al. is the seminal work in the FL literature. It outlines a basic FL algorithm where an initial global model is distributed from an aggregation server to clients participating in the FL scheme. Each client then trains their copy of the global model on their local data. The local models are then returned to the aggregation server which obtains the updated global model's parameters as a weighted average of the local models' parameters, where the weight of each client's contribution is determined by the portion of training samples it holds.

However, there are several hurdles to overcome before FL's potential can be fully realized. Of principal concern is the privacy of client-data. Although data is never exchanged between clients, Zhu et al. [1] and Geiping et al. [2] showed it is possible to reproduce training samples using the gradient updates alone, a problem known as gradient leakage. Another concern is attacks against the performance of the model itself with clients working together, tailoring their outputs to avoid detection. Byzantine attacks [23] use this approach to reduce model accuracy. Model backdoor [24] attacks also use this approach but to cause misclassifications for specific feature values only, without affecting the performance on the remainder of the feature space. These attacks either maliciously craft updates directly using a modified loss function (model poisoning [25]) or indirectly using modified samples (data poisoning [26]). In Sybil attacks [27], adversaries register extra simulated clients into the FL network whose behaviour help to accomplish malicious goals.

Privacy concerns were addressed in [28] by Li et al. by using differential privacy to distort gradients to provide a minimum privacy guarantee while minimally affecting accuracy. They also used selective parameter sharing in which only gradients of sufficient magnitude are shared and only after clipping. Zhang et al. [29] used the pallier HE to hide clients' gradients before aggregation. Bonawaits et al. [30] used SMC to provide a private vector summation framework for FL weight aggregation. Their framework is also resilient to clients dropping out of the FL network, ensuring results are still correct even if clients leave part way through the secure summation procedure.

However, even after addressing the threats of privacy and model performance attacks vis-á-vis client updates, it remains possible to infer the presence of a given sample in a training set using the finished model alone. This is known as a membership inference attack. In [31], Choquette-Choo et al. noted that data points used to train a NN are confidently classified by it and thus implemented various techniques for estimating a model's prediction confidence using only its outputted labels. They demonstrated that it was possible to predict a data point's membership in a training set with better performance than other methods that leveraged more than just label predictions (i.e. class probabilities). Shokri et al. [32]

used a different approach that trained several "shadow models" on corresponding "shadow datasets" generated using various techniques, such as features' marginal distributions. Using the outputs of the shadow models on their corresponding training data and a disjoint test set, an attack model was then trained to detect if a sample was in the training set or out of the training set of the shadow models, with the expectation that this predictive ability will extend to the model targeted by the inference attack. Hu et al. [33] built on membership inference attacks with the source inference attack, which estimated the FL participant that contributed a given training sample. In their approach, they calculated each client's updated model's loss on a given training sample and predicted the client whose update had the lowest loss as the source of said sample.

It is also possible to recreate training samples using only the final model, without the use of gradients or other data obtained during FL training. Such techniques are known in the literature as model inversion attacks. In [34], a generative adversarial network (GAN) was trained to create images that closely resembled real images using a dataset disjoint from the dataset used to train the model targeted by the attack. Then, they solved an optimization problem that penalized unrealistic images while maximizing the likelihood with the targeted model. The result was convincing recreations of images used during training.

Attacks like these make sharing the global model after training a possible privacy risk. This makes it necessary to rely on the private inference techniques of scenario 1, discussed in the previous section, to allow predictions on private data without sharing the global model. However, if participants in the FL network cannot be trusted, then an FL method that does not require sharing the global model with each participant at the end of each round of training would provide security against these types of attacks.

There have been several approaches designed to improve convergence with non-IID data. Chen et al. [35] addressed label quality disparity as a source of non-IIDness by adjusting the weight of clients' updates according to their credibility which is assessed using a public benchmark dataset. Li et al. [36] introduced FedProx, which improves on FedAvg by penalizing large updates with the addition of a "proximal term" to the clients' local objective function. The proximal term prevents local updates from pulling the global model away from the global optimum. Ye et al. [37] addressed non-IIDness due to low quality clients with varying resources by using contract theory to incentivize high quality clients to maximize their contribution so that most of the participating clients' data may be more similarly distributed.

Karimireddy et al. [38] introduced SCAFFOLD to account for "client drift," when a client's local optimum is not aligned with the average local optimum across all clients, by approximating the ideal unbiased local update which would be the average gradient of the

local model across all clients' data. [39] introduced FedDC which improved on SCAFFOLD by adding a loss term that allows clients to learn and correct their drift before sending updates to the aggregating server. Duan et al. [40] created Astraea to account for varying data and class distributions across clients by putting clients into groups with close to uniform distributions. Each group then performs standard FedAvg for some number of epochs before aggregating each group's jointly trained model into one global model. This is repeated until convergence. Wang et al. [41] created FedMA to address permutation invariance in NNs. There are many variants of a NN that only differ in the ordering of parameters, meaning learnt weights may not correspond across clients even though each client's local model is learning the same thing. FedMA permutes local model weights to align neurons across clients.

There are several approaches that give each client a personalized global model to improve performance on the local data distribution while still benefiting from other clients' data. Huang et al. [42] provided personalization by training a multi-view recommender system using standard FedAvg and differential privacy. The model provides predictions personalized to a specific client, a specific view (e.g. a software application) or both. Bui et al. [43] proposed FURL which provides personalization by learning a user embedding which is provided as supplementary information at inference time. In [44], Huang et al. introduced CBFL in which clients are clustered by representing each client as the average encoding of their data. Encodings are obtained using an autoencoder trained using a single round of FedAvg. Personalization is then provided by using standard FedAvg to train a different model for each cluster of clients. At test time, the test sample's encoding is used to determine which cluster's model to use for prediction. Liu et al. [45] presented FADL which first trains a global model using standard FedAvg, then clients receive a personalized model by freezing the first part of the global model before locally training the remaining layers.

In contrast to the typical approach with a centralized aggregating server, Guha et al. [46] presented a peer-to-peer FL adaptation of FedAvg where a peer was randomly selected for the job of aggregating other peers' models. In [47], Zheng et al. addressed the vertical FL scenario where each client possesses a different set of features for the same set of samples. They presented FL-LRBC, a LR model with bounded constraints to enable interpretability by keeping all weights positive. Privacy was ensured using the pallier HE scheme. Lui et al. [48] also enabled vertical FL by training a NN at each client which learns a common projection by using a loss function that ensures corresponding points projected using different clients' NNs are similar to each other in the common space and obtain the same labels. Privacy is guaranteed using HE. Yoon et al. [49] adapted continuous learning, where a model continuously trains on a sequence of tasks, to the FL setting, in an approach they call

FedWeIT. They used general (not FL specific) solutions to address catastrophic forgetting, where parameters trained on previous tasks drift towards the solution of the current task. However, they also provided a novel solution to ensure a task learnt at one client does not interfere with a task learnt at another by using a learnt sparse mask and attention mechanism to selectively incorporate knowledge from other clients.

2.2.1 Federated Dimensionality Reduction

Oftentimes, it is necessary to reduce the dimensionality of data to remove noisy or uninformative features. This improves the efficiency and performance of the resulting model. However, unlike in the centralized private ML setting, where the data owner can perform dimensionality reduction without the cooperation of other participants, in the federated setting, dimensionality reduction requires cooperation so that the resulting data projection is coherent across participants without the need for sharing raw data between them. Below, we briefly survey the many approaches which perform dimensionality reduction in the federated setting. We focus on one of the most common dimensionality reduction techniques, singular value decomposition (SVD) [50], which is used to decompose a matrix into three separate matrices with useful properties. For example, performing SVD on the covariance matrix of a data matrix or on the mean-centered data matrix itself yields its principal components. This process is called principal component analysis (PCA), a technique that finds a projection of a data matrix that preserves the maximum variance of the original data.

In Balcan et al. [51], each participant performs PCA on subspace embeddings of their local data and then sends the first t_1 local right singular vectors and values to the server. The server concatenates the singular vectors obtained from each participant and performs randomized PCA on the resulting matrix to get a close approximation of the first t_2 global singular vectors that would be obtained if PCA was performed on all the participants' data at once. They showed theoretically and empirically that the approximation introduces negligible error, while also significantly decreasing computation time. However, they did not mention privacy concerns.

Polat et al. [52] introduced private SVD-based collaborative filtering by masking user scores with random values sampled from either a uniform or normal distribution and recovering the desired statistics from the disguised values.

In Han et al. [53], the authors proposed privacy-preserving SVD for data shared by two parties, but left the extension to more parties as future work. They used a secure Gram–Schmidt process for the QR decomposition of data shared by two parties that relied on primitives built upon random shares and HE. The intermediate results at the end of each

iteration of the Gram–Schmidt process and the final SVD result were split into two private portions using random shares in order to preserve the privacy of each party's data.

Hartebrodt et al. [54] presented an iterative federated SVD algorithm for highly dimensional data in genome-wide association studies (GWAS). Their algorithm ensured only partial right singular values were returned to each participant. To this end, they also presented a federated Gram–Schmidt orthonormalization algorithm, which was used as a subroutine in their algorithm to ensure orthonormality of candidate eigenvectors at the end of each iteration. Full right singular values were kept private because Nasirigerdeh et al. [55] showed that sharing right singular vectors in conjunction with some federated linear regression techniques in GWAS pipelines causes data leaks.

In Hegedűs et al. [56], using a federated iterative gradient descent method, the authors computed the SVD of a data matrix A, the rows of which were partitioned among participants. Their approach yielded matrices X and Y containing scaled versions of the top left and right singular vectors, respectively, without blatantly exposing A. Randomly initialized approximations of the rows of X were partitioned among participants, while different randomly initialized approximations of Y existed at each participant. In each round, participants updated the current SVD approximation using their local data and sent the updated Y to a randomly selected peer. This was repeated iteratively until X and Y converged.

In Chai et al. [57], FedSVD was introduced, which performed lossless SVD with the help of a trusted masking server on data partitioned across various participants, using random orthogonal matrix masks (ROMMs) to ensure participants' data privacy was not compromised. In subsequent work [58], the authors showed that by replacing the ROMMs with blocked masks that consist of much smaller ROMMs along the diagonal, and by using tailored disk offloading techniques, FedSVD could factor a matrix containing 50 billion elements in 16.5 hours.

A major issue with the above approaches is that they do not consider label information which may be used for finding more useful projections for supervised learning tasks. To the best of our knowledge, there are no private federated supervised PCA or SVD methods in the literature, so we briefly list some centralized supervised PCA approaches below.

In Li et al. [59], the authors presented a parametric approach for modelling a data matrix while incorporating auxiliary information using a modified expectation–maximization algorithm to predict latent variables that provide the supervised SVD decomposition for said data matrix.

In Bair et al. [60] and Rakotomalala et al. [61] supervision was introduced into standard PCA by using a preprocessing step that removed features with a small regression coefficient or correlation coefficient, respectively, before applying PCA as usual.

In Barshan et al. [62], the authors were the first to propose a closed-form supervised principal component analysis (SPCA) as a generalization of standard PCA. Unlike standard PCA, which finds a subspace that retains the maximum variance of the input data, SPCA finds a subspace that retains the maximum dependence on the input data's labels. They also introduced kernel supervised principal component analysis (KSPCA), which uses the kernel method to extend SPCA to nonlinear mappings of the input data. We discuss the details of their approach in Section 5.1.1, where we extend it to the federated setting.

CHAPTER 3

Machine Learning in Precision Medicine to Preserve Privacy via Encryption

Private ML has many applications, but one of pressing concern, and the focus of this chapter, is precision medicine. Precision medicine is a departure from one-size-fits-all medicine toward the customization of disease treatment and prevention for individuals by leveraging their variability in genes, environment, and lifestyle. While significant progress has been made in processing personalized data in other domains, the ability to develop actionable knowledge that facilitates individualized care through precision medicine has lagged behind.

Despite the exciting prospects of precision healthcare, it faces several technical and societal hurdles related. Security and privacy concerns are such hurdles. While precision health provides tremendous benefits by enabling better care, it can lead to personal privacy breaches through genetic disclosure or genetic discrimination. To deliver targeted, personalized care, personal data (e.g. specific human genome sequencing) must be shared with many professionals in possibly diverse geographic locations or jurisdictions and sometimes over unreliable channels, such as the Internet. This poses several risks, such as insider threats, social engineering, distributed denial of service (DDoS), illicit data inferences, cyberbullying/blackmailing, etc. [63].

In principle, a ML model can be trained on either confidential or public data, allowing more training samples and data distributions and, therefore, more complex, predictive, and generalizing models. These complex models can theoretically achieve higher predictive performance and find novel associations within precision healthcare. Since these complex models require large datasets and time investments, they may not be made publicly available. Their owners may request that patient data be shared with them rather than the model being shared with other institutions. However, performing analytics on new cases provided by

hospitals or medical centers should be treated with the utmost concern for privacy, given the reasons introduced above. Therefore, a ML method that can perform inference on encrypted data while preserving the trained model's privacy is required.

The present chapter makes the following contributions:

- We propose a machine learning with encryption (MLE) framework that considers the requirements and constraints of private ML in precision medicine and facilitates performing analysis on a real precision healthcare dataset while preserving its privacy.
- We illustrate the framework's success by considering a case study using the MSK-IMPACT dataset, one of the most recent comprehensive genomic datasets in the field [6], and we produce a predictive model for cancer with higher accuracy than the most recent publications on the same dataset [64] — based on 5-fold cross-validation (CV) rather than an independent testing set, as will be detailed in Section 3.2 — while preserving the privacy of the cases as required.
- To facilitate the capabilities of different communities, such as software engineering, ML, and precision medicine, and for the extension and validation of this work, we provide the following as an open-source repository: (1) the system design and implementation of the MLE framework, (2) all the ML code and experiments on the MSK-IMPACT dataset, and (3) a free cloud service for medical practitioners to predict their own cases using the MLE framework.

3.1 A Machine Learning with Encryption (MLE) Framework

In this section, we propose a simple four-component system architecture to accommodate any of the eight scenarios of encryption in Table 1. This simple architecture, illustrated in Fig. 2, fulfills the privacy-preserving requirements that are mandatory for future ML-based precision medicine. The components of this architecture are explained below.

***Database* (tr)** is a reservoir for both publicly available genetic datasets, which do not require preserving privacy, and private datasets, which need encryption prior to public sharing. Whenever a new dataset is revealed, it can be added to this reservoir for more accurate future analytics.

***ML Construction* ($\mathcal{M}$)** is the engine that constructs models—including transformation, feature selection, resampling, etc.—from the datasets in the *database* module. This module

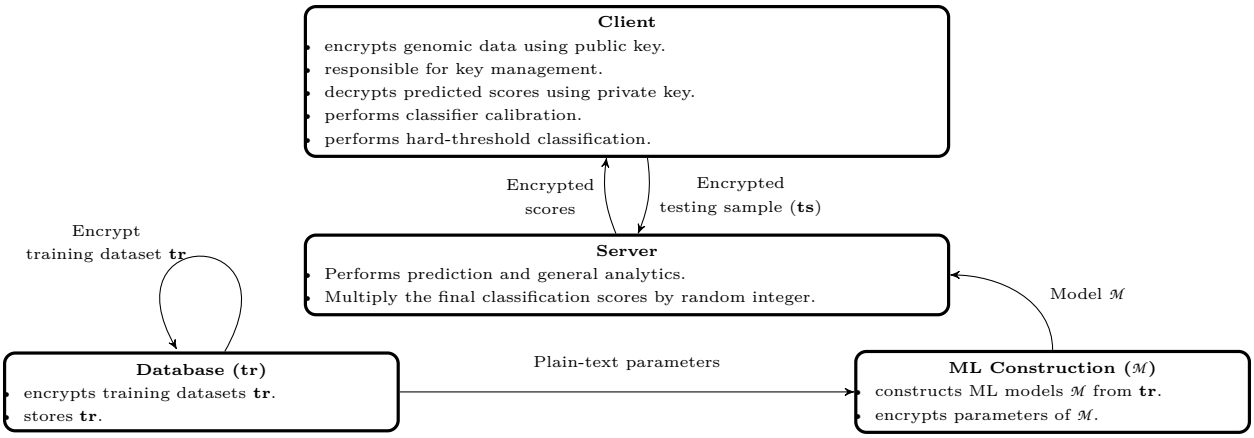

Figure 2: A block diagram of a privacy-preserving MLE framework for precision medicine that can accommodate any of the eight scenarios of Table 1. Each block contains the functionalities that can be performed in that block depending on the adopted scenario.

can be open-sourced for the entire community and can always be updated as new ML methods merge or more accurate models are constructed. In addition, the module can train on its own private dataset, which is not part of the *database* module, and then encrypt its model parameters $\mathcal{M}$. Alternatively, it can establish a protocol with the *database* module to train on the private dataset without encryption for a wider range of algorithms and then encrypt the model parameters to preserve the dataset's privacy (Cases 01x in Table 1).

***Client* (ts)** is where the testing data, which is probably sensitive and confidential, resides and needs analytics. The owner of this data can opt to encrypt it, and this encryption can be provided via simple software components installed on the *client* side available via communication with the *server*. Next, the encrypted testing data is sent to the *server* for prediction. Finally, the encrypted predicted scores are received back. The *client* should be responsible for setting the threshold on the scores for the final hard decision or classification. This is to achieve a required level of aggressiveness to control the per-class sensitivity, such as in the case of the binary classification problem, in which the threshold provides the trade-off between the sensitivity and specificity and thus controls the operating point on the receiver operating characteristic (ROC) curve.

Server is the cloud engine for prediction. On the one hand, it interfaces with the *client* to receive the encrypted dataset for prediction and sends back encrypted predictions, and on the other hand, it interfaces with *ML construction* to receive a particular predictive model. Based on the underlying encryption scenario (Table 1), the *server* receives the appropriate public key from these two modules. In addition, for the C-class classification

Table 2: Different configurations of feature pre-processing and classifiers tried on the dataset: $5 \times 2 \times 4 \times (8 + 36 + 12) = 2,240$ different configurations.

Feature Preprocessing			Classifier	Parameters
Transformation	Selection	Reduced p		
None Standardize MinMax $\log(x)$ $\log(x+1)$	MI χ^2	0 2500 3500 4500	LR	penalty: l1, l2; C: 0.1, 1, 10; solver: liblinear, lbfgs
			SVM	kernel: linear, poly, RBF; penalty: l1, l2; C: 0.1, 1, 10; loss: hinge, squared_hinge
			RF	criterion: gini, entropy; n_estimators: 100, 200, 300; bootstrap: False, True

problem and for greater privacy preservation for the model and/or the dataset (**tr**), the server can optionally multiply the scores $s_c(x), c = 1, \ldots, C$, where $x \in$ **ts**, by a random integer. This keeps the relative C scores unaffected. However, this disallows the *client* from inferring information about the model weights ($\mathcal{M}$) by sending pseudo-feature vectors in the form $x = (0, \ldots, 1, 0, \ldots)$ (only one feature is 1; the others are zeros).

To illustrate the utility of this simple architecture, we demonstrate how scenario 1 can be implemented in a very practical setup. When ML training is based on public data (**tr**), the weights of the trained model $\mathcal{M}$ are deployed on the *server* in unencrypted form, while the queries (**ts**) must be encrypted for security sensitivity. Under a hospital's public key, many parties may also be eligible to upload data (e.g. doctors and patients). The *server* is used for deploying ML implementations $\mathcal{M}$. In this case, the hospital sends encrypted data to the *server*. In the *server*, many computations can be done on the encrypted data and the results sent back to the hospital. Only the hospital can decrypt the data because the private key is provided only on the hospital side. In the following section, we conduct a large set of ML experiments under the MLE framework and scenario 1 on one of the most recent high-quality genomic datasets.

3.2 Experiments

In this section, we concisely describe the MSK-IMPACT dataset[1], explain the different ML experiments to build the predictive model, demonstrate computational aspects of the encryption process, and finally introduce the open-source platform of the whole project.

[1]More details are provided in the online supplementary materials at https://doi.org/10.1016/j.patrec.2021.07.004

Table 3: The best three configurations in Table 2. The first two are the LR and SVM, which are HE-friendly.

Feature Preprocessing			Classifier	Parameters	Accuracy (%)
Transformation	Selection	Reduced p			
Standardize	χ^2	2500	LR	C: 1; Penalty: l2; Solver: liblinear	77.47
None	χ^2	2500	SVM	kernel: linear; C:0.1; penalty: l2; loss: squared_hinge	77.23
Standardize	MI	2500	RF	criterion: gini; n_estimators: 200; bootstrap: False	73.92

Table 4: Accuracy and percentage of predicted class score rankings which agree with the full precision results, of the best model, at different multiplier precisions.

Precision	10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9
Accuracy (%)	73.88	77.41	77.47	77.41	77.41	77.41	77.41	77.41	77.41
agreeing ranks (%)	36.49	70.80	95.82	99.41	99.94	100.0	100.0	100.0	100.0

3.2.1 MSK-IMPACT Dataset

MSK-IMPACT, a clinical sequencing cohort dataset [6], comprises genomic patient records extracted from tumor-tissue samples taken from 10,336 patients. Since tumors are usually the results of many mutations, there are more than 100,000 discovered mutations. The dataset consists of 11 files linked together with `sample_ID` and `Patient_ID` and contains various information about the somatic mutations (i.e. genetic alterations acquired by cells that are the progeny of cancerous cells) within the genomic samples, including mutation signature, copy number alternation, and gene fusion data files. "*With maturing clinical annotation of treatment response and disease-specific outcome", according to [6], "this dataset will prove a transformative resource for identifying novel biomarkers to inform prognosis and predict response and resistance to therapy... Tumor molecular profiling is a fundamental component of precision oncology, enabling the identification of genomic alterations in genes and pathways that can be targeted therapeutically*".

The authors of the dataset tried to associate "biomarkers" with a particular type of cancer using simple methods of association, such as relative frequency. Then, to illustrate the usefulness of their DNA-sequence approach, they leveraged the Oncology Knowledge Base [65] to see how many of the mutations they detected (stratified by cancer type) were known to be actionable, that is, have an associated treatment or gene therapy. Recently, a subset of the MSK-IMPACT-2017 dataset containing 7,791 patients and with a portion

of these features, somewhat lightly engineered, was used by [64] with a more ML-oriented approach. Using LR, they achieved an overall accuracy of 73.8%—estimated by a 5-fold CV—in detecting the cancer type from genetic information; yet, their approach did not consider privacy preservation. They also achieved an accuracy of 74.1% on an independent cohort of 11,644 patients; however this independent testing set was not available to us.

3.2.2 Building the Model

In addition to the predictive power required for any ML model, the objective of privacy preservation via FHE requires the final ML model be FHE-friendly, that is, based only on addition and multiplication operations, as was explained in Section 2.1. Some ML models cannot satisfy both of these objectives. For example, a random forest (RF) has binary decision splits that are not FHE-friendly. However, although linear models, LR, support vector machine (SVM), and many others are all HE-friendly, they may not perform well on a particular dataset.

We obtained the same subset of the MSK dataset, used by [64], using the code referenced in their paper. Using this dataset, we tried 2,240 different ML configurations that were the cross products of five methods for transforming features, two methods for dropping the least important features, four values for the number of dropped features p, and three classifiers, LR, SVM, RF, each with several sets of tuning parameters. The detailed parameters of these 2,240 experiments are listed in Table 2. The ML experiments were run using the powerful Compute Canada Cedar Cluster[2]. The best three configurations of the 2,240 are listed in Table 3 and were achieved by LR, SVM, and RF, respectively. LR, after feature standardization, χ^2 selection, and dropping the least informative 2,500 features, achieved the highest accuracy of 77.47% (using 5-fold CV), which was higher than that obtained by [64] on the same dataset and features.

Although the feature values will be encrypted at test time, we do not have to encrypt the final model parameters (Eq. (1)) because SEAL with the CKKS HE scheme (see 3.2.3) allows the multiplication of a plain text number with an encrpyted number to obtain an encrypted result. The next section explains the encryption of the testing dataset. Since we are using FHE, we must convert all floats to integers. To do this, we scale all floats by some 10^d, where d is the number of decimal places included in the scaled floats, therefore controlling the computational precision. After scaling, we round off any remaining decimal to achieve the final integers. The effect of precision on the accuracy of the best model is illustrated in Table 4, where it is clear that a multiplier of 10^4 would be adequate.

[2] https://www.computecanada.ca/

Table 5: Effect of encryption parameters on encryption time. All libraries support automatic selection of `CoeffModulus`. NA indicates noise budget ~ 0

#	polyModulusDegree	PlainModulus	Time in Sec.
1	8192	2048	3456
2	2048	1024	96
3	1024	512	NA
4	1024	1024	NA
5	2048	1024	87
6	2048	512	88
7	2048	1499	94
8	2048	786433	89

3.2.3 Encrypting the Testing Dataset

The SEAL library

Different libraries exist for implementing HE [66]; among them, SEAL [16] is an open-source HE library developed by the cryptography and privacy research group at Microsoft. The library is written in C++ and can run in many environments. SEAL allows addition and multiplication to be performed on numbers. Other operations, such as encrypted comparison, sorting, and regular expressions, are in most cases not feasible on encrypted data using this technology. SEAL supports two FHE schemes: the Brakerski/Fan-Vercauteren (BFV) scheme, which allows modular arithmetic to be performed on encrypted integers, and the Cheon-Kim-Kim-Song (CKKS) scheme, which allows addition and multiplication on encrypted real or complex numbers, but this latter scheme yields only approximate results.

Computational Aspects of Encryption Operations

In this section, we explain how the encryption-prediction-decryption process is computed, which is summarized in Algorithm 1. From the previous section, a C-class LR model was the winner for this dataset; formally, this model is given by

$$\Pr(G = c|X = x) = \frac{\exp(s_c)}{1 + \sum_{l=1}^{C-1} \exp(s_l)}, \tag{1a}$$

$$s_c = w_{c0} + w_c' x, \ c = 1, \ldots, C, \tag{1b}$$

where, $C = 22$ types of cancer, the patient feature vector is $x \in \mathbf{R}^p, p = 5,599$; and the testing dataset $\mathbf{ts}$ to be encrypted has $N = 7,791$ patient records. By construction, the

MLE framework requires sending the encrypted score of each testing observation to the client rather than the final hard decision for trading off the types of error. In addition, from (1), the numerator is a monotonic exponential function and the denominator is only for scaling, so probabilities sum to 1. Therefore, it is sufficient to encrypt the linear term s_c and treat it as the final score sent to the *client*. We applied the BFV scheme implementation of SEAL to perform this weighted summation term. The encryption operations are explained as follows. (1) Scale and encrypt the feature list. (2) Multiply encrypted features by plain text weights and sum encrypted values along with the scaled bias step. (3) Decrypt the results and repeat step 2 for each set of coefficients, i.e. each class. We tested different encryption parameters to compare computational time. Table 5 illustrates the computational time required as a function of a subset of the parameter space. Rows 3 and 4, caused the ciphertext noise budget to reach zero. This noise budget is determined by the encryption parameters, and once the noise budget of a ciphertext reaches zero, it becomes too corrupted to be decrypted. Thus, it is essential to choose parameters large enough to support the desired computations; otherwise, the correct result is impossible to obtain, even with the secret key. The values in row five give the best average per sample prediction time after testing on the entire dataset (7791 records), which spanned over seven days of computations on an i7core–2.5GHz–16G machine. From Eq. (1b), this time is obviously $T = NC\big((p{+}1)(E{+}M{+}A){+}D\big)$, where E, M, A, and D are the encryption, multiplication, addition, and decryption times, respectively. During this experiment, `IntegerEncoder` was used to encode integers to BFV plain-text polynomials. `IntegerEncoder` is easy to understand and uses simple computations; however, there are more efficient approaches such as BatchEncoder, which can be investigated in future works.

3.2.4 MLE Framework: Opensource and Deployment

The official repository of this project [67] contains three sets of open-source resources. (1) The Python code that produced all the ML experiments from Section 3.2 is offered, organized, and commented on to challenge the ML community to develop more accurate, predictive models. (2) The system design and client-server implementation and implementation code of the MLE framework from Section 3.1 is offered to the software engineering community to propose more system functionalities. (3) A free cloud service that encapsulates the best ML model and the client-server design is offered as a simple end-user interface to individuals in the medical field to test on particular cases. We hope this kind of dissemination to different communities helps the evolution of privacy-preserving ML for precision medicine.

3.2.5 Discussion

Our results attest to the promise of HE in the precision healthcare domain. We can see from Table 5 that predictions can be obtained in a reasonable amount of time with the simple computing resources of an i7core–2.5GHz–16G machine. In addition, the results in Table 4 indicate that despite losing very small precision when encrypting real numbers, identical performance to that of the plain text case can be achieved with only a modest rescaling (10^4) of the weights and feature values—much of the full precision in real values is superfluous. It is remarkable that despite the limitations on model type and mathematical operations (as a price paid by choosing the HE), we were able to exceed the 5-fold CV accuracy of cutting edge approaches on the same dataset (see Table 3). The results prove the feasibility and practicality of augmenting genomic analysis with HE and show that in this case very little, if anything, is lost while ensuring complete patient privacy.

3.3 Summary

Toward building privacy-preserving ML models for precision medicine, this chapter has made three contributions. First, we proposed and implemented a MLE framework that accommodates different scenarios for encrypting the ML training-testing process. Second, and most importantly, we analyzed the recent high-quality clinical sequencing cohort dataset MSK-IMPACT and provided a predictive model that is both secure and outperforms the most recent predictive model built for the same dataset. Third, we offered the ML, software engineering, and precision medicine communities free resources: respectively, the client-server implementation of the framework, the Python code of all the ML experiments, and a cloud service to test genomic cases. These offerings contribute to the evolution of the privacy-preserving analytics of precision medicine.

Algorithm 1 Encrypted Class Prediction: N, M, K denote number of samples, features, and classes, respectively.

Require: Array of samples X

Ensure: Predicted class probabilities Y_{pred}

Client Side

$d \leftarrow$ Select precision

$X \leftarrow$ Load $N \times M$ feature array

$P_{priv} \leftarrow \text{KeyGenerator}_{\text{SEAL}}()$

$X = X \times 10^d$

$X_{enc} = \text{Encrypt}_{\text{SEAL}}(X,\ P_{priv})$

sendToServer(X_{enc}, d)

Server Side

$W \leftarrow$ Load $K \times M$ weight array

$W = W \times 10^d$

$S_{enc} \leftarrow$ Initialize to zeros an $N \times K$ weighted sums array

for each sample n in [0,1, ...N] **do**

  for each class k in [0,1, ...K] **do**

    for each feature m in [0,1, ...M] **do**

      $S_{enc}[n][k]$ += $\text{MultPlain}_{\text{SEAL}}(X_{enc}[n, m], W[k, m])$

sendToClient(S_{enc})

Client Side

$S = \text{Decrypt}_{\text{SEAL}}(S_{enc},\ P_{priv})$

$Y_{pred} \leftarrow$ initialize $N \times K$ predicted class matrix

for n in [0,1, ...N] **do**

  $Y_{pred}[n] = \text{softMax}(S[n])$

return Y_{pred}

CHAPTER 4

FLAMED: Federated Learning via Aggregating Multivariate Estimated Densities

In this chapter, we propose a new alternative framework for FL that uses multivariate density estimation to bypass the challenges posed by non-IID data (see Section 1.1), while also making simultaneous privacy and performance protection more convenient. The detailed algorithm is presented in Section 4.1.1, but in short, each client models the probability distribution of their local data and sends this information to the aggregating server, allowing the aggregating server to simulate centralized global training. This is a general framework in that the methods used for modelling client distributions and for obtaining a global model from the information shared by clients can be decided upon by the practitioner. The methods used in this chapter are but one specific implementation, while our framework offers the flexibility to adopt more efficient or performant techniques. We call our approach federated learning via aggregating multivariate estimated densities (FLAMED).

The present chapter makes the following contributions:

- FLAMED is a general framework for simulated centralized learning that serves as a conceptual basis for alternative FL methods and allows non-IIDness, privacy, and model security to be addressed simultaneously.
- To the best of our knowledge, FLAMED is also the only approach that enables the aggregating server to obtain a global model that is not known to any other participants. Restricting knowledge of the global model to a single participant secures the model's intellectual property and guards against a malicious participant using shared model weights to attack training data privacy (see membership inference attack [31] and model inversion attack [34]).

- We evaluated our approach against baseline and state-of-the-art FL approaches on a variety of synthetic datasets and a real-world healthcare dataset from a federated setting with 132 participants. This sparse dataset contains 3,069 features and demonstrates FLAMED's ability to handle high-dimensionality data.
- We performed a security analysis of the proposed framework and evaluated its resilience against backdoor attacks as defined in [24] using the real dataset.
- We present a technique for detecting model backdoor attacks via data poisoning. The proposed approach is applicable to not only FLAMED, but any case where data are aggregated from multiple clients.
- Our proposed method, experiments, and discussion serve to compare and contrast the maturing FL literature with an alternative approach, FLAMED, and highlight the relative advantages of both areas of research.

4.1 Method

4.1.1 FLAMED: The General Framework

As discussed in Section 1.1, non-IID data pose a significant challenge when applying FL in real-world scenarios. Non-IID data cause local models, which are constructed client-side, to be biased towards the local solution conditioned only on the locally held data. This slows convergence towards the global solution conditioned on all the data held across all clients or prevents it entirely. Following [3], there are five types of non-IIDness. Simply put, for any observation x in the IID setting, we have $x \sim P$, while in the non-IID setting, we have $x \sim P_i$ for each client C_i, where $P_i \neq P_j \ \forall i \neq j$. Although different approaches in the literature address or mitigate manifestations of non-IID data in different ways, we are not aware of any approach that explicitly addresses all five types. In order to bypass all manifestations of non-IIDness, we need to estimate the maximum likelihood estimator (denoted *MLE*, italicized, to distinguish it from machine learning with encryption, which was discussed above in Chapter 3) $\widehat{P}^{MLE}$ of the global data distribution P.

Theorem 4.1.1. *FLAMED bypasses all manifestations of non-IIDness by approximating the global MLE $\widehat{P}^{MLE}$ of the global data distribution P.*

Proof: It is obvious that $\widehat{P}_i^{MLE} \neq \widehat{P}_j^{MLE}$, where $\widehat{P}_i^{MLE}$ is the *MLE*, called the empirical nonparametric distribution, that simply puts a mass of $1/n_i$ on each observation, where

n_i is the number of observations at C_i. In FLAMED, each P_i is modeled using a density estimation technique to obtain $\widehat{P}_i$; then one of two approaches is possible:

1. Each client simulates a dataset $\widetilde{X}_i \sim \widehat{P}_i$, which the server aggregates into a global dataset.

2. The server aggregates the estimated $\widehat{P}_i$, or its summary statistics, from each client, which the server uses to simulate a global dataset.

The first approach is the default in FLAMED and what we adopt in the present article's proof of concept implementation, which can be seen as just one of many possible implementations of the first general approach. In both approaches, a global model is constructed at the server from the global dataset, which then follows a mixture distribution

$$\widehat{P} = \sum_i \alpha_i \widehat{P}_i, \ \sum_i \alpha_i = 1, \tag{1}$$

where the weight of the convex combination, α_i, controls the importance of each client, and the distributions $\widehat{P}_i, \ i = 1, \cdots, K$ are the empirical distributions $\widehat{P}_i^{MLE}$ of the data simulated at the client side (approach 1) or the estimated distributions themselves (approach 2). Therefore, the aggregating server obtains $\widehat{P}$, an estimation of $\widehat{P}^{MLE}$.

■

In this way, FLAMED simulates a centralized learning task, thus bypassing all forms of non-IIDness resulting from varying data distributions $P_i \neq P_j$. Furthermore, in contrast to standard FL, there is only a single round of communication. This means each client can contribute once they are available, and the aggregating server can wait to train the global model only when all clients have contributed, without holding up other participants. Therefore, in addition to bypassing non-IIDness resulting from differing P_i, FLAMED also addresses non-IIDness resulting from client selection bias due to nonuniform client availability.

The general framework of FLAMED is shown in Algorithm 2. In the following, $i \in [1, 2, ...K]$ is the client index, and K is the number of clients. FLAMED proceeds as follows:

1. (Optional) Each of the clients uses a dimensionality reduction technique to obtain a transformation $\mathcal{T}$ of their combined data $X = [X_1, ..., X_K]$ such that $\mathcal{T}(X)$ has a dimensionality low enough to enable tractable density estimation, distribution modeling, etc., depending on the method used to derive the global model. This step is optional for low-dimensionality datasets, but in most practical settings, it is essentially mandatory.

2. The clients then performs the statistical analysis sufficient for the learning task on their transformed data (if $\mathcal{T}$ was skipped, the analysis is performed on their raw data), and the resulting information I_i (e.g. $\widetilde{X}_i$ or $\widehat{P}_i$ for approach 1 or 2, respectively) is sent to the aggregation server. We stress that here we are only giving the schema of a general approach which contrasts standard FL approaches. A specific implementation is given in Section 4.1.3.

3. Once the server has I_i from each client, it constructs a global model $\mathcal{M}$, which is sent back to each client.

An important consideration during step 2 is to take care that clients are not being asked to expose sensitive information. For example, summary statistics or a simple scaled histogram may be considered permissible for sharing with the server but, kernel density estimation (KDE), which uses observations from the given data sample in its estimated density function, would leak these observations if said density function was sent to the server. However, in cases where sharing the estimated density function would compromise privacy, clients could follow approach 1, simulating the data themselves and sending only the simulated data to the server. This is why we consider approach 1 to be the default for FLAMED. Further, some applications may require that prototypes and summary statistics are not learned by the aggregating server, but it is not clear if standard FL approaches are capable of keeping such information private, and in many cases, these extreme restrictions are not necessary (see discussion in Section 4.2.1).

4.1.2 FLAMED: Practical Benefits

FLAMED has several practical benefits that distinguish it from traditional FL methods:

- Many FL use cases are resource constrained. FLAMED offers an alternative to standard FL methods, which require numerous rounds of communication and training on participants' devices. FLAMED requires only a single execution of dimensionality reduction and density estimation and a single communication of low-dimensionality simulated data. This alone justifies the development of FLAMED, as many use cases simply may not permit standard FL. For example, low-powered participants can only perform FL for short periods of time when other tasks are paused.

- Clients do not need to be consistently available for the duration of the FL learning process. They only need to communicate a single time once they have the opportunity. In comparison, standard FL typically requires hundreds if not thousands of rounds of communication.

Algorithm 2 FLAMED General Framework

Input: Data $X = [X_1, ..., X_K]$

output: Global model $\mathcal{M}$

parallel for i in 1 to K **do**: \\`At Client i`
    if dimensionality reduction **then**:
        $X_i' \leftarrow \mathcal{T}(X_i)$
    $I_i \leftarrow$ StatisticalAnalysis(X_i')
    Send I_i to server
end for
$I \leftarrow$ Aggregate(I_i) \\`At Server`
$[\mathcal{M}_1, ..., \mathcal{M}_g] \leftarrow$ GlobalModel.GridSearch(I)
Send $[\mathcal{M}_1, ..., \mathcal{M}_g]$ to clients
Receive test scores $[t_1, ..., t_g]$
Send $i = \text{argmax}([t_1, ..., t_g])$ to clients

- If a client is found to have poor-quality data (whether for benign or malicious reasons), the aggregation server can simply reconstruct the global model while imposing less weight (even zero) on that client (Eq. (1)).

- Similarly if a client's data drifts, or if a new client joins the FL network, the global model can be updated by reconstructing it after including the new information in the aggregation. In short, the participation of all clients is not required in order to update the global model.

- There may be several ways to assess the quality of a client's contribution, depending on the type of information supplied to the server (e.g. anomaly detection) because each contribution can easily be individually included or excluded from global model training. This has the dual benefit of making detecting malicious contributions (see Section 4.4.3) and evaluating the impact of specific contributions, and therefore the associated reward, easier.

- FLAMED allows for the streamlined expert design of the global model because grid search can be performed with little coordination or communication overhead. This is in contrast to standard FL methods, where the entire procedure must be repeated for each choice of hyperparameters or model architecture. In practice, FLAMED allows grid search to be performed in one round of communication. The server trains g global models for each hyperparameter choice and sends the g trained models to each client,

who reports each model's evaluation score on their local test data. This only requires two rounds of communication total.

4.1.3 FLAMED: Specific Implementation and Parameters

The specific algorithm and implementation of FLAMED used in our experiments is depicted in Algorithm 3. In summary, dimensionality reduction is performed to enable tractable density estimation to simulate new data in the new transformed space. Then, the simulated data from each client is sent to the aggregating server to train a global model.

In our experiments, to contend with the curse of dimensionality and make simulation tractable, we consider SVD for dimensionality reduction. SVD decomposes matrix $X \in \mathbb{R}^{m \times n}$ as $X = U\Sigma V^\top$, where the columns of $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are called the left and right *singular vectors* of X, and $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix whose entries are called the *singular values* of X. Using the r columns of U corresponding to the r largest singular values in Σ, denoted $U_{:r}$, a low dimensional transformation $\bar{X} \in \mathbb{R}^{r \times n}$ is obtained with $\bar{X} = U_{:r}X$.

However, this approach cannot be directly applied to our problem because each client would obtain a different U_i, each biased towards their local dataset, and it would not be obvious which to apply at inference time. Thus, we use FedSVD [57], which, with the help of a trusted masking server, can compute $U_{:r}$ of the combined data matrix $X = [X_1, ..., X_K]$, which is composed of all K clients' data matrices X_i, without compromising the privacy of any of the clients' data. Once each client receives $U_{:r}$, they compute the common r-dimensional transformation of their data. For low-dimensional datasets, we can skip this step and immediately simulate the original data. In our experiments, to ensure FedSVD was tractable, we sampled 20,000 observations across all clients, with each client contributing to this total in proportion to the size of their local dataset. Because FedSVD requires a masking server, the specific implementation of FLAMED in the present section also requires this second server. However, this is not an inherent trait of FLAMED.

We test two different simulation methods. The first is KDE, which maps any point in the feature space to estimates of the probability density using a weighted sum of kernel distances from said point to each observation in the training set. In our case, we use the Gaussian kernel. The second method is the Wasserstein conditional generative adversarial network with gradient penalty (WCGAN-GP). A GAN [68] is an ML framework that trains two NNs, a generator that generates synthetic data by sampling a vector from random noise and transforming it, and a discriminator that attempts to distinguish the synthetic data from the real data. Working against each other, the generator learns to create more realistic observations in an attempt to fool the discriminator. Class labels can be supplied

to the generator to specify the class of generated observations, this is known as a conditional GAN [69]. Adjusting the standard GAN to minimize the Wasserstein metric improves the convergence rate and ensures the simulated data captures the full complexity of the real data distribution [70]. The standard Wasserstein GAN uses weight clipping to keep theoretical considerations satisfied during training. However, [71] found this approach biases the GAN towards simpler functions. They remedied this by penalizing the norm of the discriminator's gradient. We also perform grid search with LR and a feed-forward neural network (FFNN) to determine the best architecture for the global model. The time, space, and communication complexities for the general FLAMED framework and the specific implementations in our experiments are shown in Table 6.

Algorithm 3 FLAMED Using Simulation

Input: Data $X = [X_1, ..., X_K]$

output: Global model $\mathcal{M}$

if applying FedSVD **then**:
    FedSVD$(X_1, ..., X_K)$ \\`Clients receive` $U_{:r}$
parallel for i in 1 to k **do**:\\`At Client i`
    if applying FedSVD **then**:
        $X_i' \leftarrow U_{:r} X_i$
    else
        $X_i' \leftarrow X_i$
    $P_i \leftarrow$ KDEorWCGAN-GP(X_i')
    $\widetilde{X}_i' \sim P_i$
    Send $\widetilde{X}_i'$ to server
end for
$\widetilde{X}' \leftarrow [\widetilde{X}_1', ..., \widetilde{X}_K']$ \\`At Server`
$[\mathcal{M}_1, ..., \mathcal{M}_g] \leftarrow$ GlobalModel.GridSearch$(\widetilde{X}')$
Send $[\mathcal{M}_1, ..., \mathcal{M}_g]$ to clients
Receive test scores $[t_1, ..., t_g]$
Send $i = \text{argmax}([t_1, ..., t_g])$ to clients

4.1.4 Comparison Approaches

We compare FLAMED with FedAvg, FedProx, and FedDC. We included FedAvg, which comes from a frequently cited paper with over 8,000 citations, as a baseline. The authors of FedProx and FedDC proved convergence rates, giving these techniques a strong theoretical

foundation. Table 11 in the survey presented by [72] shows FedProx reported the largest accuracy increase over FedAvg. FedDC is a recent approach that reportedly outperforms FedAvg, FedProx, and other popular approaches from the literature. We thus include FedProx and FedDC as a comparison to state-of-the-art FL techniques. In the following, we discuss the defining features of the comparison approaches, specifically their different objective functions. We discuss only what is relevant to the present work and refer the reader to the original papers for more detail.

FedAvg uses weighted summation to average the gradient updates across all clients:

$$w_{t+1} = \sum_{i=1}^{K} \frac{n_i}{n} w_{t+1}^i, \tag{2}$$

where n_i and n are the number of observations at client i and across all clients, respectively, while w_{t+1}^i and w_{t+1} are client i's weight update and the global model after training round t, respectively.

FedProx uses FedAvg aggregation, but adds a proximal term h_i to the clients' objective functions, which penalizes large updates to prevent a client's local update from pulling the global model away from the global optimum. Formally,

$$\min_{w_{t+1}^i} h_i = L_i(w_{t+1}^i) + \frac{\mu}{2}||w_{t+1}^i - w_t||^2, \tag{3}$$

where $L_i(\cdot)$ is the typical empirical local loss, μ scales the strength of the proximal term, and w_t is the previous global model's weights.

FedDC also uses FedAvg aggregation, but with several key changes. First, added to the clients' objective functions is the gradient correction term G_i, which penalizes large variations in model updates, as measured with $g_{t+1}^i = w_{t+1}^i - w_t^i$. They also add the term R_i, which penalizes the difference between $h_{t+1}^i + w_{t+1}^i$ and w_t, where h_t^i is the learned "local drift variable." The full objective function is

$$F(w_{t+1}^i; h_{t+1}^i, w_t) = L_i(w_{t+1}^i) + \frac{\mu}{2} R_i(w_{t+1}^i; h_{t+1}^i, w_t) + G_i(w_{t+1}^i; g_{t+1}^i, \mathbb{E}_{i\in[K]} g_{t+1}^i) \tag{4}$$

In this way, the drift of the local model away from the global model can be learned by using the partial derivative of h_{t+1}^i with w_{t+1}^i and w_t fixed. This drift can then be corrected for before sending local updates for aggregation. In practice, local drift variables are approximated with $h_{t+1}^i = h_t^i + (w_{t+1}^i - w_t^i)$ in order to save compute time.

Table 6 shows the time, space, and communication complexities for FedAvg, FedProx, and FedDC implemented with a basic FFNN.

Method	Time	Space at Client	Space at Server	Communication	
				Size	Rounds
FLAMED(General)	$O(D) + O(A) + O(G)$	$O(D)$	$max(O(A), O(G))$	$O(K(size(I_i) + size(\mathcal{M})))$	2
FLAMED(KDE/FFNN)	$O(n_I^2) + O(lm^2ne)$	$O(n_i)$	$O(lm^2)$	$O(nm + Klm^2)$	2
FLAMED(GAN/FFNN)	$O(lm^2n_Ie) + O(lm^2ne)$	$O(lm^2)$	$O(lm^2)$	$O(nm + Klm^2)$	R
FedAvg/Prox/DC(FFNN)	$O(lm^2n_IeR)$	$O(lm^2)$	$O(lm^2)$	$O(RKlm^2)$	R

Table 6: Time, space, and communication complexities for all methods. D, A, and G are placeholders for density estimation, aggregation, and building the global model, respectively. n, n_i, and n_I are the total number of observations across all clients, the number of observations at client i, and the max number of observations at a single client, respectively. KDE complexities are based off the naive direct evaluation in [73]. Assuming a FFNN with l layers no larger than the number of features m, O(lm^2) computations result from multiplying a $1 \times m$ matrix of layer activations by an $m \times m$ weight matrix for each layer. Accounting for n observations, e epochs, or R rounds, we arrive at the given results.

4.2 Security

In the following, we discuss the security features of FLAMED. We start by outlining our threat model and giving a proof of security in the context of FL. Then, we discuss the security advantages FLAMED holds over standard FL approaches.

4.2.1 Threat Model

We assume any subset of the K participants, including the aggregating server, could be malicious and use any means necessary to attempt to learn something about a particular data sample x belonging to a benign participant. Although in some settings, it may be inadmissible to allow global properties of data distributions to be leaked, it is not obvious that standard FL approaches can prevent this [1, 2, 74]. Most data privacy approaches, ours included, focus on the privacy of any particular sample, whereas hiding functions of distributions is a separate goal altogether (see [75, Sec. 2] for a detailed discussion on the matter). Further, the European Union's General Data Protection Regulation (GDRP), a major incentive for the development of FL algorithms in the first place, only applies to "personal data," which is data relating to an identified or identifiable individual (see [76, Sec. 2.1.2]). Therefore, we allow knowledge of empirical probability densities of the private data to be learned by adversaries.

Further, since FLAMED is a general method, its security would have to be proved for each individual implementation. Specifically, if approach 1 is followed, it must be shown that sharing $\widetilde{X}$ conforms to a particular privacy requirement, which is application-dependent. If

approach 2 is followed, sharing $\hat{P}_i$ must be shown to conform to a particular privacy requirement (e.g. sharing an approximation of KDE's PDF is proven secure in [77]). There are many varieties of privacy requirements. Differential privacy requires that any synthetic datasets generated from neighbouring private datasets (i.e. datasets that differ by one element) have a near equal probability of occurring [78]. This can be accomplished by using methods like PrivBayes for generating synthetic datasets [79]. We leave investigation of this approach to future work. Below, we prove the security of the specific FLAMED implementation defined in 4.1.3 using a weaker privacy requirement. Specifically, we require that FLAMED releases no certain information about a particular sample x_i.

Theorem 4.2.1. *FLAMED is secure with respect to our threat model. That is, FLAMED leaks no certain information about a particular* $x_i \in X = [X_1, ..., X_K]$ *other than* $\widehat{P}$.

Proof: In approach 1, where each client simulates a dataset $\widetilde{X}_i \sim \widehat{P}_i$, the server only receives $\widetilde{X}_i$ from each client, which cannot be used to accurately reconstruct any particular x_i with certainty. The server could only use $\widetilde{X} = [\widetilde{X}_1, ..., \widetilde{X}_K]$ to construct an empirical probability density function that approximates the estimated density $\widehat{P}$ from Eq. (1). Even if the server colludes with all but one participant j, therefore obtaining $\widetilde{X} = [X_1, ..., X_{j-1}, \widetilde{X}_K, X_{j+1}, ..., X_K]$ (i.e. the concatenation of private data belonging to clients $i \neq j$ and simulated data from client j), the server will obtain at best a closer approximation to $\widehat{P}$, which does not leak any certain information about a particular x_i at the non-colluding participant. Similarly, in approach 2, the server only receives $\widehat{P}_i$ from each client, which, according to our assumption, also cannot be used to accurately reconstruct any particular x_i. Therefore, in both approaches, so long as care is taken in specifying $\widehat{P}_i$ when using approach 2, the server does not learn any certain information about a particular x_i. Conversely, the clients may only obtain, at most, the global model trained on $\widetilde{X}$, although this is optional. Even if they were able to reconstruct much of the input data with model inversion attacks, they would have less certain information than the server when colluding with all but one participant, and, at best, they would only be able to reconstruct $\widehat{P}$. Therefore, FLAMED leaks no certain information about a particular x_i other than an approximation of $\widehat{P}$.

■

4.2.2 Security Advantages

As discussed in Section 2.2, there are several attacks on the standard FL aggregation scheme that use the weight updates sent from clients to reconstruct training observations [1, 2]. This

means that privacy can be compromised when using FedAvg, FedProx, FedDC, and other standard FL methods if no other measures are taken, such as adding noise to training data. In FLAMED, because gradients are not exchanged, this type of attack is ineffective. These gradient-based attacks are also more effective against datasets with lower dimensionality, and this is exactly where we expect to see FLAMED perform best because, due to the curse of dimensionality, statistical analysis better represents these datasets. Further, [24] showed that directly manipulating the gradient updates using a maliciously modified loss function (model poisoning) is far more effective than manipulating the training data alone (data poisoning). This makes FLAMED inherently more robust against model backdoor attacks because no gradients are exchanged, as will be demonstrated in Section 4.4.3. For the default FLAMED approach and the specific implementation adopted in the present work, where data are simulated at the client side and then shared with the server (Algorithm 3), the aggregating server can examine simulated data for signs of poisoning. Conversely, in many cases, standard FL approaches must hide each client's gradients from the server, which must therefore blindly aggregate them.

Importantly, FLAMED addresses non-IIDness, gradient leakages, and gradient poisoning simultaneously. Other approaches require observing raw gradients and sharing client state information to correct biases due to non-IIDness or to detect poisoned gradients. This makes securing such methods against gradient leakage attacks more difficult. Similarly, addressing gradient leakage by obscuring raw gradient information makes correcting bias and detecting data poisoning more difficult. FLAMED presents no such trade-off and allows careful analysis of the information used by the aggregating server (i.e. simulated data), without exchanging gradient information.

Lastly, to the best of our knowledge, FLAMED is the only FL method that allows the aggregating server to obtain a global model that is not known to other participants by default. To share this property with FLAMED, all standard FL methods would require intensive and complex redesign. It is easy to see the use cases for such an FL method. For example, consider an FL scenario with untrusted participants, such as cell phone users. The participants may want to isolate the trained model at the aggregating server to maintain the aggregating server's sole proprietorship of the global model or to strengthen privacy guarantees because sharing the global model with participants may allow them to perform model inversion attacks [80], exposing participants' private data to one another.

4.3 Experiments

In this section, we present the configurations for our experiments. We begin by presenting our experiments which compare the performance of FLAMED to leading standard FL approaches from the literature. Then, we present experiments comparing FLAMED's robustness to attacks on model performance to the same standard FL approaches used in the previous experiment.

4.3.1 Performance Comparison: Setup and Configurations

The following experiments were designed to compare the performance of FLAMED to several FL approaches from the literature using both synthetic datasets, offering versatile customization options, and real-world datasets, which give a more realistic indication of FLAMED's performance.

Synthetic Dataset

To evaluate FLAMED against the FL techniques under a variety of scenarios, we used synthetic datasets generated by modifying the code published by the authors of FedProx, who themselves followed an approach similar to [81]. In the following, c is the number of classes, m the number of features, and n_i the number of observations held at client C_i. Following FedProx, for each client, we generate a local dataset $(X_i \in \mathbb{R}^{n_i \times m}; Y_i \in \mathbb{R}^{n_i})$ by first sampling each element of the mean vector v_i from $\mathcal{N}(B_i, 1), B_i \sim \mathcal{N}(0, \beta)$ and then using v_i to define the multivariate normal $x_i \sim \mathcal{N}(v_i, \Sigma)$, where the covariance matrix Σ is diagonal with $\Sigma_{j,j} = j^{-1.2}$. Therefore, β controls how much local data differs across clients. Next, we sample weights $W_i \in \mathbb{R}^{c \times m}$ and biases $b_i \in \mathbb{R}^c$ from $\mathcal{N}(u_i, 1), u_i \sim \mathcal{N}(0, \alpha)$. Class labels in Y_i are then determined with $y = \text{argmax}(\text{softmax}(W_i x + b_i))$. Thus, α controls how much the relation between labels Y_i and features X_i differ across clients. Following FedProx, we set α equal to β.

Where we depart from FedProx is in the configurations of the synthetic datasets used for experiments. The mean number of observations held across all clients is determined in proportion to the number of features as ρm, where ρ is an experimental hyperparameter. The distribution of the number of observations across all clients is either uniform (i.e. $n_i = \rho m \forall i$) denoted $\mathcal{U}$, or a modified log-normal distribution $\mathcal{LN} = n_i^* + \frac{\rho m}{2}$ where $n_i^* \sim \text{lognormal}(\mu, 2)$ and μ and 2 are the mean and standard deviation, respectively, of the underlying normal distribution; μ is chosen such that $E[n_i^*] = \frac{\rho m}{2}$, and thus, the mean number of observations at each client C_i is $E[\mathcal{LN}] = E[n_i^*] + \frac{\rho m}{2} = \rho m$. We also ensure no class has only one observation

so that it cannot appear in both the train and test sets.

Our experiments were divided into successive sets:

- **Initial experiments**: we applied FLAMED to the set of experiments containing 1,536 synthetic distributed dataset configurations defined by the cross product: $K \in \{2, 4, 8, 16\} \times c \in \{2, 4, 8, 16\} \times m \in \{8, 32, 128, 512\} \times \rho \in \{5, 10, 20\} \times \alpha = \beta \in \{\text{IID}, 0, 0.5, 1\} \times \mathcal{D} \in \{\mathcal{U}, \mathcal{L}n\}$. The case where data are independent and identically distributed across clients is denoted by $\alpha = \beta = \text{IID}$ and corresponds to setting $v_i = \vec{0}$ and $u_i = 0, \forall i$.

- **High non-IIDness**: After our results from these initial experiments (discussed in Section 4.4.1), we wanted to further study how FLAMED handled higher levels of non-IIDness. Therefore, we repeated our initial experiments, but with $\alpha = \beta \in \{1.5, 2\}$, adding 768 configurations.

- **High number of clients**: In light of our results on the real dataset (Section 4.4.2), and to further supplement our initial experiments with larger numbers of clients as a source of non-IIDness, we repeated our initial experiments, but with $K \in \{32, 64, 128\}$ and $\rho \in \{5, 10\}$ except the configurations with $K = 128 \wedge m = 512 \wedge \rho = 10$ were excluded due to time constraints. This added 736 configurations.

- **Low dimensionality**: To test the feasibility of plain simulation without FedSVD and evaluate FLAMED in lower dimensionality, we repeated our initial experiments as above, but with $m \in \{2, 4, 8\}$ and $\rho \in \{10, 20, 40, 80\}$ without first applying FedSVD. Higher values of ρ were used because for low dimensionalities, there would be very few observations per client, making simulation difficult. These experiments added another 1,536 configurations.

In total, our experiments covered 4,576 configurations. In the cases where $K \geq 64$, we ran FedSVD without the secure aggregation protocol [30]. Secure aggregation was used in the summation of the masked matrices across clients, but since it is lossless, its exclusion does not affect our results. It was omitted to save compute and memory resources because, in our experiments only, the secure summation requires holding K^2 one-time pads in memory at once. Normally, each client would only need to hold a much more manageable K one-time pads in memory, but in our case, clients are simulated on a single machine.

Real Datasets

To evaluate FLAMED in a real-world federated setting, we used the eICU Collaborative Research Database [82]. This dataset contains real-world medical data from over 200,000

ICU admissions to more than 200 medical centres across the United States. Unlike other commonly used datasets, the eICU dataset represents a real-world federated setting instead of a contrived one obtained by separating a centralized dataset. Therefore, it provided a more realistic evaluation of FL techniques. Further, the datasets commonly used for comparison in FL papers (e.g. MNIST, CIFAR-10) consist of high-dimensionality image data. Because our proposed approach uses simulation, which is not tractable in higher dimensions, we target low dimensionality tabular data. We use only the data contained in the drug infusions table, which tracks the drugs administered to patients during their stay in the ICU. Each row in our feature matrix $\mathbf{X}$ corresponds to a patient, while each column corresponds to a drug. If a patient i receives any dose of a certain drug j across all of their ICU admissions, then $\mathbf{X}_{ij}$is set to 1. Otherwise, it is set to 0. We specify a binary classification problem where a patient is assigned label 0 if their hospital discharge status specified in the patient table is "alive", and 1 otherwise. After removing any hospital with less than 10 observations, we are left with a total of 3,069 features and 72,959 patients held across 132 hospitals. Although this dataset has a large number of features, it is very sparse. Thus we suspect it can be reduced to lower dimensionality while retaining an adequate amount of information. In this way, we can push the limits of FLAMED with FedSVD dimensionality reduction. In addition to using all 132 clients, we test different configurations defined in the set $K \in \{2, 4, 8, 16, 32, 64, 128\} \times \mathcal{S} \in \{\text{smallest}, \text{middle}, \text{largest}\}$. Here, K is the number of hospitals used and $\mathcal{S}$ denotes the strata of hospitals we select from. That is, if $\mathcal{S} = \text{middle}$, then we select the K hospitals with nearest to the median number of patients, while if $\mathcal{S} = \text{smallest}$ or $\mathcal{S} = \text{largest}$, then we select the K hospitals with the least or most number of patients, respectively. This adds 22 configurations to our experiments. As already stated, we do not use secure aggregation in FedSVD when $K \geq 64$.

Model Parameters

For FedAvg, FedProx, and FedDC, we performed 200 rounds of standard FL to train a FFNN. Intermediate layers have $\lfloor \frac{m}{2} \rfloor$ neurons, unless $m = 2$, in which case they have 2 neurons. Other parameters are determined through grid search using balanced accuracy for evaluation to account for class imbalances. For FLAMED, FedSVD is used to transform the dataset into $r \in \{2, 4, 8\}$, dimensions except when it is skipped in the low-dimensionality configurations. Grid search is used to determine the optimal simulation parameters which minimize the log-likelihood score of a held-out local test set for KDE and the Wasserstein training loss for WCGAN-GP. After simulation and the training of a global model, the optimal values for r and the hyperparameters of the global model are determined using balanced accuracy on a

Method	Parameter Space		Evaluation Metric
FedAvg	Number of layers:	{1,2,3}	Balanced accuracy
	Activation function:	{ReLU, linear}	
FedProx/FedDC	Number of layers:	{1,2,3}	Balanced accuracy
	μ:	{0.001, 0.01, 0.1, 1}	
	Activation function:	{ReLU, linear}	
FLAMED	KDE bandwidth:	{0.01,0.5,0.1,0.5,1,5}	{Log-likelihood on held out 10% of training data }
	WC-GAN noise dimensionality:	{2,4}	Loss on training data
	WC-GAN number of layers:	{3,4,5}	
	WC-GAN hidden layer dimensionality:	{32,64,128}	
	WC-GAN gradient penalty coefficient:	{0.1, 1, 10}	
	Global model complexity:	{LR, 2 layer NN, 3 layer NN }	Balanced accuracy
	Global model hidden layer size:	Same size as input data	

Table 7: Grid search parameter space and evaluation metrics used to determine optimal hyperparameters for each method

validation set that consists of ≈10% of each local dataset. For the global model, LR and a NN were compared. The parameter space searched and the evaluation metrics used for all methods are shown in Table 7. Parameters not included in the grid search, such as learning rate or decay and the bounds of our grid search, were selected using manual optimization on a subset of configurations or using recommended values from the literature. We used the FedProx and FedAvg implementation used in the original FedProx paper and the FedDC implementation used in the FedDC paper. Scikit learn's [83] KDE and LR were used with WCGAN-GP and the global model NN implemented in PyTorch [84].

4.3.2 Security Analysis: Setup and Configurations

As previously discussed, FL algorithms that exchange gradients are vulnerable to privacy attacks [1, 2], which use weight updates sent from clients to reconstruct training observations. Since privacy is a requirement for most FL use cases, often mandated by regulating authorities, these attacks must be protected against with a high degree of confidence. A popular technique to protect client gradient privacy, and therefore client data privacy, without sacrificing accuracy is the secure multiparty vector aggregation framework by [30].

However, another serious threat faced by FL algorithms are poisoning attacks. Poisoning attacks target model performance, attempting to cause incorrect classifications by either using manipulated observations (data poisoning) or directly manipulating model updates

(model poisoning). Model backdoor attacks [24] use poisoning to cause the global model to only misclassify observations with certain feature values, called the backdoor, without affecting the overall accuracy of the global model. This is accomplished by causing the global model to learn a specific trigger that, when present in an observation, causes it to be misclassified as a specific class desired by the attacker. The remainder of the training proceeds as normal so that observations that do not contain the backdoor are not misclassified.

Ideally, we want to detect these attacks during training so the model's integrity is preserved once deployed. Common methods for detecting such attacks rely on computing gradient norms and statistics using the plain text gradient updates from each client. However, these types of defenses are not compatible with the above-mentioned secure vector aggregation because client gradient updates are not visible during aggregation.

Since FLAMED is not vulnerable to gradient-based privacy attacks, and secure aggregation adequately addresses such privacy concerns in the standard FL approaches, we only compare the effectiveness of backdoor attacks on FLAMED and the comparison approaches. For this, we use the full eICU dataset after preprocessing as described in Section 4.3.1. For all methods, we vary the number of attacking clients, using the same clients, those with the nearest to median number of observations, in each poisoning experiment. We also use the same backdoor, setting column 377 to 1 with target label 1. We chose this as the backdoor because only one observation has this column set to 1, and it has the label 0. In this way, the poisoning objective is not too difficult for the attacker because it is not contradictory to the majority of the benign data. It is not the trivial case either, where the backdoor is attempting to induce classifications that a model trained on only benign data would have produced anyway. The poisoned training data consists of the backdoor, with all other columns set to 0, except for some noise in the form of random columns set to 1 so that each of the poisoned observations have the real data's average feature value sum. This helps the model to generalize the learned backdoor so it is more robust, while also making the poisoned observations' superficial statistics align with those of the benign data.

For attacking FedAvg and FedProx, we follow the model poisoning approach presented in [24], reusing much of their code. We also perform the backdoor attack via data poisoning as a baseline comparison with the data poisoning attack against FLAMED. At the server, we test two different defenses that were also presented in [24], computing the cosine distance and the L2 distance between each client's weight update and the global model. It is assumed that updates with higher L2 or cosine distances are anomalous and represent poisoning attempts. In practice, these defenses cannot be deployed in conjunction with secure aggregation; however, we report their effectiveness here as a best case scenario. Further, also following [24], in order to evade detection, the attacker modifies their loss function to include an "anomalous

loss" term. This term penalizes weight updates with large L2 or cosine distances, depending on the defense deployed by the aggregating server (see [24, eq. 4]). In this way, we have two evasion techniques, each with a specific loss term that corresponds to the aggregating server's defense method. We also test the ability of the server to detect model poisoning attacks when no evasion is used by the attacker; that is, the anomalous loss term is ignored. For attacking FedDC, which was not used in [24], we use a similar approach, weighting the anomalous loss term with $1 - \alpha$. However, where the adversarial clients in [24] only use the class loss weighted by α and the anomalous loss, ignoring the "proximal term" in FedProx, we use the sum of the three terms in the FedDC loss weighted with α, along with the anomalous loss term. We vary the strength of the attackers' poisoned update and the weight of the anomalous loss term using the hyperparameters denoted γ and α in the original paper. To represent a worst-case scenario for the defenders and really test all methods involved, we allow the attackers to be selected in every round of the federated training process.

For FLAMED, we are confined to data poisoning, that is, generating poisoned observations and to send to the aggregation server for training, because no gradients are exchanged, so we cannot directly manipulate them. We inject the poisoned training data before the FedSVD dimensionality reduction and vary the number of poisoned training observations as a percentage p of the training data at the attacking client(s). We assume that the backdoor, being inserted by only a few clients and by definition not present in the original data, will be rare. Therefore, any observations containing the backdoor will be an outlier so we use anomaly detection to find poisoned observations. We test six different defense methods by applying two anomaly detection algorithms, local outlier factor (LOF) [85] and isolation forest [86], to the simulated data, to the centroid of each client's simulated data and to the centroid of each class's simulated observations for each client. The full set of attack configurations and parameters are displayed in Table 8.

To evaluate the poisoning attacks, we use the backdoor success rate (BSR) which is the percentage of observations in the poisoned test data that fool the global model into predicting the target label. The poisoned test data contain the same backdoor and noise as the poisoned training data, with the noise serving to test the backdoor's robustness. We also record the area under curve (AUC) of the L2 and cosine defenses by assigning positive labels to model updates that were poisoned either by direct manipulation or by training on poisoned data and using the L2 or cosine distance as the predicted scores. We compute the AUC for FLAMED's poisoning defenses by assigning positive labels to the poisoned observations and the centroids of simulated data that contain poisoned observations, while using the outlier score as the predicted scores. The attacker aims to insert an effective model backdoor with a high BSR while also going undetected, resulting in a low AUC for the defense method

Method	Parameter Space		Defense
Any	Number of attackers:	{1,2,4}	
Model Poisoning	γ:	{50,75,90}	L2 and cosine distance
	α:	{1,0.7,0.5,0.4}	None
	Aggregator:	{FedAvg, FedProx}	
Data Poisoning	Number of poisoned observations:	$n_i \times p \in \{0.01, 0.05, 0.1, 0.2, 0.3, 0.5$ $0.8, 1, 2, 4, 8, 16, 32, 64\}$	LOF with n neighbours in {2,4,8,16,32,64} Isolation forest with n trees in $\{2, 4, 8, 16, 32, 64, 128, 256, 512\}$

Table 8: Backdoor attack and defense parameters used in poisoning experiments

used. This is also the only result that is bad for the defender, because in any other case, the attack is either detected or ineffective. We note that this is the first work we are aware of that studies backdoor attacks against FL using tabular data that was actually distributed in the real world and not split among clients in a contrived manner.

4.4 Results

In this section, we present the results of our experiments using synthetic datasets, our experiments using the real-world eICU dataset, and our experiments investigating FLAMED's robustness to model poisoning attacks. Alongside the results of each experiment, we provide commentary and analysis outlining their significance.

4.4.1 Synthetic Data

In real-world FL settings, entirely IID datasets are exceedingly rare. We also stand to gain more by learning from non-IID datasets, because each client's contribution tells us something different about the global learning task. Therefore, we are more interested in the non-IID configurations than the entirely IID configurations or the overall performance across both. We thus provide separately the average balanced accuracy scores, including and excluding the configurations with entirely IID data. Throughout the analysis of our results, we also put emphasis on the non-IID experiments. The averaged balanced accuracy scores for each method across all dataset configurations in our initial experiments are shown in Table 9. FedAvg, FedProx, and FedDC were the overall winners, performing mostly at par, but FLAMED, using either KDE or WCGAN-GP, remained competitive in the non-IID experiments. This can already be considered a success because, in addition to

FLAMED's performance (which was within 3% balanced accuracy of the best performing method across all nonIID settings), it offers the security advantages that are discussed in Section 4.2 and demonstrated later in Section 4.4.3. It also offers the practical benefits mentioned in Section 4.1.2, which may make FLAMED the only option in some constrained application settings. We saw a noticeable decrease in the performance of both FLAMED methods when entirely IID configurations were included, but as discussed, this setting is of little concern to us, and we include it here for completeness only. Simulation with WCGAN-GP requires tens of thousands of epochs to train and more hyperparameters to optimize during grid search. Despite the large reduction in complexity, using KDE for simulation still results in slightly better performance in most cases. Therefore, due to time constraints, we excluded FLAMED with WCGAN-GP from the extended experiments.

		FedAvg	FedProx	FedDC	FLAMED:KDE	FLAMED:WCGAN-GP
Balanced	All Configs. Except $\alpha = \beta =$ IID	0.7962	0.8009	**0.8023**	0.7741	0.7725
Accuracy	All Configs.	0.8092	**0.8125**	0.7952	0.7226	0.7111

Table 9: Balanced accuracy for each method averaged over the initial synthetic dataset configurations including and excluding the entirely IID configurations.

It was not expected that any one approach would perform best across all scenarios, and as we will see, the results in Table 9 represent only a superficial glance at the true utility of each method. To identify interesting subsets of our synthetic experiments, we break down the results with respect to configuration parameters found to heavily affect the relative performance of the compared methods. Because our approach was motivated by bypassing the challenges presented by non-IID data, we first consider the effect of the non-IID control parameters $\alpha = \beta \in \{\text{IID}, 0, 0.5, 1\}$. The overall results show that high non-IIDness helped FLAMED, motivating us to extend our experiments with $\alpha = \beta \in \{1.5, 2\}$.

The results with respect to the level of non-IIDness for our initial experiments and extended experiments with greater levels of non-IIDness are shown in Fig. 3. The plot shows the average balanced accuracy across all configurations with a given value of $\alpha = \beta$ for each method. We can see that the first introduction of even slight non-IIDness resulted in a very large improvement in the performance of FLAMED relative to FedAvg, FedProx, and FedDC. This trend continued as the non-IIDness increased, albeit with diminishing returns, until FLAMED with KDE scored just 0.007 average balanced accuracy below the best performing standard FL approach, FedDC. The fact that the difference between the entirely IID configurations and configurations where $\alpha = \beta = 0$ alone is so great illustrates the importance of considering the overall performance and the performances on just the non-IID configurations separately.

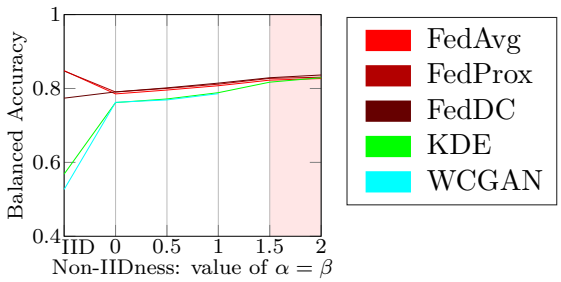

Figure 3: Average balanced accuracy for each method across initial experiment configurations (unshaded) and configurations with high non-IIDness (shaded) with respect to the level of non-IIDness. FLAMED with WCGAN-GP was excluded from the high non-IIDness experiments.

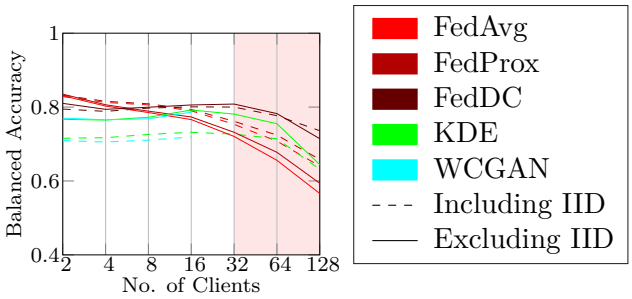

Figure 4: Average balanced accuracy with respect to the number of clients across the initial (unshaded) and extended configurations with a greater number of clients (shaded). For $K > 32$ we remove $\rho = 20$, and when $K = 128$ and $m = 512$ we remove $\rho = 10$. FLAMED with WCGAN-GP was excluded from the extra experiments with more clients.

FLAMED performed worse on IID data regardless of the simulation method used, suggesting it is an issue affecting this implementation of the framework with FedSVD. Because the comparison methods do not behave in the same way, we cannot consider this an artefact of the synthetic dataset generation. Rather, it is likely due to FedSVD not being able to preserve information relevant to classification in these settings. As we will see, when FedSVD is not used, or when the dimensionality is low enough that FedSVD can preserve nearly the entire feature space, the relationship reverses and the less complex non-IID data are easier for FLAMED's resulting model to classify correctly. The reason FedSVD is not able to preserve all relevant information in the IID setting may be due to it not necessarily preserving maximum dependence on the class label.

In light of FLAMED's performance with respect to higher non-IIDness as controlled by $\alpha = \beta$, it is interesting to examine how the number of clients affects the results given that it is another source of non-IIDness. Fig. 4 shows, for each method, the balanced accuracy with respect to different numbers of clients averaged across all configurations in our initial experiments and in our extended experiments with a greater number of clients. We show both the averages including the entirely IID configurations and the averages excluding them with a solid and dashed line, respectively. The plot indicates that as the number of clients increased, the relative performance of FLAMED improved. When not considering entirely IID configurations, FLAMED with KDE performed better than FedAvg and FedProx, and

was competitive with FedDC, if the number of clients $K \geq 16$. It can also be seen that when $K > 32$, the performance for all methods declined. This is because for $K > 32$, we remove $\rho = 20$, and when $K = 128$ and $m = 512$, we remove $\rho = 10$ as well due to computational constraints. This means all methods have fewer observations to learn from relative to the number of clients than for other values of $K \leq 32$.

A principal characteristic of any ML problem is the number of features being used for prediction. In our case, because transformation to a low dimensionality feature space is required for tractable simulation, the number of features in the raw data is a bottleneck, and studying its effect should highlight he true potential, as well as some limitations, of FLAMED. Fig. 5 shows the average balanced accuracy for each method with respect to different dataset dimensionalities for the initial experiments and the extended set of low-dimensionality experiments that did not use FedSVD feature reduction. Again the performance both including and excluding the entirely IID setting is shown. From the figure, we can see that in the initial experiments, when the dimensionality was lowest, FLAMED with KDE performed best. The results extend almost seamlessly when FedSVD is not used in the extended experiments with dimensionality $m \in \{2, 4, 8\}$. The results of the initial experiments and the extended experiments at $m = 8$ are not an exact match because, in the low-dimensional experiments, the inclusion of $\rho \in \{40, 80\}$ and the exclusion of $\rho = 5$ slightly improved FLAMED's results because higher values of ρ synthesize a greater number of samples at each client.

We also demonstrate that there are some scenarios where FLAMED with FedSVD performs well, even with up to 128 features. Fig. 6 shows the average balanced accuracy for each method with respect to different dataset dimensionalities for the experiments with a greater number of clients and high non-IIDness. The results show that when non-IIDness is increased by increasing the number of clients, FLAMED with FedSVD excels, even at higher dimensionalities, where simulation is not tractable without first using dimensionality reduction. This outcome is corroborated by our results on the real dataset discussed in Section 4.4.2. Further, when non-IIDness was directly increased by using higher values for $\alpha = \beta$, FLAMED's performance improved compared to the initial experiments (see right side of Fig. 5). The results also show that for the experiments with a greater number of clients, as the number of features grew, the performance improved for all methods. The same trend is present, but to a lesser degree, in the initial experiments (see Fig. 5). Because the same behaviour was exhibited by all methods, it can be explained as an artefact of the synthetic dataset generation, as this is the only common denominator. Specifically, because each weight in the matrix W_i (see Section 4.3.1) is sampled identically and independently for all features and for each class, as the number of features grows, so too does the likelihood that informative features (i.e. features with large weights) for different classes at different

clients do not contradict. Therefore, weight updates are less likely to contradict each other and hamper convergence, while density estimations are less likely to overlap and confuse the global model. Said differently, as the dimensionality increases, class separability is improved. We can also see that as the number of features increases, the performance increases more in the extra experiments with a greater number of clients. This supports our hypothesis, because with dimensionality constant, the likelihood of overlapping distributions increases with the number of clients and the overcrowding effect is more significant. This also explains why both with and without FedSVD, when the number of features is low ($m \leq 8$), the inclusion of IID data helps FLAMED's performance, since non-IID data are more likely to have overlapping class densities. Once the number of features is large enough, non-IID densities are unlikely to overlap.

Taken all together, our results show that there are several scenarios where FLAMED outperforms the standard FL methods, forming a seemingly contiguous region of the configurations' parameter space. That is, the region where the number of features is low, so as to keep simulation tractable (of course this is dependent on the number of informative features, which determines SVD's ability to successfully preserve all meaningful information) and where non-IIDness is high because of highly heterogenous client distributions or a large number of slightly heterogenous client distributions.

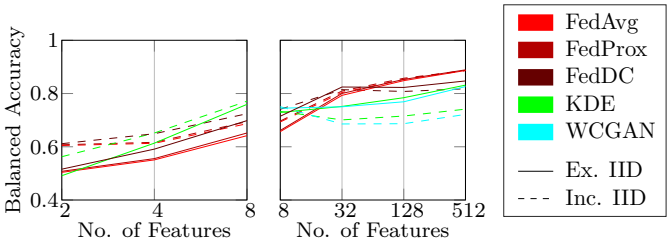

Figure 5: Average balanced accuracy with respect to the number of features for each method across all configurations in the initial experiments (right) and the extended experiments with low dimensionality which did not apply FedSVD before simulation (left). Note: FLAMED with WCGAN-GP was dropped after the initial experiments.

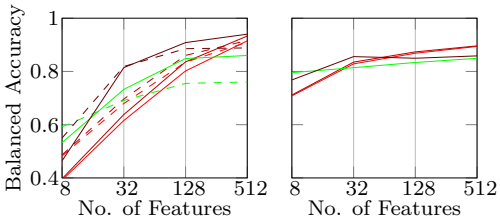

Figure 6: Average balanced accuracy with respect to the number of features for each method across the extra configurations with more clients (left) and higher non-IIDness (right).

4.4.2 Real Data

The eICU dataset is a real-world healthcare dataset from a real-world federated setting. Not only are the data from real patients, but the distribution of observations (patients) among

clients is not arbitrary or contrived, as in many other works. Rather it is a distribution of observations and resulting non-IIDness that exists in the real world. For these reasons, we consider the results on this dataset to be potentially the most important in the present work. We display the results of our experiments on the eICU dataset in Fig. 7. The plot shows the best balanced accuracy achieved by each method across varying numbers of clients. As described earlier, we applied each method to the K clients with the smallest, nearest to median, and largest numbers of observations. It is important to note that the test set is taken from each client that was used in training, meaning the higher accuracies of some methods in some of the experiments with fewer clients does not indicate that training with fewer clients results in a better global model. Unsurprisingly, when clients with a smaller number of observations were used, the relative performance of FLAMED was lowest because many observations are needed to reliably train a simulator versus a classifier, and the smallest clients have only a handful of observations ($n_i < 20$). When larger clients were used, the relative performance of all methods became much closer. As in our previous experiments, when we had a high number of clients, FLAMED had good performance, beating FedAvg and FedProx when using more than 32 of the clients with the largest number of samples, and remaining competitive with FedDC. Most significantly, as shown in the bottom plot, FLAMED performed best (with 0.6% higher balanced accuracy than the second best approach), when using the full eICU dataset, excluding only the three clients with the least number of samples. FLAMED uses simulated data for model training, and the three excluded clients likely contributed poorly simulated data, given that not enough local data were present to obtain an accurate model of the data distribution at these clients. Further, the best performing standard FL methods all used two layers and $\mu = 0.1$ when applicable, which was in the center of our grid search for optimal hyperparameters. This indicates that the reason FLAMED perform relatively well was not because the comparison methods had better hyperparameters lying outside our grid search area.

4.4.3 Security Analysis: Poisoning Comparison

The performance of FLAMED just discussed is encouraging, but predictive accuracy alone is not the sole metric of evaluation. As previously mentioned, FLAMED has natural advantages when it comes to security and privacy because gradients are not exchanged. Here, we present the results of our experiments comparing the practicality of model backdoor attacks against FedAvg, FedProx, FedDC, and FLAMED.

In Fig. 8, we show the resulting AUC and BSR of the backdoor attacks via data and model poisoning against FedAvg, FedProx, and FedDC, as described in Section 4.3.2. Each point on

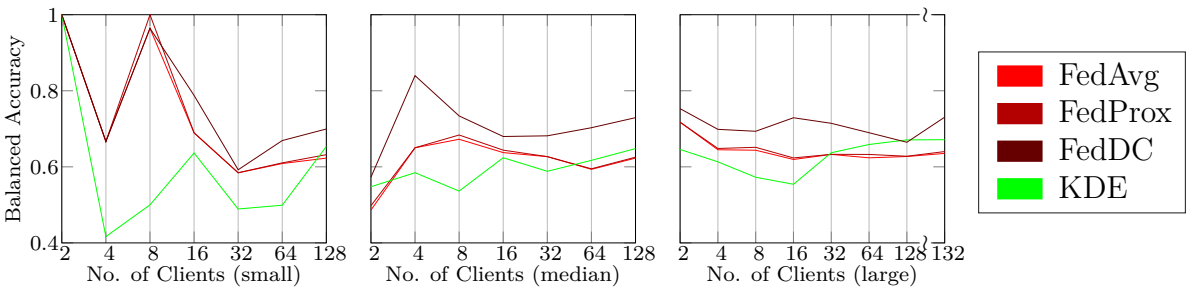

Figure 7: The balanced accuracy for each method evaluated on the eICU dataset with varying number of clients using the clients with the smallest (top left), nearest to median (top right), and largest (bottom) number of samples.

the four plots corresponds to different attack parameters. In the top left plot, data poisoning was used, in which case no evasion can be performed by the attacker while the maximum AUC from either the L2 or cosine defense is presented. In the remaining plots, model poisoning is used. In the two bottom plots, the AUC resulting from the specified evasion technique's corresponding defense method is used. In the top right plot, no evasion technique is used, so the maximum AUC from either the L2 distance or cosine distance defense is presented to illustrate the best case scenario for the defender, in which the attacker is not aware of which defense method the server is using. The results indicate there are many instances where the model poisoning attack is successful, with over 90% BSR, while going undetected by the defense methods used. However, the data poisoning attack was always detected with a high AUC, even when it was unsuccessful. This corroborates the results from [24] and highlights the strength inherent in FLAMED, which does not exchange gradients but only simulated data, meaning it is not possible to perform the stronger model poisoning attack against it. Regardless of the FL method and the defense method used, or lack thereof, there were attack configurations where the BSR was greater than 90% and the AUC lower than 60%, with minimal effect on the benign learning task, meaning the attacks were undetected while being potent.

We also note that in these experiments, FedProx and FedAvg were implemented in Pytorch, while in the real dataset experiments, we used the FedProx authors' TensorFlow implementations. Therefore, the similar benign accuracy across both sets of experiments helps validate both implementations.

The reader may notice the AUC is below 50 and think that, therefore, the defender can flip the threat score prediction to achieve a ∼80% AUC, but that would be incorrect. The AUC is so low because the attacking clients have the median number of observations, which is much less than many of the other clients who, having a larger variety of observations, would

have learned more information and thus give larger weight updates. Therefore, the poisoned updates appeared more modest than many of the benign updates. Removing any update that appears small like the evasive poisoned updates would remove many benign updates as well, heavily damaging accuracy. Further, the evasion techniques used were conceived of in accordance with Kerckhoffs's principle [87]. Therefore, if the defender was to also remove any update that appeared too small, we would assume the attacker to be aware of this and accordingly improve their evasion technique. This is beyond the scope of the present work, in which we aim to simply recreate the attack and defense techniques used in [24]. Additionally, the AUCs presented here all represent a best case scenario because, in practice, due to privacy concerns, the client updates are not visible to the aggregating server.

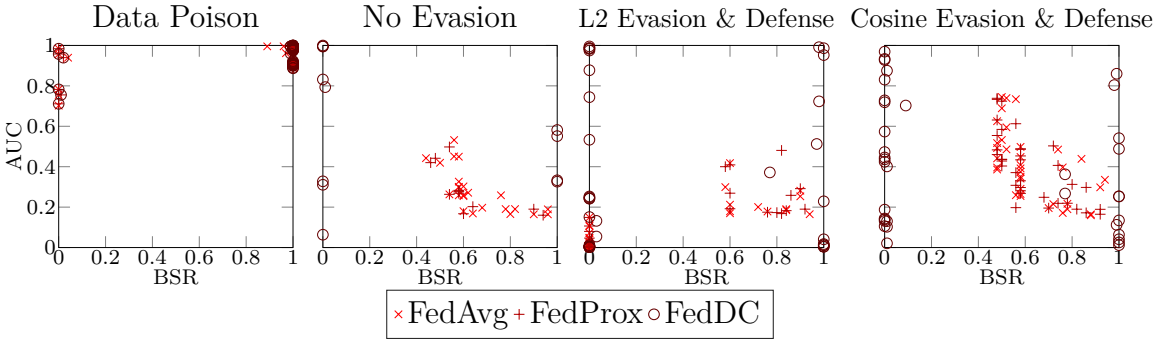

Figure 8: The AUC of the defense method used and BSR obtained with the different poisoning attack configurations, each represented by a mark with a different colour for each standard FL method. In the top left plot, the attacker used data poisoning, so they could not use any evasion technique, but for illustration purposes, the maximum AUC from either the L2 or cosine defense is presented. In the remaining plots, model poisoning was used. In the top right plot, no evasion technique was used. Here, we again show the maximum AUC from the L2 or cosine defense. In the two bottom plots, L2 and cosine defense were used by the server, and the attacker used the corresponding evasion technique.

Next, in Fig. 9, we show the results of the backdoor attacks via data poisoning against FLAMED with KDE. Isolation forest used on the raw simulated data was the clear winner, scoring a high average AUC regardless of the attack configuration used. This demonstrates the security advantage of FLAMED. Unlike the standard FL methods, FLAMED enables effective attack detection without exposing plain text gradients to the aggregating server. It should also be noted that many of the data poisoning attacks achieved poor results, but there is no reliable method for an attacker to determine a good attack parameter setting (i.e. the number of poisoned observations) without knowing the number of observations held at each client. Even then, the optimal number of poisoned observations may be dataset dependent.

Instead, attackers must hope to correctly guess how many poisoned observations to use. In doing so, the attackers risk an ineffective attack or being discovered by the aggregating server, which can then simply drop the attackers' observations and train the global model without any coordination from the remaining clients. Therefore, a data poisoning attack on FLAMED is impractical. This is not the case for model poisoning, where the authors in [24] present methods for obtaining good attack parameters with little prior knowledge of the FL network. Since, gradients are not exchanged in FLAMED, model poisoning attacks cannot be used against FLAMED. Thus, FLAMED has broad resistance against both types of poisoning attacks.

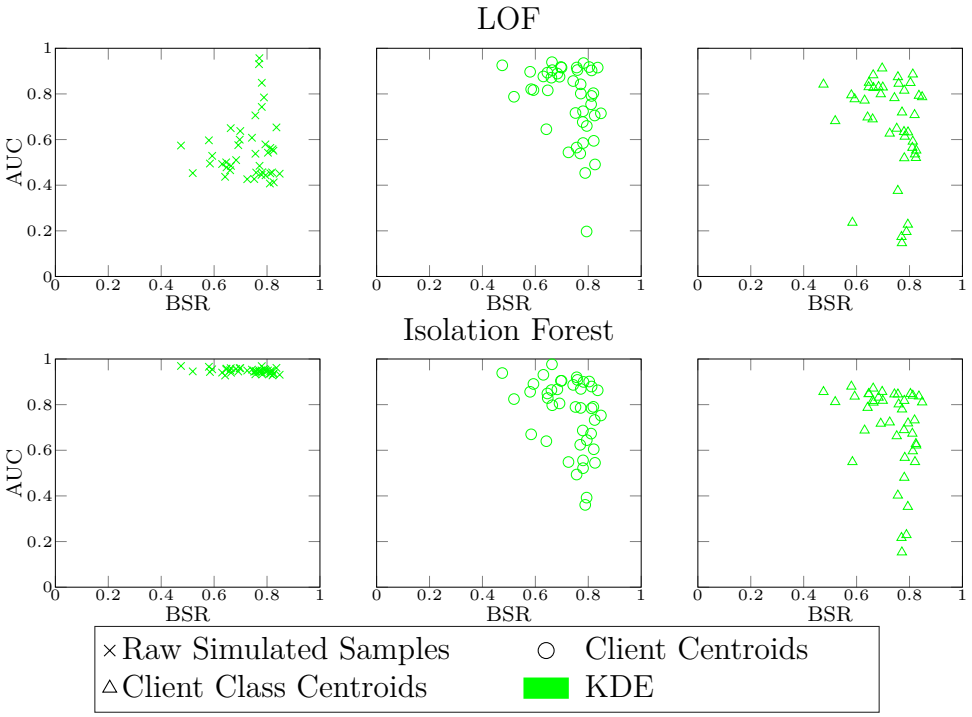

Figure 9: Each plot shows the BSR (x-axis) of the data poisoning attacks against FLAMED versus the average AUC across all parameter settings (e.g. number of neighbours) for the corresponding defense method (y-axis). Repeated marker shapes correspond to different attack configurations. Results with LOF (top 3 plots) and isolation forest (bottom 3 plots) are broken down by whether raw data, client centroids, or client class centroids were analyzed (see Section 4.3.2).

4.5 Summary

In the present chapter, we introduced a generic and specific implementation of the FLAMED FL framework for learning from non-IID data while preserving security and privacy. We compared FLAMED with FedAvg, FedProx, and FedDC on a variety of synthetic datasets and found that in cases where accurate simulation is most tractable, such as when dimensionality is low, the proposed approach performs best. Furthermore, with greater non-IIDness, due to increasing the total number of clients or the heterogeneity of client distributions, the proposed approach performs well even for a larger number of features. This suggests that FLAMED handles non-IIDness better than some of the comparison methods, so as to make up for the shortcomings of using simulated data. Our results on the real-world eICU dataset corroborated our results on the synthetic datasets. Most importantly, when a subset of the eICU dataset made up of clients with a large number of samples was used, FLAMED outperformed FedProx and FedAvg, and was competitive with FedDC, despite the high dimensionality of the data (3,069 features).

We also analyzed the privacy and security aspects of all approaches. We observed that, by its nature, FLAMED is not susceptible to gradient-based privacy attacks and the comparison methods can also remediate these attacks using SMC. However, SMC costs the aggregating server the ability to analyze model updates, making detecting attacks on model performance much more difficult. We compared the resilience of FLAMED and the comparison approaches against attacks on model performance. To this end, we implemented backdoor attacks via model poisoning and data poisoning for FedAvg, FedProx, and FedDC and via data poisoning for FLAMED. We found that although all methods are susceptible to backdoor attacks, the model poisoning method used against FedAvg, FedProx, and FedDC is much harder to detect, while the data poisoning attack used against FLAMED is much easier to detect. Further, if attack parameters exist that would result in a successful covert attack against FLAMED, the attacker likely has no way of knowing these parameters.

CHAPTER 5

Federated Supervised Principal Component Analysis

Research in FL is rapidly expanding to cover a variety of ML and data modelling techniques such as SVD and PCA (see 2.2.1). A federated version of SVD, called FedSVD (discussed in Section 5.1.1), which can be used to perform PCA without compromising privacy, was proposed by Chai et al. [57].

One shortcoming of PCA is that it is unsupervised, meaning it does not take into account any extra information, such as categories or labels for the samples within a data matrix. This is of particular interest in ML and FL, where supervised models are trained to predict the label of input samples. In this case, we do not necessarily wish to preserve the maximum variance in our transformed data, but rather we would like to preserve the variance in the directions most important for our task. Said differently, we would like to find the transformation of the data matrix that has maximum dependence on its corresponding labels. SPCA, dual SPCA, and KSPCA are techniques that can find such a transformation, which were proposed by Barshan et al. [62], and discussed in Section 5.1.1. However, to the best of our knowledge, there are no federated versions of SPCA and its variants.

In this chapter, we propose federated versions of SPCA, dual SPCA, and kernel SPCA called FeS-PCA, dual FeS-PCA, and FeSK-PCA, respectively. The federated versions of SPCA and dual SPCA are lossless, while the federated version of kernel SPCA is an approximation that uses the centroids of participants' data rather than the actual data. The participants in FeS-PCA and FeSK-PCA keep their data and its labels private, while the participants in dual FeS-PCA keep just their data private but must share their labels in exchange for an efficiency gain. FeS-PCA and dual FeS-PCA preserve privacy by using ROMMs, an approach inspired by Chai et al., and FeSK-PCA preserves privacy by virtue of the approximation. In all three cases, the participants gain the principal components of their data. We report the time complexities for the proposed methods as well as their em-

pirical runtimes. We also recreate several experiments from Barshan et al. to test the utility of the proposed approaches' transformations for visualization, classification, and regression tasks. Our analysis and results indicate that the proposed FeS-PCA and dual FeS-PCA are lossless and private versions of their originals, while FeSK-PCA exhibits performance equal to standard kernel SPCA.

The present chapter makes the following contributions:

- We extend supervised PCA and its variants to the federated setting. This improves the regression and classification accuracy of FL approaches that use federated standard (unsupervised) PCA.
- We prove the privacy of our approach in theory, and confirm this theory experimentally with the Kolmogorov–Smirnov test.
- We evaluated the performance of our approach by recreating experiments from the original SPCA paper by Barshan et al., and add experiments using a real world federated dataset.

5.1 Method

In this section, we review some preliminaries before presenting our proposed federated supervised PCA approach along with its variants.

5.1.1 Preliminaries

Before presenting our approach, we review some preliminaries which are used in FLAMED. Namely, we review SVD, PCA, SPCA, Dual SPCA, KSPCA, FedSVD and secure summation, while discussing how they are related.

SVD

Singular value decomposition [88] is a common method used for dimensionality reduction. SVD decomposes a data matrix $X \in \mathbb{R}^{m \times n}$ as $X = U\Sigma V^\top$, where the columns of $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are called the left and right *singular vectors* of X, and $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix whose entries are called the *singular values* of X. Using the r columns of U corresponding to the r largest singular values in Σ, denoted $U_{:r}$, a low dimensional transformation $Z \in \mathbb{R}^{r \times n}$ is obtained with $Z = U_{:r}^\top X$.

PCA

Principal component analysis [89] is used to find a transformation of a matrix $X \in \mathbb{R}^{m \times n}$ into $r << m$ dimensions that retains the maximum amount of variance using $Z = U_{:r}^{\top} X$. The columns of $U_{:r}$ are referred to as the principal components of X and can be shown to be the eigenvectors of the covariance matrix of X. Therefore, SVD plays a key role in performing PCA. The corresponding eigenvalues indicate the amount of variance each principal component retains.

SPCA

Supervised PCA, presented by Barshan et al. in [62], maximizes the dependence between transformed points and some response variables (e.g. class labels). Formally, given data matrix $X \in \mathbb{R}^{m \times n}$ with corresponding multivariate response $Y \in \mathbb{R}^{l \times n}$, SPCA finds the subspace $U^{\top} X$ maximally dependent on Y. Let L be the kernel of Y (e.g. the linear kernel $Y^{\top} Y$) and H be the centering matrix. Then U can be computed in closed form

$$\begin{aligned} M &\leftarrow XHLHX^{\top} \\ U &\leftarrow \text{eigenvectors of } M \text{ corresponding to} \\ &\quad\ \text{the } r \text{ largest eigenvalues} \end{aligned}$$

Standard PCA can be seen as a special case of SPCA where L equals the identity matrix, I, because finding the eigenvectors and eigenvalues of $XHIHX^{\top}$ is equivalent to finding the eigenvectors and eigenvalues of the covariance matrix of X.

Dual-SPCA

Barshan et al. also implemented a dual form of SPCA with a runtime that is dependent on the number of samples n for cases where the number of features $m >> n$. In this case, we decompose L such that $L = \Delta^{\top} \Delta$, and then find U with

$$\begin{aligned} \Psi &\leftarrow XH\Delta^{\top} \\ V &\leftarrow \text{eigenvectors of } \Psi^{\top}\Psi \in \mathbb{R}^{n \times n} \text{ corresponding to} \\ &\quad\ \text{the } r \text{ largest eigenvalues} \\ \Sigma &\leftarrow \text{diagonal matrix of square roots of the top } r \\ &\quad\ \text{eigenvalues of } \Psi^{\top}\Psi \\ U &\leftarrow \Psi V \Sigma^{-1} \end{aligned}$$

KSPCA

Kernel Supervised PCA, also presented in [62], uses a kernel function $k(\cdot, \cdot)$ and its associated kernel matrix K in place of X, where $K_{i,j} = k(x_i, x_j)$, and x_i is the i^{th} column (i.e. sample) in X. This allows one to perform SPCA in the larger feature space implied by the kernel without having to explicitly map X to higher dimensions. The result is a nonlinear transformation of X in the subspace $U^\top K$, which like the above, is maximally dependent on Y. With L again the kernel of Y, U can be found with

$$\begin{aligned} M &\leftarrow KHLHK^\top \\ U &\leftarrow \text{generalized eigenvectors of } (M, K) \text{ corresponding to} \\ &\quad \text{the } r \text{ largest eigenvalues} \end{aligned}$$

FedSVD

In federated SVD by Chai et al. [57], a trusted masking server distributes ROMM P and shares, Q_i, of ROMM $Q^\top = [Q_1^\top, \ \ldots, \ Q_C^\top]$ to C participants holding shares, X_i, of global data matrix $X = [X_1, \ \ldots, \ X_C]$. The participants then send $X' = PXQ = \sum_{i=1}^{C} PX_iQ_i$ to a factorization server using secure summation. The factorization server then computes the SVD $PXQ = U'\Sigma V'^\top$ and sends U' to the participants, who obtain the eigenvectors U of the unmasked global data X using $U = P^\top U'$. Throughout this process, the masking and factorization servers and the participants learn nothing of the raw data held by others except the eigenvalues Σ, while the participants obtain U.

Secure Summation

To securely compute matrix sums, we use the secure aggregation protocol with one-time pads proposed in [90] and used by Chai et al. The protocol guarantees the factorization server only learns the sum of participants' inputs, $W = \sum_{i=1}^{C} W_i$, where $W_i \in \mathbb{R}^{m \times n}$ is some matrix held by participant i, and the participants learn nothing. This is accomplished by each pair of participants agreeing on a random pad $S \in \mathbb{R}^{m \times n}$ and one participant adding this pad to their input while the other subtracts it from theirs. Then, when the factorization server sums the padded inputs, the random pads cancel out, and the true sum W is obtained without revealing the inputs W_i. Refer to Section 4.0.1 in [90] for details.

5.1.2 Private SPCA

In this section, we leverage and combine the above techniques and introduce our novel private federated versions of SPCA, dual SPCA, and KSPCA. In the following, assume we have C

participants who hold a share, X_i, of global data matrix $X = [X_1, \ldots, X_C]$. The participants also hold shares of corresponding labels $Y = [Y_1, \ldots, Y_C]$. Further, $X \in \mathbb{R}^{m \times n}$ and $Y \in \mathbb{R}^{l \times n}$, where $m, n,$ and l are the number of features, samples, and classes, respectively. This means that $X_i \in \mathbb{R}^{m \times n_i}$ and $Y_i \in \mathbb{R}^{l \times n_i}$ where n_i is the number of samples at participant i and $n = \sum_{i=1}^{C} n_i$. The participants wish to perform PCA on their combined data matrix X so they may find the subspace $U^\top X$ maximally dependent on Y. To accomplish this, a trusted masking server that can distribute ROMMs and a factorization server that runs standard SVD are available. The ROMMs are generated using QR decomposition as proposed by Chai et al. In the process, nothing may be learned about the data held at participant i by participant $j \neq i$, except for U. The factorization server may only learn Σ, that is, the eigenvalues of X, and if unavoidable, Y. This is permissible because Σ only reveals the proportion of variance associated with each eigenvector of the data matrix. Y is only learned by the server in the case of the dual form of SPCA and, although this reveals general information about the distribution of classes at each client, cannot be used to recreate X.

FeS-PCA

We start with federated supervised principal component analysis (FeS-PCA), which uses a technique similar to the one suggested by Chai et al. The masking server distributes ROMM $P \in \mathbb{R}^{m \times m}$ and shares, Q_i, of ROMM $Q^\top = [Q_1^\top, \ldots, Q_C^\top] \in \mathbb{R}^{n \times n}$ to the participants. Because we are performing PCA, it is necessary to first centre X. In Barshan et al., this was done using a centering matrix. However, because that would block the application of the ROMM Q, we used secure summation to get the row-wise average of X to the factorization server, which sends the result to the participants to mean centre their data. The participants then use secure summation to send $PXQ = \sum_{i=1}^{C} PX_iQ_i$ and $Q^\top Y^\top = \sum_{i=1}^{C} Q_i^\top Y_i^\top$ to the factorization server, which computes $(PXQ)(Q^\top Y^\top)$ to obtain $PXY^\top$ because $QQ^\top = I$. The factorization server then performs standard SVD on $(PXY^\top)(PXY^\top)^\top = PXY^\top YX^\top P^\top = PXLX^\top P^\top = PMP^\top$ and obtains U', the matrix composed of the masked eigenvectors of M, which are sent back to each participant. Each participant then unmasks U' to get the eigenvectors of M using $U = P^\top U'$ by Theorem 4.1 in Chai et al. The procedure is outlined in algorithm 4.

In algorithm 4, $PXY^\top$ is obtained at the server as an intermediate step. This does not compromise the security of X because $PXY^\top$ is still masked by P, nor does it change the collusion threshold, i.e. the number of participants required to compromise the privacy of X, as shown in Section 5.2.2. This means it is possible to avoid generating Q and calculating PXQ and $Q^\top Y^\top$ by using $PXY^\top = \sum_{i=1}^{C} PX_iY_i^\top$. However, in many cases, it is necessary

Algorithm 4 FeS-PCA

At Participant i **do**:

$\vec{\sigma}_i \leftarrow$ `rowWiseSum`(X_i)

At Factorization Server **do**:

$\vec{\mu} \leftarrow$ `secureSummation`$(\vec{\sigma}_1\ , \dots\ \vec{\sigma}_C)/n$

At Participant i **do**:

$X_i \leftarrow\ X_i - \vec{\mu}$

$X_i' \leftarrow\ PX_iQ_i$

$Y_i'^{\top} \leftarrow\ Q_i^{\top}Y_i^{\top}$

At Factorization Server **do**:

$PXQ \leftarrow$ `secureSummation`$(X_1'\ , \dots\ X_C')$

$Q^{\top}Y^{\top} \leftarrow$ `secureSummation`$(Y_1'^{\top}\ , \dots\ Y_C'^{\top})$

$PMP^{\top} \leftarrow\ PXQQ^{\top}Y^{\top}(PXQQ^{\top}Y^{\top})^{\top}$

$U' \leftarrow$ `eigenvectors`$(PMP^{\top})$

\\ `using standard SVD`

At Participant i **do**:

$U \leftarrow P^{\top}U'$

$Z^{test} \leftarrow\ U^{\top}X^{test}$

to add the identity matrix to L in order to avoid the problem of rank deficiency. This is not possible if the factorization server has access to only $PXY^\top$ and not the intermediate result of $Q^\top Y^\top YQ = Q^\top LQ$. Observe

$$\begin{aligned} Q^\top Y^\top YQ + I = Q^\top LQ + I &= Q^\top LQ + Q^\top Q \\ &= Q^\top LQ + Q^\top IQ \\ &= Q^\top (L + I)Q \end{aligned}$$

Therefore, masking X on both sides is useful for ensuring rank sufficiency with the following

$$PXHQ(Q^\top LQ + I)Q^\top X^\top P^\top = PXH(L + I)HX^\top P^\top$$

FeS-PCA provides better security than dual FeS-PCA, which follows, but only allows for the use of the linear kernel $L = Y^\top Y$. However, this can be remediated at the expense of security (see Section 5.2.2) by sharing Y in plain text and obtaining PX at the server through the concatenation $[PX_1, \ \ldots, \ PX_C]$.

Dual FeS-PCA

In the case of dual SPCA, federation comes with looser security guarantees, as discussed in Section 5.2.2. Because we must compute the matrix $L = Y^\top Y$, each participant must share Y_i with the factorization server, which uses it to obtain the decomposition $L = \Delta^\top \Delta$. The participants mask their data and send PX_i to the factorization server, which concatenates it to get $PX = [PX_1, \ \ldots, \ PX_C]$. The factorization server then computes $P\Psi = PXH\Delta^\top$ and finds the eigenvectors V of $\Psi^\top P^\top P\Psi = \Psi^\top \Psi$ with standard SVD. Finally, it returns $U' = PU = P\Psi V\Sigma^{-1}$ (since it cannot unmask Ψ) where Σ is a diagonal matrix of the square roots of the eigenvalues of $\Psi^\top \Psi$. The participants then obtain the eigenvectors U of Ψ (see equation 12 in Barshan et al.) with $U = P^\top U'$. The procedure is outlined in algorithm 5.

Although this result is less expensive than plain FeS-PCA in the case where $m > n$, because the dimensionality of $\Psi^\top \Psi$ is dependent on n rather than m, FeS-PCA suffers from other drawbacks. Y must be shared, and the matrix X is only masked on one side by P, which is shared by all participants, meaning only the factorization server and one other participant must collude to compromise the privacy of all participants' data. Although computation is again saved by not computing $Q \in \mathbb{R}^{n \times n}$, the reader should keep in mind that in the case of the dual formulation, n is smaller than m. Furthermore, any kernel can be used for Y, but FeS-PCA permits this as well, but at the same cost of sharing Y and not double masking X.

Algorithm 5 Dual FeS-PCA

At Participant i **do**:

$X'_i \leftarrow PX_i$

At Factorization Server **do**:

$PX \leftarrow$ **concatenation**$(X'_1, \ldots X'_C)$

$Y \leftarrow$ **concatenation**$(Y_1, \ldots Y_C)$

$\Delta^\top\Delta \leftarrow$**decompose**$(Y^\top Y)$

$P\Psi \leftarrow PXH\Delta^\top$

$V, \Sigma \leftarrow$ **eigendecomposition**$(\Psi^\top P^\top P\Psi)$

\\ `using standard SVD`

$\Sigma \leftarrow$ **elementwiseSqrt**(Σ)

$U' \leftarrow P\Psi V\Sigma^{-1}$

At Participant i **do**:

$U \leftarrow P^\top U'$

$Z^{test} \leftarrow U^\top X^{test}$

FeSK-PCA

Federated supervised kernel principal component analysis (FeSK-PCA) departs from the use of ROMMs and must instead settle for an approximation of the closed form computation suggested by Barshan et al. It is possible to compute the kernel matrix K and obtain the solution $U^\top K$ using ROMMs to ensure no sensitive information is exposed, but at test time, a new kernel matrix K^{test} will need to be computed, where $K^{test}_{i,j} = k(x_i, x^{test}_j)$, $x_i \in X$ and $x^{test}_j \in X^{test}$. This means that at test time, each participant would need a copy of X, thus compromising privacy. Therefore, we replaced X with support data X^{sup} which is derived from X and can be shared with participants without compromising privacy.

Each participant finds ρn_i centroids, where $\rho << 1$, using k-means clustering, and obtains $X^{sup}_i \in \mathbb{R}^{m\times\rho n_i}$. They label their centroids locally using k-nearest neighbours to obtain $Y^{sup}_i \in \mathbb{R}^{l\times\rho n_i}$. The factorization server concatenates each participant's support matrices to get $X^{sup} = [X^{sup}_1, \ldots, X^{sup}_C] \in \mathbb{R}^{m\times\rho n}$ and $Y^{sup} = [Y^{sup}_1, \ldots, Y^{sup}_C] \in \mathbb{R}^{l\times\rho n}$. It then computes the kernel matrix K^{sup} where $K^{sup}_{i,j} = k(x^{sup}_i, x^{sup}_j)$ and x^{sup}_i is the i^{th} column/centroid in X^{sup}. Finally, it finds, U, the generalized eigenvectors of $M = K^{sup}HL^{sup}HK^{sup}$ constrained by K^{sup} (i.e. $U^\top KU = I$), where L^{sup} is the kernel of Y^{sup}. The eigenvectors U and X^{sup} are sent to participants who can compute the transformation of the original and test data using $Z = U^\top K$ and $Z = U^\top K^{test}$, respectively. The procedure is outlined in algorithm 6.

Algorithm 6 FeSK-PCA

At Participant i **do**:

$X_i^{sup} \leftarrow$ `kMeans`(X_i)

$Y_i^{sup} \leftarrow$ `kNearestNeighbours`(X_i^{sup})

At Factorization Server **do**:

$X^{sup} \leftarrow$ `concatenation`$(X_1^{sup}, \dots X_C^{sup})$

$Y^{sup} \leftarrow$ `concatenation`$(Y_1^{sup}, \dots Y_C^{sup})$

$K^{sup} \leftarrow$ `kernel`(X^{sup})

$L^{sup} \leftarrow$ `kernel`(Y^{sup})

$M^\top \leftarrow K^{sup} H L^{sup} H K^{sup}$

$U \leftarrow$ `generalizedEigenvectors`$(U, M^\top)$

\\ `using standard generalized eigendecomposition`

At Participant i **do**:

$K^{test} \leftarrow$ `kernel`(X^{sup}, X^{test})

$Z^{test} \leftarrow U^\top K^{test}$

Although this results in an approximation, the use of centroids lowers the dimensionality of K and allows for the use of any kernel on X^{sup} and Y^{sup}. Further, there is no need for ROMMs and, therefore, no need for a trusted masking server or even a factorization server if one of the participants volunteers to perform the factorization.

The time complexities for each of the proposed methods are shown in Table 10. Time complexities at participant i result from multiplying the local data by ROMMs for FeS-PCA and dual FeS-PCA. FeS-PCA has the further computational cost of generating C pads in $\mathbb{R}^{n \times m}$ for each participant in the secure aggregation protocol. For FeSK-PCA, the time complexity results from applying k-means and then k-nearest neighbours, each of which are $O(n^2)$. At the factorization server, the runtime results from computing eigenvalues and eigenvectors [91]. The masking server's computation time results from performing QR decomposition to generate ROMMs [57]. As noted in Chai et al., the time complexity of generating ROMMs can be greatly reduced by replacing a single large ROMM with a matrix that consists of $b = \frac{n}{c}$ smaller ROMMs of size c. This brings the complexity of this step down from $O(n^3)$ to $O(c^2 n)$. We refer the reader to Chai et al. for the details of this approach and an empirical analysis of the effect it has on the time complexity.

Method	Time Complexity		
	Participant i	**Factorization Server**	**Masking Server**
FeS-PCA	$O(m^2n_i + mn_i^2) + O(Cnm)$	$O(m^3)$	$O(n^3) + O(m^3)$
Dual FeS-PCA	$O(m^2n_i)$	$O(n^3)$	$O(m^3)$
FeSK-PCA	$O(2mn_i^2)$	$O((n\rho)^3)$	n\a

Table 10: Time complexities (with keeping all proportional constants) for the proposed methods. Note: generating ROMMs at the masking server can be made $O(c^2n)$ using the building blocks method proposed in Chai et al.

5.2 Privacy Properties

In the following we give proof of privacy that is similar to Chai et al.'s, for FeS-PCA, dual FeS-PCA, and FeSK-PCA. We also visualize the effects ROMMs have on raw data, again inspired by Chai et al. We then improve on the subjective visual demonstration of the privacy of ROMMs by performing an empirical analysis of the randomness of masked data.

5.2.1 Proof of Privacy

We adopt a privacy definition similar to that used in Chai et al., but with slightly different requirements depending on the version of FeS-PCA.

Definition 5.2.1. *FeS-PCA and FeSK-PCA are private if:*

1. Participant i learns nothing about X_j and Y_j where $i \neq j$, except for principal components U.

2. Factorization and masking servers learn no more about X_i and Y_i than participant $j \neq i$, except for possibly Σ_X and Σ_Y.

Definition 5.2.2. *Dual FeS-PCA is private if:*

1. Participant i learns nothing about X_j where $i \neq j$, except for principal components U.

2. Factorization and masking servers learn no more about X_i than participant $j \neq i$, except for possibly Σ_X. However, the factorization server can learn Y.

Theorem 5.2.1. *In FeS-PCA, participant $i \in [1, C]$ learns nothing about X_j and Y_j where $i \neq j$, except for principal components U.*

Proof: Each participant i only receives U', P, and Q_i. ROMMs P and Q_i are randomly generated independently of U', meaning that by lemma 2 in Chai et al., the participants only receive new knowledge U'. By Theorem 4.1 in Chai et al. $U = P^\top U'$; therefore, each participant i learns no more than U.

Theorem 5.2.2. *In FeS-PCA, the factorization and masking servers learn no more about X_i and Y_i than participant $j \neq i$, except for Σ_X and Σ_Y.*

Proof: The masking server receives no information from any participant or the factorization server and therefore does not learn any more about X_i than participant $j \neq i$. In contrast, the factorization server receives masked matrices PXQ and $Q^\top Y^\top$. By Theorem 4.2 in Chai et al., PXQ can only reveal Σ_X to the factorization server, which reveals no further information about X. Further, since Q is uniformly randomly sampled from the set of all $n \times n$ orthogonal matrices, $Q^\top Y^\top$ follows a random uniform distribution over $\{Q^\top Y^\top\} = \{Q^\top Y^\top | Q^\top = Q^{-1} \in \mathbb{R}^{n \times n}, Y \in \mathbb{R}^{l \times n}\}$. Hence, the factorization server only learns $Q^\top Y^\top$ is uniformly sampled from $\{Q^\top Y^\top\}$. Following the same logic as Theorem 4.2 in Chai et al., except for omitting the right-hand side ROMM, performing SVD on $Q^\top Y^\top$ to obtain $Q^\top Y^\top = Q^\top U \Sigma_Y V^\top$ allows the factorization server to learn the same amount from $Q^\top Y^\top$ as Σ_Y.

Theorem 5.2.3. *In FeSK-PCA, participant $i \in [1, C]$ learns nothing about X_j and Y_j where $i \neq j$, except for principal components U of the kernel of X^{sup}.*

Proof: Each participant only receives U and X^{sup}. X^{sup} contains ρn centroids, each corresponding to the mean of at most n_i samples, which alone does not allow for the deduction of any of the individual samples' feature values. Y^{sup} is the label of said centroids and only indicates the majority class of nearby neighbours in the corresponding participant's dataset.

Theorem 5.2.4. *In FeSK-PCA, the factorization and masking servers learn no more about X_i and Y_i than participant $j \neq i$, except for X^{sup} and Y^{sup}.*

Proof: The masking server receives no information from any participants or the factorization server and therefore does not learn any more about X_i and X_i than participant $j \neq i$. The factorization server receives matrices X^{sup} and Y^{sup}, where each column x_j in the former and y_j in the latter correspond to aggregate data of at most n_i samples, which alone does not allow for the deduction of any of the individual samples' feature values or labels. Therefore, the factorization and masking servers learn no more about X_i and Y_i than participant $j \neq i$.

Theorem 5.2.5. *In dual Fes-PCA, participant $i \in [1, C]$ learns nothing about X_j where $i \neq j$, except for principal components U.*

Proof: Identical to the proof for Theorem 5.2.1.

Theorem 5.2.6. *In dual FeS-PCA, the factorization and masking servers learn no more about X_j than participant $j \neq i$.*

Proof: Identical to the proof for Theorem 5.2.2, except we accept that Y is learned by the factorization server.

Theorem 5.2.7. *FeS-PCA and FeSK-PCA satisfy Def. 5.2.1.*

Proof: According to Theorem 5.2.1 and 5.2.3, FeS-PCA and FeSK-PCA satisfy point 1 in Def. 5.2.1 because each data holder only learns the principal components of the combined data matrix or of the kernel matrix of X^{sup}. By Theorem 5.2.2 and 5.2.4, FeS-PCA and FeSK-PCA satisfy point 2 in Def. 5.2.1 because the factorization and masking servers learn no more than Σ_X and Σ_Y.

Theorem 5.2.8. *Dual FeS-SPCA satisfies Def. 5.2.2*

Proof: According to Theorem 5.2.5, Dual FeS-PCA satisfies point 1 in Def. 5.2.2 because each data holder only learns the principal components of the combined data matrix. By Theorem 5.2.6, Dual FeS-PCA satisfies point 2 in Def. 5.2.2 because the factorization and masking servers learn no more than Σ_X.

5.2.2 Collusion Threshold

In FeS-SVD or Dual FeS-SVD, if the masking server leaks to the factorization server P and Q or just P, respectively, then all participants will have their data exposed. If the masking server is trusted, then the collusion required to expose data always includes the factorization server, because only it has access to masked data, plus some number of participants.

Dual FeS-PCA has the lowest collusion threshold among participants because each participant i sends PX_i to the factorization server and each participant has P, meaning if one participant and the factorization server collude, all participant data can be exposed. Standard FeS-PCA is in this camp if changes are made (see Section 5.1.2) to allow a nonlinear label kernel L.

FeS-PCA has a high collusion threshold of $C - 1$ participants plus the factorization server, even though the factorization server has access To PXY. Without loss of generality, assume participants 1 to $C - 2$ collude with the factorization server to obtain $PXY =$

$\sum_{i=1}^{C-2} PX_iY_i + X_{C-1}Y_{C-1} + X_CY_C$ where PXY and $\sum_{i=1}^{C-2} PX_iY_i$ are known. The equation still has infinite solutions for $X_{C-1}Y_{C-1}$ and X_CY_C if $X_i \in \mathbb{R}^{m\times n}$. However, if the participants 1 to $C-1$ collude to obtain $PXY = \sum_{i=1}^{C-1} PX_iY_i + X_CY_C$, where PXY and $\sum_{i=1}^{C-1} PX_iY_i$ are known, then the non-colluding participant C's information is exposed since X_CY_C has a unique solution.

Because no masked data is exchanged in FeSK-PCA, the privacy of non-colluding participants is always maintained regardless of how many participants collude, even if the factorization server defects.

5.2.3 Privacy Visualization

To give the reader an idea of the privacy provided by ROMMs, as well as some intuition for our approach, we visualized the effects ROMMs have on image data. Inspired by the results shown in Fig. 7 of Chai et al., we visualized the effect masking has on the MNIST [92] dataset. The MNIST dataset is a collection of 70,000 images of handwritten digits commonly used as a benchmark dataset for evaluating ML techniques. The samples in MNIST contain 784 features, corresponding to pixel values in $\{0, 1\}$, which make up a 28×28 black and white image of a handwritten digit. We took the first 10 samples from the MNIST dataset as $X \in \mathbb{R}^{784\times 10}$ and applied one ROMM with PX, as in dual FeS-PCA, and two ROMMs with PXQ, as in FeS-PCA. We then visualized the first two samples from X before and after masking. Fig. 10 shows our visualization results, which clearly depicts our proofs that the masked images, using either one ROMM or two, leak no information about the original image.

5.2.4 Empirical Privacy Analysis

Because the visualization results above are inherently subjective, we empirically verified the randomness of masked data matrices PX and PXQ using the MNIST and Iris [93] datasets. To do this, we used the Kolmogorov–Smirnov goodness-of-fit test [94] to verify that the sample probability distribution of the feature values in the masked data matches their theoretical distribution, which we derive below.

By Theorem 2.2 of Zhang [95], if $P \sim \text{Uniform}(\mathbb{O}^m)$, where $\mathbb{O}^m$ is the set of all $m \times m$ orthogonal matrices, and $v \in \mathbb{R}^m$ where $\|v\| \neq 0$, then $\frac{Pv}{\|v\|} \sim \text{Uniform}(S^{m-1})$, where S^m is the unit m-sphere, i.e. $S^m = \{v \in \mathbb{R}^{m+1} : \|v\| = 1\}$. In dual FeS-PCA, we have $PX = [Px_1, Px_2, \ldots, Px_n]$. Therefore, it is easy to see that for the i^{th} column or masked sample Px_i we have $\frac{Px_i}{\|x_i\|} \sim \text{Uniform}(S^{m-1})$. Also, consider the columns or masked samples

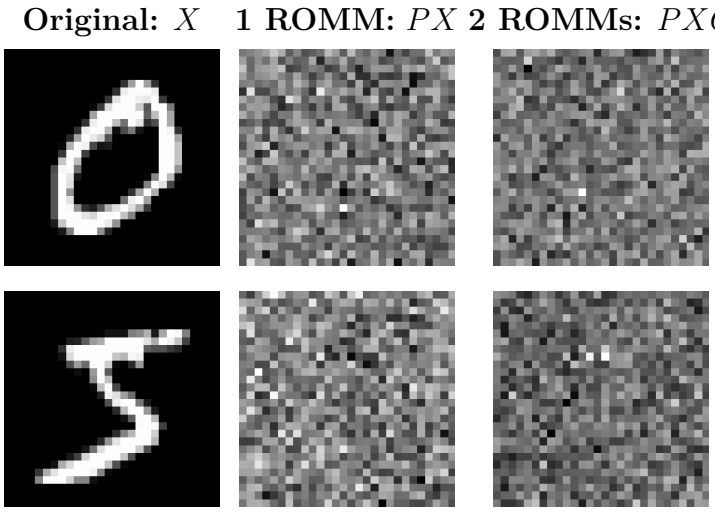

Figure 10: Visualization of two samples from the MNIST dataset (left), after applying one ROMM (middle), and two ROMMs (right).

of PXQ in FeS-PCA as Px'_i, where x'_i is the i^{th} column of XQ. It again follows, using the same theorem, that $\frac{Px'_i}{\|x'_i\|} \sim \text{Uniform}(S^{m-1})$.

It is well known that the surface area of an m-hypersphere with radius 1 is

$$A^s_m = \frac{2\pi^{\frac{m}{2}}}{\Gamma(\frac{m}{2})}$$

where Γ is the gamma function. Further, the surface area of an m-hyperspherical cap with radius 1 and height h is

$$A^c_m(h) = \frac{1}{2}A^s_m I_{(2h-h^2)}(\frac{m-1}{2}, \frac{1}{2}), h \leq r = 1$$

where $I_x(a, b)$ is the incomplete beta function. Therefore, we can obtain the CDF for random variable Z, corresponding to a masked feature value independently drawn from some $\frac{Px_i}{\|x_i\|}$ or $\frac{Px'_i}{\|x'_i\|}$, as

$$\begin{aligned}
\text{P}(\text{Z} \leq z) &= \begin{cases} \frac{\frac{1}{2}A^s_m I_{(2(1+z)\text{-}(1+z)^2)}(\frac{m\text{-}1}{2}, \frac{1}{2})}{A^s_m} & \text{if -1} \leq z \leq 0 \\ \frac{A^s_m - \frac{1}{2}A^s_m I_{(2(1\text{-}z)\text{-}(1\text{-}z)^2)}(\frac{m\text{-}1}{2}, \frac{1}{2})}{A^s_m} & \text{if } 0 < z \leq 1 \end{cases} \\
&= \begin{cases} \frac{1}{2}I_{(1\text{-}z^2)}(\frac{m\text{-}1}{2}, \frac{1}{2}) & \text{if -1} \leq z \leq 0 \\ 1 - \frac{1}{2}I_{(1\text{-}z^2)}(\frac{m\text{-}1}{2}, \frac{1}{2}) & \text{if } 0 < z \leq 1 \end{cases}
\end{aligned}$$

Intuitively, the above computes the surface area of the hyperspherical cap with height $1 + z$ over the surface area of the $(m-1)$-sphere when -1 $\leq z \leq$ 0. Alternatively, when

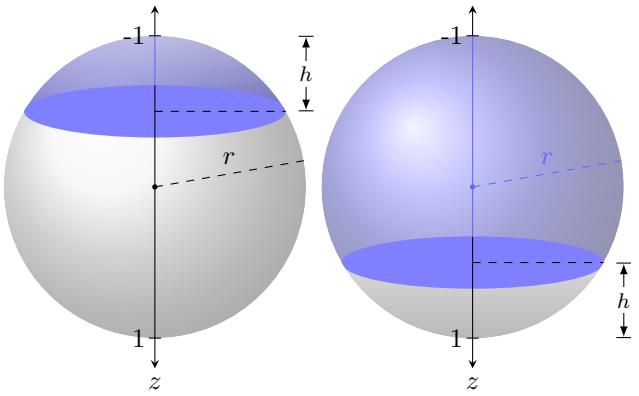

Figure 11: On the left is a sphere with a spherical cap of height h. A point uniformly randomly distributed on the surface of this sphere has a probability of occurring in the blue region equal to the surface area of the cap, divided by the total surface area of the sphere. Similarly, the right also shows a sphere with a spherical cap, except it is at the positive pole of the z-axis. A similarly distributed point has a probability of occurring in the blue region equal to the sphere's surface area, minus the cap's surface area, all divided by the total surface area of the sphere.

$0 < z \leq 1$, the above computes the surface area of the $(m-1)$-sphere, minus the surface area of the hyperspherical cap with height $1-z$, over the surface area of the $(m-1)$-sphere. The graphical representation of this concept is shown in Fig. 11 for $m = 3$, but the same intuition applies to higher dimensions.

In our experiments, we used the MNIST dataset, with data matrix $X \in \mathbb{R}^{784\times 70{,}000}$. The sample CDF of all the unmasked feature values taken together across all samples is shown in Fig. 12.

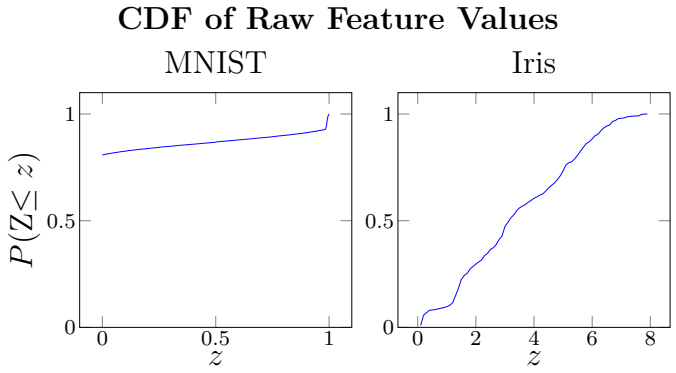

Figure 12: Sample CDF of all feature values taken together across all samples, before masking, for the MNIST and Iris datasets.

For the j^{th} block, with $j \in \{1, 2, \ldots, 70\}$, of size $n_b = 1000$, we sampled orthogonal

matrices P_j and Q_j, and then computed P_jX_j and $P_jX_jQ_j$ where P_j, Q_j and X_j are in $\mathbb{R}^{784\times784}$, $\mathbb{R}^{n_b\times n_b}$ and $\mathbb{R}^{784\times n_b}$, respectively. The true CDF of the elements of $\frac{P_jx_{j,i}}{\|P_jx_{j,i}\|}$ and $\frac{P_jx'_{j,i}}{\|P_jx'_{j,i}\|}$, where $P_jx_{j,i}$ and $P_jx'_{j,i}$ are the i^{th} masked samples or columns in P_jX_j and $P_jX_jQ_j$, respectively, is

$$\mathrm{P}(\mathrm{Z}\leq z)=\begin{cases}\frac{1}{2}I_{(1\text{-}z^2)}(391.5,\frac{1}{2}) & \text{if -1}\leq z\leq 0\\ 1-\frac{1}{2}I_{(1\text{-}z^2)}(391.5,\frac{1}{2}) & \text{if } 0<z\leq 1\end{cases}$$

Our null hypothesis H_0 is that the true CDF and the sample CDFs obtained using all the masked feature values from P_jX_j and $P_jX_jQ_j$ for $j\in\{1,2,\ldots,\frac{70{,}000}{n_b}\}$ are equivalent. The alternative hypothesis H_a is that the true CDF and sample CDFs are not equivalent. The true CDF and the sample CDFs are shown in Fig. 13. As expected, the true CDF and the sample CDFs are nearly identical. We confirmed this to be the case, accepting our null hypothesis, using the Kolmogorov–Smirnov test, the results of which are shown in Table 11.

To be thorough, because the above CDF for the MNIST dataset was derived for very high dimensionality $m=784$, we repeated the above experiment with the Iris dataset, which contains 150 samples and $m=4$ numerical features. These features record the physical characteristics of three different types of Iris flowers. The sample CDF of all the unmasked feature values taken together across all samples is again shown in Fig. 12. The true CDF of the masked feature values is

$$\mathrm{P}(\mathrm{Z}\leq z)=\begin{cases}\frac{1}{2}I_{(1\text{-}z^2)}(\frac{3}{2},\frac{1}{2}) & \text{if -1}\leq z\leq 0\\ 1-\frac{1}{2}I_{(1\text{-}z^2)}(\frac{3}{2},\frac{1}{2}) & \text{if } 0<z\leq 1\end{cases}$$

In this case, because there are only 150 samples, we obtained 1,000 different P_jX and P_jXQ_j using the full data matrix $X\in\mathbb{R}^{4\times150}$ and sampling P_j and Q_j for $j\in\{1,2,\ldots,1{,}000\}$. Again, H_0 is that the true CDF and the sample CDFs are the same across all j, while H_a is that the true CDF and the sample CDFs are different. The true CDF and the sample CDFs are also shown in Fig 13, and as expected, the CDFs are nearly identical. We confirmed this using the Kolmogorov–Smirnov test. The results are shown in Table 11.

5.3 Quality of Transformation and Scalability Experiments

In this section, we present two sets of experiments. Those which measured the quality of the transformation derived from our approaches, and those which measured the scalability of our approach in a federated setting. We start with a discussion of the experimental setup, then

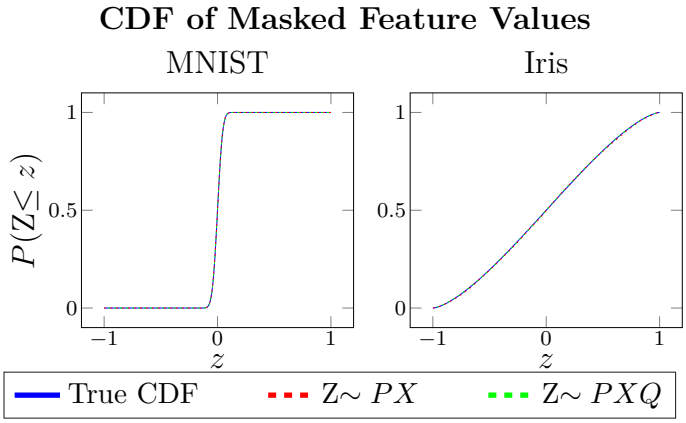

Figure 13: Sample CDF of all masked feature values taken together across all samples, along with their theoretical CDF, for the MNIST and Iris datasets.

	Masking	**KS Statistic**	**Critical Value** ($\alpha = 0.01$)	**Result**
MNIST	Z $\sim PX$	0.0013	0.1948	Accept H_0
	Z $\sim PXQ$	0.0002		Accept H_0
Iris	Z $\sim PX$	0.0067	0.0515	Accept H_0
	Z $\sim PXQ$	0.0032		Accept H_0

Table 11: The Kolmogorov–Smirnov statistic for the masked feature values in P_jX_j and $P_jX_jQ_j$ for all j, totalling 70 and 1,000 observations for the MNIST and Iris datasets, respectively. The results indicate acceptance of the null hypothesis with at least 0.01 significance level.

present the individual experiments and their results in turn, along with some commentary and analysis.

5.3.1 Setup and Configurations

In the former we recreated Barshan et al.'s visual experiments shown in their Fig. 1, 4, and 5, classification experiments from their Fig. 7.c and Table 3, and regression experiments in their Table 4 and Fig. 8. To the classification experiments, we added the eICU [82] dataset, discussed below. Across these experiments we used our proposed approaches, FeS-PCA, dual FeS-PCA, and FeSK-PCA, and their unsupervised counterparts SPCA, dual SPCA, and KSPCA, respectively. Even though the datasets were not split among multiple participants, we applied the ROMMs without partitioning. This allowed us to compare the utility of the original techniques against the private federated versions. We also used PCA in

Dataset	No. of Samples	No. of Features	No. of Classes
XOR	400	2	2
Rings	313	2	2
Iris	150	4	3
Ionosphere	351	34	2
SRBCT	83	2,308	4
eICU	72,959	3,069	2
Synthetic	see Section 5.3.3		
DLBCL	240	7,399	Regression

Table 12: Datasets Characteristics

the classification experiments, and principal component regression (PCR) in the regression experiments, to contrast the performance gain resulting from supervision. We added a classification experiment with a varying number of participants form the eICU dataset in order to test the effect federation has on the utility of the projection. In the scalability experiments we measured the execution time on synthetic datasets generated using the same approach as Chai et al.

The eICU dataset was used because it represents a genuinely federated real-world dataset. That is, it is a real-world dataset collected from separate participants, thus representing a realistic federation rather than a contrived one. The dataset contains medical data from over 200,000 ICU admissions to more than 200 medical centres across the United States. We used the data contained in the drug infusions table, which tracks the drugs administered to patients during their stay in the ICU. Each sample corresponds to a patient, and each feature corresponds to a drug. If a patient receives any dose of a certain drug across all of their ICU admissions, then the corresponding feature value in the corresponding sample was set to 1; otherwise it was set to 0. We specified a binary classification problem where a patient was assigned label 0 if their hospital discharge status in the patient table was "Alive", and 1 otherwise. After removing any hospital with less than 10 observations, we were left with a total of 3,069 features and 72,959 patients held across 132 hospitals. The details for each dataset can be found in Table 12.

For FeSK-PCA in the experiments recreating Barshan et al., except for the visualization experiments and with the toy datasets, the best result was reported across all combinations $(\rho, \gamma) \in \{\frac{1}{6}, \frac{1}{3}\} \times \{0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100\}$, where ρ times the train set size is the number of centroids used and γ is a parameter for the radial basis function (RBF).

5.3.2 Quality of Transformation

In the present section, we present the procedure and results for our experiments comparing the quality of the transformation derived using our proposed approaches with the transformation derived using their unfederated counterparts.

Procedure

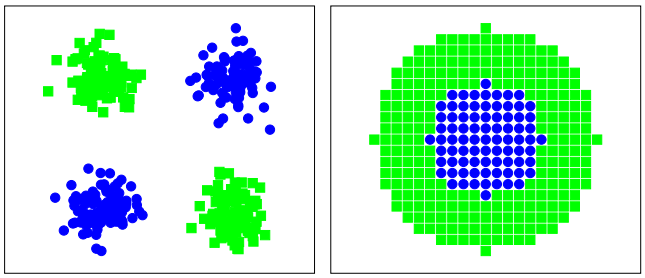

Figure 14: The toy datasets XOR (left) and Rings (right) from Barshan et al. before transformation. Different marker styles denote different classes.

We used three toy datasets from Barshan et al., the XOR, Rings, and Iris [93] datasets, and visualized them using the top two principal components obtained from applying our approaches. The XOR and Rings datasets before transformation are shown in Fig. 14. We used a random 70/30 train/test split, applied our approaches to the train set, then used the obtained principal components to transform both the test and train sets. Labels Y were one-hot-encoded, meaning $L = Y^\top Y$ resulted in a label kernel equivalent to the delta label kernel used in Barshan et al. For FeSK-PCA, we used the RBF kernel for the data kernel K, but since we were interested in recreating Barshan et al., we selected the parameter γ manually to get as close a visual match as possible to the results in Barshan et al. We set the number of centroids used in X^{sup} to 50. We also visualized each toy dataset using the unfederated versions of our approaches as they were originally proposed in Barshan et al. in order to verify our results.

For the classification experiments, we followed Barshan et al. and recorded the error rate obtained by a 1-nearest-neighbour (1-NN) classifier with respect to the number of projection dimensions. Like them we used the Ionosphere [93] and SRBCT [96] datasets, but also added the eICU dataset. This allowed us to test all three proposed approaches because the SRBCT dataset has more features than samples and benefits from the dual formulation of Fed-SPCA. The eICU dataset was used because it represents a genuinely federated real-world dataset and is much larger and more complex than the other datasets. Using our proposed approaches, a projection was obtained on a train set, and the 1-NN classifier was

created using the projected train set. The same projection was applied to a test set that was used to evaluate the 1-NN classifier. For the SRBCT and Ionosphere datasets, the train set represented 70% of the data and the test set was 30% of the data. For the eICU dataset, we used a 90/10 train/test split, but only applied the given PCA technique to 20,000 samples randomly drawn from the train set. Still, the full train set was used to train the classifier. The error rate was averaged over 40 random train/test splits for the SRBCT and Ionosphere datasets, but we averaged over only five splits with the eICU dataset because of its size and time constraints. Also, following Barshan et al., we added an identity matrix to the label Kernel $L = Y^\top Y$. For comparison and validation, we added the original versions of SPCA from Barshan et al. and the standard unsupervised PCA.

To verify that the performance with multiple participants does not change in a decentralized setting and that the proposed approaches are indeed lossless, we applied FeS-PCA and FeSK-PCA to the eICU dataset and measured the error rate with a varying number of participants. To test the performance under different conditions, we used C participants in $[2, 4, 8, 16, 32, 64, 128, 132]$ and we selected the participants with the C smallest, C largest, and C nearest to median number of samples. Just as in the previous experiment, a 90/10 train/test split was used, and when the total number of samples was over 20,000, PCA was applied to a random sample of only 20,000 data points. For FeSK-PCA, we set $(\rho, \gamma) = (\frac{1}{6}, 5)$ because it performed well in the centralized classification experiments. The top eight supervised principal components were used to transform the entire train set and test set which were then used to create and test a 1-NN classifier.

For the regression experiments, again following Barshan et al., we recorded the root mean square error (RMSE) obtained with a LR model. We used their synthetic dataset configurations along with the DLBCL dataset [97], which has more features than samples, allowing us to test the dual form of FeS-PCA on a regression task. The projections were obtained on a train set, and the LR model was trained on the projected train set. The same projection was applied to a test set used to evaluate the LR model. For the synthetic datasets, the RMSE was averaged over 40 samples of size 100 with random 70/30 train/test splits using a single projection dimension. For the DLBCL dataset, we averaged the RMSE over 40 160/80 train/test splits and reported the performance with a varying number of projection dimensions. Again, for comparison and validation, we also used the original versions of SPCA from Barshan et al. and standard unsupervised PCR. In all cases, we used a linear label kernel ($L = Y^\top Y$) instead of the RBF label kernel used by barshan et al. This was done because FeS-PCA does not support nonlinear label kernels, and in this way we could directly compare only the effect of nonlinearity in the data kernel K. Further, the extra parameter γ used in the label kernel was not given in Barshan et al. so recreating the

exact results would have been unlikely.

Results

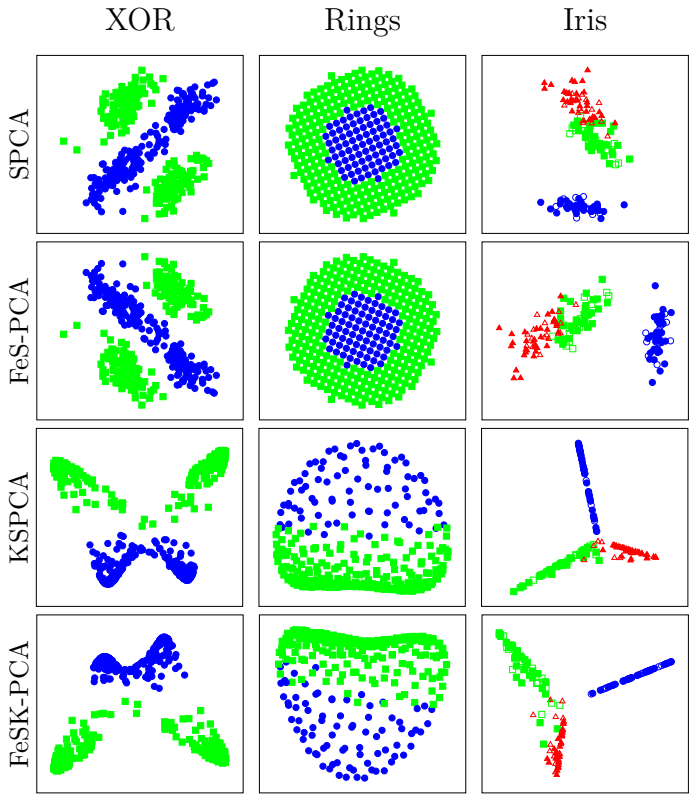

Figure 15: Visualization of the XOR, Rings, and Iris datasets using the top two principal components obtained using FeS-PCA, FeSK-PCA, and their unfederated counterparts.

The transformed toy datasets using Fed-SPCA, FeSK-PCA, and the originals proposed by Barshan et al. are shown in Fig. 15. The transformations resulting from our implementation of the Barshan et al. and from our proposed approaches bear a very close resemblance to the originals in the Barshan et al. paper. There were small variations in the XOR and Rings datasets due to the precise distributions and choice of *gamma* not being given, as well as some randomness in the train and test sampling. The FeS-PCA and SPCA transformations were identical, while the FeSK-PCA and KSPCA transformations had only a small variation despite only 50 centroids being used in X^{sup}. The results indicated the fidelity of the approach, showing that the masking was lossless, while the approximation using centroids was nearly lossless.

The results from the classification experiments are shown in Fig. 16. As expected, the results with FeS-PCA and its original SPCA on both the Ionosphere and eICU datasets were

nearly identical. The same was true for dual FeS-PCA's and the original dual SPCA's performance on the SRBCT dataset. FeSK-PCA performed mostly at par with KSPCA on the Ionosphere and eICU dataset but with increased variation on the larger eICU dataset. On the SRBCT dataset, FeSK-PCA performed considerably worse than its centralized counterpart, KSPCA. However, as pointed out by Barshan et al., in this setting, it is expected and, indeed, corroborated by our results that linear kernels will perform better than nonlinear kernels on microarray gene datasets such as SRBCT. Therefore, we would not expect to use KSPCA in this setting anyway. More importantly, however, FeSK-PCA still outperformed standard unsupervised PCA, which performed the worst on all datasets, as expected, especially in lower dimensions. Taken together, the results proved the utility of the proposed approaches for classification tasks (see Table 10).

In Fig. 17, the error rate is shown for FeS-PCA and FeSK-PCA with various numbers of participants selected from those with the smallest, the nearest to median, and the largest number of samples. The same methods were also applied on the selected participants' data aggregated at one participant in a centralized setting. Regardless of the number of participants, both methods performed as well with centralized data as they did with data partitioned among participants. This indicated that federation did not result in any loss of information. This is especially relevant to FeSK-PCA, where each client finds and labels their centroids using local data. Compared to the centralized setting, where each centroid is found and labelled using the data held across all clients, the resulting centroids, which may be more representative of the input space, and their resulting labels, which may be more accurately assigned, do not provide better results.

The results from the regression experiments on the synthetic datasets are shown in Table 13. FeS-PCA and SPCA performed about the same, and again, we saw some variability with the utility of the FeSK-PCA approximation. It performed a little worse than its unfederated counterpart on synthetic dataset A but a little better on synthetic dataset B and much better on synthetic dataset C. For all three datasets, one of the federated supervised approaches performed better than standard PCA, proving the utility of the proposed approaches for regression tasks. The results differed from those reported by Barshan et al. in that for synthetic dataset B, PCR outperformed SPCA, and on dataset C, SPCA outperformed KSPCA. We suspect this results from using a linear label kernel rather than an RBF label kernel. Regardless, SPCA and KSPCA performed very similarly in both our results and those in Barshan et al.'s on synthetic dataset B, while on synthetic dataset C, SPCA and KSPCA performed similarly to each other in both our results and Barshan et al.'s results.

Results for the regression experiment with the DLBCL dataset are shown in Fig. 18. Here, the federated algorithms dual FeS-PCA and FeSK-PCA performed about the same as their

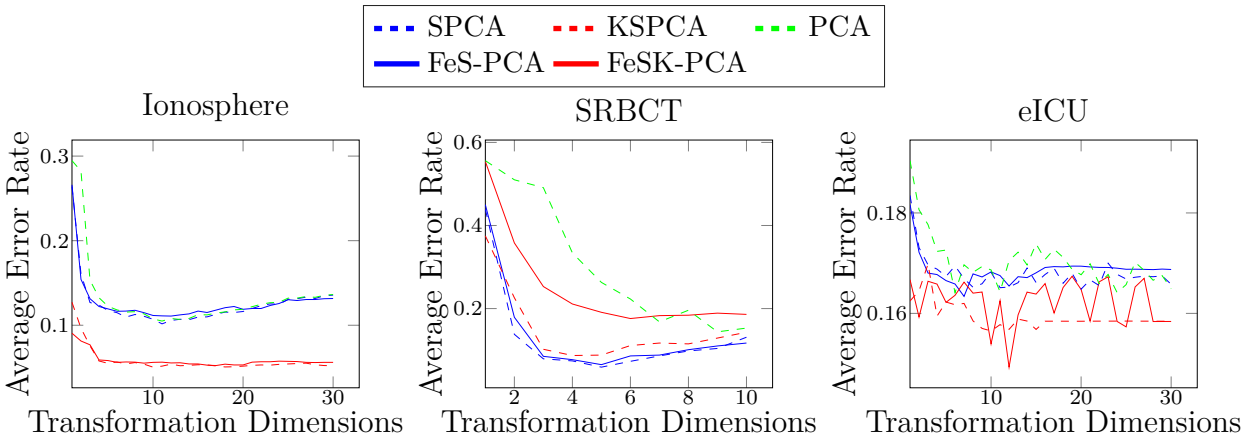

Figure 16: The average error rate with respect to the number of transformation dimensions of FeS-PCA and FeSK-PCA and their unfederated counterparts on the Ionosphere, SRBCT, and eICU datasets. The dual formulations of FeS-PCA and SPCA were used on the SRBCT dataset.

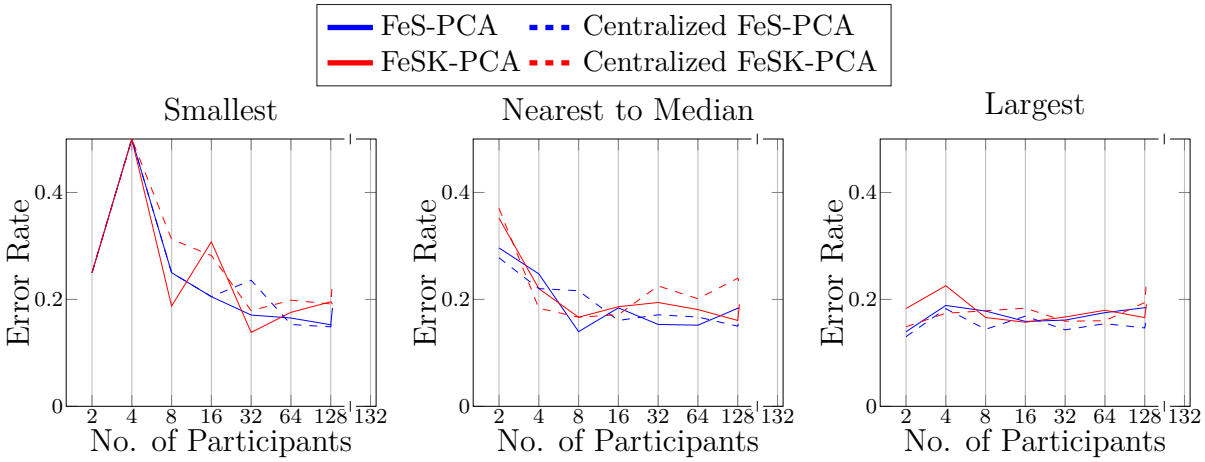

Figure 17: The error rate of FeS-PCA and FeSK-PCA with respect to the number of participants on the eICU dataset, using the participants with the smallest, the largest, and the nearest to median number of samples.

unfederated counterparts, and importantly, dual FeS-PCA and FeSK-PCA outperformed standard PCR. FeS-PCA and SPCA did not perform as well as SPCA did in Barshan et al. but this could be attributed to a difference in label kernel.

5.3.3 Scalability

Here, we present the procedure and results for our experiments measuring the scalability of the proposed approaches by recording the runtime in various configurations.

Dataset	FeS-PCA	FeSK-PCA	SPCA	KSPCA	PCR
Synthetic A	1.728	1.433	1.747	1.322	1.847
Synthetic B	0.499	0.336	0.429	0.392	0.409
Synthetic C	0.755	0.249	0.517	0.559	0.985

Table 13: The average RMSE of FeS-PCA and FeSK-PCA and their unfederated counterparts on synthetic regression datasets.

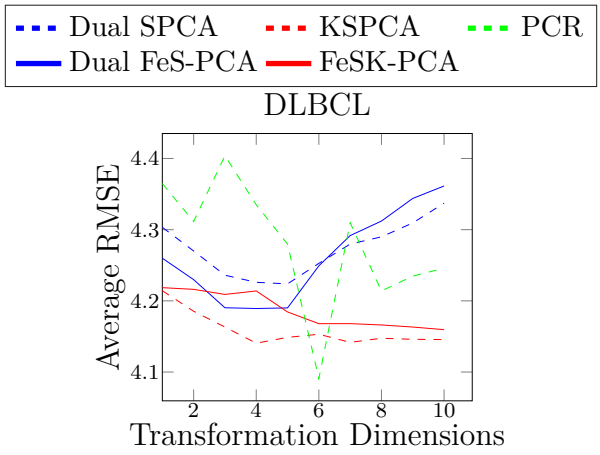

Figure 18: The average RMSE of dual FeS-PCA and FeSK-PCA and their unfederated counterparts on the DLBCL dataset.

Procedure

To test the scalability of our proposed methods, we measured the runtime for applying FeS-PCA, dual FeS-PCA, and FeSK-PCA to synthetic datasets generated using the same approach as Chai et al. We ran two sets of experiments, one where we varied the dataset size and the number of participants and another where we varied just the number of participants. In the former, for FeS-PCA and FeSK-PCA, we set the number of samples, n, equal to 2,000 for each participant, and the number of features, m, equal to 1,000. However, for dual FeS-PCA, because its computation time scales with n rather than m, and it is designed to be used when $n << m$, we set $n = 1,000$ and $m = C \times 2,000$ for a more meaningful comparison. We then varied C from 1 to 10. For the experiments with a varying number of participants and fixed m and n, we set $n = 10,000$ and $m = 1,000$ for FeS-PCA and FeSK-PCA, while for dual FeS-PCA, we set $n = 1,000$ and $m = 10,000$. We then varied C from 1 to 20. Since each participant can perform their steps in parallel, the execution time was recorded only for a single participant. Thus, the total execution time was the compute time at the masking server plus the compute time at a single participant plus the compute time at the

factorization server (see Table 10).

Results

Fig. 19 shows the runtime results from the scalability experiments. As the number of samples increased, the empirical runtime for FeS-PCA and dual FeS-PCA were dominated by the $O(n^3)$ runtime of ROMM generation. Because, the smaller of the two ROMMs were not generated in dual FeS-PCA, it was slightly quicker for the same size data matrix. FeSK-PCA was much faster than FeS-PCA and dual FeS-PCA because generating ROMMs was not required. As the number of participants increased, and the number of samples was fixed, FeS-PCA and dual FeS-PCA's execution times grew very little because the generated ROMMs remained the same size. In contrast, FeSK-PCA's compute time decreased because the number of samples at each participant shrank, making k-nearest neighbours and k-means quicker to execute. Therefore, FeSK-PCA actually benefited from being an approximation because the runtime was reduced but it was still able to achieve nearly identical results to standard KSPCA, as we have shown. Taken together, the results were in line with the theoretical runtimes discussed earlier.

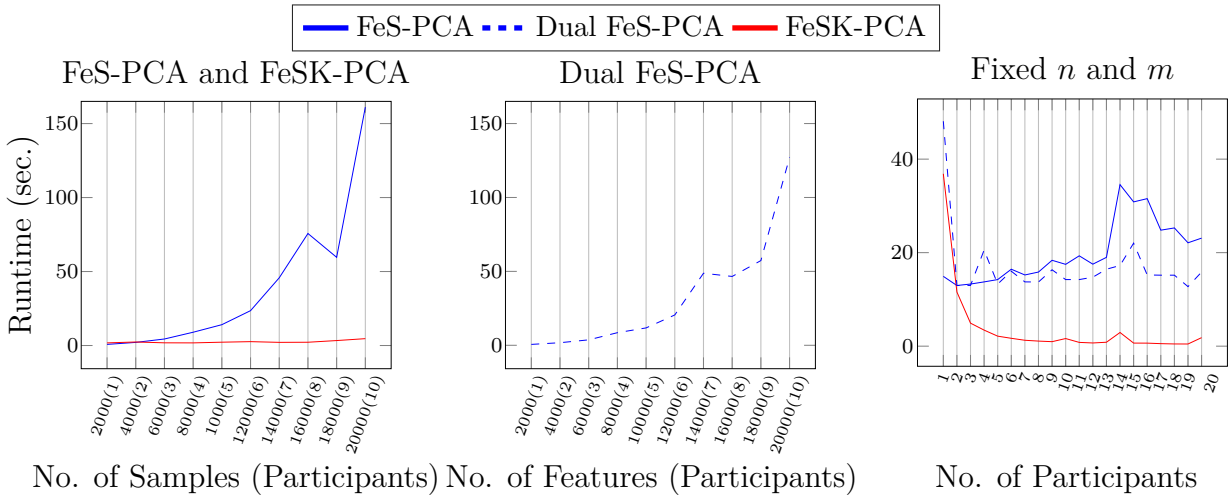

Figure 19: The runtime of FeS-PCA and FeSK-PCA with increasing numbers of participants and samples (left), of dual FeS-PCA with increasing numbers of participants and features (middle), and of all three methods with increasing numbers of participants and n and m fixed at $1,0000$ and $1,000$, respectively, for FeS-PCA and FeSK-PCA and $1,000$ and $10,000$, respectively, for dual FeS-PCA (right).

5.4 Summary

In this chapter, we proposed private federated versions of SPCA and its variants. We compared our approaches, FeS-PCA, dual FeS-PCA, and FeSK-PCA, with their unfederated equivalents by recreating visualization, classification, and regression experiments from the original SPCA paper. Our results show that our implementation of SPCA was faithful to the original and that the federated versions of SPCA and its dual formulation were lossless and thereby achieved the same performance. Even in the case of FeSK-PCA, which only approximated kernel SPCA, the performance was nearly identical to the original in the majority of cases. We also studied the effects of federation with multiple participants on FeS-PCA and FeSK-PCA using the eICU dataset, a real-world federated dataset collected from over 100 hospitals across the United States. Results with this dataset using a varying number of participants indicate that federation does not impact the quality of the resulting transformation. Finally, we recorded the runtime of our experiments in order to empirically support our theoretical time complexities.

CHAPTER 6

FLAMED-PICAFE: A Federated Learning and Private Inference Framework

In Section 2.1.2, we identified the eight possible scenarios for MLE, the most restrictive of which, scenario 7 in Table 1, was deemed impractical when relying solely on HE. In the chapters that followed, we introduced techniques for making inferences on private data using a trained LR model (Chapter 3), techniques for federated supervised PCA (Chapter 5), and a framework for model agnostic FL using density estimations (Chapter 4). In the present chapter, we introduce cooperative activation functions (CAFs), which privately compute the output of nonlinear activation functions and extend our private inference approach to NNs. This approach is combined with FLAMED and FeS-PCA in order to functionally satisfy scenario 7. We call this end-to-end FL and private inference framework federated learning via aggregating multivariate estimated densities with private inference via cooperative activation functions and encryption (FLAMED-PICAFE).

CAFs enable private inference with conventionally or jointly trained NN by using HE to compute the weighted summation of a user's data and the model owner's NN weights. Then, the resulting pre-activation is masked and returned to the user to compute nonlinear activation functions, which are not HE-friendly, on the decrypted result. The resulting masked activation is again encrypted with HE and returned to the model owner, who unmasks the result and computes the next layer's pre-activation (see Fig. 21 in Section 6.1.2). This cooperative approach readily permits the use of popular activation functions such as the ReLU because it does not rely solely on HE for the entire inference, unlike many other private inference techniques. In this way, private inference can be achieved without leaking information about a user's data or a model owner's weights, all without requiring changes to an already trained model.

FLAMED-PICAFE is a combination of CAFs and federated supervised PCA (Chapter 5) and FLAMED (Chapter 4), a FL technique which, as we saw, retains the privacy of the jointly trained model. Therefore, FLAMED-PICAFE maintains the privacy of both the training data during FL and the live data during the inference phase. It also keeps the model private to a single participant, the aggregating server, during both training and testing. To the best of our knowledge, FLAMED-PICAFE is the only FL and private inference approach that can make such strong model security guarantees without the use of costly homomorphically encrypted local training. All previous FL approaches require sharing the jointly trained model with all participants.

The present chapter contains the following contributions:

- FLAMED-PICAFE is a general framework for FL and private inference that conserves model privacy in the training and testing phases.
- CAFs enable private inference without requiring changes to model training and architecture, and it can be used outside the FLAMED-PICAFE framework.
- We identify vulnerabilities in a comparison approach for private inference that compromise the privacy of model weights and, therefore, training data.
- We evaluated FLAMED-PICAFE by comparing it to baseline and state-of-the-art FL approaches on a variety of synthetic datasets.
- We also evaluated FLAMED-PICAFE using a real-world healthcare dataset from a genuine federated setting. This dataset was collected from separate institutions and thus represents a realistic federation rather than a contrived one obtained by separating a centralized dataset, as is common in related work.
- We evaluated CAFs using the best-performing model on the healthcare dataset with various levels of precision for the homomorphic operations. We compare two contrasting HE schemes and measure the effect they have on model accuracy and compute time.

6.1 Method

In FLAMED-PICAFE, the model owner obtains a jointly trained model by using an updated version of FLAMED (see Chapter 4), which is improved by replacing its federated PCA preprocessing step from Chai et al. [57] with supervised federated PCA (see Chapter 5). Once the jointly trained model is obtained, the model owner can allow others to use it

for inferences on private data using CAFs. In the following subsection, we review the HE techniques used in the present chapter and then introduce CAFs. This is followed by a proof that CAFs provide privacy for both the jointly trained model and the live data it enables inferences on. Finally, we combine the three components, federated supervised PCA, FLAMED, and CAFs, into FLAMED-PICAFE, before discussing its benefits and novelty.

6.1.1 Preliminaries

Before presenting our method, we give some brief background on HE and review the HE techniques used by CAFs.

Homomorphic Encryption

As previously discussed (see Chapter 2.1.1), HE was originally proposed in [7], and allows the evaluation of certain operations on encrypted data so that upon decryption, the same result is obtained as operating on the unencrypted data. Recall that in SWHE, both addition and multiplications are supported, but only for a limited number of times. SWHE is more computationally efficient than FHE, but noise added during encryption in order to mask the underlying data grows with each subsequent operation until a certain point where decryption fails, hence the limited number of encrypted operations. Multiplying encrypted values grows this noise much quicker than addition, so we typically aim to keep the multiplicative depth of functions implemented with SWHE as small as possible.

Two popular examples of SWHE schemes are the BFV [98] and CKKS [99] schemes, both of which where used in Section 3.2 and are use again in the present chapter. Recall that the former allows operations to be carried out on encrypted integers with exact results, while the latter allows operations to be carried out on real or complex numbers but with approximate results. Just as in Section 3.2, because BFV only allows operations on encrypted integers, real numbers must be encoded as integers before encryption. The method we adopt in the present chapter is to simply scale real numbers by a large whole number then truncate the decimal part. After decryption, the result is then correspondingly downscaled. Multiplying two scaled values together produces a result that is doubly scaled, so care must be taken to track the scale of encrypted values during the evaluation of encrypted functions. Although CKKS permits real numbers, the noise added during decryption causes approximate results upon decryption, so we may also wish to scale the input values to retain more precision since the noise affects the least significant digits of the result first. Both the BFV and CKKS schemes use different methods to map plaintext vectors to plaintext polynomials. A noise polynomial and another polynomial derived from the public key is then multiplied by

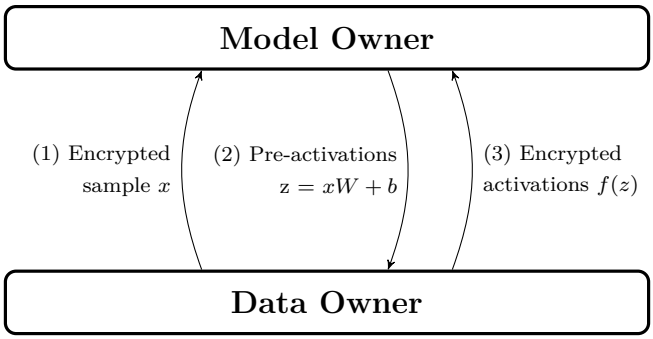

Figure 20: A diagram of the private inference data flow followed by CAFs and Orlandi et al. In CAFs, the pre-activations z are masked, and the activation is computed by the data owner in such a way that the mask can be removed by the model owner.

this plaintext polynomial to obtain the encrypted polynomial. After performing encrypted operations, the encrypted polynomial can be decrypted using the public key.

Although CAFs can in principle be implemented using any PHE or FHE scheme, we use BFV and CKKS in the present chapter because of their contrasting strengths, namely the ability to compute exact results versus operate on real numbers, and we are interested in the effect these differences have on the performance of our proposed CAFs. For details on the BFV and CKKS schemes, we refer the reader to the original papers. For the present chapter, the reader should keep in mind the above mentioned differences and the limiting effect noise has on the depth of encrypted functions.

6.1.2 Cooperative Activation Function

As mentioned in Section 2.1.2, Orlandi et al. [18] also implemented private inference with NNs using HE. In their approach, the client (data owner) encrypted their private data and sent the encrypted vectors to the server (model owner). The server, which wants to keep its NN's weights private, partially computed the output of the first layer of its NN. Only the dot product between the client's encrypted vectors and the first layer's weight matrix was computed, with biases added, because this portion of the computation is homomorphic friendly. The result was an encrypted vector of the first layer's pre-activations, which was sent back to the client. The client then decrypted the pre-activations and applied a nonlinear activation function before encrypting the resulting activations, which were returned to the server and used as input in the next layer. The general data flow for this approach, which we also use in CAFs, is illustrated in Fig. 20. Obviously, the client can leak one row of the weight matrix per inference by submitting a vector with one non-zero element. Orlandi et al. claim to provide privacy against such an attack by randomly permuting pre-activations

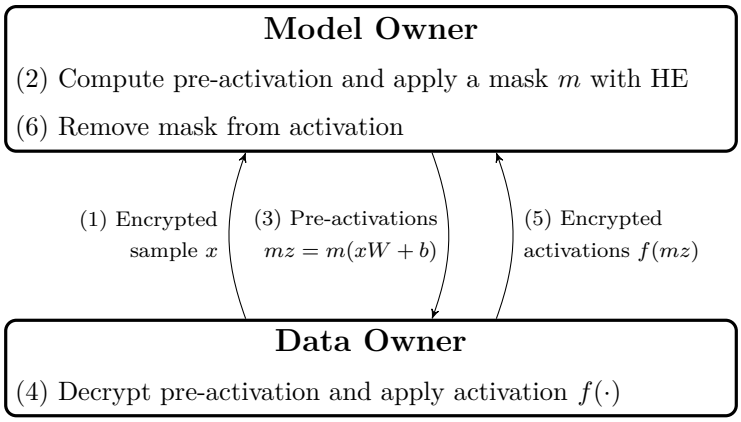

Figure 21: A diagram of the private inference data flow followed by CAFs. The exact method used for removing masks differs depending on the activation function used.

and randomly flipping their sign before returning them to the client. Therefore, although the client may obtain one row of the weight matrix per inference, the elements in different rows will not have corresponding order, with $n!$ possible permutations for each row where n is the size of the rows in the weight matrix. Further, each element will have a random sign, so if the true permutation was discovered, there would be 2^n different possible choices for the signs of the elements. The server can reverse these defense measures by returning the elements to their original order and taking advantage of the fact that many activation functions are symmetric about zero (e.g. sigmoid), so flipping the sign of their output gives the same result as flipping the sign of the input.

However, Orlandi et al.'s method suffers from a weakness that allows an attacker to reconstruct the weight matrices of the jointly trained model. The weakness stems from the fact that although any vector returned to the attacker may be one of over $n!$ permutations of the true vector obtained by the model owner, the attacker has extra knowledge that can be used to obtain a much smaller set of permutations, often a singleton set, which must contain the permutation applied by the server. This extra information is the fact that any vector returned by the server that is not masked (Orlandi et al. only support masking for their thresholding function) is a permutation of a vector resulting from a linear combination of the rows of the weight matrices held by the server. The attack is detailed in the Appendix.

Due to the weakness of the Oralndi et al. method, we propose a novel approach to enable private inference with encrypted data. Our approach is called CAF. As its name suggests, it enables inferences with private data using cooperation between the client (data owner) and server (model owner). Fig. 21 illustrates the data flow of our proposed approach. In general, our approach follows Orlandi et al. in that the client encrypts their data and sends it to the model owner, who homomorphically computes pre-activations for a given layer in their NN.

However, in our approach, the server always masks pre-activations before returning them to the client to compute HE incompatible activation functions. The masked activations are then encrypted and returned to the server and are then unmasked before being used as input to the next layer. We also extend our technique to the popular ReLU activation function. The objective of our approach is to enable private inference with popular NN activation functions, namely the ReLU and sigmoid activation functions, as well as thresholding and softmax for converting raw class scores to a predicted class label. In the following, we detail these functions and prove the privacy of the proposed techniques.

For all the CAFs discussed below, random permutation is still used to further obfuscate the pre-activations returned to the client; however, permuting encrypted vectors in the BFV and CKKS HE schemes is not trivial. To get around this, the columns of each layer's weight matrix are permuted before each encrypted inference. Because this would result in permuted inputs to subsequent layers, the rows of each layer are also permuted to match the column permutation of the previous layer. In this way, the vectors obtained by the client will still be permuted without requiring extra manipulations of encrypted vectors, while also still preserving the result.

Sigmoid

Our technique builds on a sigmoid approximation derived by Jian et al. [100] for privately training LR in federated settings using HE in combination with hardware isolation. Their approximation is given as

$$g(z) = \frac{0.5z}{\sqrt{2.722 + z^2}} + 0.5 \tag{1}$$

Notably, this approximation is an algebraic function, so it does not require computing the exponential function that, in practice, is difficult to mask and unmask without losing precision. The approximation is illustrated in Fig. 22 alongside the original sigmoid function, $\sigma(z) = \frac{1}{1+e^{-z}}$. Jian et al. found the above approximation by minimizing the mean squared error between the algebraic function $g(z) = \frac{\alpha z}{\sqrt{b+z^2}}$ and the original sigmoid. However, because they proposed a technique for privately training a LR model and they could not compute the inverse square root homomorphically, they used Newton's algorithm to approximate the inverse square root in equation 1 using only additions and multiplications; that is, they used an approximation of an approximation. However, because we have the client to compute the inverse square root, we do not need to rely on this second approximation and can directly use equation 1, as shown below. Further, the cooperative approach allows the client to reset the noise in the encrypted results while computing each activation function, allowing for more accurate predictions when using deeper NNs. This would be too costly during training, but

because we are focused on the inference phase, we can rely on this cooperative approach rather than using trusted execution environments, like Jian et al.

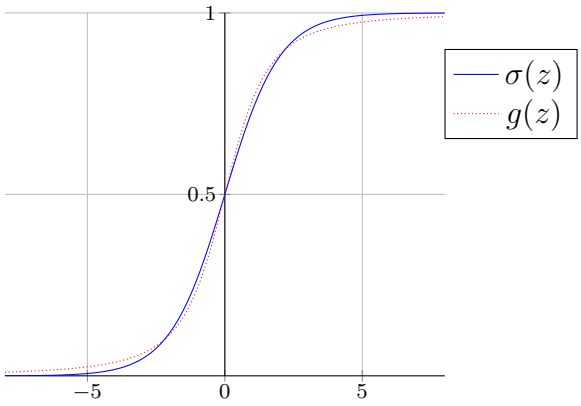

Figure 22: The true sigmoid function, $\sigma(z)$, and the approximation, $g(z)$, given by Jian et al. and used in CAFs.

We leverage this approximation for secure inference with NNs containing the sigmoid activation function. The client encrypts their data x using HE and sends it to the server, which computes the pre-activations for each node in a given layer with $z = wx + b$. The server then computes the masked numerator and denominator $n0.5x$ and $m^2 2.722 + (xm)^2$, respectively, where the masks n and m are randomly sampled real values. The server returns these masked values to the client, who decrypts them and computes

$$\begin{aligned} g'(z) &= \frac{n0.5z}{\sqrt{m^2 2.722 + (zm)^2}} \\ &= \frac{n0.5z}{m\sqrt{2.722 + z^2}} \end{aligned} \tag{2}$$

The results are encrypted and returned to the server, which unmasks them and finishes the approximate sigmoid computation with

$$\begin{aligned} g(z) &= \frac{m}{n} g'(z) + 0.5 \\ &= \frac{m}{n} \frac{n0.5z}{m\sqrt{2.722 + z^2}} + 0.5 \\ &= \frac{0.5z}{\sqrt{2.722 + z^2}} + 0.5 \end{aligned} \tag{3}$$

The server knows the plaintext values of m and n, so it can easily compute the above, but the result is encrypted, keeping $g(z)$ hidden from the server. Because the multiplicative depth of functions evaluated using HE is limited, the unmasking in equation 3 and the application

of the next layer's weights and masks are performed simultaneously with a single encrypted multiplication. So, the numerator for a neuron in the next layer is calculated as

$$\begin{aligned} n'0.5\left(\sum_{i=0}^{k} w_i g(z)_i + b\right) &= \sum_{i=0}^{k} n'0.5 w_i \left(\frac{m_i}{n_i} g'(z)_i + 0.5\right) \\ &\quad + n'0.5b \\ &= \sum_{i=0}^{k} \frac{n'0.5 w_i m_i}{n_i} g'(z) + n' w_i 0.25 \\ &\quad + n'0.5b \end{aligned} \tag{4}$$

where n' and w_i are the mask and weight applied in the next layer, respectively, to the activation $g(\cdot)_i$ from the i^{th} neuron in the previous layer. The terms $\frac{n'0.5 w_i m_i}{n_i}$ and $n' w_i 0.25$ can be computed in plaintext. Similarly, the next layer's denominator is calculated with

$$\begin{aligned} & m'^2 2.722 + \left(m'\left(\sum_{i=0}^{k} w_i g(z)_i + b\right)\right)^2 \\ &= \left(\sum_{i=0}^{k} m' w_i \left(\frac{m_i}{n_i} g'(z)_i + 0.5\right) + m'b\right)^2 + m'^2 2.722 \\ &= \left(\sum_{i=0}^{k} \frac{m' w_i m_i}{n_i} g'(z) + m' w_i 0.5 + m'b\right)^2 + m'^2 2.722 \end{aligned} \tag{5}$$

where m' is the mask applied to the activations in the next layer. The terms $\frac{m' w_i m_i}{n_i}$, $m' w_i 0.5$, $m'b$, and $m'^2 2.722$ can be computed in plaintext. This ensures the multiplicative depth throughout the private inference process does not exceed two, one for removing old masks and applying new masks and weights, and another for computing the z^2 in the denominator.

Thresholding and ReLU

We follow Orlandi et al.'s implementation of the threshold function and build on it to provide a private ReLU implementation. We review their threshold function approach here (with a slight reformulation for brevity). The output of the threshold function for input z and some threshold δ is

$$\tau(z, \delta) = \begin{cases} 1 & z - \delta \geq 0 \\ 0 & z - \delta < 0 \end{cases} \tag{6}$$

However, because the input z is encrypted, the server cannot compute $\tau(z, \delta)$. Instead, for some random $m \neq 0$, the server returns $m(z - \delta)$ to the client, which decrypts it and checks if $m(z - \delta) \geq 0$. If it is, the client sets the corresponding element in the vector of activations

a to 1; otherwise it will set it to 0. The binary vector a is then encrypted and returned to the server. For each element in a, if its corresponding mask m was greater than 0, the server uses the result from the client as is because, in this case, $\text{sign}(m(1-\delta)) = \text{sign}(1-\delta)$, which implies the result returned from the client, $\tau(mz, m\delta)$ equals $\tau(z, \delta)$. If m was less than 0, the server uses $1 - \tau(mz, m\delta)$ because $\tau(z, \delta) = 1 - \tau(-z, -\delta)$. This ensures the server does not learn the output of the threshold function, while the client does not learn the value or the sign of the input to the threshold function.

However, we make one improvement to this approach. To reduce the multiplicative depth to 1, the mask m is applied to each weight and bias in the computation of the pre-activation z because the same can be done in plain text. Therefore, we have $\sum_{i=0}^{n} mw_i x_i + mb - m\delta = m(z - \delta)$, where x_i is the i^{th} element of the input to the current neuron, w_i is the weight applied to x_i, n is the number of elements in x, and b is the current neuron's bias. Further, reversing the masking when m is negative increases the multiplicative depth of the encrypted operation since $\tau(mz, m\delta)$ is multiplied by -1. In the next layer, $w(1 - \tau(mz, m\delta))$ will be computed when calculating the pre-activations. So, to ensure the same multiplicative depth for all activations, we instead compute $w + -w \cdot \tau(mz, m\delta) = w(1 - \tau(mz, m\delta))$ when calculating the pre-activation in the next layer.

Second, we allow the sign of the output to be kept secret even when thresholding in the final layer to produce a prediction in $\{0, 1\}$. The server simply provides a dictionary of class names, indicating to the client which class labels 0 and 1 correspond to. In the event that m is negative, this dictionary is reversed. The client does not know the original label corresponding to each class, so they will not know when the dictionary has been reversed, meaning they will not know whether they were given an input with a flipped sign for thresholding.

Our ReLU is implemented in a similar way using the threshold function with $\delta = 0$ and returning z when z is greater than 0. Formally,

$$\text{ReLU}(z) = \begin{cases} z & z \geq 0 \\ 0 & z < 0 \end{cases} \tag{7}$$

Because simply permuting the activations is not sufficient to retain the privacy of the model's weights, the pre-activation z is masked using the same technique as the threshold function before being sent to the client. However, for the ReLU, the client returns both $a_+ = \text{ReLU}(mz)$ and $a_- = \text{ReLU}(\text{-}mz)$. The server then computes $\frac{w}{m}a = \frac{w}{m}1_{\geq 0}(m)a_+ + \frac{w}{m}1_{<0}(m)a_-$, where $1_{\geq 0}(\cdot)$ and $1_{<0}(\cdot)$ are the indicator functions that returns 1 if the the input is positive and negative, respectively, and 0 otherwise. $\frac{w}{m}1_{\geq 0}$ and $\frac{w}{m}1_{<0}$ can be calculated in plaintext, giving the encrypted calculation a multiplicative depth of 1. Because $1_{\geq 0}$ is 0 whenever $1_{<0}$ is 1, and vice versa, $\frac{w}{m}a$ will equal either $\frac{w}{m}a_+$ or $\frac{w}{m}a_-$. When m is positive, $\frac{w}{m}a$ will

equal $\frac{w}{m}a_+ = \frac{w}{m}\text{ReLU}(mz)$, meaning $\frac{w}{m}a$ will equal $\frac{w}{m}mz$ when z is positive and 0 otherwise. Further, when m is negative, $\frac{w}{m}a$ will equal $\frac{w}{m}a_- = \frac{w}{m}\text{ReLU}(\text{-}mz)$, again meaning $\frac{w}{m}a$ will equal $\frac{w}{m}mz$ when z is positive and 0 otherwise. So, in both cases, the result is $\frac{w}{m}mz = wa$ when z is positive and 0 otherwise. In this way, the server computes the weighted input to the next layer and removes the mask applied to z with a single encrypted multiplication. If a multiplicative mask m' is applied in the next layer to cooperatively compute another sigmoid, ReLU, threshold, or softmax function (see 6.1.2), then the server computes $m'\frac{w}{m}1_{<0}$ and $m'\frac{w}{m}1_{\geq 0}$ in plaintext to obtain $m'wa$ with a single encrypted multiplication.

Multiclass Predictions

Our thresholding can easily be used to deliver a prediction in the binary classification scenario. In the multiclass scenario with C classes, one route may be returning $\lceil \frac{C}{2} \rceil$ pairs of scores to the client, obtaining a vector indicating the greater member of each pair, and repeating for $\lceil \log_2(C) \rceil$ rounds. However, this incurs a larger communication overhead and multiple rounds of communication while needing to account for the ability of the client to craft arbitrary vectors instead of ones that faithfully indicate the greater member of each pair. Instead, we opt for returning a matrix of encrypted values in a single round of communication. The server computes each class score and then computes the difference between each pair of scores. Each difference is then uniquely masked by a random positive multiplicative mask and put into an upper triangle $C \times C$ array $\hat{Y}$ such that $\hat{Y}_{i,j|j>i} = m_{i,j}(a_i - a_j)$. The client can then use this matrix to find the maximum class if $\hat{Y}_{i,j} > 0$ then $a_i > a_j$. If $\hat{Y}_{i,j}$ is not in the upper triangle, then $\hat{Y}_{j,i}$ is and will give the same information. Once they find the index i for which no element in $\hat{Y}$ implies $a_i < a_j$, the client can use this to determine the corresponding class label using a dictionary returned by the server. This also allows the client to determine the entire ordering of the class predictions, but does not enable them to see the relative probabilities.

6.1.3 CAF Privacy Properties

In the following, we give a proof of privacy for our private NN inference using cooperative activation functions. In particular, we prove the privacy for each of the components of our approach, namely sigmoid, ReLU, thresholding, and multi-class predictions. In each proof, we show that the server learns nothing about the client's data, while the client learns nothing about the server's NN parameters. We also assume the existence of a secure HE scheme that supports addition and multiplication, such as CKKS or BFV. Throughout this section, encrypted components are indicated with $[\cdot]_e$.

Definition 6.1.1. *The approaches in 6.1.2 are private if*

1. *the server learns nothing about the client's input x, including its predicted label.*

2. *the client learns nothing about the server's NN weights.*

Theorem 6.1.1. *The cooperative sigmoid activation function in 6.1.2 is private according to definition 6.1.1.*

Proof: During the computation of the cooperative sigmoid function, the server only learns $[x]_e$ and $[\frac{n0.5z}{m\sqrt{2.722+z^2}}]_e$. Both these terms are homomorphically encrypted and any result computed using them will necessarily be encrypted as well. Therefore no information about their plaintext counterparts is leaked, provided the HE scheme used to encrypt them is secure and the client does not share their private key with the server. This implies that no information about the client's data can be learned by the server. In contrast, the client obtains $n0.5z$ and $m\sqrt{2.722+z^2}$ in plaintext. Each of these terms are some product of random variables m and n and are therefore themselves random variables. Further, these terms are always returned as elements within a vector of size $|l|$, where $|l|$ is the size of the l^{th} layer whose activations are currently being calculated, and each element has a different random m and n. This vector is randomly permuted with $|l|!$ possible permutations and, unlike in Orlandi et al.'s private sigmoid, the masks m and n ensure there are there is no deterministic relationship between such vectors obtained from multiple chosen inputs. Therefore, there is no way to sample the masked values $n0.5z$ and $m\sqrt{2.722+z^2}$ outputted from a specific neuron for a given input. An attacker can only obtain a sample of the masked values from all neurons, without knowing from which neuron each masked value originated. Thus, the client could only, at best, obtain a empirical distribution of $n0.5z$ and $m\sqrt{2.722+z^2}$ over an entire layer for a given input, but learns nothing about the true value of z. The client cannot obtain individual weights, let alone entire weight matrices with elements in the correct positions. ■

Theorem 6.1.2. *The cooperative threshold and the ReLU activation functions in 6.1.2 are private according to definition 6.1.1.*

Proof: During the evaluation of the cooperative threshold function, the server again only learns encrypted values. So, for the same reasons as in the proof for Theorem 6.1.1, no information about the client's data can be learned by the server. The client only obtains mz, which again is some product of random variable m; therefore, it is itself a random variable. Here, too, this value is returned as an element in a vector with $|l|!$ possible permutations, and the different masks m applied to each element ensure there is no deterministic relationship

between such vectors obtained from multiple chosen inputs. Therefore, following the same logic as Theorem 6.1.1, the client could only, at best, obtain an empirical distribution of mz over an entire layer for a given input, and again, this leaks no information about individual weights. ■

Theorem 6.1.3. *The cooperative multiclass prediction function 6.1.2 is private according to definition 6.1.1.*

Proof: Just as before, the server cannot learn anything about the client's data because all information received by the server is securely encrypted. The client obtains a matrix $\hat{Y}$ where $\hat{Y}_{i,j|j>i} = m_{i,j}(a_i - aj)$ for some random variable $m_{i,j}$. Similarly, this implies that each nonzero element in $\hat{Y}$ is a product of IID random variables. Also, because this activation function is only used in the last layer, the inputs are expected to have been randomly permuted and masked. Any any arbitrarily chosen input will be randomly permuted and randomly masked (because the server will still apply the inverse of the permutation and mask used in the last layer) before the computed differences are masked again. This means there is no way to compute an attack that leverages chosen inputs because the attacker will have no way of knowing what the actual input became. The client can only determine what changes in inputs to the first layer produced a classification change, which is always the case in all private inference methods. Thus, the client cannot learn anything about the true value of $a_i \forall i \in \{1, 2, \ldots, C\}$ except for the ordering of these values in relation to each other. ■

6.1.4 FLAMED-PICAFE

We can now bring together the techniques just discussed to define FLAMED-PICAFE. FLAMED-PICAFE consists of three phases. (1) FeS-PCA or FeSK-PCA is performed in order to project participant data into a subspace with maximum dependence on class labels. This projection is found using all participants' data and without compromising their data privacy. (2) Perform FL using FLAMED. The participants' data is again kept private throughout the process, while the jointly trained model is known only to the aggregating server. (3) Allow other clients to send their homomorphically encrypted live data to the model owner to carry out inference using CAFs. Here, the client's live data are kept private from the model owner, while the model weights are kept private from the client.

FLAMED-PICAFE has numerous benefits:

- It is the first approach of its kind that enables FL where the aggregating server retains sole ownership of the jointly trained model and can retain ownership even when allowing an arbitrary amount of inferences on live data. This incentivizes participants to

serve as the aggregating server because as the model's sole owner, they can monetize, or otherwise benefit from, the use of their model.

- Because the jointly trained model can be more readily used by others who were not involved in the FL process without the need for them to share their data, the potential use cases for FL expands to scenarios where the jointly trained model is used on private data in production.

- Data owners can be motivated to participate in joint training by entering into agreements regarding the distribution of profits and the like obtained through use of the jointly trained model. This is like any other industry in which producers are compensated for their work despite not having ownership of the final product. Such compensation can be viewed as similar to royalties, where the model owner acts like a publisher, and the data owners, who may not possess the means to publish or monetize the model themselves, receive a share of the profits from the model owner, who has control of the model's publication and monetization.

- By using methods other than HE to keep training data and the jointly trained model private, FLAMED-PICAFE satisfies the most restrictive privacy scenario in [101], which is otherwise impractical when using purely HE to implement privacy. The framework can be adapted to any more relaxed privacy requirement as well. If the training data are not required to be kept private, standard ML training methods can be used. If the jointly trained model is not required to be kept private from participants in the FL network, standard FL methods could be used, or if the jointly trained model is not required to be private at all, then it can be distributed to whoever would like to use it. Lastly, if the live data are not required to be kept private, it can be sent to the model owner unencrypted.

- FLAMED-PICAFE inherits FLAMED's relative robustness against model poisoning attacks when compared with other FL methods in the literature.

6.2 Experiments and Results

In this section, we present the configurations and results of our experiments. We with our experiments which compare the performance of FLAMED using FeS-PCA or FeSK-PCA to the original FLAMED approach. Then, we present experiments illustrating the susceptibility of standard FL approaches to membership inference attacks. Finally, we present experiments

evaluating the accuracy of CAFs compared to their non-private counterparts when using various HE precisions.

6.2.1 Performance on Learning Task

The following experiments compare FLAMED's performance as it was originally proposed with FLAMED-PICAFE's performance which uses FLAMED with FeS-PCA or FeSK-PCA in place of FedSVD.

Experiment Setup

To evaluate the performance gain from upgrading FLAMED using FeS-PCA or FeSK-PCA for use in FLAMED-PICAFE, we compared it against FLAMED using Chai et al.'s FedSVD. We generated 432 synthetic federated datasets according to the method in [81], which was the same method used in the chapter on FLAMED and the FedSVD paper. We briefly review this method here so the section can remain self contained. In the following, K is the number of participants in the FL network, c is the number of classes, and m is the number of features in their data. The total number of samples across all clients is N. For the i^{th} client, we generate a local dataset $(X_i \in \mathbb{R}^{\frac{N}{K} \times m}; Y_i \in \mathbb{R}^{\frac{N}{K}})$ by first sampling each element of the mean vector v_i from $\mathcal{N}(B_i, 1), B_i \sim \mathcal{N}(0, \beta)$ and using v_i to define the multivariate normal $x_i \sim \mathcal{N}(v_i, \Sigma)$, where the covariance matrix Σ is diagonal with $\Sigma_{j,j} = j^{-1.2}$. Therefore, β controls how much local data differs across clients. Next, we sample weights $W_i \in \mathbb{R}^{c \times m}$ and biases $b_i \in \mathbb{R}^c$ from $\mathcal{N}(u_i, 1), u_i \sim \mathcal{N}(0, \alpha)$. Class labels in Y_i are then determined with $y = \text{argmax}(\text{softmax}(W_i x + b_i))$. Thus, α controls how much the relationship between labels Y_i and features X_i differ across clients. Following FedSVD, we set α equal to β. The 432 datasets covered different configurations defined in the cross product $K \in \{2, 8, 32\} \times c \in \{2, 4, 8, 16\} \times m \in \{8, 32, 128, 512\} \times \alpha = \beta \in \{\text{IID}, 0.5, 1\} \times N \in \{5m, 10m, 20m\}$ where larger values of $\alpha = \beta$ indicate higher levels of non-IIDness, and $\alpha = \beta = \text{IID}$ corresponds to the completely IID setting in which we set v_i to the zero vector and $u_i = 0, \forall i$.

After generating each dataset, 10% of the samples at each client were used for evaluation, while the rest were used to obtain the jointly trained model. Training proceeded by following the standard FLAMED framework. We first applied FedSVD and FeSK-PCA to transform the dataset into r dimensions for $r \in \{2, 4, 8\}$. We then applied KDE at each client to obtain an estimated density for each of the transformed datasets at each client. The estimated densities were used to simulate new data points from each client's transformed data. Finally, the simulated data from each client were aggregated and used to train a simple FFNN. The best value for r, the hyper-parameters for KDE and the NN, and the NN's architecture were

determined as those that resulted in a jointly trained model that performed the best on the evaluation data. Because of the class imbalance in the datasets, performance was evaluated using balanced accuracy.

To extend the above comparison to a real-world federated setting, we used the eICU Collaborative Research Database [82]. This dataset contains real-world medical data from over 200,000 ICU admissions to more than 200 medical centres across the United States. We chose this dataset because, unlike other datasets, the eICU dataset represents a real-world federated setting instead of a contrived one obtained by separating a conventional dataset. Thus, it provides a more realistic evaluation of FL techniques. We used only the drug infusion data, which tracks the drugs administered to patients during their stay in the ICU. Each row in the feature matrix $\mathbf{X}$ corresponds to a patient, while each column corresponds to a drug. If a patient i receives any dose of a certain drug j during any ICU admission, then $\mathbf{X}_{ij}$ is set to 1. Otherwise, it is set to 0. We defined a binary classification problem in which a patient is assigned label 0 if their hospital discharge status specified in the patient table was "alive" and 1 otherwise. After removing hospitals with less than 10 observations, we were left with a total of 3,069 features and 72,959 patients across 132 hospitals. In addition to one configuration using all 132 hospitals, we tested different configurations in the cross product $K \in \{2, 4, 8, 16, 32, 64, 128\} \times \mathcal{S} \in \{\text{smallest}, \text{middle}, \text{largest}\}$, where K is the number of hospitals used and $\mathcal{S}$ denotes the strata of hospitals we selected from. If $\mathcal{S} = \text{middle}$, then we selected the K hospitals with the nearest to the median number of patients, while if $\mathcal{S} = \text{smallest}$ or $\mathcal{S} = \text{largest}$, then we selected the K hospitals with the least or most number of patients, respectively. This added 22 configurations to our experiments.

Results

Fig. 23 shows the average balanced accuracy across all synthetic dataset configurations for different values of $\alpha = \beta$ for FLAMED as it was originally proposed using FedSVD and FLAMED upgraded using FeS-PCA and FeSK-PCA. For comparison, the popular benchmark FedAvg and the state-of-the-art FedProx were also evaluated. The results indicated that using FeS-PCA slightly improves performance against standard FedSVD, but FeSK-PCA performs poorly. In the most non-IID setting, FLAMED with FeS-PCA outperforms FedAvg as well. We did see a large decrease in the performance of FLAMED when entirely IID configurations were included. However in real-world FL settings, entirely IID datasets are exceedingly rare, and there is much more utility in non-IID datasets because each client contributes new information to the global learning task, so we are much more interested in the results on non-IID datasets. Therefore, we present the rest of the results both including

and excluding the IID setting. Fig. 24 shows the average balanced accuracy across all synthetic dataset configurations for different numbers of clients and features (before applying PCA dimensionality reduction). The results are similar for all versions of FLAMED, with the FeS-PCA version performing slightly better throughout.

Fig. 25 shows the balanced accuracy obtained by the different versions of FLAMED and the comparison methods evaluated on the eICU dataset with varying numbers of clients using the clients with the least (top left), nearest to median (top right), and most (bottom) number of samples. Here, FLAMED with FeSK-PCA performed much nearer to its counterparts, indicating the potential of the method, which notably outperformed the standard version of FLAMED when all 132 clients were used. We also see an improvement across the board when using FLAMED with FeS-PCA compared with the standard version of FLAMED. This version of FLAMED even beat out all comparison methods when 4 and 8 of the largest clients were used. This may indicate that FLAMED, when given enough data at each participant to reliably simulate training data, can adequately address non-IIDness and make up for any loss in accuracy attributable to simulation.

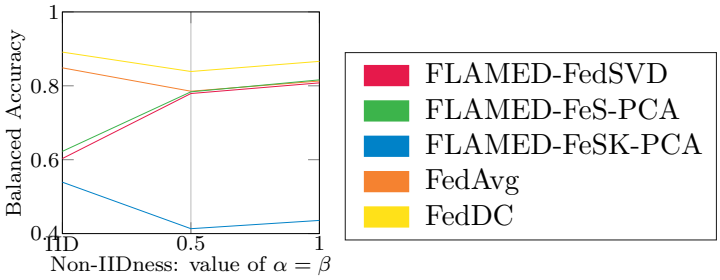

Figure 23: Average balanced accuracy for each method across all configurations with respect to the level of non-IIDness.

6.2.2 Membership Inference Attack

The experiments in this section illustrate the susceptibility of a state-of-the-art standard FL approach, FedDC, to membership inference attacks.

Experiment Setup

One key motivation for keeping a model private and forcing clients to send their data to the model owner instead is the risk of membership inference and model inversion attacks. Membership inference attacks use the output of a model, activations from intermediate layers, or some other information to infer whether a given sample was used to train a model.

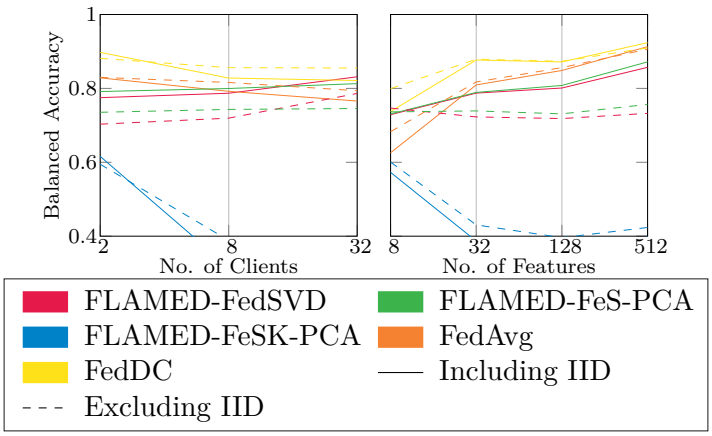

Figure 24: Average balanced accuracy with respect to the number of clients (left) and features (right) for each method across all configurations.

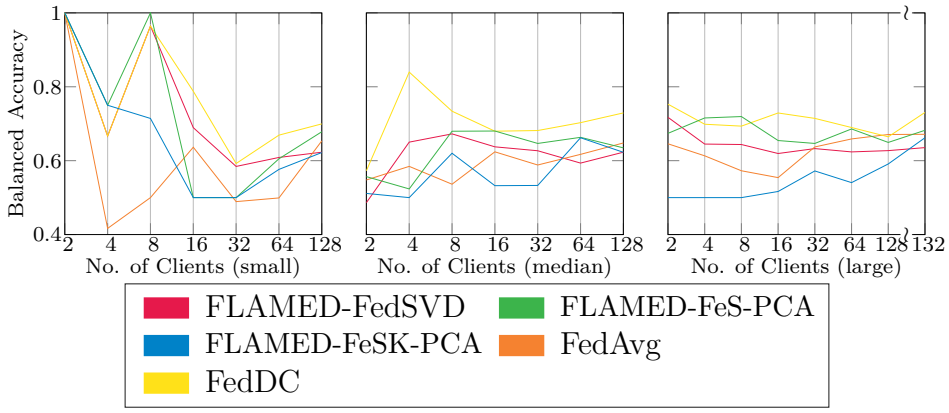

Figure 25: The balanced accuracy for each method evaluated on the eICU dataset with varying number of clients, using the clients with the smallest (top left), nearest to median (top right), and largest (bottom) number of samples.

Model inversion uses similar information to recreate input samples. As noted in the literature review, the latter is especially performant in recreating image samples.

FLAMED is the only FL method in which the participants involved in training do not necessarily receive a copy of the jointly trained model at the beginning of every epoch. However, even in cases where training participants can be trusted, making the jointly trained model public can make it susceptible to model inversion or membership inference attacks, risking the privacy of the training data. Of course, the same risk applies when a model is trained on private data in the conventional centralized manner. To demonstrate the risk of membership inference attacks when models are made public and, therefore, support the utility of CAFs in enabling inference on homomorphically encrypted data while keeping model weights private, we performed a model inversion attack against the state-of-the-art

FL technique FedDC [39].

We trained a simple 2-layer NN on the eICU dataset using all 132 clients with FedDC. We then selected the K_a clients with the nearest to median number of samples to act as attackers. Because the attackers were present during training, they knew that their training sets were, of course, used to train the jointly trained model. However, they can only assume the samples in their test sets were not present in any of the other clients's training sets. The attackers generated 20,000 random samples by duplicating their test data and randomly flipping the value of 1 or 2 other features. The attackers also assumed these randomly generated samples were not present during training. The attackers used their copy of the jointly trained model obtained at epoch e to find the activations from the first and second layers for each sample in their combined training and test sets and their randomly generated samples. The attackers then used these activations as features to train another simple FFNN to predict 1 if a sample was present in the training set and 0 if it wasn't. To evaluate the resulting membership inference model, we used all the samples that were held at the other clients. We repeated this for multiple configurations defined by the cross-product $K \in \{8, 32, 132\} \times K_a \in \{1, 2, 4\} \times \mathrm{e} \in \{2, 8, 32, 128\}$ to test the efficacy of the attack under different circumstances.

Results

Fig. 26 shows the balanced accuracy of the membership inference model across different attack configurations. The results indicate that in all cases, information has been leaked, which is shown by the balanced accuracy being always above random and with the best performing attack obtaining 66.2% balanced accuracy. The number of adversaries contributing to the real data that are known to be used during joint training has little effect on the outcome, suggesting that the main contributor to the membership inference model's performance is the randomly generated samples. The epoch from which the jointly trained model was taken in order to obtain the activations used to train the membership inference model seems to have a substantial effect on performance, but without a consistent trend. This indicates that attackers should perform the attack at several epochs and evaluate the resulting membership inference models to obtain the best-performing one. Surprisingly, as the total number of participants grew, the performance of the membership inference attack increased. The best performing attacks, with 8, 32, and 132 participants total, achieved 59.4%, 63%, and 66.2% balanced accuracy, respectively. This may suggest that as the jointly trained model is exposed to more training data, less noise is present in the activations used to train the membership inference model, making inference more accurate.

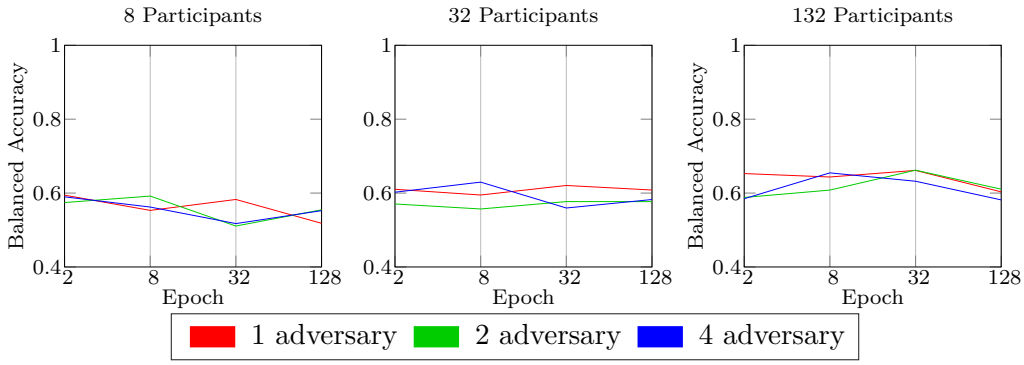

Figure 26: The balanced accuracy of the membership inference model across different attack configurations using the 8, 32, and 132 clients from the eICU dataset with the nearest to median number of samples.

6.2.3 CAFs Homomorphic Precision

In this section, we present experiments comparing the accuracy of the ReLU and sigmoid CAFs to their non-private counterparts when using various HE precisions and schemes.

Experiment Setup

As discussed in Section 6.1.1, some HE schemes, such as BFV, require real values to be encoded as integers before they can be encrypted. Typically, this amounts to scaling all values by 2^p and truncating whatever remains after the decimal point. After encrypted operations are carried out, the result is decrypted and descaled. Further, in the CKKS scheme, although real values can be directly used, noise added for privacy grows during encrypted operations and affects the least significant digits first, so scaling values here can help protect the accuracy of less significant digits. Of course, both of these approaches incur some loss in precision, which is dependent on the value of p and which may affect model accuracy, so we wanted to evaluate the performance of CAFs with different values of p. To do this, we implemented our ReLU CAF using the BFV and CKKS HE schemes (see Section 6.1.1) and the sigmoid in only the CKKS scheme because when using the BFV scheme, even with a modest scale, the squared term in the denominator caused overflow. We used the TenSEAL [102] Python library's implementation of CKKS and BFV, which is built on top of Microsoft's SEAL C++ library [103]. We then took the best-performing model using ReLUs and the best performing model using sigmoids obtained using FLAMED with FeSK-PCA on the full eICU dataset and performed private inference on the test sets from

each participant using CAFs in place of standard activation functions. In the case of the BFV scheme, the scale was 2^p, but for the CKKS scheme, which permitted a larger scale but also wiped out all accuracy with noise for $p < 20$, the scale was 2^{20+p}. We repeated this for $p \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ to evaluate the scale's effect on performance for both the sigmoid and ReLU CAFs with each HE scheme. We also recorded the execution time of the private inferences for $n \in \{1, 2, 4, \ldots, 4096\}$ samples for all HE scheme configurations. Because the scale does not affect execution time, the average execution time was taken across all scales for each HE scheme.

We also tested a slightly different formulation of the approximation in Equation 1. Although the difference between that approximation and the original sigmoid is very small, the relative difference (the difference divided by the value of the original sigmoid) is very large when $z < -4$. However, slightly adjusting the approximation to obtain

$$g(z, c) = \frac{(0.5 + c)z}{\sqrt{2.722 + z^2}} + 0.5 \tag{8}$$

results in a much smaller relative difference when $-10 < z < -4$ for small c, but a slightly larger difference elsewhere. Fig. 27 shows the relative difference between the original sigmoid and the approximations in Equation 1 and Equation 8 for $c \in \{0.008, 0.01, 0.012, 0.014\}$. We repeated the above experiment with this adjusted approximation for each of these values of c.

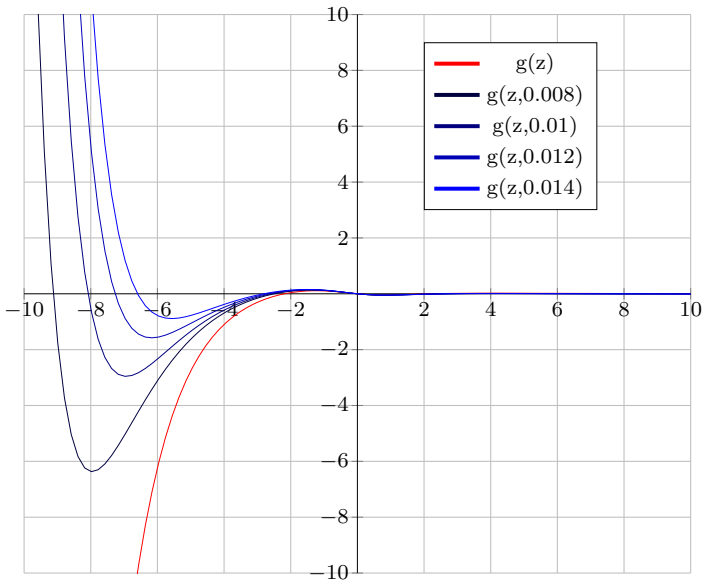

Figure 27: The relative differences between the original sigmoid and the approximations obtained using Equation 1 or Equation 8 with $c \in \{0.008, 0.01, 0.012, 0.014\}$.

Results

Fig. 28 shows the balanced accuracy for each model and HE scheme across different values of p. For the adjusted approximation from Equation 8, the best performing values of c were used. For comparison, the performance of each model using full precision without HE is also shown. The results indicate that both schemes were able to deliver full accuracy when using the ReLU CAF. The CKKS scheme was able to accurately compute the approximations in Equation 1 and 8, as evidenced by the fact that higher scales past $p = 20 + 7$ and increasing precision did not improve the accuracy, but the approximations themselves result in a modest accuracy loss of 2.6% in the best case at $p = 8$. Notably, the lightly modified approximation achieved slightly better performance for all values of p. In the same figure, we can also see the BFV scheme was significantly slower, about 10 times so, than the CKKS scheme when using the ReLU function, while the sigmoid with CKKS was comparable in compute time to the ReLU with CKKS.

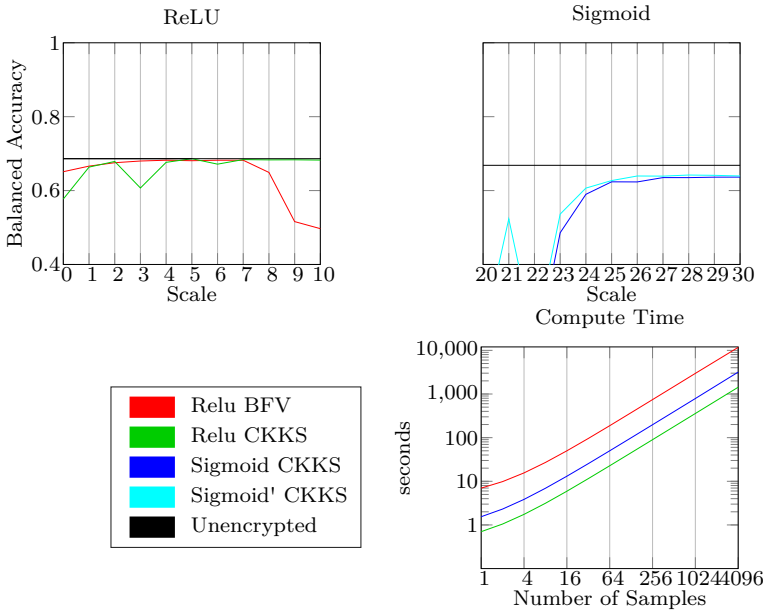

Figure 28: The balanced accuracy of the different CAF methods compared to their unencrypted counterparts for the BFV and CKKS HE schemes, using various scales and the inference time with respect to the number of samples for each method and HE scheme combination.

6.3 Summary

In this chapter, we introduced FLAMED-PICAFE, an end-to-end FL and private inference framework. The approach is a combination of the techniques discussed in Chapter 4 and Chapter 5, as well as CAFs, which were introduced in this chapter. We evaluated FLAMED-PICAFE on 432 synthetic dataset configurations and compared it to the original version of FLAMED, which did not use supervised dimensionality reduction, and to FedAvg and FedDC, a benchmark and state-of-the-art FL technique, respectively. Our results indicated a slight increase in the performance of FLAMED using FeS-PCA over the original. We also evaluated the same approaches on the eICU dataset, a real-world FL dataset from over 132 healthcare institutions. These results indicated a more significant increase in the performance of FLAMED using FeS-PCA, which now defeats FedAvg almost across all configurations and defeats FedDC in some settings when using only the clients in the eICU dataset with the largest number of samples. Further, FLAMED-PICAFE is the only FL approach that keeps the jointly trained model private to the aggregating server. The performance of the CAFs were evaluated using the best-performing model architecture from the experiments with the full eICU dataset. The results indicate that both the CKKS and BFV HE schemes were able to deliver lossless or near-lossless performance. However, a HE-friendly approximation used to compute the sigmoid activation function causes minor accuracy loss. It is important to note that CAFs can be used on their own to enable private inference for any model, jointly trained using FL or otherwise, so long as the activation functions are supported by the implemented CAFs.

CHAPTER 7

Conclusion

7.1 Contribution Summary

This dissertation introduces a general end-to-end FL and private inference framework called FLAMED-PICAFE. FLAMED-PICAFE maintains the privacy of both the training data during FL and the live data during the inference phase, all while keeping the model private to the aggregating server, during both training and testing. All previous FL approaches require sharing the jointly trained model with all participants. Therefore, FLAMED-PICAFE is, to the best of our knowledge, the only FL and private inference method that functionally satisfies scenario 7 in Table 1, where the training data, testing data, and model weights are all encrypted with HE. This was deemed impractical when relying solely on homomorphic encryption, but FLAMED-PICAFE achieves the same privacy guarantees and does so for the FL setting, not just the conventional centralized ML setting.

FLAMED-PICAFE is comprised of three separate techniques, also detailed in this dissertation. The model owner obtains a jointly trained model by using FLAMED (Chapter 4) with FeS-PCA (Chapter 5) for supervised dimensionality reduction as a preprocessing step. Once the jointly trained model is obtained, the aggregating server, as the sole model owner, can allow others to use their model for inferences on private data using CAFs (Section 6.1.2). We evaluated FLAMED-PICAFE by comparing it against baseline and state-of-the-art FL approaches on a variety of synthetic datasets. We also evaluated FLAMED-PICAFE using a real-world healthcare dataset from a real-world federated setting with 132 participants.

FLAMED was detailed in Chapter 4 and is an alternative framework for FL that bypasses the challenges posed by non-IID data (see Section 1.1), while also contributing to the simultaneous protection of data privacy and model performance. FLAMED is a FL approach that enables the aggregating server to obtain a global model that is not known to any other participants. Each client models the probability distribution of their local data and sends this information to the aggregating server, allowing the aggregating server to simulate centralized

global training. We evaluated FLAMED against baseline and state-of-the-art FL approaches on the same synthetic and real-world datasets used to evaluate FLAMED-PICAFE. We also evaluated FLAMED's resilience to model backdoor attacks.

Both FLAMED-PICAFE and FLAMED allow the model to be known solely by the aggregating server. This is important because restricting knowledge of the global model to a single model owner protects the model's intellectual property and guards against a malicious participant using shared model weights to attack training data privacy. As mentioned before, providing protection against such attacks and securing the model owner's propriety of the jointly trained model motivates further innovation.

FeS-PCA and its variants, dual FeS-PCA and FeSK-PCA, were detailed in Chapter 5. Although a federated version of PCA already exists in the literature, one shortcoming of PCA is it does not take into account data labels for supervised learning tasks. FeS-PCA therefore extends supervised PCA to the federated setting. Dual FeS-PCA provides an alternative technique that is more efficient for datasets with more features than samples while FeSK-PCA leverages the kernel trick to find nonlinear transformations. In all three cases, participants gain the principal components of their combined data without needing to share their local data, without compromising its privacy. FeS-PCA can therefore be used as a supervised privacy-preserving preprocessing step in FL settings to find a transformation of data into a lower dimensionality.

CAFs were detailed in Section 6.1.2 and they enable private inference with conventionally or jointly trained NN by using HE and masking with multiple rounds of communication between the model owner and data owner. This cooperative approach permits the use of the ReLU and sigmoid activation functions. This means a model can be used for private inference without changing the training process or model architecture, which can have a negative effect on model performance. Further, the multiplicative depth of homomorphic operations can be kept to a minimum, enabling the accurate evaluation of deeper NNs. CAFs ensure the privacy of both input data and model weights. To motivate CAFs, we identified a vulnerability in a comparison approach for private inference that compromises the privacy of model weights.

7.2 Future Work

It is interesting to consider repeating FLAMED for several rounds, such as in standard FL, to iteratively improve the global model with a technique perhaps similar to AdaBoost. An improvement on FLAMED, which relies on a version of FedSVD that doesn't require a trusted masking server, would make the dimensionality reduction step more convenient in practice.

FLAMED can also be extended with histogram probability mass function estimation for discrete features. In vertical FL, all participants have samples which correspond to the same individual instance, but each participant has a different set of features. Methods for privately linking samples that correspond to the same instance could extend FLAMED to the vertical FL setting. Online dimensionality reduction techniques extended for FL, combined with online multivariate density estimation (e.g. [104]) and online anomaly detection [105] can extend FLAMED to the online FL setting. Lastly, for data poisoning detection, more advanced methods than those put forward in this chapter can be investigated.

For FeSK, it would be beneficial to eliminate the need for a trusted masking server by perhaps using a federated QR decomposition technique [106] to generate the ROMMs. A method for applying the ROMMs without the need for any one participant to hold the left-hand side ROMM would also raise the collusion threshold for dual FeS-PCA to the factorization server plus $K-1$ participants. Similarly, a federated decomposition technique used in dual FeS-PCA could remove the need to share labels when calculating and decomposing the label kernel matrix. There is also the question of using distributed k-means [107] to improve the approximation of FeSK-PCA. Lastly, methods for securely computing a nonlinear label kernel in FeS-PCA can be proposed and studied.

It remains to be determined what the exact cause is of the drop in accuracy when using the IID synthetic dataset configurations in chapters 4 and 6. It is possible that the estimated densities of the similarly distributed datasets interfere with each other, confusing the global model during training. Therefore, perhaps a federated version of KDE or some other density estimation technique will help performance in this setting. Regardless of the solution, this improvement may even help in the non-IID settings as well. Our results in Chapter 6 also indicated that while FeSK-PCA can find useful transformations on some datasets, its performance is inconsistent, possibly due to the selection of sensitive hyperparameters like *gamma* used in the RBF kernel. So, future work should investigate improving FeSK-PCA's hyperparameter selection or find alternative kernel functions that give more stable performance. Lastly, CAFs currently support four activation functions, so, this list should be extended following similar approaches as those discussed in Chapter 6 to enable private inference using a wider variety of model architectures.

In standard ML, interpretability is the degree to which a human can understand the cause of a prediction [108, Sec. 3.0]. Although not discussed in this dissertation, enabling interpretability methods when performing private inference would extend the benefits of interpretability (e.g. prediction error debugging, improving model trust, etc.) to the private ML setting. Data used in private ML settings is in many cases private because it specifies personal characteristics. If the reasons behind a ML model's predictions are not investigated,

then the model could produce unintentionally discriminatory results. Another common use case for private ML is in the healthcare setting. Here, due to the potentially serious and life-threatening nature of a misclassification, practitioners may be hesitant to use a prediction from a model they have no intricate knowledge of without further explanation and justification for a given prediction. However, in the case of FLAMED-PICAFE and private ML, the data owner at inference time is necessarily dealing with a black box and cannot easily determine the reason behind a given prediction while the model owner, unaware of the prediction and the feature values that caused it, would not be aware if the prediction is discriminatory or otherwise erroneous. An interpretability method that lets the data owner know the reason behind a prediction, without compromising the privacy of the private data or the model, could be used to improve the trust of the data owner in the prediction they receive, and allow the data owner to report erroneous associations learnt by the model to the model owner.

FLAMED-PICAFE, which leverages CAFs, currently has no way of providing interpretability for private inferences. However, since in FLAMED-PICAFE, the aggregating server has access to all the simulated data used to train the jointly trained model, a modification of approaches like Ribeiro et al.'s [109] surrogate method, where several interpretable classifiers are trained with samples in different regions weighed more heavily, can be used. Some interpretability methods can backpropagate NN predictions using various techniques to determine the most important input features [110]. Using a combination of HE or SMC, this approach can be extended to the secure ML setting. Regardless of the particular solution, care should be taken that the privacy of the jointly trained model is preserved.

Bibliography

[1] L. Zhu and S. Han, *Deep Leakage from Gradients*. Cham: Springer International Publishing, 2020, pp. 17–31.

[2] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?" *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 937–16 947, 2020.

[3] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis *et al.*, "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[4] W. Briguglio, P. Moghaddam, W. A. Yousef, I. Traoré, and M. Mamun, "Machine learning in precision medicine to preserve privacy via encryption," *Pattern Recognition Letters*, vol. 151, pp. 148–154, 2021.

[5] W. Briguglio, W. A. Yousef, I. Traore, and M. Mamun, "Federated supervised principal component analysis," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 646–660, 2024.

[6] A. Zehir, R. Benayed, R. H. Shah, A. Syed, S. Middha, H. R. Kim, P. Srinivasan, J. Gao, D. Chakravarty, S. M. Devlin *et al.*, "Mutational landscape of metastatic cancer revealed from prospective clinical sequencing of 10,000 patients," *Nature medicine*, vol. 23, no. 6, p. 703, 2017.

[7] R. Rivest, L. Adleman, and M. Dertouzos, "On Data Banks And Privacy Homomorphism," Massachusetts Institute of Technology, Tech. Rep., 1978.

[8] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–35, jul 2018.

[9] C. Gentry, "Fully homomorphic encryption scheme," Ph.D. Dissertation, Stanford University, Departement of Computer Science, 2009.

[10] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *33rd Int. Conf. Mach. Learn. ICML 2016*, vol. 1, 2016, pp. 342–351.

[11] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep Neural Networks over Encrypted Data," *arXiv:1711.05189v1*, 2017.

[12] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015, p. 4325.

[13] T. Graepel, K. Lauter, and M. Naehrig, "ML confidential: Machine learning on encrypted data," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7839 LNCS, 2013, pp. 1–21.

[14] L. J. M. Aslett, P. M. Esperan ç a, and C. C. Holmes, "Encrypted statistical machine learning: new privacy preserving methods," *arXiv:1508.06845v1*, 2015.

[15] K. Nandakumar, "Towards Deep Neural Network Training on Encrypted Data," *IEEE Conf. Comput. Vis. Pattern Recognit. Work.*, 2019.

[16] Microsoft, "Microsoft SEAL (release 3.6)," https://github.com/Microsoft/SEAL, Nov. 2020, microsoft Research, Redmond, WA.

[17] M. Barni, C. Orlandi, and A. Piva, "A privacy-preserving protocol for neural-network-based computation," in *Proceedings of the 8th Workshop on Multimedia and Security*. New York, NY, USA: Association for Computing Machinery, 2006, p. 146–151. [Online]. Available: https://doi-org.ezproxy.library.uvic.ca/10.1145/1161366.1161393

[18] C. Orlandi, A. Piva, and M. Barni, "Oblivious neural network computing via homomorphic encryption," *EURASIP Journal on Information Security*, vol. 2007, no. 1, p. 037343, Jul 2007. [Online]. Available: https://doi.org/10.1155/2007/37343

[19] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 619–631. [Online]. Available: https://doi.org/10.1145/3133956.3134056

[20] T. Graepel, K. Lauter, and M. Naehrig, "Ml confidential: machine learning on encrypted data," in *Information Security and Cryptology - ICISC 2012 (15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers)*, ser. Lecture Notes in Computer Science, T. Kwon, M.-K. Lee, and D. Kwon, Eds. Germany: Springer, 2013, pp. 1–21.

[21] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin, and S. Bakas, "Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation," in *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*. Cham: Springer International Publishing, 2019, pp. 92–104.

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

[23] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[24] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 26–28 Aug 2020, pp. 2938–2948.

[25] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 634–643.

[26] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *ArXiv*, vol. abs/1708.06733, 2017.

[27] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *CoRR*, vol. abs/1808.04866, 2018.

[28] W. Li, F. Milletarì, D. Xu, N. Rieke, J. Hancox *et al.*, "Privacy-preserving federated brain tumour segmentation," in *Machine Learning in Medical Imaging*, H.-I. Suk, M. Liu, P. Yan, and C. Lian, Eds. Cham: Springer International Publishing, 2019, pp. 133–141.

[29] J. Zhang, B. Chen, S. Yu, and H. Deng, "Pefl: A privacy-enhanced federated learning scheme for big data analytics," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE Press, 2019, p. 1–6.

[30] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1175–1191.

[31] C. A. Choquette-Choo, F. Tramer, N. Carlini, and N. Papernot, "Label-only membership inference attacks," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 1964–1974. [Online]. Available: https://proceedings.mlr.press/v139/choquette-choo21a.html

[32] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 3–18.

[33] H. Hu, Z. Salcic, L. Sun, G. Dobbie, and X. Zhang, "Source inference attacks in federated learning," in *2021 IEEE International Conference on Data Mining (ICDM)*, 2021, pp. 1102–1107.

[34] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song, "The secret revealer: Generative model-inversion attacks against deep neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[35] Y. Chen, X. Yang, X. Qin, H. Yu, B. Chen, and Z. Shen, "FOCUS: dealing with label quality disparity in federated learning," *CoRR*, vol. abs/2001.11359, 2020.

[36] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[37] D. Ye, R. Yu, M. Pan, and Z. Han, "Federated learning in vehicular edge computing: A selective model aggregation approach," *IEEE Access*, vol. 8, pp. 23 920–23 935, 2020.

[38] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5132–5143.

[39] L. Gao, H. Fu, L. Li, Y. Chen, M. Xu, and C.-Z. Xu, "Feddc: Federated learning with non-iid data via local drift decoupling and correction," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 10 102–10 111.

[40] M. Duan, D. Liu, X. Chen, Y. Tan, J. Ren, L. Qiao, and L. Liang, "Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications," *2019 IEEE 37th International Conference on Computer Design (ICCD)*, Nov 2019.

[41] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," in *International Conference on Learning Representations*, 2020.

[42] M. Huang, H. Li, B. Bai, C. Wang, K. Bai, and F. Wang, "A federated multi-view deep learning framework for privacy-preserving recommendations," *International Workshop on Federated and Transfer Learning for Data Sparsity and Confidentiality in Conjunction with IJCAI*, 2021.

[43] D. Bui, K. Malik, J. Goetz, H. Liu, S. Moon, A. Kumar, and K. G. Shin, "Federated user representation learning," *arXiv preprint arXiv:1909.12535*, 2019.

[44] L. Huang, A. L. Shea, H. Qian, A. Masurkar, H. Deng, and D. Liu, "Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records," *Journal of Biomedical Informatics*, vol. 99, p. 103291, 2019.

[45] D. Liu, T. Miller, R. Sayeed, and K. D. Mandl, "Fadl: Federated-autonomous deep learning for distributed electronic health record," *arXiv preprint arXiv:1811.11400*, 2018.

[46] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "Braintorrent: A peer-to-peer environment for decentralized federated learning," *arXiv preprint arXiv:1905.06731*, 2019.

[47] F. Zheng, K. Li, J. Tian, X. Xiang *et al.*, "A vertical federated learning method for interpretable scorecard and its application in credit scoring," *arXiv preprint arXiv:2009.06218*, 2020.

[48] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intelligent Systems*, vol. 35, no. 4, p. 70–82, Jul 2020.

[49] J. Yoon, W. Jeong, G. Lee, E. Yang, and S. J. Hwang, "Federated continual learning with weighted inter-client transfer," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 12 073–12 086.

[50] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.

[51] M.-f. F. Balcan, Y. Liang, V. Kanchanapally, and D. Woodruff, "Improved distributed principal component analysis," *Advances in neural information processing systems*, vol. 27, 2014.

[52] H. Polat and W. Du, "Svd-based collaborative filtering with privacy," in *Proceedings of the 2005 ACM Symposium on Applied Computing*, ser. SAC '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 791–795.

[53] S. Han, W. K. Ng, and P. S. Yu, "Privacy-preserving singular value decomposition," in *2009 IEEE 25th International Conference on Data Engineering*, 2009, pp. 1267–1270.

[54] A. Hartebrodt, R. Röttger, and D. B. Blumenthal, "Federated singular value decomposition for high dimensional data," *arXiv preprint arXiv:2205.12109*, 2022.

[55] R. Nasirigerdeh, R. Torkzadehmahani, J. Baumbach, and D. B. Blumenthal, "On the privacy of federated pipelines," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1975–1979.

[56] I. Hegedűs, M. Jelasity, L. Kocsis, and A. A. Benczúr, "Fully distributed robust singular value decomposition," in *14-th IEEE International Conference on Peer-to-Peer Computing*, 2014, pp. 1–9.

[57] D. Chai, L. Wang, L. Fu, J. Zhang, K. Chen, and Q. Yang, "Federated singular vector decomposition," *arXiv preprint arXiv:2105.08925*, 2021.

[58] D. Chai, L. Wang, J. Zhang, L. Yang, S. Cai, K. Chen, and Q. Yang, "Practical lossless federated singular vector decomposition over billion-scale data," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 46–55.

[59] G. Li, D. Yang, A. B. Nobel, and H. Shen, "Supervised singular value decomposition and its asymptotic properties," *Journal of Multivariate Analysis*, vol. 146, pp. 7–17, 2016, special Issue on Statistical Models and Methods for High or Infinite Dimensional Spaces.

[60] E. Bair, T. Hastie, D. Paul, and R. Tibshirani, "Prediction by supervised principal components," *Journal of the American Statistical Association*, vol. 101, no. 473, pp. 119–137, 2006.

[61] R. Rakotomalala and F. Mhamdi, "Combining feature selection and feature reduction for protein classification," in *Proceedings of the 6th WSEAS International Conference on Simulation, Modelling and Optimization, Lisbon, Portugal*, 2006, pp. 444–451.

[62] E. Barshan, A. Ghodsi, Z. Azimifar, and M. Z. Jahromi, "Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds," *Pattern Recognition*, vol. 44, no. 7, pp. 1357–1371, 2011.

[63] M. S. Rahman, I. Khalil, A. Alabdulatif, and X. Yi, "Privacy preserving service selection using fully homomorphic encryption scheme on untrusted cloud service platform," *Knowledge-Based Syst.*, vol. 180, pp. 104–115, sep 2019.

[64] A. Penson, N. Camacho, Y. Zheng, A. M. Varghese, H. Al-Ahmadie, P. Razavi, S. Chandarlapaty, C. E. Vallejo, E. Vakiani, T. Gilewski *et al.*, "Development of genome-derived tumor type prediction to inform clinical cancer care," *JAMA oncology*, vol. 6, no. 1, pp. 84–91, 2020.

[65] D. Chakravarty, J. Gao, S. Phillips, R. Kundra, H. Zhang, J. Wang, J. E. Rudolph, R. Yaeger, T. Soumerai, M. H. Nissan, M. T. Chang, S. Chandarlapaty, T. A. Traina, P. K. Paik, A. L. Ho, F. M. Hantash, A. Grupe, S. S. Baxi, M. K. Callahan, A. Snyder, P. Chi, D. C. Danila, M. Gounder, J. J. Harding, M. D. Hellmann, G. Iyer, Y. Y. Janjigian, T. Kaley, D. A. Levine, M. Lowery, A. Omuro, M. A. Postow, D. Rathkopf, A. N. Shoushtari, N. Shukla, M. H. Voss, E. Paraiso, A. Zehir, M. F. Berger, B. S. Taylor, L. B. Saltz, G. J. Riely, M. Ladanyi, D. M. Hyman, J. Baselga, P. Sabbatini, D. B. Solit, and N. Schultz, "Oncokb: A precision oncology knowledge base," *JCO Precision Oncology*, no. 1, pp. 1–16, 2017.

[66] A. Carey, "On the explanation and implementation of three open-source fully homomorphic encryption libraries," Undergraduate Thesis, University of Arkansas, Fayetteville, 2020.

[67] W. Briguglio, P. Moghaddam, W. A. Yousef, I. Traore, and M. Mamun, "Machine Learning via Encryption (MLE) Framework in Precision Medicine to Preserve Privacy," https://github.com/isotlaboratory/Healthcare-Security-Analysis-MLE, 2021.

[68] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014.

[69] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[70] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 214–223.

[71] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[72] Y. Liu, L. Zhang, N. Ge, and G. Li, "A systematic literature review on federated learning: From a model quality perspective," *arXiv preprint arXiv:2012.01973*, 2020.

[73] V. C. Raykar, R. Duraiswami, and L. H. Zhao, "Fast computation of kernel estimators," *Journal of Computational and Graphical Statistics*, vol. 19, no. 1, pp. 205–220, 2010.

[74] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. IEEE Press, 2019, p. 2512–2520. [Online]. Available: https://doi.org/10.1109/INFOCOM.2019.8737416

[75] Z. Lin, S. Wang, V. Sekar, and G. Fanti, "Summary statistic privacy in data sharing," 2023.

[76] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, vol. 10, no. 3152676, pp. 10–5555, 2017.

[77] T. Wagner, Y. Naamad, and N. Mishra, "Fast private kernel density estimation via locality sensitive quantization," in *International Conference on Machine Learning*. PMLR, 2023, pp. 35 339–35 367.

[78] B. Ding, M. Winslett, J. Han, and Z. Li, "Differentially private data cubes: optimizing noise sources and consistency," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 217–228. [Online]. Available: https://doi.org/10.1145/1989323.1989347

[79] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao, "Privbayes: Private data release via bayesian networks," *ACM Trans. Database Syst.*, vol. 42, no. 4, oct 2017. [Online]. Available: https://doi.org/10.1145/3134428

[80] X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton, "A methodology for formalizing model-inversion attacks," in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, 2016, pp. 355–370.

[81] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate newton-type method," in *International conference on machine learning*. PMLR, 2014, pp. 1000–1008.

[82] T. J. Pollard, A. E. W. Johnson, J. D. Raffa, L. A. Celi, R. G. Mark, and O. Badawi, "The eicu collaborative research database, a freely available multi-center database for critical care research," *Scientific Data*, vol. 5, no. 1, p. 180178, Sep 2018.

[83] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[84] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[85] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[86] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ser. ICDM '08. USA: IEEE Computer Society, 2008, p. 413–422.

[87] F. A. P. Petitcolas, *Kerckhoffs' Principle*. Boston, MA: Springer US, 2011, pp. 675–675.

[88] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM, 1997, pp. 25–30.

[89] I. T. Jolliffe, *Principal Component Analysis*. New York, NY: Springer New York, 2002, pp. 1–9.

[90] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. Mcmahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1175–1191.

[91] V. Y. Pan and Z. Q. Chen, "The complexity of the matrix eigenproblem," in *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, ser. STOC '99. New York, NY, USA: Association for Computing Machinery, 1999, p. 507–516.

[92] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[93] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[94] F. J. Massey, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American Statistical Association*, vol. 46, no. 253, pp. 68–78, 1951. [Online]. Available: http://www.jstor.org/stable/2280095

[95] L. Zhang, "On security properties of random matrix masking," Ph.D. dissertation, University of Florida, 2014.

[96] J. Khan, J. S. Wei, M. Ringnér, L. H. Saal, M. Ladanyi *et al.*, "Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks," *Nat Med*, vol. 7, no. 6, pp. 673–679, Jun. 2001.

[97] A. Rosenwald, G. Wright, W. C. Chan, J. M. Connors, E. Campo *et al.*, "The use of molecular profiling to predict survival after chemotherapy for diffuse large-b-cell lymphoma," *N Engl J Med*, vol. 346, no. 25, pp. 1937–1947, Jun. 2002.

[98] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Paper 2012/144, 2012, https://eprint.iacr.org/2012/144. [Online]. Available: https://eprint.iacr.org/2012/144

[99] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology – ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds. Cham: Springer International Publishing, 2017, pp. 409–437.

[100] Y. Jiang, J. Hamer, C. Wang, X. Jiang, M. Kim, Y. Song, Y. Xia, N. Mohammed, M. N. Sadat, and S. Wang, "Securelr: Secure logistic regression model via a hybrid cryptographic protocol," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 16, no. 1, pp. 113–123, 2019.

[101] W. Briguglio, P. Moghaddam, W. A. Yousef, I. Traoré, and M. Mamun, "Machine learning in precision medicine to preserve privacy via encryption," *Pattern Recognition Letters*, vol. 151, pp. 148–154, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865521002403

[102] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal, "Tenseal: A library for encrypted tensor operations using homomorphic encryption," 2021.

[103] "Microsoft SEAL," https://github.com/Microsoft/SEAL, Jan. 2023, microsoft Research, Redmond, WA.

[104] M. Kristan, A. Leonardis, and D. Skočaj, "Multivariate online kernel density estimation with gaussian kernels," *Pattern Recognition*, vol. 44, no. 10, pp. 2630–2642, 2011, semi-Supervised Learning for Visual Content Analysis and Understanding.

[105] W. A. Yousef, I. Traoré, and W. Briguglio, "Un-avoids: Unsupervised and nonparametric approach for visualizing outliers and invariant detection scoring," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 5195–5210, 2021.

[106] H. Straková, W. N. Gansterer, and T. Zemen, "Distributed qr factorization based on randomized algorithms," in *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 235–244.

[107] G. Jagannathan and R. N. Wright, "Privacy-preserving distributed k-means clustering over arbitrarily partitioned data," in *Proceedings of the Eleventh ACM SIGKDD International*

Conference on Knowledge Discovery in Data Mining, ser. KDD '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 593–599.

[108] C. Molnar, *Interpretable Machine Learning*, 2019, https://christophm.github.io/interpretable-ml-book/.

[109] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should i trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1135–1144. [Online]. Available: https://doi.org/10.1145/2939672.2939778

[110] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller, *Layer-Wise Relevance Propagation: An Overview*. Cham: Springer International Publishing, 2019, pp. 193–209. [Online]. Available: https://doi.org/10.1007/978-3-030-28954-6_10

Appendix: Attack on Model Privacy During Private Inference

In Orlandi et al. [18], private prediction was allowed with NNs by using HE to compute the product between each layer's input and weight matrix. The result is then returned to the client, who unencrypts it to use as input to nonlinear activation functions before sending the re-encrypted activation values back to the server to be used as input for the next layer. For their sigmoid activation function, they claim to provide privacy for the model weights by randomly permuting the returned products so that malicious clients cannot reconstruct the NN by submitting vectors with one non-zero element. That is because the inferred weights would be in different orders for successive input vectors, with the correct order being one of $m!$ possible permutations. They also add to this privacy by randomly flipping the sign of some of the pre-activations since this can be reversed after the sigmoid is applied by simply flipping the sign of the resulting activation since the sigmoid is symmetric about 0.

We will show that for many weight matrices, when using Orlandi et al.'s sigmoid activation, the client can obtain the weight matrix and bias vector with their columns and elements, respectively, permuted in the same way and with a reasonable number of queries. This attack does not apply to their threshold activation function because they mask the values before returning them permuted to the client. We also only show the attack for a binary classification model that uses all sigmoid activation functions. This is already a severe flaw that must be remedied, but it also illustrates that the permutation and flipped signs alone do not ensure privacy. First, we will show that this attack is possible for the individual layers by taking advantage of the fact that upon submitting a vector $x \in \mathbb{R}^{1\times n}$ to the server as input to some layer, the server must return a permutation of the vector $xW + b$ with random signs, where $W \in \mathbb{R}^{n\times m}$ and $b \in \mathbb{R}^{1\times m}$ are the layer's weight matrix and bias vector, respectively. We break this down into two parts: the first shows that the attack is possible when the returned vector is just permuted, and the second builds on this and shows it is possible when the returned vector is permuted and the element's signs are randomly flipped. Then, we will extend the attack to all layers in the network. Finally, we give the

time complexity and the upper bound on the number of queries required for a successful attack.

In Orlandi et al., pre-activations are multiplied by a learnt scalar α before being sent to the client. Since this scalar is always the same during inference, it is mathematically equivalent to returning $\alpha(xW + b) = \alpha xW + \alpha b$ to the client. Meaning that in the following attack, which assumes the pre-activations are not scaled as such, the attacker will obtain αW and αb. So, even though the attacker learns different weights, their model will produce the same pre-activations for the same input and, therefore, have the same performance. These weights will still hold just as much information about the training data making model inversion attacks just as practical and similarly causing the server to lose sole ownership over their model's utility.

Defeating Permutation Defense

The attack starts by constructing a set of candidates for a given row of the weight matrix. Crucially, this set may be much smaller than the $m!$ set of possible permutations assumed by Oralndi et al. After, it is shown that this set can be reduced to the single candidate, which must correspond to a permutation of the original held at the server, and this permutation will be the same for each row, thus resulting in the attacker obtaining a column-wise permutation of the original matrix held by the server. Lastly, we discuss how to reorder results across layers so the permutation of the weight matrices' columns is irrelevant.

Greek letters	A scalar value
lower case letter	A vector
upper case letter	A matrix
x_i	The i^{th} element in the vector x
$W_{i:}$	The i^{th} row of matrix W
$W_{i,j}$	The j^{th} element in the i^{th} row of matrix W
$\mathbf{1}_i$	Vector whose i^{th} element is 1
$(\cdot)^{\pi_i}$	Vector or matrix whose elements or columns, respectively, are permuted according to some permutation π_i

Table 14: Notations used throughout the appendix.

Privacy Attack Against Permutation Defense: Given the randomly permuted results of the expression $xW + b$ for a set of vectors $\{x | x \in \mathbb{R}^{1\times n}\}$ chosen by an attacker with no prior knowledge of $W \in \mathbb{R}^{n\times m}$ or $b \in \mathbb{R}^{1\times m}$, the attacker finds W and b with their columns and elements, respectively, permuted in the same way.

Step 1: The attacker first submits a vector of all zeros to obtain a randomly permuted bias vector b^{π_1}. The attacker then submits $\mathbf{1}_1$ and -$\mathbf{1}_1$ to obtain $p^{\pi_2} = (W_{1:} + b)^{\pi_2}$ and $q^{\pi_3} = (\text{-}W_{1:} + b)^{\pi_3}$, respectively. Noting that $W_{1:}+b+\text{-}W_{1:}+b = 2b$, the attacker finds all permutations π_j and π_k of p^{π_2} and q^{π_3}, respectively, such that $(p^{\pi_2})^{\pi_j}+(q^{\pi_3})^{\pi_k} = 2b^{\pi_1}$. Note that for each π_j, all permutations π_k that satisfy this property result in the same $(q^{\pi_3})^{\pi_k}$ so all but one such permutation is discarded. Therefore, we can index these pairs of permutations by j, the index of the first element π_j. Further, for any two permutations π_j and $\pi_{j'}$ where $j \neq j'$, if $(p^{\pi_2})^{\pi_j} = (p^{\pi_2})^{\pi_{j'}}$, then all pairs containing $\pi_{j'}$ can be discarded. Therefore, each pair of permutations π_j, π_k results in a vector $(p^{\pi_2})^{\pi_j}$, that is unique across all j. The worst case for the number of permutations $|j|$ is $m!$ but as we will discuss in the next subsection, the likelihood of valid permutation pairs is much lower than the likelihood of invalid pairs so the true value of $|j|$ may be much closer to 1 rather than $m!$.

Step 2: For at least one of these pairs of permutations, $((p^{\pi_2})^{\pi_j}, (q^{\pi_3})^{\pi_k}) = (p^{\pi_1}, q^{\pi_1})$. Given this pair of permutations, $(p^{\pi_2})^{\pi_j} - b^{\pi_1} = p^{\pi_1} - b^{\pi_1} = (W_{1:} + b)^{\pi_1} - b^{\pi_1} = (W_{1:})^{\pi_1}$, meaning the attacker will have the first row of W and the bias vector b after applying the *same permutation* π_1 to each. However, the attacker does not know which pairs of permutations will satisfy this property. So, for all pairs of permutations found in the previous step, the attacker computes $(p^{\pi_2})^{\pi_j} - b^{\pi_1} = \widehat{(W_{1:})}_j^{\pi_1}$, where $\widehat{(W_{1:})}_j^{\pi_1}$ is a candidate for $(W_{1:})^{\pi_1}$.

Step 3: Once the attacker has all candidates $\widehat{(W_{1:})}_j^{\pi_1}$, they use the fact that $\alpha \cdot \widehat{(W_{1:})}_j^{\pi_1} + b^{\pi_1} = y^j \in \mathbb{R}^{1\times m}$ defines a set of linear equations of the form $y_i^j = \alpha \cdot \widehat{(W_{1,i})}_j^{\pi_1} + b_i^{\pi_1}$. Because, by construction, two candidates $\widehat{(W_{1:})}_j^{\pi_1}, \widehat{(W_{1:})}_{j'}^{\pi_1}, j \neq j'$ must differ in at least one position, then the linear equations in $y^j = \alpha \cdot \widehat{(W_{1:})}_j^{\pi_1} + b^{\pi_1}$ and $y^{j'} = \alpha \cdot \widehat{(W_{1:})}_{j'}^{\pi_1} + b^{\pi_1}$ must also differ in at least one position. When the values in b are not unique, two sets of linear equations y^j and $y^{j'}$ will differ but be permutations of each other. In this case, the attacker drops all but one such y^j so none that remain are permutations of each other. This means the y^j corresponding to the $\widehat{(W_{1:})}_j^{\pi_1}$ that is equal to the true $(W_{1:})^{\pi_1}$ may be discarded. However, one $\widehat{(W_{1:})}_j^{\pi_1}$ will remain that equals the true $(W_{1:})^{\pi_1}$, except that the elements in the same position as the non-unique elements in b^{π_1} will be permuted. We deal with this in the next step.

Because no two y^j are permutations of each other, by Lemma 7.2.1, there are at most m values of α where y^j is a permutation of $y^{j'}$. Further, if we have $|j|$ candidates $\widehat{(W_{1:})}_j^{\pi_1}$, then there are at most $|j|m$ values of α where any y^j and $y^{j'}$ are a

permutation of each other. Because there are infinitely many values of α, the attacker can randomly sample α_* and with high likelihood obtain a y_*^j for each $\widehat{(W_{1:})}_j^{\pi_1}$ where no two y_*^j are permutations of each other. After finding this α_* the attacker then submits $\alpha_* \mathbf{1}_1$ to the server, obtaining $y_*^{\pi 4} = (\alpha_*(W_{1:}) + b)^{\pi 4}$. Then, because no two $y_*^j = (\alpha_* \widehat{(W_{1:})}_j^{\pi_1} + b)^{\pi 1}$ are permutations of each other, but only one corresponds to a $\widehat{(W_{1:})}_j^{\pi_1}$ that is equal to $W_{1:}^{\pi_1}$ at positions corresponding to unique elements in b, then exactly one $y_*^j = (\alpha_*(W_{1:}) + b)^{\pi 1}$ is a permutation of $y_*^{\pi_4}$. If the elements in b are unique, then the $\widehat{(W_{1:})}_j^{\pi_1}$ corresponding to this y_*^j must therefore be the true $(W_{1:})^{\pi_1}$. If they are not unique, the attacker may have dropped the y^j corresponding to the true $(W_{1:})^{\pi_1}$, but the one y_*^j that is a permutation of $y_*^{\pi 4}$ will have a corresponding $\widehat{(W_{1:})}_j^{\pi_1}$ where the elements in the same position as the non-unique elements in b^{π_1} will be permuted and the other elements will be in the correct order. For the elements in $\widehat{(W_{1:})}_j^{\pi_1}$ that are in the same position as the non-unique elements in b^{π_1}, we ensure their correct permutation in the next step.

Step 4: The attacker repeats the above for each row in W and obtains one $\widehat{(W_{i:})}_j^{\pi_1}$ for i in $1, 2, \ldots, n$

a: If the elements in b^{π_1} are unique, then the attacker can concatenate $\widehat{(W_{i:})}_j^{\pi_1}$ for i in $1, 2, \ldots, n$ to get W^{π_1}.

b: If elements in b^{π_1} are not unique, then we have a set of indices $\overline{U}$ where $b_i^{\pi_1}$ for all i in $\overline{U}$ is not unique in b^{π_1} and $|\overline{U}| > 1$. The attacker has one candidate for each row that may have elements in the incorrect order among the indices in $\overline{U}$, but they can concatenate the candidate rows into a candidate matrix, and the columns at indices not in $\overline{U}$ will match the corresponding columns in W^{π_1}. The attacker then submits $\alpha \mathbf{1}_1 + \mathbf{1}_2$ to obtain $(\alpha W_1 + W_2 + b)^{\pi_5}$ from the server for some random α. Using the columns of the candidate matrix that correspond to unique elements in b, they can determine the values of $(\alpha W_{1:} + W_{2:} + b)^{\pi_5}$ corresponding to these columns and remove them from $(\alpha W_{1:} + W_{2:} + b)^{\pi_5}$. The attacker now has a set of true pre-activations $(\alpha W_{1:\overline{U}} + W_{2:\overline{U}} + b)^{\pi_5}$ from the server and a permutation of $W_{1:\overline{U}}$ and $W_{2:\overline{U}}$, where $W_{i:\overline{U}}$ is the elements at the indices in $\overline{U}$ from row i of W. The permutation of $W_{1:\overline{U}}$ corresponds to π_1 (thanks to the previous step), but may not correspond to the permutation of $W_{2:\overline{U}}$. Using the set from $W_{2:\overline{U}}$, the attacker finds all permutations π_v of weights for row 2 that when used in $(\alpha W_{1:\overline{U}})^{\pi_2} + (W_{2:\overline{U}})^{\pi_v} + b_{\overline{U}}^{\pi_1}$, give a permutation of $(\alpha W_{1:\overline{U}} + W_{2:\overline{U}} + b)^{\pi_5}$. Again, the likelihood of a valid permutation is much lower than an invalid one, so after finding all valid permutations, the attacker will likely

be left with only one, and if there are more than one, the attacker can repeat this last step with different values of α to eliminate remaining invalid permutations. Once the correct permutation π_v is found, the attacker will have a permutation π_* of $W_{2:}$ (where π_* is the permutation resulting from using the permutation of $W_{1:}$ obtained in step 3 for the elements that are not indexed by $\overline{U}$ and using π_v for the ones that are), which corresponds to b^{π_1} and the permutation of $W_{1:}$ obtained in step 3, even among indices in $\overline{U}$. The attacker repeats this step with $\sum_{i=1}^{j-1} \alpha(W_{i:\overline{U}})^{\pi_*} + W_{j:\overline{U}} + b_{\overline{U}}^{\pi_1}$ for all $j \in 1, 2, \ldots n$ to obtain a column-wise permutation W^{π_*}. At each step, $\sum_{i=1}^{j-1} \alpha(W_{i:\overline{U}})^{\pi_*}$ and $b_{\overline{U}}^{\pi_1}$ are held constant, and a π_v is found to make $W_{j:\overline{U}}$ correspond to the rows $1, \ldots, j-1$. π_* is only different from π_1 in that the columns corresponding to a repeated value in b^{π_1} may be permuted among other columns corresponding to the same repeated value. So, $b^{\pi_1} = b^{\pi_*}$.

Lemma 7.2.1. *Given the two sets of linear equations defined by $y = \alpha \cdot x + b$ and $y' = \alpha \cdot x' + b$ where $y, y', x, x', b \in \mathbb{R}^{m \times 1}$ and α is a scaler variable, and at least one element in x is not in x', then there are at most m values of α where y is a permutation of y'.*

Proof:

Assume there are more than m values of α in which y is a permutation of y'. Because at least one element of x is not in x', at least one element in the system of equations $y = \alpha \cdot x + b$ is not in $y' = \alpha \cdot x' + b$, let this be $y_i = \alpha \cdot x_i + b_i$. At each value of α where the system of equations y evaluated at α is a permutation of the system of equations y' evaluated at α, one line defined in y' must intercept y_i at α; otherwise, y_i evaluated at α would not be in y' evaluated at α so y evaluated at α could not be a permutation of y' evaluated at α. However, after each line in y' has intercepted y_i at α once, for a total of m different values of α, no line may do so again for any other value of α or else said line would intercept y_i more than once, which is only possible if it equals y_i. This means y_i is in $y' = \alpha \cdot x' + b$, which contradicts our assumption. ■

Privacy Attack Against Permutation and Random Sign Defense: Building on the above result to account for randomly flipped signs is somewhat simple. Note, for each vector x the attacker submits, they only need any permutation of the true result, because π_1 is arbitrary, and π_2 and π_3 will be permuted to correspond with π_1 anyway.

Step 1: To account for the randomly flipped bits, when the attacker submits $\pm\mathbf{1}_i$, they count the number of occurrences of each distinct absolute value in the returned vector. In this way, the attacker obtains a list of frequencies of each distinct absolute value in

p and q for $W_{1:}$, after submitting $\mathbf{1}_1$ and $-\mathbf{1}_1$. They also obtain the same list for b, and create a version of b^{π_1} with the absolute value of its elements. As already stated, it doesn't matter what π_1 actually is as long as it is held constant for the remainder of the steps in the attack. The attacker must find the true signs of the elements in $|b^{\pi_1}|$.

The attacker again finds all permutations π_j and π_k of p and q, respectively, such that $(p^{\pi_2})^{\pi_j} + (q^{\pi_3})^{\pi_k} = |2b^{\pi_1}|$. Now the worst case for the number of valid permutations pairs $|j|$ is ${}_{2m}P_m$ because the permutations are actually created from twice as many values, given that each of the m values in p have two choices of sign. However, the likelihood of valid permutation pairs is still much lower than the likelihood of invalid pairs, so the true value of $|j|$ may be much closer to 1 than $m!$. For each element in $(p^{\pi_2})^{\pi_j}$ and $(q^{\pi_3})^{\pi_k}$ in each of the valid permuted pairs, if the sign of the corresponding element in $2b^{\pi_1}$ is positive, its sign will be correct. Otherwise, its sign will be flipped because the right side of the equation $(p^{\pi_2})^{\pi_j} + (q^{\pi_3})^{\pi_k} = |2b^{\pi_1}|$ has a flipped sign. We deal with this in Step 6.

Step 2: Computing $(p^{\pi_2})^{\pi_j} - b^{\pi_1}$ for each valid pair previously obtained will return candidate rows $\widehat{(W_{1:})}_j^{\pi_1}$ just as before, except that now, for each element in $\widehat{(W_{1:})}_j^{\pi_1}$, if the sign of the corresponding element in $2b^{\pi_1}$ is negative, its sign will be flipped from what it would be if the attacker knew the true sign of $2b^{\pi_1}$.

Step 3: The attacker again finds α_* but this time such that no two y_*^j derived from all candidates $\widehat{(W_{1:})}_j^{\pi_1}$ are permutations of each other when considering the absolute value of their elements evaluated at α_* instead, meaning the sign of the values in $y_*^{\pi 4}$ and y_*^j evaluated at α_* can be ignored. Again, only one choice of $\widehat{(W_{1:})}_j^{\pi_1}$ will be valid, except in the case where b^{π_1} has repeated values, but this is again dealt with by selecting one such choice of $\widehat{(W_{1:})}_j^{\pi_1}$ and then correcting it in Step 4.b. Still, the signs of some elements in $\widehat{(W_{1:})}_j^{\pi_1}$ may be flipped as previously noted.

Step 4: The attacker can repeat the above process for the remaining rows in W. They again obtain one $\widehat{(W_{i:})}_j^{\pi_1}$ for i in $1, 2, \ldots, n$ and all will have flipped signs where $2b^{\pi_1}$ is negative.

a: This step remains unchanged.

b: The attacker performs the same process as before but only considers absolute values in the results $(\alpha W_{i:} + W_{j:} + b)^{\pi_5}$ and $\sum_{i=1}^{j-1} \alpha (W_{i:\overline{U}})^{\pi_*} + W_{j:\overline{U}}^{\pi_v} + b_{\overline{U}}^{\pi_1}$ for all possible choices of sign in $W_{i:\overline{U}}$.

Step 5: By this step, the attacker has a column-wise permutation of W and b with the corresponding elements permuted in the same way, except that for each element in b that is negative, the sign of the same element in the attacker's copy and the sign of the elements in the corresponding columns in W are flipped. Using their copy, the attacker can compute a permutation of the result returned by the server, except with some signs flipped, and they cannot yet tell if the server flipped the sign or if their corresponding weights and biases have flipped signs. However, the attacker can find an input to the first layer that produces an output with all unique elements, regardless of sign. The attacker submits the same input to the server and receives the same output but in permuted order, with some signs flipped. Because the absolute value of the elements are unique, the attacker knows which position in the randomly permuted output from the server corresponds to each position in their calculated output, so they can submit $\pm\mathbf{1}_i$ to the next layer in the permuted order expected by the server. Because the columns of the client's matrix are out of order, so the rows returned by the server will be in the corresponding order rather than the order they appear in at the server. This does not matter, however, because permuted columns in one layer's weight matrix will have no effect given that the corresponding permutation is applied to the rows in the next layer. The attacker obtains p and q by submitting $\mathbf{1}_i$ twice, once with the sign of the non-zero element the same as the corresponding element returned by the server and again with the opposite sign. However, the attacker does not know which result will be p or q because regardless of the sign of the non-zero element, the attacker does not know if the server will flip it, meaning the attacker might submit -$\mathbf{1}_i$ when they mean to submit $\mathbf{1}_i$ and vice versa. So, the attacker assumes that the server will not flip the sign, meaning the response is p when the client submits $\mathbf{1}_i$ with the sign of the non-zero element the same as the corresponding element returned by the server and q when the client submits $\mathbf{1}_i$ with said sign flipped. Of course, this may not be the case, so the client computes $\widehat{(W_{1:})}_j^{\pi_1}$ as before, and after repeating Steps 1-2 using this method, in Step 3, they also test all $\widehat{(W_{1:})}_j^{\pi_1}$ with reversed signs, because each candidate's value might have been accidentally computed from $(q^{\pi_3})^{\pi_k} - b^{\pi_1} = -\widehat{(W_{1:})}_j^{\pi_1}$ instead of $(p^{\pi_2})^{\pi_j} - b^{\pi_1} = \widehat{(W_{1:})}_j^{\pi_1}$, as before. Again, in the end, only a single candidate will remain, this time using the $|2b^{\pi_1}|$ from the second layer. For the signs that are positive in $2b^{\pi_1}$, only one y_*^j will be valid and the signs of the corresponding $\widehat{(W_{1:})}_j^{\pi_1}$ will be correct. For the signs that are negative in $2b^{\pi_1}$, the single valid y_*^j will correspond to a $\widehat{(W_{1:})}_j^{\pi_1}$ with flipped signs. Step 4 would then proceed as before. The attacker can repeat this for all layers.

Step 6: By this step, the attacker has a column-wise permutation of the weight matrix from each layer, with the rows of successive layers correspondingly permuted. However, at each step, the previous layer's absolute values where used to identify corresponding positions in the next layer for submitting $\pm\mathbf{1}_i$ and the inability to ensure the correct sign for $\pm\mathbf{1}_i$ was dealt with only with respect to the $|2b^{\pi_1}|$ obtained for each layer. But, as mentioned, taking the absolute value of the bias vector at each layer means some columns in each layer's weight matrix recreated by the client are flipped, meaning, some outputs of the client's layers are flipped. This means the output produced using any two successive layers of the client's copy may be incorrect. Starting with the first layer, the client can determine the correct sign of the first neuron's output by submitting only the output of said neuron to the second layer of the server's NN, with all other outputs set to 0, and noting the output from the server of the second layer. The client then computes the same result using the sign they assumed in the previous steps. If the client obtains the same output, ignoring the sign, for all neurons in the second layer, they can assume their choice of sign was correct. Otherwise, they know they must flip the sign of that neuron by flipping the sign of all the weights and biases in that neuron. This follows from the fact that the client is computing $|\pm\alpha W_{1:} + b|$ in their second layer, while the server is returning $|\alpha W_{1:} + b|$ for some input α. Of course, with an incorrect choice of sign of α, the result will likely be different for at least one of the elements in $|\alpha W_{1:} + b|$ unless all the lines defined in the vectors $|\alpha W_{1:} + b|$ and $|\text{-}\alpha W_{1:} + b|$ intersect at α, which is exceedingly rare. To be sure, the client can check with a second and third value for α because a set of lines can always only have one common intercept, and taking the absolute value of this set of lines only permits one other such intercept. The client repeats this for each neuron in the first layer, then for each layer, until they can obtain the output of the last layer, but potentially with a flipped sign. The client does not care if the sign of this value is flipped as long as the predicted label for said output matches the label they obtain when using the server's prediction. This can be trivially ensured by observing the predicted label when using the server's model and the sign of the output when using the attacker's model.

Attack Complexity

Of course, the above attack may not constitute a privacy leak if it is not practical because of the amount of time or queries it takes to perform the attack. Further, an attack that is practical against a nontrivial subset of possible weight matrices still invalidates the defense method used to hide said weight matrices. Therefore, in the following, we discuss the worst

and best case time complexities of each step in the above attack against the permutation and random sign defense, and we bound the number of inferences.

Step 1: To find all the permutations π_j and π_k such that $(p^{\pi_2})^{\pi_j} + (q^{\pi_3})^{\pi_k} = |2b^{\pi_1}|$ regardless of sign, in the worst case may take $O({}_{2m}P_m)$ time. However, when considering a single element, for an incorrect choice of $(p^{\pi_2})_i^{\pi_j}$, the result $2b_i^{\pi_1} - (p^{\pi_2})_i^{\pi_j}$ will be randomly distributed in the real numbers, yet the values in q represent only a small subset of the real numbers. Therefore, it is very unlikely any incorrect choice of $(p^{\pi_2})_i^{\pi_j}$ will result in $2b_i^{\pi_1} - (p^{\pi_2})_i^{\pi_j}$ being in q, and permutations containing such choice will not satisfy $(p^{\pi_2})^{\pi_j} + (q^{\pi_3})^{\pi_k} = |2b^{\pi_1}|$. So, the true number of valid permutations of p^{π_2} is actually much lower than ${}_{2m}P_m$ and in the best case may only be 1. This subset of valid permutations can be efficiently found by first creating a dictionary out of q in $O(2m)$. Then, for each choice of $(p^{\pi_2})_i^{\pi_j}$, check if $2b_i^{\pi_1} - (p^{\pi_2})_i^{\pi_j}$ is in q, and therefore a valid choice. For all possible choices of a single element, this takes $O(2m)$. To repeat this for all elements takes $O(2nm)$. Once the valid choices of each element in p^{π_2} are found, create all valid p^{π_2}. In the worst case, this takes $O({}_{2m}P_m)$ time, but because the number of valid choices for each element may be one, in the best case, we have $\Omega(1)$. Repeating this for all p, q corresponding to each row in W has worst and best case time complexities of $O(n\,{}_{2m}P_m)$ and $\Omega(n)$, respectively. This step requires two inferences per row and one inference for the beta vector, or $2n + 1$ inferences in total.

When $|\overline{U}|$ elements in b are not unique, if this step is done naively, there will be minimum $|\overline{U}|!$ candidate permutations for each row in W. However, the attacker can only work with candidate permutations of the elements corresponding to unique elements in b^{π_1}, following steps 1 through 4.a to narrow said permutations down to a single choice, while assigning a random permutation to the remaining elements indexed in $\overline{U}$. This does not add to the time complexity of this step, and the true permutation of the elements indexed in $\overline{U}$ can be resolved in step 4.b.

Step 2: This step requires computing m subtractions per permutation found in the last step. In the worst case, this may be $n\,{}_{2m}P_m$ permutations, but in the best case, we have only n. Therefore, we have a worst and best case time complexity of $O(mn\,{}_{2m}P_m)$ and $\Omega(mn)$, respectively. This step requires no inferences.

Step 3: This step requires computing $y_*^j = \left(\alpha_* \cdot \widehat{(W_{i:})}_j^{\pi_1} + b\right)^{\pi 1}$ for each candidate row, across all rows. The worst and best cases are again $O(mn\,{}_{2m}P_m)$ and $\Omega(mn)$, respectively, but in the scenario where there is only one valid permutation for each row, this

step can be skipped because we only have one choice for each row. Here, we need only a single inference for each row.

Step 4.a: This step is $O(nm)$ and requires no inferences.

Step 4.b: This step depends on the size of the set $\overline{U}$. In the worst case, $\overline{U} = m-1$ because, if it equals m, this means the entire bias vector is the same and the attacker can just subtract this bias value from the returned results each time they submit a vector. Then, they can perform an analogous attack by using $W_{i:}^{\pi_2} - (W_{i:} + W_{1:})^{\pi_3} = W_{1:}^{\pi_1}$. The attacker obtains $(\sum_{i=1}^{j-1} \alpha(W_{i:\overline{U}}) + W_{j:\overline{U}} + b_{\overline{U}})^{\pi_5}$ and creates a dictionary from it in $O(\overline{U})$. Using the sets of weights indexed by $\overline{U}$, the attacker computes $2|\overline{U}|$ sums $(\sum_{i=1}^{j-1} \alpha W_{i:\overline{U}})^{\pi_*} + W_{j:\overline{U}} + b_{\overline{U}}^{\pi_2}$ using all possible choices for the value and sign for $W_{j:\overline{U}}$, then checks if each sum is in $(\sum_{i=1}^{j-1} \alpha(W_{i:\overline{U}}) + W_{j:\overline{U}} + b_{\overline{U}})^{\pi_5}$ therefore implying the sum's corresponding choice of value and sign are valid. Repeating this for all k in $\overline{U}$ requires computing $O(|\overline{U}|2|\overline{U}|)$ sums. Once all the valid choices are found for each index in $\overline{U}$, all valid vectors $W_{j:\overline{U}}$ are found. As in Step 1, the worst case for the number of valid vectors is ${}_{2|\overline{U}|}P_{|\overline{U}|}$ but is more likely to be close to 1. So, this implies a worst case time complexity of $O({}_{2|\overline{U}|}P_{|\overline{U}|})$ but a best case of $\Omega(1)$. If there are more than one valid choices, the attacker tries again with different values of α. To repeat this for all j in $1, 2, \ldots, n$ we have a best and worst case of $O(mn\,{}_{2m}P_m) + O(n\,{}_{2|\overline{U}|}P_{|\overline{U}|})$ and $\Omega(n)$, respectively. This step requires n inferences, regardless of the size of $|U|$.

Step 5: Here, the actions taken are the same as in steps 1-4 except with some different assumptions corrected in Step 6. These actions are repeated for all layers except for the first. Therefore, the time complexity of Step 5 is $L-1$ times the time complexity of steps 1-4, where L is the number of layers.

Step 6: Denote the size of the i^{th} layer as $|l_i|$. Then, the attacker must submit at most 3 and at least 1 inference to determine the sign of each neuron. For each inference, they must calculate the output of the neurons in the next layer, which is a $|l_i| \cdot |l_{i+1}|$ operation. Therefore, the worst and best case time complexities for this step are $O(3\sum_{i=1}^{L-1} |l_i|^2 \cdot |l_{i+1}|)$ and $\Omega(\sum_{i=1}^{L-1} |l_i|^2 \cdot |l_{i+1}|)$, respectively. The worst case number of inferences is $3\sum_{i=1}^{L-1} |l_i|$.

The total time complexity for the entire attack is dominated by the term $O(mn\,{}_{2m}P_m)$, which of course makes the attack too inefficient to qualify as a threat in the *worst case*. However, the best case time complexity is $\Omega(Lmn)$, which constitutes a very efficient attack and a severe threat to the privacy of the server's model and potentially its training data.

The total number of inferences required is also only $\sum_{i=1}^{L-1} 4|l| + 2 + 3\sum_{i=1}^{L-1} |l_i|$, which is reasonable for a single attacker, let alone several working together.

Defeating Random Neuron Defense

Orlandi et al. also presented a way to hide the network topology by randomly adding some fake neurons with random weights. The server uses these fake neurons to compute extra pre-activations, which are also sent to the client, making it difficult to distinguish real pre-activations from fake ones. However, because these pre-activations are random, they will change upon successive inferences with the same input. This means that in the above attack, each time the attacker wants to know the true pre-activations for a given input x, the attacker submits it multiple times instead of just once. The attacker then counts the occurrences of each distinct absolute value in the vectors returned by the server. The minimum number of occurrences of each distinct absolute value across the permuted vectors returned when submitting the same x multiple times is the number of occurrences of true pre-activations that have that absolute value. If the minimum is zero, the pre-activation value is ignored because it corresponds to a random fake pre-activation; otherwise, this pre-activation would occur in every permuted vector. The more copies of x submitted, the more confident the attacker can be in determining the true pre-activations. The only expense is multiplying the number of inferences by ρ, the number of times the attacker submits each vector and adding a $O(\rho m)$ operation to determine the minimum number of occurrences of each distinct absolute value.