Privacy-Preserving Transfer Learning for Human Activity Recognition

David Melanson

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Computer Science and Systems

University of Washington

2021

Reading Committee:

Dr. Martine De Cock, Chair

Dr. Hee-Seok Kim

Program Authorized to Offer Degree:
Computer Science and Systems

University of Washington

Abstract

Privacy-Preserving Transfer Learning for
Human Activity Recognition

David Melanson

Chair of the Supervisory Committee:
Professor Dr. Martine De Cock
School of Engineering and Technology

Neural network models yield high accuracy in different application domains, making them an impactful tool for machine learning system developers. In the context of Human Activity Recognition (HAR), it is common that the data a deep neural network was originally trained on will not translate optimally to new users. To remedy this, one can apply Transfer Learning (TL) to personalize the model to new end-users, leading to significant accuracy improvements. Such TL is by design done based on personal information of the end-users that, if not protected, can be (mis)used by application developers beyond the professed scope.

We propose a cryptography-based solution for privacy-preserving TL to personalize a Convolutional Neural Network (CNN) for HAR with the data of end-users, without requiring the end-users to reveal their sensitive data in an unencrypted manner, and without requiring the owner of the CNN to disclose their trained model parameters or any other sensitive or proprietary information with anyone in plaintext. To this end, we use techniques from Secure Multi-Party Computation (MPC) to encrypt all data and model parameters, and to allow all parties to compute functions over that encrypted data. To demonstrate the effectiveness of our privacy-preserving solution, we compare the accuracy and runtimes against the TL algorithm in-the-clear, i.e. when no measures to protect privacy are in place. We performed tests on two datasets using two different MPC based protocols to ensure the security of

data. Our tests not only demonstrate significant increases in accuracy, but also show that our approach may be fast enough to use in practice.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Dr. Martine De Cock who has patiently taught me much of what I know now, and has guided me through my entire Thesis. I also wish to thank Dr. Hee-Seok Kim, who has diligently provided valuable feedback, helping me to refine my Thesis. Additionally, I extend my thanks to Dr. Anderson C.A. Nascimento who has helped me understand the cryptographic techniques necessary to succeed in my academic ventures. Lastly, I would like to thank Ricardo Maia, who has helped me to significantly improve my code base, allowing us to get efficient runtimes in our secure protocols.

Chapter 1
INTRODUCTION

Machine Learning (ML) is a powerful tool for application developers to exploit information found in data. In principle, the more data one has available, the more accurate ML models can be made. This is especially true for state-of-the-art deep learning models, which are notoriously data hungry, meaning that a lot of data is required to train such models. In many application domains however, the data is scarce. Transfer Learning (TL) aims to overcome this data scarcity issue by combining data from different, related domains, through techniques for training a model in one domain and then transferring ("updating") it in another domain [28].

An important use-case of TL is in Human Activity Recognition (HAR), with at least 170 papers published during the years 2014-2019 regarding the topic [28]. Recognition of human activity – such as standing, walking, or running – from sensors in wearable devices like wristbands and smartphones is used in a growing variety of applications, including for early detection of diseases, strokes or seizures, mental health assessment, fall detection, sports monitoring, etc. [39, 47]. As physical characteristics and activity patterns differ from one user to the next, the problem of data heterogeneity is well known in applications that rely on bio-signal data, and various methods have been proposed to address end-user calibration challenges (see e.g. [28, 35, 36, 51] and references therein).

Algorithms to create well-calibrated, personalized models for HAR take as input information from the source domain (such as a model trained on data from source users) and from the target domain (calibration data from the end-user). Sensors worn on the human body however collect very personal data. Giving an application developer (model owner) free access to the calibration data from the target end-user opens the door to potential data mis-

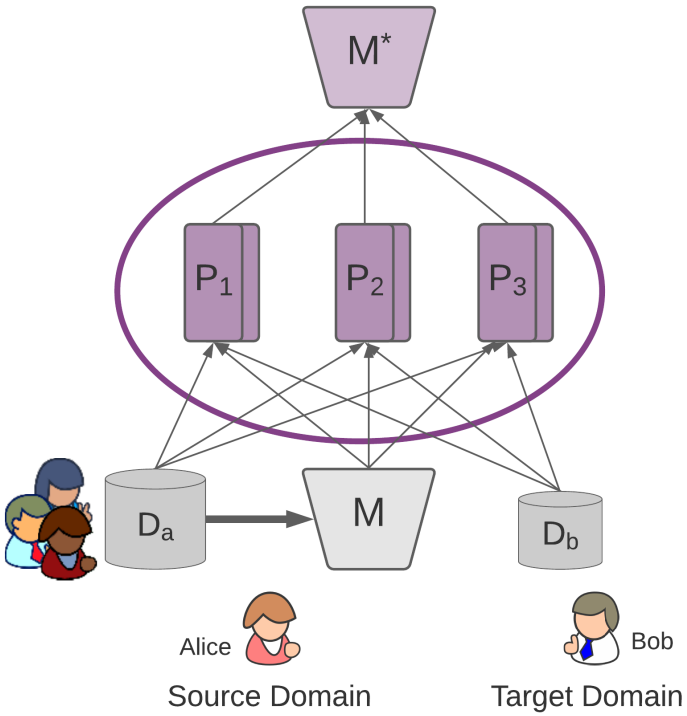

Figure 1.1: Diagram illustrating privacy-preserving TL between Alice (representing the source domain) and Bob (representing the target domain). Alice and Bob send encrypted shares of D_a, M, and D_b to computing parties P_1, P_2, and P_3. These parties subsequently execute MPC protocol π_{pers} to construct a personalized model M^*.

use. Giving the target-user free access to the pre-trained parameter values of a proprietary model is equally problematic, even more so because trained models can leak information about the training data [9], in other words about the bio-signal data of the source users. Despite the acknowledgement of the need for *privacy-preserving* aggregation of information from the source and the target domain (see e.g. [35]), there is a wide gap in the literature on technologies to this end.

In this Thesis we propose the cryptographic protocols that enable the same kind of model personalization and calibration as in-the-clear (i.e. when no encryption is used), without re-

quiring the model owner and the target user to disclose their model parameters or calibration data to anyone in an unencrypted manner. Consider the following use-case to motivate the benefit of our secure[1] personalization method, and to describe how it works at a high level. As illustrated in Figure 1.1, an application developer nick-named *Alice* has trained a model M for HAR on source data set D_a. A new end-user *Bob* is interested to use M to monitor his physical condition via sensor data gathered from his watch [6]; however, M may not be very accurate on Bob's data because of a domain shift between the data D_a that M was trained on, and Bob's data D_b [35]. To remedy this situation, Alice and Bob could use a transfer learning technique to personalize M to Bob's data.This creates a potential issue for both Bob and Alice as it would require either Bob to reveal D_b to Alice, or Alice to reveal D_a and/or M to Bob. From Alice's perspective, the latter is a problem since D_a contains bio-signal data of source users (depicted on the left in Figure 1.1), so it would be unethical, and perhaps illegal, to leak this data. In addition, Alice wants to keep the parameters of the proprietary model M hidden, as it gives her an edge over her competitors. Furthermore, Bob does not want to reveal his data D_b to Alice either for his own privacy concerns.

To resolve this situation, we use Secure Multi-Party Computation (MPC) techniques [12] to enable the construction of a personalized model M^* from D_a, M, and D_b, without requiring Alice to disclose D_a and the trained model parameters of M to anyone in an unencrypted manner, and without requiring Bob to show his calibration data D_b to anyone in plaintext. To this end, Alice and Bob each send encrypted shares of their respective inputs D_a, M, and D_b to untrusted servers $P_1, P_2, \ldots$ (see Figure 1.1). These servers or *computing parties* subsequently perform computations over the encrypted shares – as defined in MPC protocols proposed in this Thesis – to arrive at encrypted shares of the personalized model M^*.

MPC is concerned with the protocol execution coming under attack by an adversary. We assume in this Thesis that the adversaries are *semi-honest*. Parties corrupted by a semi-honest adversary still follow the instructions of the MPC protocols, but the adversary

[1]Throughout this Thesis, we use the terms "secure" and "privacy-preserving" as synonyms.

attempts to learn private information from the internal state of the corrupted parties and the messages that they receive. The MPC protocols proposed in this Thesis and the MPC schemes that they rely on, are designed to prevent such attacks being successful, both in *honest-majority* settings, i.e. when more than half of the parties can assumed to be honest (not corrupted), and in *dishonest-majority* settings where the adversary is able to corrupt half of the parties or more if he wants.

The primary contributions of this Thesis are the design, implementation, and evaluation of privacy-preserving MPC protocols for personalization of a CNN for HAR system through TL. Algorithms to achieve higher accuracy in HAR through TL already exist in-the-clear, i.e. without any encryption or regards for privacy. Our aim is to develop cryptographic protocols that enable the same kind of TL accuracy gains for an end-user through personalization, *without leaking any sensitive information from the end-user (target domain) to the model owner (source domain) or vice versa.*

After reviewing related work in Chapter 2 and recalling premilinaries about MPC in Chapter 3, in Chapter 4 we present our MPC protocols π_{pers} for privacy-preserving transfer learning and π_{infer} for secure activity recognition with the personalized model. Chapter 5 contains a detailed evaluation of the MPC protocols in terms of accuracy and runtime in various security settings.

A journal manuscript based on the results of this work is in preparation:

> **David Melanson**, Ricardo Maia, Hee-Seok Kim, Anderson Nascimento, Martine De Cock. Privacy-Preserving Transfer Learning for Human Activity Recognition. In preparation, 2021

Chapter 2
RELATED WORK

Over recent years, there has been great development in the literature for Privacy-Preserving Machine Learning (PPML). State-of-the-art PPML approaches rely on a variety of privacy-enhancing technologies, most prominently Federated Learning (FL), Differential Privacy (DP), and cryptographic approaches such as Secure Multiparty Computation (MPC) and Homomorphic Encryption (HE). Research on the application of PPML for human activity recognition from wearable sensors, as we do in this Thesis, has been fairly limited so far. Below we discuss the most relevant related work.

Federated Learning for HAR. In Federated Learning (FL), clients (users) collaborate in solving a machine learning problem under the coordination of a central server, without the need for the clients to transfer or exchange their raw data [32]. Instead, each client trains a model locally on their own data and sends the trained model parameters to the central server. The central server commonly averages the parameter values across all clients and pushes the result back to the clients who subsequently use them to update their models and continue training in an iterative fashion. In this process the original raw data never leaves the clients, as only model parameter values are exchanged. Exploration of FL for human activity recognition from wearable sensor data is still in its infancy [21, 48]. The weaknesses of FL compared to the method we propose in this Thesis, is that a model trained in a federated manner is typically less accurate than a model trained in the centralized paradigm, and that the model parameters themselves still leak information about the training data, i.e. FL is not fully privacy preserving. Furthermore, standard FL in itself does not result in personalized models for the end-users. To address the latter, Federated Transfer Learning (FTL) has been proposed as an extension of FL which is concerned with personalizing the global model to

particular users.

Federated Transfer Learning for HAR. One prominent work in the field is FedHealth [10], which is specifically designed for HAR systems leveraging TL for users with wearable sensors in the health sector. FedHealth assumes that the server owner Alice has access to a public dataset, on which she trains an initial Deep Neural Network (DNN), such as a CNN. Alice then distributes this model to the individual end-users, who each push their data through the DNN, calculate the error, and then update the model parameters to minimize said error on their end. At this point, the models that the end-users updated are encrypted via HE, and then sent to the server. The server owner then takes the average of the individual user models, giving us a new global model, which is then decrypted and revealed to both the server owner and the clients. This is how the FL portion of FedHealth operates. To perform TL, the individual users take the global model and push their data through to calculate the loss again, at which point only the last few dense layers of the DNN are updated to minimize the loss. The idea is that the first layers of the DNN learned high level representations of the sensor data that should be true of all users, and that the last few layers learn more intimate features of the data and thus should be personalized to the end-user. FedHealth protects the individual users exact values of data, but with the global model being public, it is still possible to uncover potentially sensitive information regarding the users' data. In our work, we offer end-to-end encryption of all data, including the model parameters at every step.

Privacy-Preserving SVM for HAR. Very different from the above, a Privacy-Preserving Support Vector Machine (PPSVM) for HAR approach [27, 26] has been proposed that is applicable in a scenario like the one in Figure 1.1 in which there is an entity Alice with source data and a baseline model trained on said data, and an entity Bob with target data. The PPSVM approach works by obfuscating client Bob's data by performing matrix multiplication between Bob's data and a random matrix. Ideally, for the sake of TL, the clients would have access to some of the original source data. Since this violates Alice's privacy, the authors propose multiple methods, some of which reveal the original source data, while others do not. While these methods do not require disclosure of the exact values of the

source data, they do require that the parameters of the PPSVM be revealed to the end-user. Despite the obfuscation of exact data values, revealing the parameters can still reveal sensitive information regarding the nature of the source data [5, 29]. Furthermore, it is likely that the entity Alice owning the model would not want to reveal proprietary model parameters. As mentioned before, we offer end-to-end privacy for both the client Bob and the model owner Alice, ensuring that all the data remains secure, including the parameters of Alice's pre-trained model.

Chapter 3
PRELIMINARIES ON MPC

In this section, we discuss the background necessary to understand how we maintain privacy in our pipeline.

3.1 Introduction

In PPML, there are two kinds of prominently used approaches that provide formal guarantees about the privacy protection they offer, namely Differential Privacy (DP), and cryptographic approaches such as Secure Multiparty Computation (MPC) and Homomorphic Encryption (HE). DP works by injecting noise into the data [30, 19]. Although this technique can help protect the privacy of data and model owners [20], the injected noise reduces the accuracy of the results [50]. Furthermore, the amount of noise to be injected depends, among other things, on a privacy budget (a real valued number ϵ), and once this budget is exhausted, any further use of the data will result in privacy violations.

In contrast, Secure Multi-Party Computation (MPC) – an umbrella term for cryptographic approaches that allow two or more parties to jointly compute a specified output from their private information in a distributed fashion, without actually revealing the private information to each other – has theoretical guarantees for correctness of the function the parties are computing [23], and does not require a privacy budget. The benefits introduced by MPC protocols, namely mathematically provable privacy protection without loss of accuracy, are typically met with the cost of more time-consuming computation. Specifically, MPC requires frequent communication between all of the parties involved in the computation, which adds a significant communication overhead to MPC protocols. The privacy guarantees provided by MPC have made it an attractive field of research in PPML. It has prompted

development in secure inference with pre-trained models [13, 15, 24, 31, 34, 38, 44, 45, 46], as well as in the privacy-preserving training of machine learning models, such as linear regression models [2, 17, 42], decision tree models [1, 18, 37], and neural network architectures [3, 16, 25, 41, 49].

In Section 3.2, we recall what MPC is, and present common notation involving MPC. We also discuss how data and model owners should represent their data for consumption by MPC protocols. In Section 3.3, we present details about the security model considered in this thesis.

3.2 MPC Idea and Notation

In MPC, *data owners* encrypt information by distributing shares of the data across multiple *computing parties* in such a way that no one party can reconstruct values of the original data based only on the shares that they are given. With these shares, the parties can jointly compute functions that maintain the secrecy of the data. To this end, the parties execute protocols, which are typically denoted by π (e.g. π_{mul} for multiplication, cfr. Section 3.2.3). In the remainder of this thesis, by "data" we mean the data sets D_a and D_b from respectively Alice and Bob, as well as the trained parameters of Alice's model M, i.e. both Alice and Bob are considered "data owners", and model parameters are broadly referred to as "data" as well.

3.2.1 Fixed Point Representation in a Ring

In our work, during the execution of any protocol, operations are performed on integers (shares) in the ring $\mathbb{Z}_q$. In some MPC schemes, q is a prime, but for our purposes, we let $q = 2^\lambda$, for some $\lambda \in \mathbb{Z}^+$. The model parameters in Alice's classifier M are natively real numbers represented in a floating point format, as are the sensor measurements in Alice and Bob's data sets D_a and D_b. Before the data owners send any sort of shares to the computing parties, they must map any continuous value x into the ring. To do so, they use the function $Q : \mathbb{R} \to \mathbb{Z}_{2^\lambda}$, defined as [16]:

$$Q(x) = \begin{cases} 2^{\lambda} - \lfloor 2^{a} \cdot |x| \rfloor & \text{if } x < 0 \\ \lfloor 2^{a} \cdot x \rfloor & \text{if } x \geq 0 \end{cases} \tag{3.1}$$

In (3.1), a represents the amount of bits we reserve for fractional accuracy. So for example, if $a = 16$, then we only reserve 16 bits to maintain the fractional value of x. When we convert $Q(x)$ to its bit string representation, we have λ bits in total. For finite λ, Q defines a bijection between the reals represented by fixed point notation with λ bits in total and a bits of decimal precision, and the integers in $\mathbb{Z}_q$, making it simple to go from one representation to the other.

3.2.2 Secret Sharing based MPC

In practice, there exist several MPC schemes with different security guarantees, accommodating various amounts of computing parties. MPC protocols that accommodate computation between n parties are referred to as nPC. In the context of our work, we have two data owners, Alice and Bob, but may have a different number of computing parties. To maintain the secrecy of the data owner's information, we use *secret sharing* schemes. To secret share a value x, we first require that the data owners map their data into $\mathbb{Z}_q$ by equation (3.1). They then split the data into secret shares on their end (cfr. Section 3.2.3 and 3.2.4) and send the shares to the computing parties. Depending on the specifics of the secret sharing scheme, a party can own up to $n-1$ shares of z, but will never own all n shares unless we are revealing the true value of z to that party, or if the party was the original owner of z. When denoting a secret shared value z, we write it as $[\![z]\!]_q$, where $[\![z]\!]_q = (z_0, z_1, ..., z_{n-1})$. All computations in our work are assumed to be done in the ring $\mathbb{Z}_q$ unless otherwise specified, and so for the remainder of this thesis, we will write $[\![z]\!]_q$ as $[\![z]\!]$ for convenience.

We test our work in two different secret sharing schemes, namely Additive Secret Sharing and Replicated Secret Sharing. We apply these techniques to a 2PC and a 3PC settings, respectively. We discuss these in greater details in Section 3.2.3 and 3.2.4.

3.2.3 Additive secret Sharing

One of the ways we test our work is with a 2PC paradigm reliant on a secret sharing technique called *Additive Secret Sharing*. The specific MPC protocol based on additive secret sharing that we use to ensure security is the *semi2k-party.x* protocol, with *Extended daBits* [22] (For details, see here [33]). For simplicity, when referring to this approach, we will simply call it the "additive secret sharing protocol". This technique works in a way such that even if $n-1$ out of n shares are in possession of an adversary, still he will not be able to learn the secret shared value. In the context of our 2PC setting, this means that if only one computing party is corrupted, then we can maintain the secrecy of both the data owner's sensitive information, because none of the original data values can be recovered only from the shares owned by one computing party.

To additively secret share a value $z \in \mathbb{Z}_q$ between two computing parties P_1 and P_2, the data owner splits z into two shares $z_0, z_1 \in \mathbb{Z}_q$ chosen uniformly at random bound to the constraint $z = z_0 + z_1 \mod q$. The data owner then distributes these shares to the computing parties (servers), i.e. z_0 to S_0 and z_1 to S_1. Now that we know how the data is distributed as shares between the computing parties, we must know how to perform operations on said shares.

As noted in Section 3.1, the security achieved by MPC based paradigms typically come at the cost of time consuming operations. These operations are lengthy largely because of the communication required to be performed between the parties. Not all protocols, however, actually require communication. In the Additive Secret Sharing scheme we use, there are four common operations that do not require communication. Let $[\![x]\!] = (x_0, x_1)$ and $[\![y]\!] = (y_0, y_1)$, where S_i owns x_i, y_i, and let $c \in \mathbb{Z}^+$, then, we have the following operations that can be performed locally:

- Addition ($z = x + y$): $x + y = (x_0 + x_1) + (y_0 + y_1) = (x_0 + y_0) + (x_1 + y_1)$, thus the parties can simply sum up their respective shares of x, y. This operation is denoted as $[\![z]\!] = [\![x]\!] + [\![y]\!]$.

- Subtraction ($z = x - y$): $x - y = (x_0 + x_1) - (y_0 + y_1) = (x_0 - y_0) + (x_1 - y_1)$, thus the parties can simply take the difference of their shares of x, y, respectively. This operation is denoted as $[\![z]\!] = [\![x]\!] - [\![y]\!]$.
- Multiplication by a constant ($z = c \cdot x$): $c \cdot x = c \cdot x_0 + c \cdot x_1$, thus each party simply multiplies their share of x by c. This operation is denoted as $c[\![x]\!]$.
- Addition of a constant ($z = x + c$): $x + c = x_0 + x_1 + c$, thus without loss of generality, S_0 adds c to their share. This operation is denoted as $[\![z]\!] = [\![x]\!] + c$.

A non-trivial operation to perform is secure multiplication. Suppose that the computing parties own additive shares of x, y, and would like to compute the product $x \cdot y$, i.e., execute the protocol $\pi_{\mathsf{mul}}(x, y)$. To perform this operation, we make use of Beaver triples [8]. There are multiple ways to generate Beaver Triples, but in our work we use *oblivious transfer* (OT) [43], which is a two-party protocol that is known to imply any secure two-party computations. The process of performing OT is not free, and requires communication to be performed between the parties and computation that is equivalent to the ones used in public key cryptography. For continued discussion on OT, we refer the reader to [43, 23]. Beaver triples consist of three additively shared values, a, b and c. Values a and b are chosen at random, while $c = a \cdot b$. To perform multiplication, the parties S_0 and S_1 locally compute $d = [\![x]\!] - [\![a]\!]$ and $e = [\![y]\!] - [\![b]\!]$ with their respective shares. By doing this, S_1 and S_2 have masked the true values of their sensitive information with their shares of a, b. As such, they can *open* d and e, making these values public, without revealing anything about their shares of x or y. Now, consider the following:

$$\begin{aligned} x \cdot y &= ((x - a) + a)((y - b) + b) \\ &= (d + a)(e + b) \\ &= de + db + ae + ab \\ &= de + db + ae + c \end{aligned} \tag{3.2}$$

This identity allows the parties to compute shares of the product $x \cdot y$. Specifically, one

party computes $d[\![b]\!] + e[\![a]\!] + [\![c]\!]$, and another computes $de + d[\![b]\!] + e[\![a]\!] + [\![c]\!]$.

With only two computing parties and two data owners (Alice and Bob), it is very natural for Alice to represent one computing party, and Bob to represent the other, but this does not always have to be the case. Alice and Bob could offload their computations to representatives (servers) that can act as their computing parties.

3.2.4 Replicated Secret Sharing

The other secret sharing scheme we use is a 3PC paradigm based on Replicated Secret Sharing (RSS) [4], which works by breaking data into *replicated secret shares*. The specific MPC protocol based on RSS that we use to ensure security is the *replcated-ring-party.x* protocol, with Extended daBits [22] (For details, see here [33]). For the sake of simplicity, when referring to this approach, we will simply call it the "RSS protocol". This technique is an honest majority paradigm that allows for 1 of the 3 parties to be corrupted without breaching the data owners secret values. Similar to Section 3.2.3, to use this technique we require Alice and Bob to map their data x into the ring via equation (3.1), creating z. Then, to secret share z, the data owners will split z into $z_1, z_2, z_3 \in \mathbb{Z}$ uniformly at random bound to the constraint $z = z_1 + z_2 + z_3 \mod q$. At this junction, two of the three shares of z must be sent to the computing parties S_1, S_2, and S_3. Specifically, we have (z_1, z_2) go to S_1, (z_2, z_3) to S_2, and (z_3, z_1) to S_3, i.e., party S_i owns $(z_i, z_{i+1 \mod 3})$.

Similar to Section 3.2.3, there are four common operations that can be performed locally without communication between the parties. Let $[\![x]\!] = (x_1, x_2, x_3)$ and $[\![y]\!] = (y_1, y_2, y_3)$, where S_i owns $x_i, x_{i+1 \mod 3}$ and $y_i, y_{i+1 \mod 3}$, and let $c \in \mathbb{Z}^+$, then, we have the following operations that can be performed locally:

- Addition ($z = x + y$): Party S_i computes $(x_i + y_i, x_{i+1 \mod 3} + y_{i+1 \mod 3})$, yielding proper shares of z. This operation is denoted as $[\![z]\!] = [\![x]\!] + [\![y]\!]$.
- Subtraction ($z = x - y$): Party S_i computes $(x_i - y_i, x_{i+1 \mod 3} - y_{i+1 \mod 3})$, yielding proper shares of z. This pperation is denoted as $[\![z]\!] = [\![x]\!] - [\![y]\!]$.
- Multiplication by constant a ($z = c \cdot x$): Party S_i computes $(c \cdot x_i, c \cdot x_{i+1 \mod 3})$, yielding

proper shares of z. This operation is denoted as $[\![z]\!] = c \cdot [\![x]\!]$

- Addition of a constant ($z = x+c$): Parties S_1, S_3 add c to x_1. Despite S_2 not performing any operations, each party now holds a proper share of z. This operation is denoted as $[\![z]\!] = [\![x]\!] + c$.

Multiplication is a more involved operation, requiring communication between the computing parties. To demonstrate this, suppose that the computing parties own shares of two values, v, w, and wish to compute the product $v \cdot w$. Then with the equality $v \cdot w = (v_1 + v_2 + v_3) \cdot (w_1 + w_2 + w_3)$ in mind, consider the following:

$$\begin{aligned} S_1 \text{ computes} \quad & v_1 \cdot w_1 + v_1 \cdot w_2 + v_2 \cdot w_1, \\ S_2 \text{ computes} \quad & v_2 \cdot w_2 + v_2 \cdot w_3 + v_3 \cdot w_2, \\ S_3 \text{ computes} \quad & v_3 \cdot w_3 + v_3 \cdot w_1 + v_1 \cdot w_3 \end{aligned} \tag{3.3}$$

Performing (3.3) gives the computing parties additive secret shares of $v \cdot w$. Now the parties need to transform their shares so that they are valid in a RSS setting. This operation requires that each party P_i sends a single value to party $P_{i+1 \mod 3}$, making this operation relatively cheap when compared to the approach in Section 3.2.3. For more information on the specifics of how this conversion takes place, we point the reader to [4].

This method of secret sharing is typically faster than common 2PC approaches because the multiplication protocol is relatively lightweight, only requiring the parties to send each other a single value, completely skipping OT which was required in the additive sharing scheme. This is possible because of the assumption that an honest majority of players exist. This increase in speed makes RSS an attractive scheme to use in MPC operations where runtime is important and where the honest-majority assumption applies.

At first sight, this setting may seem less natural than what we saw in section 3.2.3 since we have 2 data owners and 3 computing parties. However, our use-case can be extended to accommodate three computing parties. For example, it could be the case that Alice acts as one party, Bob as another, and there could be a third party, Charlie, perhaps a company

who provides their services to accommodate such protocols. It is also possible for Alice and Bob to outsource the secure computation to three servers. Alice and Bob secret share their inputs with the servers, these servers run the 3PC protocol based on RSS, and the shares of the results are sent to Alice, or Bob, who can recover the final result.

In the case where a third party is undesirable, or not possible, our protocols will work in a 2PC setting, potentially at the cost of efficiency. We remark that if more efficient protocols are proposed to compute Beaver Triples, they can immediately be applied to our solutions, resulting in improved run times for the 2PC setting. In Section 5.4, we compare the run times of both approaches.

3.2.5 Cryptographic Building Blocks

In Section 3.2.3 and 3.2.4, we described how to securely perform some of the most primitive operations, including addition and multiplication. With these operations, one can construct much more complicated protocols in a secure manner. Below are all of the cryptographic building blocks we use to create our novel protocols, π_{pers} and π_{infer}, as seen in Section 4.2. Each of the protocols listed below require each of the computing parties to communicate with each other. At the end of each protocol, the parties will all have secret shares of the output of the protocol.

- π_{eq}: This protocol takes as input two secret shared values, $[\![a]\!], [\![b]\!]$, and returns $[\![1]\!]_2$ if $a = b$, and $[\![0]\!]_2$ otherwise. Notice the subscript of 2, this means that after the execution of π_{eq}, the computing parties are left with a share of the value 0 or 1 in the ring Z_2. For more details, we refer to the reader to [14]
- $\pi_{\mathsf{scalar\_mul}}$: This is the scalar multiplication protocol. It takes a secret shared scalar $[\![a]\!]$, and a secret shared vector $[\![\mathbf{x}]\!] = \langle [\![x_1]\!], ..., [\![x_n]\!] \rangle$, and outputs $\langle [\![a \cdot x_1]\!], ..., [\![a \cdot x_n]\!] \rangle$. This protocol is a trivial extension of the multiplication protocol [23]
- π_{div}: Given two secret shared values $[\![x]\!], [\![y]\!]$, the computing parties execute π_{div}, giving each of them a proper share of the quotient between x, y, i.e., each party has the following share $[\![x/y]\!]$. At a lower level, this protocol uses many operations to estimate

the quotient, including Newton's method. For more details, see [33]

- π_{dot}: Given two vectors $[\![\mathbf{x}]\!] = \langle [\![x_1]\!], ..., [\![x_n]\!] \rangle, [\![\mathbf{y}]\!] = \langle [\![y_1]\!], ..., [\![y_n]\!] \rangle$, the computing parties take the element wise product of their shares $[\![x_i]\!]$ and $[\![y_i]\!]$, and then locally take the sum of said products. See [33] for additional details
- π_{Euclid}: Given a vector $[\![\mathbf{x}]\!] = \langle [\![x_1]\!], ..., [\![x_n]\!] \rangle$, the parties calculate a share of the Euclidean distance, $[\![\sqrt{x_1^2 + ... + x_n^2}]\!]$. To calculate π_{Euclid}, the computing parties use π_{dot} between $[\![\mathbf{x}]\!]$and a copy of itself. They also execute the secure square root protocol, which among other things also utilizes Newton's method. See [33] for more details on the secure square root function
- π_{argmax}: Given a vector $[\![\mathbf{x}]\!] = \langle [\![x_1]\!], ..., [\![x_n]\!] \rangle$, the parties perform comparisons between the elements of $[\![\mathbf{x}]\!]$ to find the largest value in the vector, which we will call x_i. After x_i is found, the parties each receive a secret share of i. For more details see [33]

With these basic building blocks, we have everything we need to construct π_{pers} and π_{infer}.

3.3 Security Models

We say something is *in-the-clear* if we make no attempt to hide the data. When converting data to secret shares, and designing protocols to process said secret shared data, there are two main kinds of adversarial models we consider. The first is the *honest-but-curious* adversary (sometimes known as the *semi-honest* adversary). An honest-but-curious adversary is one who follows the protocols as agreed upon by all parties, but will try to capture additional information during the execution of protocols [23]. This security setting is common in MPC based PPML (see e.g. [18]) as it lends itself to faster computations when compared to the alternative adversarial model, referred to as the *malicious adversary*. A malicious adversary is one who will arbitrarily deviate from the protocols [23]. The malicious parties could, for example, send false values to the other parties to ensure the output of the function is incorrect. MPC protocols that guard against these malicious adversaries tend to introduce additional layers of complexity. In our work, we guard against honest-but-curious adversaries.

Chapter 4

METHODOLOGY

4.1 Personalization without Privacy

Recently, Lin and Marculescu [35] proposed a lightweight TL algorithm that can leverage the knowledge gained from a Convolutional Neural Network (CNN) trained on source domain data, and personalize that model to work better with specific people, representing the target domain in this case. The authors note in their paper that a limitation of their work is that their personalization method requires the aggregation of source domain and target domain data, which could be problematic since the source domain may need to be kept private. In this section, we propose protocol π_{pers} (see Protocol 1) which extends the personalization method from [35] in a way that not only hides the values of the source data D_a, but also the end-user's target data D_b, and the CNN M itself.

Pseudocode for the personalization method in-the-clear is shown in Algorithm 1. The algorithm makes use of a feature extractor ϕ, which is Alice's pre-trained CNN M without the output layer, i.e. ϕ consists of all the layers of M up to and including the last hidden layer. To use ϕ, we feed data x into M like normal, but instead of taking the output from the output layer, we take the output from the last hidden layer of M. Assuming M is an accurate CNN, $\phi(x)$ should give valuable information about the raw data x. When we feed x through ϕ, we say that we are *projecting* x into the feature space. This is because in CNNs perform end-to-end feature extraction of data, which is to say that CNNs can take raw data from some input source, and transform it into a more meaningful representation. Note that x is actually a matrix of data, consisting of several sensor readings gathered over some period of time. In contrast, $\phi(x)$ is a single vector of dimension n, where n is the number of nodes in the last hidden layer in M.

In addition to ϕ, Algorithm 1 uses a subset D_a^* of the source data set D_a – which was presumably used by Alice to train the CNN M that the feature extractor ϕ stems from – and a target data set D_b. D_a and D_b have the same data schema, having ordered pairs (x, y) where x is sensor data indicating a users movements, and y is an activity label denoting what those movements corresponds to. The set of all possible labels occurring in D_a and D_b are called the label space, denoted as Y. Bob's data D_b consists of a few labeled instances of Bob's own data that serve as calibration data to assist the personalization method.

The personalized model is created with a *k-shot learning approach*. In the context of our work, k-shot means that D_a^* and D_b each contain k data samples of each activity label. So for example, if there are 3 labels, namely *walking*, *jogging*, and *standing*, and we are in a 5-shot setting, then we collect 5 samples of data corresponding to the *walking*, *jogging*, and *standing* activity each from both data owners' data, yielding 30 samples in total to train the personalized model. For the purposes of Algorithm 1, Alice creates a random subset of her data D_a called D_a^* such that it is the same size as D_b, both of which should have k of samples of each label.

The personalization method works by combining a small subset D_a^* of labeled source domain data with a small set D_b of labeled calibration data from the target domain, creating the data set C. We then project C into the feature space using ϕ. At this point, we average out projections with common class labels, and normalize them. In other words, for the class label j, we calculate a *weight vector* $\mathbf{w}_j$ which is the normalized average of all the data in C with the class label j that was projected into the feature space via ϕ. At a high level, the idea behind taking the average of projected values with the same class label is that we find some 'common ground' between the data of the source and target domain. The collection of all weight vectors creates the *weight matrix*, W. To infer the label for a new instance of data d we project d into the feature space as well, and determine which label j maximizes the dot product between $\mathbf{w}_j$ and $\phi(d)$. Whichever label maximizes said dot product suggests which label most accurately describes the sensor data in d.

For a more precise description of the personalization method, we reference Algorithm 1.

Algorithm 1 : Personalization Method in-the-clear

Input ϕ, a feature extractor; D_a^*, source data; D_b, target data; d, unlabeled instance of data; Y, label space

Output y, class label inferred for d

1: $C = D_a^* \cup D_b$
2: **for** j `in` Y **do** ▷ for each label j in Y
3: $X_j = \{x|(x, j) \in C\}$ ▷ all data with label j
4: $\tilde{\mathbf{w}}_j = mean\{\phi(x)|\ \mathrm{x} \in X_j\}$ ▷ element wise average of projections of X_j
5: $\mathbf{w}_j = \tilde{\mathbf{w}}_j / \|\tilde{\mathbf{w}}_j\|_2$ ▷ normalize vector
6: **end for**
7: $y = argmax_{j \in Y}\{\mathbf{w}_j \cdot \phi(d)\}$
8: **return** y

Line 1 has us combine D_a^* and D_b into C. Line 2 has us iterate over the labels of Y in a for-loop. In this loop, we have lines 3-5, which have us collect all samples in C with the label j (line 3), which are then projected into the feature space, and then averaged to yield a vector $\tilde{\mathbf{w}}_\mathbf{j}$ which is prototypical for the training instances from class j (line 4). Finally ending the loop, we normalize the vector $\tilde{\mathbf{w}}_\mathbf{j}$ by dividing by its L2 norm, giving us our weight vector $\mathbf{w}_\mathbf{j}$ (line 5). Finally on line 7, we project d into the feature space, and then calculate its similarity with the data of the jth label. The value of j which maximizes the dot product $\mathbf{w}_j \cdot \phi(d)$ tells us which labeled data points are most similar to d, and thus how we should classify d.

The model that we personalized to Bob is essentially a *cosine similarity* classifier. The cosine similarity of two vectors $\mathbf{a}$ and $\mathbf{b}$ is defined to be

$$sim(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|_2\ \|\mathbf{b}\|_2}.$$

In the personalized method, the vectors that we wish to observe the similarity of are $\mathbf{w}_j$ and

$\phi(d)$. Recall that we only need to find the value of j which maximizes the cosine similarity. With this in mind, observe the following

$$\frac{\tilde{\mathbf{w}}_j \cdot \phi(d)}{\|\tilde{\mathbf{w}}_\mathbf{j}\|_2 \ \|\phi(d)\|_2} \geq \frac{\tilde{\mathbf{w}}_{j'} \cdot \phi(d)}{\|\tilde{\mathbf{w}}_{j'}\|_2 \ \|\phi(d)\|_2}$$

iff

$$\frac{\tilde{\mathbf{w}}_j}{\|\tilde{\mathbf{w}}_j\|_2} \cdot \phi(d) \geq \frac{\tilde{\mathbf{w}}_{j'}}{\|\tilde{\mathbf{w}}_{j'}\|_2} \cdot \phi(d).$$

Thus, since we defined $\mathbf{w}_j = \tilde{\mathbf{w}}_{j'}/\|\tilde{\mathbf{w}}_{j'}\|_2$, we only need to find the label $j \in Y$ such that for all $j^{'} \in Y$, the following inequality is satisfied

$$\mathbf{w}_j \cdot \phi(d) \geq \mathbf{w}_{j'} \cdot \phi(d),$$

which is precisely what we do on line 7 of Algorithm 1

4.2 MPC Protocols for Private Transfer Learning

4.2.1 Privacy-Preserving Personalization Protocol

Algorithm 1 is an in-the-clear algorithm that operates on plaintext, without any regards for privacy. In Protocol 1, we present an MPC based protocol (called π_{pers}) for privacy-preserving model personalization. Table 4.1 tracks who owns what in our secure protocols. Alice owns D_a and also D_a^*, which is a subset of D_a. She also owns the feature extractor ϕ, which can be used to project data into the feature space. After the execution of π_{pers}, the computing parties are expected to have shares of the weight matrix W. With this matrix, the computing parties can execute Protocol 2 (called π_{infer}) along with Bob's new data instance d to make an inference on what the label for d should be.

All sub-protocols used in our secure protocols that require communication between the parties will be denoted with π. So for example, the MPC sub-protocol that the parties execute to securely compute the dot product of two vectors is denoted as π_{dot}. Operations that do not require communication between parties, such as addition, will not be denoted

Table 4.1: Ownership Table

Owner	Data	Description
Alice	D_a	– Original source data
	D_a^*	– A randomly selected subset of the source data. This subset should contain k instances of each label.
	ϕ	– Feature extractor derived from Alice's pre-trained CNN M that was originally trained on the source data.
Bob	D_b	– Labeled target data which will be used for calibration
	d	– unlabeled instance of data to be classified by π_{infer}
Secret shared among computing parties	W	– Weight matrix that is computed by the computing parties during the execution of π_{pers}. This weight matrix is Bob's personalized model. No single party owns W, instead, they own shares of W. To classify new data with W, the parties jointly execute π_{infer}.
Public	Y	– The label set with all possible labels.
	k	– Hyperparameter for k-shot learning. A number agreed upon by the data owners before execution. Note that Bob is expected to have at least k instances of labeled data in D_b for each label in Y.

by π. The protocol π_{pers} requires 2 inputs to be secret-shared before the execution of the protocol, namely ϕ, and D_b.

In line 1 of Protocol 1 Alice starts in an offline phase. Because she owns her own feature extractor ϕ, she can project her data into the feature space with no MPC, as seen on line 3. She is also responsible for initializing each weight vector $\tilde{\mathbf{w}}$, which acts as an intermediate value for the vectors that create the secret-shared matrix W. In line 4 of the protocol,

Alice increments the values row of $\tilde{\mathbf{w}}$ that corresponds to the i'th label. On line 7, Alice sends secret shares of the preliminary version of the weight matrix $\tilde{W}$ to all computing parties. The computing parties can use the secret shares of $\tilde{W}$ in combination with the secret-shared inputs that are listed at the top of Protocol 1. The computations between all computing parties start on line 9. In line 10-12 the parties jointly and securely project all of Bob's calibration data into the feature space, which is done with π_ϕ. The protocol π_ϕ we use here was implemented by us with the specific purpose to project data into the feature space. It is similar to MPC based CNN implementations designed for image classification [3, 13, 31, 34, 40, 33]. The main difference here is that image classification is based on 2-dimensional (2D) CNNs, while CNNs designed to process sensor data commonly use 1-dimensional (1D) CNNs. The aforementioned MPC based protocols for inference with 2D CNNs can be straightforwardly adjusted to 1D CNNs. Line 14 has the parties iterate over the label set, and line 15 has the parties iterate over Bob's data. The secure comparison protocol, π_{eq}, returns a secret sharing of 1 modulo 2, i.e. $[\![1]\!]_2$, if $y^{(i)}$ is equal to j, and $[\![0]\!]_2$ otherwise. Thus $[\![q_{i,j}]\!]$ on line 16 acts as a secret-shared Boolean value indicating whether the currently considered instance from Bob's data has the jth class label or not. If it does not, then line 17 "zeros" out the projected data $X_\phi[i]$ with scalar multiplication so that it cannot contribute to $\tilde{\mathbf{w}}_j$ in line 18. If equality is achieved, then the value of $X_\phi[i]$ is maintained, affecting $\tilde{\mathbf{w}}_j$.

After the for-loop that started on line 15 terminates, $\tilde{\mathbf{w}}_j$ accounts for all feature representations of all instances in C with the jth class label. We then take the average the projected feature values (see $\frac{1}{2k}$ on line 23), and normalize the vectors, which is performed between lines 21 and 23. Finishing up the for-loop that started on line 14 yields all vectors for the weight matrix W, concluding π_{pers}.

4.2.2 Privacy-Preserving Personalized Inference Protocol

Protocol 1 allows to personalize a pre-trained CNN to a specific user using some of their labeled data as calibration data. After its execution, all computing parties are left with

Protocol 1 π_{pers}: Secure Personalization Method

Input $[\![\phi]\!]$, secret-shared feature extractor; $[\![D_a^*]\!]$, secret-shared source data; $[\![D_b]\!]$, secret-shared target data; Y, label space (publicly known); k, number of examples for k-shot learning (publicly known)

Output $[\![W]\!]$, secret-shared weight matrix

start offline: Alice
  for $i \leftarrow 1$ `to` $size(D_a^*)$ **do**
    $\mathbf{x}_\phi = \phi\left(x^{(i)}\right)$
    $\tilde{\mathbf{w}}_{\mathbf{y}^{(i)}} \mathrel{+}= \mathbf{x}_\phi$
  end for
end offline
Alice secret-shares $\tilde{W}$ with all computing parties

start online: All Computing Parties
  for $i \leftarrow 1$ `to` $size(D_b)$ **do**
    $[\![X_\phi]\!][i] = \pi_\phi\left([\![x^{(i)}]\!]\right)$
  end for

  for j in Y **do**
    for $i \leftarrow 1$ `to` $size(D_b)$ **do**
      $[\![q_{i,j}]\!] = \left([\![y^{(i)}]\!], j\right)$ ▷ Compare the i'th label to j
      $[\![\mathbf{n}]\!] = ([\![q_{i,j}]\!], [\![X_\phi]\!][i])$ ▷ "zeros" out $X_\phi[i]$ if needed
      $[\![\tilde{\mathbf{w}}_j]\!] \mathrel{+}= [\![\mathbf{n}]\!]$
    end for

    $[\![\tilde{w}_j^{L2}]\!] = \pi_{\mathsf{Euclid}}\left([\![\tilde{\mathbf{w}}_j]\!]\right)$
    $[\![\tilde{w}_{j_{rec}}^{L2}]\!] = \pi_{\mathsf{div}}\left(1, [\![\tilde{w}_j^{L2}]\!]\right)$
    $[\![\mathbf{w}_j]\!] = \pi_{\mathsf{scalar\_mul}}\left(\frac{1}{2k} \cdot [\![\tilde{w}_{j_{rec}}^{L2}]\!], [\![\tilde{\mathbf{w}}_j]\!]\right)$
  end for
end online
return $[\![\mathbf{W}]\!]$ ▷ Collection of all weight vectors $\mathbf{w}_j$

Protocol 2 π_{infer}: Secure Inference Method

Input $[\![\phi]\!]$, secret-shared feature extractor; $[\![W]\!]$, secret-shared weight matrix; $[\![d]\!]$, unlabeled instance of data

Output $[\![y]\!]$, secret-shared class label inferred for $[\![d]\!]$

1: $[\![\mathbf{d}^*]\!] = \pi_\phi([\![d]\!])$
2: **for** j in Y **do**
3: $\quad [\![\mathbf{y}^*]\!][j] = \pi_{\mathsf{dot}}([\![\mathbf{w}_j]\!], [\![\mathbf{d}^*]\!])$
4: **end for**
5: $[\![y]\!] = \pi_{\mathsf{argmax}}([\![\mathbf{y}^*]\!])$
6: **return** $[\![y]\!]$

shares of the weight matrix W, which, combined with Alice's feature extractor ϕ, is Bob's personalized model, M^*. As described in section 4.1, this model is essentially a cosine similarity classifier.

To infer a label for Bob's new instance d, the computing parties execute Protocol 2. The protocol starts by having the computing parties jointly execute protocol π_ϕ to project d into the feature space, yielding the vector $\mathbf{d}^*$. Then on line 3, for each class label, the parties take the secure dot product between $[\![\mathbf{w}_j]\!]$ and $[\![\mathbf{d}^*]\!]$, calculating their similarity. The parties then find the largest of these dot products on line 5. This yields the inferred class label of d, being y. The parties terminate π_{infer} with shares of y. All computing parties will then send their shares of y to Bob, giving him the class label for his data instance d.

Chapter 5

RESULTS

We implemented our MPC protocols in MP-SPDZ [33], a benchmarking library that supports a variety of MPC schemes. We ran our tests using an additive secret-sharing scheme with two parties, and a replicated secret-sharing scheme using 3 parties. All tests were performed with Standard F48s v2 on Azure cloud computing servers. These machines have 48 vCPUs, 96 Gib of RAM and Gib Ethernet. It should be noted that despite high amounts of vCPUs and RAM, they are not often utilized. Our Protocol uses roughly 20 Gib of RAM at most, and is mostly single threaded. The primary benefit of larger machines is that it gives us more regular access to high network throughput, which is important in the context of MPC. All of our protocols run over a ring of 64 bits, and we use extended daBits, as proposed by Escudero et. al. [22]. These settings work for both the 3PC and 2PC approach we use.

5.1 datasets

We performed our tests on two datasets that the authors of the personalization method tested their work on [35]. The datasets are the Sports and Daily Activity dataset (SDA) [7], and the Sensors Activity dataset (SAC) [47].

The SDA dataset gathered sensor data from 8 participants. Measurements from these participants were taken with three different sensors, namely an accelerometer, a gyroscope, and a magnetometer, each with an x, y, z coordinate, resulting in 9 measurement values. The original data was collected with smartphones over 5 different body parts, so the dataset totals with 45 columns, plus a class label that denotes the activity. Each participants performed 19 different activities such as sitting, walking, rowing, and jumping. Sensor data was

collected from each participant 25 times a second for 5 minutes, resulting in 7500 samples corresponding to each class label.

The other dataset, SAC, used three of the same sensors as SDA, but introduced an additional sensor, a linear acceleration sensor. Just as before, each sensor produced three coordinates, resulting in 12 measurement values. Additionally the sensors were placed on five different body parts, resulting in 60 total columns, plus a class label. Each participants performed 8 different activities such as sitting, walking, and running. Sensor data was collected from each participant 50 times a second for 3 minutes, resulting in 9000 samples corresponding to each class label

To mimic a realistic use case with measurements collected through smart wristbands or watches, we only use sensor data from a single position. For the SDA dataset, we use the data gathered while the phone was on the participants' right arm (exact position not specified). For the SAC dataset, we used sensor data gathered from participants' right wrist. We used these locations because they most closely resemble the position of a smart watch, which is a fairly non-invasive device.

5.2 Data preprocessing

We normalized the data column wise by subtracting each value in the column by the mean of the column, and then dividing by the standard deviation of the column. Since our algorithm is done in a privacy-preserving way, it can be advantageous for us to normalize the source data D_a and the target (i.e., test) data D_b according to their own distribution, as opposed to letting the target data be normalized according to the mean and variance of the source data. This allows us to avoid division, which can be an expensive operation in the context of MPC. We show runtime results attempting both methods.

Each dataset we performed tests on contains sensor data gathered several times a second. In a preprocessing step, we broke the data into 1 second windows, i.e., each data point contains all the sensor data gathered in a second, making our datapoints matrices of sensor data. All the sensor data that was put into each datapoint contained the same activity,

and thus the class label we associate with each data point is simply the activity all the sensor data corresponded to. In doing this, each participant of the SDA dataset are left with 5700 datapoints (300 of each activity), each of which have 25 rows and 9 columns. Each participant in the SAC dataset are left with 1260 datapoints (180 of each activity), each of which have 50 rows, and 12 columns.

To simulate target and source domain data for each dataset, we collected the data from $n-1$ participants, where n is the total number of participants in that dataset, and aggregated their data to make the source domain data D_a. Then, the remaining 1 user was considered to be the target user, making their data D_b. We repeated this process n times, allowing each participant to take a turn being the target user.

5.3 CNN Architecture

To infer activity labels from the sensor measurements, we use a one-dimensional convolutional neural network (1D CNN), with an architecture similar to the one proposed in [35]. The input to the CNN is a 2D matrix, where each column corresponds to either an x,y, or z coordinate from a particular sensor, and each row is the accumulation of this sensor data. The output is a probability distribution over the activity labels.

We use a lean 1D CNN architecture that is efficient (even when used in combination with the MPC protocols) for both datasets. The 1D CNN we use for the SAC and the SDA datasets have the same layers, only differing in the number of parameters. Table 5.1 express's the architectures of our CNNs, and how the shape of the data changes from one layer to the next as response to our hyperparameter choices.

The layers in the CNN are standard (see e.g. [52] for details). We use a convolutional layer followed by a maxpool layer. We then follow up with another convolutional and maxpool layer. After the use of each individual convolutional/maxpool layer pair, we use a dropout of 25%. Both of the convolutional layers use a hard sigmoid as the activation function. After we perform the last maxpool layer, we flatten it and put it through a dense layer with ReLu as the activation. Finally, we make an output layer using softmax. For the SDA dataset, the

Layer (type)	Output Shape	Param#
Conv1D	(24, 8)	152
MaxPooling1D	(12, 8)	0
Dropout (25%)	(12, 8)	0
Conv1D	(8, 128)	5248
MaxPooling1D	(1, 128)	0
Dropout (25%)	(1, 128)	0
Flatten	(128)	0
Dense	(50)	6450
Dense	(19)	969
Total Params: 12,819		

Layer (type)	Output Shape	Param#
Conv1D	(49, 16)	400
MaxPooling1D	(24, 16)	0
Dropout (25%)	(24, 16)	0
Conv1D	(17, 128)	16512
MaxPooling1D	(2, 128)	0
Dropout (25%)	(2, 128)	0
Flatten	(256)	0
Dense	(50)	12850
Dense	(7)	357
Total Params: 30,119		

Table 5.1: Tables show how the data is transformed at each layer of the CNN. The left table is our CNN we trained on the SDA dataset, while the right table is the CNN we trained on the SAC dataset.

first convolutional/maxpool layer pair had 8 filters with kernel size 2, and a pool size of 2. The second convolutional/maxpool layer pair had 128 filters with kernel size 5, and a pool size of 8. For the SAC dataset, the first pair of layers had 16 filters of kernel size 2, and a pool size of 2. For the second pair, it used 128 filters with kernel size 8, and pool size 8.

We trained these CNN models in Keras [11]. In each case, these models are trained on the source data D_a, yielding a model M. Everything from model M, except for the final softmax layer, is used as a feature extractor ϕ in protocols π_{pers} and π_{infer}.

5.4 Accuracy and Runtime Results

5.4.1 Accuracy

To obtain our accuracy results, we split the target user's data, at random, into 1/5 of its original size, creating a dataset we will call B. We did this because testing MPC protocols can be an expensive to test, so it can be nice working with a smaller dataset. Then, to collect samples for the k-shot algorithm, we randomly sample k datapoints without replacement that correspond to each activity, creating the target domain D_b. The left over values are what we test on, and we will call this dataset T. In other words, we have $B = D_b \cup T$ and $\emptyset = D_b \cap T$. Recall that the source domain data, D_a, is the aggregation of all the participants data excluding the current target user. Then, to collect the subset of the source domain data that we use for training the personalizer, D_a^*, we randomly select k datapoints with replacement that correspond to each activity from D_a. In other words, $D_a = D_a \cup D_a^*$ and $D_a^* = D_a \cap D_a^*$. We repeated this process 5 times, and took the average over all results.

Table 5.2 shows us what kind of accuracy results we get when we normalize all the data according to the distribution of the source data. Each result shows the models accuracy for each target user. So for example, looking at the first row in the table, the CNN column indicates the accuracy we obtained by training a CNN on users 2-8, and testing it on user 1. The CNN column does not involve any personalization and was done in a non-privacy preserving manner.

In each row, we also show the k-shot accuracy, for $k = 1, 5, 10$, both using the original in-the-clear personalizer [35] ("No Privacy Protection"), and our secure personalizer ("Privacy Protected"). Looking at the average results across all datasets, we see that our privacy-preserving personalization method makes a positive impact on accuracy. For the SDA dataset, we obtained roughly a 7.5% increase in accuracy, and on the SAC dataset, we see a smaller, yet still significant increase in accuracy of about 1%, demonstrating that our personalization approach can benefit multiple datasets.

You may also notice some differences in the accuracy between the personalized model with

		No Privacy Protection				**Privacy Protected**		
			Personalized Models			**Personalized Models**		
Data	User	CNN	1-shot	5-shot	10-shot	1-shot	5-shot	10-shot
SDA	1	73.09	69.85	78.62	80.33	72.03	78.75	79.72
	2	78.46	76.84	80.47	81.29	77.25	80.56	80.59
	3	78.26	73.15	83.12	81.87	72.69	81.85	80.42
	4	84.57	82.25	86.12	88.22	81.32	86.95	87.18
	5	81.35	87.73	93.96	93.78	87.15	92.89	92.98
	6	67.01	73.74	83.65	84.10	73.70	83.27	84.08
	7	82.20	80.23	89.35	88.99	79.98	89.33	88.73
	8	69.56	75.79	79.38	82.59	75.41	80.31	82.14
	avg	**76.81**	**77.45**	**84.33**	**85.15**	**77.44**	**84.24**	**84.48**
SAC	1	86.50	88.27	87.48	87.45	87.96	87.23	87.45
	2	93.55	91.43	91.18	92.99	92.00	91.68	92.81
	3	95.71	94.37	96.39	96.10	95.35	96.47	96.02
	4	97.96	94.69	96.89	98.18	95.27	97.23	98.27
	5	92.64	91.76	94.62	94.20	92.57	95.71	94.63
	6	94.28	91.59	93.70	93.33	91.92	94.12	93.42
	7	96.69	94.04	97.31	97.75	94.20	97.73	97.92
	8	95.44	95.59	95.80	96.45	96.33	96.39	96.97
	9	94.25	94.69	95.29	95.76	95.10	94.71	96.28
	10	91.83	91.10	93.03	94.89	91.76	93.45	94.81
	avg	**93.89**	**92.75**	**94.17**	**94.71**	**93.25**	**94.47**	**94.86**

Table 5.2: Accuracy results for the SDA and SAC dataset where each individual participant had a turn to represent the target domain. The results in this table were obtained by normalizing all data by the distribution of data in the source domain.

		No Privacy Protection				**Privacy Protected**		
			Personalized Models			**Personalized Models**		
Data	User	CNN	1-shot	5-shot	10-shot	1-shot	5-shot	10-shot
SDA	1	67.36	70.54	79.87	82.05	70.76	79.75	81.87
	2	64.57	67.92	74.45	76.81	67.69	74.10	76.69
	3	58.77	73.79	78.58	79.76	73.54	78.02	79.52
	4	79.17	74.27	78.75	79.91	75.02	79.51	79.67
	5	63.84	79.79	93.97	92.58	79.59	92.65	91.38
	6	61.94	68.12	71.98	74.52	68.15	71.98	74.68
	7	70.08	73.65	79.07	82.99	73.47	79.64	83.05
	8	66.34	75.61	77.50	77.40	75.47	78.09	77.86
	avg	**66.51**	**72.96**	**79.27**	**80.75**	**72.96**	**79.22**	**80.59**
SAC	1	88.60	86.26	88.24	89.18	87.14	87.90	88.92
	2	92.05	90.53	91.43	92.47	91.43	91.60	92.38
	3	96.90	94.69	96.64	98.18	95.27	96.81	98.44
	4	97.50	93.39	96.97	96.36	92.90	96.97	96.54
	5	92.27	90.12	90.84	92.73	90.86	91.34	93.25
	6	94.36	90.86	93.95	93.94	90.94	94.62	93.85
	7	96.25	96.82	97.65	97.75	96.98	97.73	97.84
	8	95.01	92.98	95.80	96.19	92.73	95.29	96.02
	9	93.41	95.10	94.03	93.77	94.20	94.45	95.41
	10	90.52	92.00	92.10	91.52	91.51	92.69	92.38
	avg	**93.69**	**92.28**	**93.77**	**94.21**	**92.40**	**93.94**	**94.50**

Table 5.3: Accuracy results for the SDA and SAC dataset where each individual participant had a turn to represent the target domain. The results in this table were obtained by normalizing all data by the datasets own distribution.

no privacy, and our model with privacy. This is to be expected, because many functions in the context of MPC are very expensive. As such, in the MP-SPDZ framework we used to write our code, there are built in functionalities that approximate the true value of functions as opposed to outright calculating them. This process of approximating several functions can have some affect on the output of our model. Despite this, the accuracy between the in-the-clear model and ours is very minimal, and within the range of $\pm 0.67\%$

Table 5.3 is similar to the previous table, but here we show the accuracies we obtain when we normalize the target and source domain data according to their own distribution, which was done in order to avoid secure division. Similar to the previous table, we see that both datasets can benefit from the personalization method. Using our secure method, the SDA dataset received an increase in accuracy of about 14%, and the SAC dataset obtained an increase of about 0.8%. Unfortunately, the overall accuracy of the SDA dataset in Table 5.3 fell by about 4%, and thus clearly suffered from our decision to normalize the data domains by their own distribution. In contrast, the SAC dataset seemed mostly unphased by this decision, possibly making it ideal to normalize the datasets according to their distribution in the context of the SAC dataset, but not the SDA dataset.

5.4.2 Runtime

Tables 5.4 and 5.5 describe the runtime, in seconds, it takes to train our personalized k-shot classifier for $k = 1, 5, 10$ using the 2PC scheme and the 3PC scheme, respectively. It also shows how long it takes, on average, to classify a new, unseen instance of data, both sequentially, and batching the classification into groups of 15. To perform sequential classifications, we used MP-SPDZ's `@for_range(length_of_test_data)` to loop over the test data, and to perform batched classifications, we used MP-SPDZ's `@for_range_parallel(15, length_of_test_data)`. The "# Params" column tells us how large the feature extractor is in terms of parameters. Recall that the feature extractor is Alice's model M without the last dense layer, this is why the number of parameters in these tables are slightly less than what is show in table 5.1. Lastly, we have the "Normalized"

Runtime Results for Additive Secret-Sharing Approach							
Data	# Params	Normalized	π_{pers}			π_{infer}	
			1-shot	5-shot	10-shot	sequential	batched (15)
SDA	11,850	Source	18.77s	87.84s	177.86s	0.98s	0.88s
		Self	16.64s	78.44s	156.75s	0.93s	0.77s
			1-shot	5-shot	10-shot	sequential	batched (15)
SAC	29,762	Source	16.98s	84.15s	163.72s	2.69s	2.61s
		Self	15.66s	75.01s	149.44s	2.51s	2.40s

Table 5.4: Runtimes for the 2PC additive sharing based protocol from section 3.2.3. #Params tell us how large the feature extractor was. Column π_{pers} shows us how long it takes to train a classifier in a 1,5, and 10-shot setting. Column π_{infer} shows us how long, on average, it takes to infer data sequentially and in batches of 15

column. In this column, we have the field "Source" which tells us that all data was normalized by the source domains distribution, thus requiring secure division. The field "Self" tells us that data was normalized according to its own distribution in-the-clear, avoiding secure division.

Table 5.4 shows our runtimes using the additive sharing 2PC approach. Times for π_{pers} are fairly efficient, scaling nearly linearly with k. This is because the amount of data used to train our personalized model is linearly dependent on k. Note that the data required to train on the SDA dataset is far more than the data required to train on the SAC dataset. This is because SDA had 19 activities, while SAC had 7. So for $k = 10$, we use $10 * 19 * 2 = 380$ samples to train π_{pers} on SDA, and only $10 * 7 * 2 = 140$ samples to train π_{pers} on SAC. Despite this, the runtimes for π_{pers} are similar between the two datasets. This is because the feature extractor for SAC is more than twice as large than that of SDA, evening out the total runtimes.

As long as π_{pers} does not take an egregious amount of time to execute, its runtimes

Runtime Results for Replicated Secret-Sharing Approach							
Data	# Params	Normalized	π_{pers}			π_{infer}	
			1-shot	5-shot	10-shot	sequential	batched (15)
SDA	11,850	Source	8.39s	39.55s	77.57s	0.42s	0.41s
		Self	8.27s	38.10s	78.00s	0.41s	0.42s
			1-shot	5-shot	10-shot	sequential	batched (15)
SAC	29,762	Source	4.44s	20.88s	42.02s	0.67s	0.65s
		Self	4.30s	20.95s	41.25s	0.67s	0.66s

Table 5.5: Runtimes for the 3PC replicated sharing based protocol from section 3.2.4 #Params tell us how large the feature extractor was. Column π_{pers} shows us how long it takes to train a classifier in a 1,5, and 10-shot setting. Column π_{infer} shows us how long, on average, it takes to infer data sequentially and in batches of 15

are not that important. This is because this model only needs to be trained once, and training could be performed when the target user is asleep, for example, not affecting their daily routine at all. What's more pressing are the runtimes for π_{infer}. Recall that data was collected in 1 second windows, so in order for our work to be performed in real time, π_{infer} must take less than a second on average. For the SDA dataset, we accomplish this by a thin margin. Performing classifications sequentially have our protocol take just under a second at 0.98 seconds when normalizing according to the source distribution, and 0.93 seconds when normalizing data according to its own distribution. Batching performs a little better, giving us some more breathing room, taking an average of 0.88 seconds with the "Source" field, which means it took a total of $15 \cdot 0.88 = 13.2$ seconds total to classify all 15 samples, and 0.77 seconds (11.55 total seconds) with the "Self" field. In contrast, our results for the SAC dataset take more than twice as long as we need them to be in all cases. Again, this is likely because the feature extractor is more than twice as large as the one for the SDA data in terms of the number of parameters. When normalizing data by the distribution of the

source domains data, we see a slight increase in runtime when compared to runtimes when we normalize the data in-the-clear by its own distribution. Although saving ourselves some time can be valuable, given how small the margin is, the potential decrease in accuracy may not be worth it.

Table 5.5 shows our runtimes using the replicated sharing 3PC approach. As we can see, these runtimes significantly outperform that of table 5.4. Runtimes for π_{pers} are very low, taking under 80 seconds for SDA in the 10-shot scenario, and are nearly half of that for the SAC dataset. In fact, π_{pers} on the SAC dataset was consistently about half the runtime of SDA. The performance on the datasets differs from what we saw in Table 5.4, whose runtimes for π_{pers} were all roughly the same time. This could be for a multitude of reasons. One possibility is that since multiplication in the context of the 3PC setting takes significantly less time than that of the 2PC setting, the overhead introduced by the other operations, such as secure comparisons in the max pooling layer, might have become more prevalent, resulting in wider runtime gaps.

Possibly even more interesting is that our protocols seem completely unphased by the division introduced by normalizing data according to the source domain, and batching samples of data in the classification phase. The former is likely because division in the context of the 3PC setting is also relatively fast, and we did not have to perform enough of it to see a serious impact in runtimes. In terms of batching operations, the main benefit is that we can batch communication rounds more efficiently. Given how little communication the RSS based protocol needs in order to perform several primitive operations, it could be that the protocol does not really benefit from batching only 15 samples of data.

Chapter 6

CONCLUSION AND FUTURE WORK

We adapted an in-the-clear algorithm for personalization based HAR with wearable sensors to a secure protocol which allows a target user to receive a boost in their classification accuracy, all the while keeping their data, and the original source owner's data, secure. We ran accuracy tests on two datasets which showed that our approach is roughly equivalent to its in-the-clear counterpart. We also ran runtime tests using two different MPC protocols, one based on additive secret-sharing, and another based on replicated secret-sharing. With the former, we discovered that our results may be fast enough to provide real time inference on the SDA dataset. For the SAC dataset, our runtimes were too long, and may require optimizing the code, or reducing the complexity of the feature extractor it uses to help classify data, which could be valuable future work. Using the replicated secret-sharing approach, we found that our runtimes were very fast, and that it may be possible to use our method in its current state to perform real time classification. The classification model we proposed in π_{infer} is a very lightweight CNN model in combination with a cosine similarity classifier that should be highly scalable in practice. Moreover, the approach we use is general enough to accommodate other model architectures besides CNN and cosine similarity models, potentially leaving room for future research and improvements in terms of both accuracy and runtime.

BIBLIOGRAPHY

[1] Mark Abspoel, Daniel Escudero, and Nikolaj Volgushev. Secure training of decision trees with continuous attributes. *Proceedings on Privacy Enhancing Technologies*, 2021(1):167–187, 2021.

[2] Anisha Agarwal, Rafael Dowsley, Nicholas D. McKinney, Dongrui Wu, Chin-Teng Lin, Martine De Cock, and Anderson Nascimento. Protecting privacy of users in brain-computer interface applications. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(8):1546–1555, 2019.

[3] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. QUOTIENT: two-party secure neural network training and prediction. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1231–1247, 2019.

[4] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 805—-817, 2016.

[5] Giuseppe Ateniese, Luigi V. Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. 2015.

[6] Serkan Balli, Ensar Arif Sağbaş, and Musa Peker. Human activity recognition from smart watch sensor data using a hybrid of principal component analysis and random forest algorithm. *Measurement and Control*, 52(1-2):37–45, 2019.

[7] Billur Barshan and Murat C. Yüksek. Recognizing daily and sports activities in two open source machine learning environments using body-worn sensor units. *The Computer Journal*, 57(11):1649–1667, 2014.

[8] Donald Beaver. Commodity-based cryptography (extended abstract). In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, 1997.

[9] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium*, pages 267–284, 2019.

[10] Yiqiang Chen, Xin Qin, Jindong Wang, Chaohui Yu, and Wen Gao. FedHealth: a federated transfer learning framework for wearable healthcare. *IEEE Intelligent Systems*, 35(4):83–93, 2020.

[11] Francois Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[12] Ronald Cramer, Ivan Damgard, and Jesper Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press Print, New York, 2015.

[13] A. Dalskov, D. Escudero, and M. Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies*, 2020(4):355–375, 2020.

[14] Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure mpc over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1102–1120, 2019.

[15] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson Nascimento, Wing-Sea Poon, and Stacey Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE Transactions on Dependable and Secure Computing*, 16(2):217–230, 2019.

[16] Martine De Cock, Rafael Dowsley, Anderson C. A. Nascimento, Davis Railsback, Jianwei Shen, and Ariel Todoki. High performance logistic regression for privacy-preserving genome analysis. *BMC Medical Genomics*, 14(1):23, 2021.

[17] Martine De Cock, Rafael Dowsley, Anderson C.A. Nascimento, and Stacey C. Newman. Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 3–14, 2015.

[18] Sebastiaan de Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. Practical secure decision tree learning in a teletreatment application. In *International Conference on Financial Cryptography and Data Security*, pages 179–194. Springer, 2014.

[19] Cynthia Dwork. Differential privacy: A survey of results. In *Proc. of 5th International Conference on Theory and Applications of Models of Computation*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2008.

[20] Cynthia Dwork and George J. Pappas. Privacy in information-rich intelligent infrastructure. *CoRR*, abs/1706.01985, 2017.

[21] Sannara Ek, François Portet, Philippe Lalanda, and German Vega. Evaluation of federated learning aggregation algorithms: application to human activity recognition. In *Adjunct Proc. of the 2020 ACM International Joint Conf. on Pervasive and Ubiquitous Computing and Proc. of the 2020 ACM International Symposium on Wearable Computers*, pages 638–643, 2020.

[22] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for mpc over mixed arithmetic-binary circuits. Cryptology ePrint Archive, Report 2020/338, 2020. `https://ia.cr/2020/338`.

[23] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends in Privacy and Security*, 2(2-3):70–246, 2018.

[24] Kyle Fritchman, Keerthanaa Saminathan, Rafael Dowsley, Tyler Hughes, Martine De Cock, Anderson Nascimento, and Ankur Teredesai. Privacy-preserving scoring of tree ensembles: A novel framework for AI in healthcare. In *Proc. of 2018 IEEE BigData*, pages 2412–2421, 2018.

[25] Chuan Guo, Awni Hannun, Brian Knott, Laurens van der Maaten, Mark Tygert, and Ruiyu Zhu. Secure multiparty computations in floating-point arithmetic. *Information and Inference: A Journal of the IMA*, 2021. iaaa038.

[26] Mina Hashemian, Farbod Razzazi, Houman Zarrabi, and Mohammad Moin. Semi-supervised and unsupervised privacy-preserving distributed transfer learning approach in HAR systems. *Wireless Personal Communications*, 117:1–18, 2021.

[27] Mina Hashemian, Farbod Razzazi, Houman Zarrabi, and Mohammad Shahram Moin. A privacy-preserving distributed transfer learning in activity recognition. *Telecommunication Systems: Modelling, Analysis, Design and Management*, 72(1):69–79, 2019.

[28] Netzahualcoyotl Hernandez, Jens Lundström, Jesus Favela, Ian McChesney, and Bert Arnrich. Literature review on transfer learning for human activity recognition using mobile and wearable devices with environmental technology. *Springer Nature Computer Science*, 1(66), 2020.

[29] Matthew Jagielski, Michael Kearns, Jieming Mao, Alina Oprea, Aaron Roth, Saeed Sharifi-Malvajerdi, and Jonathan Ullman. Differentially private fair learning. In *International Conference on Machine Learning*, pages 3000–3008, 2019.

[30] Zhanglong Ji, Zachary Chase Lipton, and Charles Elkan. Differential privacy and machine learning: a survey and review. *CoRR*, abs/1412.7584, 2014.

[31] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium*, pages 1651–1669, 2018.

[32] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. https://arxiv.org/abs/1912.04977, 2021.

[33] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1575–1590, 2020.

[34] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow: Secure TensorFlow inference. In *41st IEEE Symposium on Security and Privacy*, 2020.

[35] Ching-Yi Lin and Radu Marculescu. Model personalization for human activity recognition. In *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2020.

[36] Yuan-Pin Lin and Tzyy-Ping Jung. Improving EEG-based emotion classification using conditional transfer learning. *Frontiers in Human Neuroscience*, 11:334, 2017.

[37] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Annual International Cryptology Conference*, pages 36–54, 2000.

[38] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via MiniONN transformations. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631, 2017.

[39] Saeed Mehrang, Julia Pietila, Johanna Tolonen, Elina Helander, Holly Jimison, Misha Pavel, and Ilkka Korhonen. Human activity recognition using a single optical heart

rate monitoring wristband equipped with triaxial accelerometer. In *Joint Conference of the European Medical and Biological Engineering Conference (EMBEC) and the Nordic-Baltic Conference on Biomedical Engineering and Medical Physics (NBC)*, pages 587–590, 2017.

[40] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium*, 2020.

[41] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, 2017.

[42] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy (SP)*, pages 334–348, 2013.

[43] Michael O Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, 2005(187), 2005.

[44] Devin Reich, Ariel Todoki, Rafael Dowsley, Martine De Cock, and Anderson Nascimento. Privacy-preserving classification of personal text messages with secure multi-party computation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 3752–3764, 2019.

[45] Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Asia Conference on Computer and Communications Security*, pages 707–721. ACM, 2018.

[46] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. DeepSecure: Scalable provably-secure deep learning. In *55th Annual Design Automation Conference (DAC)*, 2018.

[47] Muhammad Shoaib, Stephan Bosch, Ozlem Incel, Hans Scholten, and Paul Havinga. Fusion of smartphone motion sensors for physical activity recognition. *Sensors*, 14:10146–10176, 2014.

[48] Konstantin Sozinov, Vladimir Vlassov, and Sarunas Girdzijauskas. Human activity recognition using federated learning. In *2018 IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications*, pages 1103–1111, 2018.

[49] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26–49, 2019.

[50] Alexandra Wood, Micah Altman, Aaron Bembenek, Mark Bun, Marco Gaboardi, James Honaker, Kobbi Nissim, David R. OBrien, Thomas Steinke, and Salil Vadhan. Differential privacy: A primer for a non-technical audience. *Vanderbilt Journal of Entertainment & Technology Law*, 21(1):209–275, 2018.

[51] Dongrui Wu, Yifan Xu, and Bao-Liang Lu. Transfer learning for EEG-based brain-computer interfaces: A review of progress made since 2016. *IEEE Transactions on Cognitive and Developmental Systems*, 2020.

[52] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.