Secure Multi-Party Computation for Personalized Human Activity Recognition

David Melanson[1*], Ricardo Maia[2], Hee-Seok Kim[1], Anderson Nascimento[1] and Martine De Cock[1,3]

[1]School of Engineering and Technology, University of Washington, 1900 Commerce Street, Tacoma, 98402-3100, WA, United States.
[2]Dept. of Computer Science, Institute of Exact Sciences, University of Brasília, Federal District, Brasília, 70910-900, Brazil.
[3]Dept. of Applied Mathematics, Computer Science and Statistics, Ghent University, Krijgslaan 281 (S9), 9000, Gent, Belgium.

*Corresponding author(s). E-mail(s): mence40@uw.edu;
Contributing authors: ricardo.jmm@gmail.com; heeskim@uw.edu; andclay@uw.edu; mdecock@uw.edu;

Abstract

Calibrating Human Activity Recognition (HAR) models to end-users with Transfer Learning (TL) often yields significant accuracy improvements. Such TL is by design done based on very personal data collected by sensors worn close to the human body. To protect the users' privacy, we therefore introduce Secure Multi-Party Computation (MPC) protocols for personalization of HAR models, and for secure activity recognition with the personalized models. Our MPC protocols do not require the end-users to reveal their sensitive data in an unencrypted manner, nor do they require the application developer to disclose their trained model parameters or any other sensitive or proprietary information with anyone in plaintext. Through experiments on HAR benchmark datasets, we demonstrate that our privacy-preserving solution yields the same accuracy gains as TL in-the-clear, i.e. when no measures to protect privacy are in place, and that our approach is fast enough for use in practice.

Keywords: Transfer Learning, Human Activity Recognition, Convolutional Neural Network, Secure Multi-Party Computation, Cryptography, Privacy

1 Introduction

Machine Learning (ML) is a powerful tool for application developers to exploit information found in data. As a general 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. However, in many application domains, the data is scarce. Transfer Learning (TL) aims to overcome this data scarcity issue by using data from different, but related domains [1–3].

An important use-case of TL is in Human Activity Recognition (HAR), with at least 170 papers published in the last decade regarding the topic [4]. Recognition of human activity – such as standing, walking, or running – from sensors in wearable devices like wristbands and smartphones is fueling a growing variety of applications, including early detection of diseases, strokes or seizures, mental health assessment, fall detection, and sports monitoring [5, 6]. 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. [4, 7–9] 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 misuse. Allowing the target user free access to the pre-trained parameter values of a proprietary model on the other hand would also be problematic, even more so because trained models leak information about the training data [10], 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. [7]), there is a wide gap in the literature on technologies to this end.

In this paper we propose cryptographic protocols that enable the same kind of model personalization and calibration as in-the-clear (i.e. when no encryption is used), without requiring 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, an application developer nick-named *Alice* has trained a model M for HAR on source dataset D_a. A new end-user *Bob* is interested to use M to monitor his physical condition via sensor data gathered from his watch [11]; 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 [7]. To remedy this situation, Alice and Bob could use a transfer learning

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

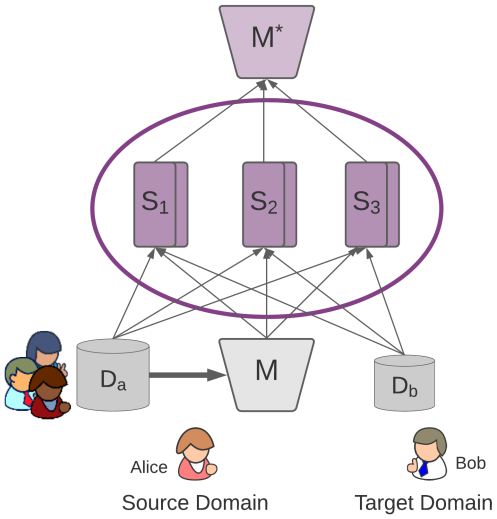

Fig. 1 Diagram illustrating privacy-preserving TL between Alice (representing the source domain) and Bob (representing the target domain). Alice and Bob send encrypted shares of D_a, M, and D_b to computing parties S_1, S_2, and S_3. These parties subsequently execute MPC protocol π_{pers} to construct a personalized model M^*.

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), 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 $S_1, S_2, \ldots$ (see Figure 1). These servers or *computing parties* subsequently perform computations over the encrypted shares – as defined in MPC protocols proposed in this paper – to arrive at encrypted shares of the personalized model M^*. These computations are performed in a way so that no single server learns anything about D_a, D_b, M and M^*. If so desired, the final trained model M^* can be reconstructed by a designated party, or it can be evaluated in an oblivious (encrypted) way by the servers. The price to be paid for the protection of privacy is an increase in runtime.

The primary contributions of this paper are the design, implementation, and evaluation of the first privacy-preserving MPC protocols for personalization of Convolutional Neural Networks (CNNs) for HAR through TL. Algorithms to improve 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) and without leaking the source domain information in the clear to the end-user.* Our main contributions are:

- An MPC protocol π_{pers} for privacy-preserving personalization of a CNN for HAR (Section 4).
- An MPC protocol π_{infer} for secure activity recognition with the personalized model (Section 4).
- A detailed evaluation of the MPC protocols in terms of accuracy and runtime in various security settings on HAR benchmark datasets, demonstrating that our privacy-preserving solution yields the same accuracy gains as in-the-clear, i.e. when no measures to protect privacy are in place, and that our approach is fast enough for use in practice (Section 5).

Our MPC protocols constitute the first approach to offer end-to-end privacy for both the client Bob and the model owner Alice (Section 2), ensuring that all the data remains secure, including the parameters of Alice's pre-trained model. Moreover, the security of our constructions follows from rigorous cryptographic models and definitions (Section 3).

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), cryptographic techniques such as Secure Multiparty Computation (MPC) and Homomorphic Encryption (HE), and combinations of all these approaches. Research on the application of PPML for human activity recognition from wearable sensors, as we do in this paper, has been fairly limited so far. Below we discuss the most relevant related work.

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 [13]. Typically, 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. The weaknesses of FL

compared to the method we propose in this paper, 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.

Exploration of FL for personalized human activity recognition from wearable sensor data is relatively new, with existing approaches addressing the above mentioned issues to some extent. Presotto et al. [14] have proposed FedCLAR, a hierarchical clustering method that clusters model parameters of Artificial Neural Networks (ANNs), that are locally trained by end-users on their own data, into similar groups, creating a global model. To personalize a model for a particular user, the user receives the aggregated parameters from the cluster they belong to, and retrains the last few layers of their ANN. While addressing the need for personalization, FedCLAR makes no attempt to hide the trained model parameters, thereby leaking information about the users to each other. In contrast, as we explain below, Chen et al. [15], Hu et al. [16], and Liu et al. [17] make an effort to obfuscate the users' parameter values, making their overall approaches more privacy-preserving.

FedHealth [15], which is specifically designed for HAR systems leveraging TL for users with wearable sensors in the health sector, 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 update are encrypted via **Homomorphic Encryption (HE)**, and then sent to the server. The server owner then takes the average of the individual user models, yielding a new global model, which is subsequently 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. As in FedCLAR, 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 user's 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.

An entirely different way of enhancing the privacy protection of FL algorithms is through **Differential Privacy (DP)** mechanisms [18]. Solutions for training HAR models that are based on DP (see e.g. [16, 17]) introduce noise in the model parameters, the gradients, or the objective function during training, to limit the amount of information leaked from the trained model.

While MPC (the approach we use in this paper) and DP both provide formal privacy guarantees that can be mathematically proven, they are of a very different nature. DP provides *output privacy* in the sense that one should not be able to conclude from the output (the trained model) whether a particular instance was used in the training dataset or not. MPC on the other hand protects *input privacy* in the sense that the model owner *Alice* and the target user *Bob* do not have to disclose their model and training samples to anyone. While the computational overhead of DP is much lighter than that of MPC, a substantial disadvantage of DP is that the use of noise causes a decrease in utility (accuracy), which is proportional to the privacy protection, i.e. the more one wants to protect privacy, the more noise is injected, and the more the accuracy drops. The use of MPC on the other hand does not cause a loss in accuracy, as we demonstrate empirically in Section 5. Moreover, we provide end-to-end privacy for all entities involved, revealing no sensitive information during intermediate stages.

Very different from the above, a **Privacy-Preserving Support Vector Machine (PPSVM) for HAR** approach [19, 20] has been proposed that is applicable in a scenario like the one in Figure 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 and making these random projections publicly known. The random projections can be used to train a support vector machine model. There are some issues with the results presented in [19, 20]. First, while preventing an exact reconstruction of Bob's inputs, the protocol does leak collective information about these entries through the random projections (that is the reason one is able to train the classifier in the first place). Second, the protocol leaks the labels: they are publicly announced and are necessary for training the final model. As mentioned before, the approach we propose in this paper offers 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. Moreover, the security of our constructions follows from rigorous cryptographic models and definitions.

3 Preliminaries on MPC

In this section, we review the MPC and cryptography background necessary to understand how we maintain privacy in our pipeline.

3.1 Introduction

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 [21]. 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 [22–30], as well as in the privacy-preserving training of machine learning models, such as linear regression models [31–33], decision tree models [34–37], and neural network architectures [38–42].

In Section 3.2, we describe the main idea of MPC and the adversarial models considered in this paper. After discussing in Section 3.3 how data and model owners should represent their data for consumption by MPC protocols, in Section 3.4 we recall details about the prominent secret-shared based MPC schemes in which we have implemented and evaluated our work. We refer to [21] for a self-contained introduction to MPC.

3.2 MPC Idea and Adversarial Models

MPC Idea. MPC provides a way for several mutually distrustful parties to jointly compute a function (which depends on secret inputs) such that: (i) the function output is correctly computed; and (ii) no information about the secret inputs is ever leaked (beyond possibly the result of the function itself). Out of several different approaches for implementing MPC, we work with secret-sharing based MPC. In secret-sharing based MPC, *data owners* encrypt information by distributing so-called shares of the data across multiple *computing parties*. Such shares are constructed in a way that no information about the secret inputs can be inferred from a secret share. However, when a sufficient number of shares is put together, they allow the recovery of the secret inputs. The main idea of secret-sharing MPC is that computations happen on the secret shares, rather than on the inputs themselves. In the end, the computing parties have shares of the desired function output and can jointly recover such result by combining all the shares.

MPC-as-a-Service. It is possible for the computing parties and the data holders to be the same. If this is the case, these computing parties first secret share their inputs among themselves and then proceed by executing MPC protocols on top of these shares. It is also possible for the data holders to *outsource* or *delegate* these computations to computing servers – an MPC-as-a-service scenario. In that case, the data holders first secret share their inputs with the computing servers. The servers then proceed to compute the desired function by executing MPC protocols on these shares. Our proposed protocols are general in the sense they can be adapted to these two-scenarios. However, for the sake of simplicity, we will present our results and analyses for the MPC-as-a-service case, as illustrated in Figure 1. In the remainder of this paper, by "data" we mean the datasets 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.

Adversarial Models. MPC is concerned with the protocol execution coming under attack by an adversary $\mathcal{A}$ which may corrupt the computing parties to learn private information or cause the result of the computation to be incorrect. MPC protocols are designed to prevent such attacks being successful. A party corrupted by a *semi-honest adversary* (sometimes known as an "honest-but-curious" adversary) still follows the protocol instructions as agreed upon by all parties, but will try to capture additional information during the execution of protocols [21]. A party corrupted by a *malicious adversary* may arbitrarily deviate from the protocol instructions, for example by sending forged values to the other parties to ensure the output of the function is incorrect. The MPC protocols π_{pers} and π_{infer} presented in Section 4 are sufficiently generic to protect against semi-honest as well as malicious adversaries. This is achieved by changing the underlying MPC scheme to align with the desired security setting. Some of the most efficient MPC schemes have been developed for 3 parties, out of which at most one is corrupted. In Section 5, we evaluate the runtime of our protocols in this honest-majority 3PC setting with semi-honest adversaries (see Section 3.4.2), which is growing in popularity in the PPML literature, e.g. [22, 26, 29, 42], as well as in a dishonest-majority 2PC setting (see Section 3.4.1).

3.3 Fixed Point Representation in a Ring

Mathematical operations in MPC protocols are done over a finite, discrete structure, such as a finite ring or a finite field. In our work, during the execution of any protocol, operations are performed on integers (shares) in the ring $\mathbb{Z}_q$, with $q = 2^\lambda$, 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 datasets 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 from their data into the ring. To do so, they use the function $Q : \mathbb{R} \to \mathbb{Z}_{2^\lambda}$, defined as [39]:

$$Q(x) = \begin{cases} 2^\lambda - \lfloor 2^a \cdot |x| \rfloor & \text{if } x < 0 \\ \lfloor 2^a \cdot x \rfloor & \text{if } x \geq 0 \end{cases} \tag{1}$$

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.

3.4 Secret-Sharing based MPC

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 secret-share a value x, we first require that the data owners map their data into $\mathbb{Z}_q$ by Equation (1). They then split the data into secret shares on their end (cfr. Section 3.4.1 and 3.4.2) 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 $[\![z]\!]_q = (z_1, z_2, ..., z_n)$. All computations in our work are assumed to be done in the ring $\mathbb{Z}_q$, and so for the remainder of this paper, we will write $[\![z]\!]_q$ as $[\![z]\!]$ for convenience.

We implement 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 setting, respectively. We discuss them in greater detail in Section 3.4.1 and 3.4.2.

3.4.1 Additive Secret-Sharing for 2PC

To additively secret share a value $z \in \mathbb{Z}_q$ between two computing parties S_1 and S_2, the data owner splits z into two shares $z_1, z_2 \in \mathbb{Z}_q$ chosen uniformly at random bound to the constraint $z = z_1 + z_2 \mod q$. The data owner then distributes these shares to the computing parties (servers), i.e. z_1 to S_1 and z_2 to S_2. 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.

There are several common operations that can be performed on secret shares without requiring communication among the parties. Let $[\![x]\!] = (x_1, x_2)$ and $[\![y]\!] = (y_1, y_2)$, where S_i owns x_i, y_i, and let c be a constant in $\mathbb{Z}_q$, then the following operations can be performed locally:

- Addition ($z = x + y$): $x + y = (x_1 + x_2) + (y_1 + y_2) = (x_1 + y_1) + (x_2 + y_2)$, thus the parties can simply sum up their respective shares of x, y. This operation is denoted as $[\![z]\!] = [\![x]\!] + [\![y]\!]$.
- Multiplication by a constant ($z = c \cdot x$): $c \cdot x = c \cdot x_1 + c \cdot x_2$, thus each party simply multiplies their share of x by c. This operation is denoted as $c[\![x]\!]$.
- Addition of a constant ($z = x + c$): $x + c = x_1 + x_2 + c$, thus without loss of generality, S_1 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 $x \cdot y$. To perform this operation, we make use of Beaver triples [43]. There are multiple ways to generate Beaver triples, but in our work we use *oblivious transfer* (OT) [44], 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 [21, 44]. Beaver triples consist of three additively secret-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_0 and S_1 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: $x \cdot y = ((x-a)+a)((y-b)+b) = (d+a)(e+b) = de+db+ae+ab = de+db+ae+c$. 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 $de + d[\![b]\!] + e[\![a]\!] + [\![c]\!]$. We denote this protocol for secure multiplication as π_{mul}, i.e. to obtain $[\![x \cdot y]\!]$ from $[\![x]\!]$ and $[\![y]\!]$, the parties run $\pi_{\mathsf{mul}}([\![x]\!], [\![y]\!])$.

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 two representatives (servers) that can act as their computing parties.

3.4.2 Replicated Secret-Sharing for 3PC

The other secret-sharing scheme we use is a 3PC paradigm based on Replicated Secret-Sharing (RSS) [45], which works by splitting data into *replicated secret shares*. It is an honest-majority paradigm that guarantees protection of the data owner's inputs provided that no more than one out of the three computing parties is corrupted. Similar to Section 3.4.1, the use of this technique requires Alice and Bob to map each data value x into the ring via Equation (1), creating z. Then, to secret-share z, the data owners 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 $(z_i, z_{(i \mod 3)+1})$.

Similar to Section 3.4.1, there are several 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 $x_i, x_{(i \mod 3)+1}$ and $y_i, y_{(i \mod 3)+1}$, and let c be a constant in $\mathbb{Z}_q$, then, we have the following operations that can be performed locally:

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

$$\begin{aligned} S_1 \text{ computes} \quad & v_1 \cdot w_1 + v_1 \cdot w_2 + v_2 \cdot w_1, \\ S_2 \text{ computes} \quad & v_2 \cdot w_2 + v_2 \cdot w_3 + v_3 \cdot w_2, \\ S_3 \text{ computes} \quad & v_3 \cdot w_3 + v_3 \cdot w_1 + v_1 \cdot w_3 \end{aligned} \tag{2}$$

Performing (2) 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 S_i sends a single value to party $S_{(i \mod 3)+1}$, making this operation relatively cheap when compared to the approach in Section 3.4.1. For more information on the specifics of how this conversion takes place, we point the reader to [45].

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 parties 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 3.4.1 since we have two data owners and three 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 computations 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 runtimes for the 2PC setting. In Section 5.4, we compare the runtimes of both approaches.

3.4.3 Cryptographic Building Blocks

In Section 3.4.1 and 3.4.2, 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. For additional details regarding these protocols, we refer the reader to [46].

- π_{eq}: This protocol takes as input two secret shared values, $[\![a]\!], [\![b]\!]$, and returns $[\![1]\!]$ if $a = b$, and $[\![0]\!]$ otherwise.
- $\pi_{\mathsf{scalar\_mul}}$: This scalar multiplication protocol takes a secret shared scalar $[\![a]\!]$, and a secret shared vector $[\![\mathbf{x}]\!] = \langle [\![x_1]\!], ..., [\![x_n]\!] \rangle$ as input, and outputs $\langle [\![a \cdot x_1]\!], ..., [\![a \cdot x_n]\!] \rangle$. This protocol is a trivial extension of the multiplication protocol π_{mul} [21].
- 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.
- π_{dot}: Given two vectors $[\![\mathbf{x}]\!] = \langle [\![x_1]\!], ..., [\![x_n]\!] \rangle, [\![\mathbf{y}]\!] = \langle [\![y_1]\!], ..., [\![y_n]\!] \rangle$, the computing parties securely compute the element-wise product of each $[\![x_i]\!]$ and $[\![y_i]\!]$, and then locally take the sum of said products.
- π_{L2}: Given a vector $[\![\mathbf{x}]\!] = \langle [\![x_1]\!], ..., [\![x_n]\!] \rangle$, the parties calculate a secret sharing of the Euclidean distance from the origin (L2 norm), i.e., $[\![\sqrt{x_1^2 + ... + x_n^2}]\!]$. To calculate π_{L2}, 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.
- π_{argmax}: Given a vector $[\![\mathbf{x}]\!] = \langle [\![x_1]\!], ..., [\![x_n]\!] \rangle$, the parties perform secure 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.

With these basic building blocks, we have everything we need to construct the MPC protocols π_{pers} and π_{infer} for privacy-preserving model personalizaton and inference in Section 4.

4 Methodology

4.1 Personalization without Privacy

Lin and Marculescu [7] 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 [7] 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 parameters of the model M itself.

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: $\widetilde{\mathbf{w}}_j = \text{mean}\{\phi(x) \mid x \in X_j\}$ ▷ element-wise average of projections of X_j
5: $\mathbf{w}_j = \widetilde{\mathbf{w}}_j / \|\widetilde{\mathbf{w}}_j\|_2$ ▷ normalize vector
6: **end for**
7: $y = \text{argmax}_{j \in Y}\{\mathbf{w}_j \cdot \phi(d)\}$
8: **return** y

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 neural network 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 as one would normally do, 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 model, $\phi(x)$ 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 learned by the model. The input 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 consumes a subset D_a^* of the source dataset D_a – which was presumably used by Alice to train the model M that the feature extractor ϕ stems from – and a target dataset D_b. D_a and D_b have the same data schema, having ordered pairs (x, y) where x is sensor data indicating a user's movements, and y is an activity label denoting what those movements correspond to. The set of all possible labels occurring in D_a and D_b is 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, e.g. *walking*, *jogging*, and *standing*, then in a 5-shot setting, we collect 5 samples of data corresponding to the *walking*, *jogging*, and *standing* activity each from both data owner's 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 samples of each label. The small subset D_a^* of labeled source domain data and the small set D_b of labeled calibration data from the target domain are combined into dataset C on line 1 in Algorithm 1. Subsequently all instances

in C are grouped according to class label on line 3. For each class label j, the projections of all instances with label j are averaged out on line 4 to yield a weight vector $\widetilde{\mathbf{w}}_j$ which is prototypical for the training instances from class j, absorbing information about such instances from both the source domain and the target domain. On line 5, this weight vector is is normalized by dividing by its L2 norm, resulting in the weight vector $\mathbf{w}_j$. All weight vectors together form the *weight matrix*, W.

To infer the label for a new instance d, it is projected into the feature space as well, and the index of the vector $\widetilde{\mathbf{w}}_j$ that has the highest cosine similarity to $\phi(d)$ is returned as the inferred class label on line 7-8. Indeed, $\text{sim}(\widetilde{\mathbf{w}}_j, \phi(d)) \geq \text{sim}(\widetilde{\mathbf{w}}_{j'}, \phi(d))$ iff

$$\frac{\widetilde{\mathbf{w}}_j \cdot \phi(d)}{\|\widetilde{\mathbf{w}}_j\|_2 \, \|\phi(d)\|_2} \geq \frac{\widetilde{\mathbf{w}}_{j'} \cdot \phi(d)}{\|\widetilde{\mathbf{w}}_{j'}\|_2 \, \|\phi(d)\|_2} \tag{3}$$

in other words if $\mathbf{w}_j \cdot \phi(d) \geq \mathbf{w}_{j'} \cdot \phi(d)$.

4.2 MPC Protocols for Private Transfer Learning

Algorithm 1 is an in-the-clear algorithm that operates on plaintext, without any regards for privacy. In this section, we present MPC protocols that enable the same functionality as Algorithm 1 but in a privacy-preserving manner, i.e. without disclosing any of Alice's or Bob's data. To this end, in Section 4.2.1 we present an MPC protocol π_{pers} for model personalization that is the privacy-preserving counterpart of line 1–6 in Algorithm 1, while in Section 4.2.2 we present an MPC protocol π_{infer} to perform the equivalent of line 7–8 in Algorithm 1 in a private manner. Table 1 tracks who owns what in our MPC protocols:

- Similarly as in Section 4.1, the model owner (application developer) 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. Target user Bob contributes a small calibration dataset D_b. He also has an unlabeled instance d that needs to be classified.
- Unlike in Section 4.1, Alice and Bob do not need to disclose their data D_a, D_a^*, ϕ, D_b, and even d to anyone in an unencrypted manner. Instead, they send secret shares of their information to the computing parties (see below). After the execution of Protocol 1 (called π_{pers}), the computing parties have shares of the weight matrix W, which constitutes the personalized model for Bob. With this matrix along with Bob's unlabeled data instance d, the computing parties can execute Protocol 2 (called π_{infer}) to make an inference on what the label for d should be.

The label space Y and the hyperparameter k are considered public information, i.e. all entities know the values of Y and k.

Table 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 model 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 the model personalized for Bob. 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. Both data owners (Alice and Bob) are expected to contribute at least k instances of labeled data for each label in Y.

4.2.1 Privacy-Preserving Personalization Protocol

Pseudocode for π_{pers} is given in Protocol 1. π_{pers} requires two inputs to be secret shared with the computing parties before the execution of the protocol, namely ϕ, and D_b. D_a^* on the other hand does not need to be secret shared: since Alice owns both the feature extractor ϕ and the source data D_a^*, she can directly project all instances of D_a^* into the feature space (line 2-3) and use these projections to initialize a weight vector $\widetilde{\mathbf{w}}_j$ for each label j. Together these initialized weight vectors, containing sums of projected values for Alice's data, constitute the initialized weight matrix $\widetilde{W}$. To encrypt this matrix, Alice converts it into secret shares, which she sends to the computing parties (cfr. line 7).

Line 1–6 correspond to an "offline phase" during which Alice performs operations by herself, using only her own data. After Alice has secret-shared the result of these computations with the computing parties on line 7, an online phase starts during which the computing parties perform operations over the encrypted (secret-shared) $[\![\widetilde{W}]\!]$ (i.e. $[\![\widetilde{\mathbf{w}}_j]\!]$ for all labels j in Y), as well as the secret-shared $[\![\phi]\!]$ and $[\![D_b]\!]$ that are given as an input to the protocol. Throughout the remainder of Protocol 1 as well as in Protocol 2, all sub-protocols that require communication between the parties are for clarity denoted with a name starting with π. For example, the MPC sub-protocol that the parties execute to securely compute the dot product of two vectors is denoted as π_{dot} (see Section 3.4.3). Operations that do not require communication between parties, such as addition, are not be denoted by π.

On line 10-17, the computing parties process each instance in Bob's calibration data D_b by performing computations of the secret shared $([\![x]\!], [\![i]\!])$.

To this end, on line 11, the parties jointly and securely project each calibration instance into the feature space, which is done with π_ϕ. A suitable MPC protocol π_ϕ can be easily adapted from existing MPC protocols for privacy-preserving inference with neural networks. A variety of such MPC protocols have been proposed in the literature for secure inference with a CNN (often applied to image classification) [22, 25, 26, 38, 46, 47]. The main difference here is that image classification is based on 2-dimensional (2D) CNNs, while for our experiments in Section 5 we use a lightweight 1-dimensional (1D) CNN. The aforementioned MPC based protocols for inference with 2D CNNs can be straightforwardly adjusted to 1D CNNs (see e.g. [48]).

Protocol 1 π_{pers}: Secure Personalization Method

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

1: **start offline**: Alice
2: **for** $(x, j) \in D_a^*$ **do**
3: $\mathbf{x}_\phi = \phi(x)$
4: $\widetilde{\mathbf{w}}_j \mathrel{+}= \mathbf{x}_\phi$
5: **end for**
6: **end offline**
7: Alice secret-shares $\widetilde{W}$ with all computing parties
8:
9: **start online**: All computing parties
10: **for** $([\![x]\!], [\![i]\!])$ in $[\![D_b]\!]$ **do**
11: $[\![\mathbf{x}_\phi]\!] = \pi_\phi([\![x]\!])$ ▷ Compute secret shares of projection
12: **for** j in Y **do**
13: $[\![b]\!] = \pi_{\mathsf{eq}}([\![i]\!], j)$ ▷ Compare the i'th label to j
14: $[\![\mathbf{n}]\!] = \pi_{\mathsf{scalar\_mul}}([\![b]\!], [\![\mathbf{x}_\phi]\!])$ ▷ "zeros" out $\mathbf{x}_\phi$ if needed
15: $[\![\widetilde{\mathbf{w}}_j]\!] \mathrel{+}= [\![\mathbf{n}]\!]$
16: **end for**
17: **end for**
18: **for** j in Y **do**
19: $[\![l]\!] = \pi_{\mathsf{L2}}([\![\widetilde{\mathbf{w}}_j]\!])$
20: $[\![\mathbf{w}_j]\!] = \pi_{\mathsf{div}}([\![\widetilde{\mathbf{w}}_j]\!], [\![l]\!])$
21: **end for**
22: **end online**
23: **return** $[\![\mathbf{W}]\!]$ ▷ Collection of all weight vectors $\mathbf{w}_j$

After the projection into the feature space, the computing parties need to add the secret-shared $[\![\mathbf{x}_\phi]\!]$ to the correct secret-shared weight vector $\widetilde{\mathbf{w}}_j$. However, since the label $[\![i]\!]$ is secret-shared between the computing parties, none of the parties know which class the current calibration instance x belongs

to. On line 12–13, the parties therefore perform a secure comparison between the secret-shared label $[\![i]\!]$ and j, for each j in Y. The secure comparison protocol, π_{eq}, returns a secret sharing of 1, i.e. $[\![1]\!]$, if i is equal to j, and $[\![0]\!]$ otherwise. Thus $[\![b]\!]$ on line 13 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 14 "zeros" out the projected data x_ϕ with scalar multiplication so that it cannot contribute to the corresponding weight vector, $\widetilde{\mathbf{w}}_j$, on line 15. If equality is achieved, then the value of x_ϕ is maintained, affecting $\widetilde{\mathbf{w}}_j$.

After line 17, $\widetilde{\mathbf{w}}_j$ contains the sum of all feature representations of all instances in $C = D_a^* \cup D_b$ with the jth class label. What remains to be done is to take the average by dividing with the total number of instances with label j (namely $2 \cdot k$), and to L2 normalize the resulting vector. Since

$$\frac{\frac{1}{2k}\widetilde{\mathbf{w}}_j}{\|\frac{1}{2k}\widetilde{\mathbf{w}}_j\|_2} = \frac{\frac{1}{2k}\widetilde{\mathbf{w}}_j}{\frac{1}{2k}\|\widetilde{\mathbf{w}}_j\|_2} = \frac{\widetilde{\mathbf{w}}_j}{\|\widetilde{\mathbf{w}}_j\|_2} \tag{4}$$

it suffices for the computing parties to obtain a secret-sharing of the L2 norm of the vector $\widetilde{\mathbf{w}}_j$ by running protocol π_{L2} on line 19, and to run the secure division protocol π_{div} on line 20 to obtain the normalized vector. Finishing up the for-loop that started on line 18 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 model to a specific user using some of their labeled data as calibration data. After its execution, all computing parties are left with 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 nearest neighbor classifier based on cosine similarity.

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

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 $\mathbf{d}_\phi$. Then on line 3, for each class label, the parties take the secure dot product between $[\![\mathbf{w}_j]\!]$ and $[\![\mathbf{d}_\phi]\!]$, 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.

5 Results

We implemented our MPC protocols in MP-SPDZ [46], a benchmarking library that supports a variety of MPC schemes. We ran our tests using an additive secret-sharing scheme with 2 parties (the *semi2k-party.x* protocol in the MP-SPDZ ecosystem), and a replicated secret-sharing scheme using 3 parties (*replicated-ring-party.x* protocol). 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. [49]. These settings work for both the 3PC and 2PC approach we use.

5.1 Datasets

We performed our tests on the Sports and Daily Activity dataset (SDA) [50], and the Sensors Activity dataset (SAC) [6].

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 participant 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 3 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 participant's right arm (exact position not specified). For the SAC dataset, we used sensor data gathered from the participant's 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 all computations in our solution need to be carried out in a privacy-preserving way, it can be advantageous 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. The latter requires secure division, which can be an expensive operation in the context of MPC. We show runtime results for both normalization methods.

Each dataset that 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 is left with 5,700 datapoints (300 of each activity), each of which have 25 rows and 9 columns. Each participant in the SAC dataset is left with 1,260 datapoints (180 of each activity), each of which have 50 rows, and 12 columns.

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 [7]. 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 2 displays the architectures of our CNNs, and how the shape of the data changes from one layer to the next as a response to our hyperparameter choices.

The layers in the CNN are standard (see e.g. [51] for details). We use a convolutional layer followed by a maxpool layer. We then follow up with another

Table 2 CNN architecture details. The left table is the CNN we trained on the SDA dataset, while the right table is the CNN we trained on the SAC dataset.

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		

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 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 [52]. Everything from each model, 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

Choice of train and test data. We perform a leave-one-user-out analysis, in which the data of 1 participant is held out as the target data (Bob), while the data of the remaining participants is combined to form the source data D_a (Alice). For a benchmark dataset with n participants, we repeat this process n times (n folds), letting each participant take a turn being the target user. In each fold:

- For the source domain, we train a CNN on the source data D_a. To collect the subset of the source domain data that we use for training the personalizer, D_a^*, we randomly select from D_a, without replacement, k datapoints that correspond to each activity.
- For the target domain, we create a dataset B by randomly sampling 20% of the available instances for the target user. From B we subsequently sample, without replacement, k datapoints corresponding to each activity. We call the set of these k calibration datapoints D_b, and test the personalized model on the remaining set T. $B = D_b \cup T$ and $\emptyset = D_b \cap T$, in other

words the calibration dataset is distinct from the test dataset. We repeat this process 5 times in each fold, and take the average over all results.

Table 3 shows the accuracy results when we normalize all the data according to the distribution of the source data. Each result shows the model 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 [7] ("No Privacy"), and our MPC protocols π_{pers} and π_{infer} ("Full Privacy").

Looking at the average results across all datasets, we see that the personalization method improves accuracy, and that larger calibration datasets (10-shot vs. 1-shot learning) yield larger accuracy improvements. 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 personalization can benefit multiple datasets. For the SDA dataset, as k increases, the variance across the different users lowers. This, with the general increase in accuracy, suggests that most users benefited from higher values of k. In contrast, the variance in the SAC dataset actually increases with k. This, along with the general increase in accuracy, suggests that while many users benefit from an increase in k, it can potentially have no positive effect on accuracy for others. This can be seen from participant 1, where $k = 1$ yielded the highest accuracy. This indicates that the personalization method as we use it may in fact be sub-optimal for some users, suggesting room for improvement in future research.

The second, and most important take-away regarding the method introduced in this paper, is that HAR models can be personalized *while protecting the privacy of the users*, without sacrificing accuracy. Indeed, the accuracy results in the "Full Privacy" columns in Table 3 are at par with those in the "No Privacy" columns. The small differences in accuracy between the algorithm in-the-clear vs. the MPC protocols are to be expected. In MPC all the computations have to be decomposed in additions and multiplications. Thus, it is common to work with approximations of functions which can have some affect on the output of our model. Despite this, the accuracy between the in-the-clear models and ours is very minimal, and within the range of $\pm 0.67\%$

Table 4 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 4 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

Table 3 Accuracy results (including average and variance) 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 (as in [7])				**Full Privacy (MPC)**		
	Target		**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**
	var	**34.64**	**28.70**	**24.38**	**19.40**	**23.22**	**21.70**	**19.69**
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**
	var	**9.09**	**4.67**	**8.15**	**8.52**	**5.63**	**8.89**	**9.00**

mostly unphased by this decision, possibly making it ideal to normalize the datasets according to their distribution in the context of the SAC dataset.

Given the results of both Tables 3, 4 we see that typically speaking the more labeled data from the target and source domain we have, the more accurate we can make the personalized model. Thus, higher values of k should be perused if at all possible. Additionally, given the differences of Tables 3, 4 we see that it is typically far safer to normalize data according to the distribution of the source domain. Thus, if we can incur the additional time spent on secure training and inference, we should normalize according to the source domain in order to more reliably obtain high accuracy.

5.4.2 Runtime

Tables 5 and 6 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,`

Table 4 Accuracy results (including average and variance) 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 (as in [7])				**Full Privacy (MPC)**		
	Target		**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**
	var	**33.18**	**13.90**	**37.09**	**26.74**	**13.86**	**32.78**	**22.96**
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**
	var	**7.43**	**8.22**	**8.52**	**7.49**	**6.67**	**8.46**	**7.61**

`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, which is why the number of parameters in these tables are slightly less than what is show in Table 2. Lastly, we have the "Normalized" column. In this column, we have the field "Source" which tells us that all data was normalized by the source domain distribution, thus requiring secure division. The field "Self" tells us that data from the source and target domain was normalized each according to its own distribution in-the-clear, avoiding secure division.

As a baseline, we also performed runtime tests on the personalization method without any security. The times were nearly instantaneous, and for that reason, we only report results for $k = 10$ on both datasets, where all test data was classified sequentially. Training a personalized model took 0.11 seconds on the the SDA dataset while an average of 0.0018 seconds was needed for classifying instances. For the SAC dataset, we obtained 0.07 seconds for training runtime, and an average of 0.0019 seconds for classifying an instance. Our lightweight classifier hardly takes any time at all to train or classify when no MPC is in play. Below we discuss Table 5 and 6 which show how long our approach takes when we apply MPC techniques to it.

Table 5 shows our runtimes using the additive sharing 2PC approach. Times for π_{pers} are fairly efficient, scaling linearly with 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 has 19 activities, while SAC has 7, hence for $k = 10$, we use $10 \cdot 19 \cdot 2 = 380$ samples to train π_{pers} on SDA, and only $10 \cdot 7 \cdot 2 = 140$ 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. We note that as long as π_{pers} does not take an egregious amount of time to execute, its runtimes are not that important. This is because a personalized model only needs to be trained once (or once in a while), and training could be performed when the target user is asleep, for example, not affecting their daily routine at all.

Table 5 Runtimes for the 2PC additive secret sharing scheme from Section 3.4.1. #Params tell us how large the feature extractor was. Column π_{pers} shows how long it takes to train a classifier in a 1,5, and 10-shot setting. Column π_{infer} shows how long, on average, it takes to infer data sequentially and in batches of 15.

Runtime Results for Additive Secret-Sharing Approach (2PC)							
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 6 Runtimes for the 3PC replicated secret sharing scheme from section 3.4.2 #Params tell us how large the feature extractor was. Column π_{pers} shows how long it takes to train a classifier in a 1,5, and 10-shot setting. Column π_{infer} shows how long, on average, it takes to infer data sequentially and in batches of 15.

Runtime Results for Replicated Secret-Sharing Approach (3PC)							
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

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 has 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. 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 domain 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 6 shows our runtimes using the replicated sharing (RSS) 3PC approach. As we can see, these runtimes are significantly better than those of Table 5, because of the difference in security setting: the 3PC approach provides security in a setting with 3 computing parties, out of which at most one is corrupted (honest majority) while for the 2PC approach only 2 computing parties are needed, one of which can be corrupted (dishonest majority). Runtimes for π_{pers} in Table 6 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, 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 scheme 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.

5.4.3 Security

The security of our protocols follows immediately from the fact we only compute on secret-shared values and from the security and composability of the protocols for addition and multiplication presented in the MP-SPDZ framework [46].

6 Conclusion

We presented, to the best of our knowledge, the first Secure Multi-Party Computation (MPC) solution for privacy-preserving personalization of Human Activity Recognition (HAR) models from wearable sensor data. While algorithms to improve accuracy in HAR through transfer learning have been proposed before, existing methods assumed that the target user discloses their calibration data to the HAR application developer. Our MPC protocols on the other hand enable model calibration and inference without leaking any sensitive information from the end-user (target domain) to the model owner (source domain), without leaking the source domain information in the clear to the end-user, and without sacrificing accuracy.

The price paid to obtain privacy while preserving accuracy is a substantial increase in runtime compared to when computations are performed in plaintext, i.e. without encryption. Reducing the computational and communication cost incurred by the computing parties who execute the cryptographic protocols is an important challenge that can be tackled from two main research directions: through optimizations of a cryptographic nature and through deliberate choice of MPC-friendly machine learning techniques. In this paper we have focused on the latter with a lightweight personalization technique that lended itself well to develop very efficient MPC protocols. We have implemented our MPC protocols for model personalization and inference in an off-the-shelf MPC framework (MP-SPDZ). We have demonstrated on benchmark datasets how our implementation can be used to personalize 1D CNNs for HAR and for real time classification, without sacrificing privacy nor accuracy.

The MPC protocols that we presented are general: they only assume the availability of some data from the source domain in which the model was trained, some calibration data from the target domain, and a feature vector (i.e. representation) extractor trained in the source domain. This means that our MPC protocols can also be used for 2D CNNs, LSTMs, RNNs, or any other neural network architecture. Indeed, plugging in a different kind of neural network architecture in our MPC solution only requires replacing the MPC sub-protocol π_ϕ used on line 11 in protocol π_{pers} and on line 1 in protocol π_{infer}. As confirmed by our experiments, the runtime of the MPC protocols is influenced by the size of the feature extractors (i.e. the number of model parameters), which means that designing small and accurate neural networks for privacy-preserving HAR – as we have done for 1D CNN – is a relevant challenge in itself.

Another direction for future work that will almost certainly yield accuracy improvements in privacy-preserving HAR is the incorporation of data preprocessing and transformation techniques, including the design of MPC protocols as needed to preserve privacy during the preprocessing stage. As we noted, a simple MPC protocol (secure division) for normalization based on the data distributions from both source and target domain resulted in an accuracy improvement in our experiments. Similarly, one could design MPC protocols for outlier detection and removal or for extraction of statistical features (such

as mean and variance of each input data channel) to augment the features extracted by the neural network. We hope that our work provides a benchmark for privacy in model calibration for HAR, and inspires improvements in accuracy and efficiency in future work.

Acknowledgments. The authors would like to thank Microsoft for the generous donation of cloud computing credits through the UW Azure Cloud Computing Credits for Research program.

References

[1] Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering **22**(10), 1345–1359 (2009)

[2] Yang, Q., Zhang, Y., Dai, W., Pan, S.J.: Transfer Learning. Cambridge University Press, United Kingdom (2020)

[3] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., He, Q.: A comprehensive survey on transfer learning. Proceedings of the IEEE **109**(1), 43–76 (2021)

[4] Hernandez, N., Lundström, J., Favela, J., McChesney, I., Arnrich, B.: Literature review on transfer learning for human activity recognition using mobile and wearable devices with environmental technology. Springer Nature Computer Science **1**(66) (2020)

[5] Mehrang, S., Pietila, J., Tolonen, J., Helander, E., Jimison, H., Pavel, M., Korhonen, I.: 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), pp. 587–590 (2017)

[6] Shoaib, M., Bosch, S., Incel, O., Scholten, H., Havinga, P.: Fusion of smartphone motion sensors for physical activity recognition. Sensors **14**, 10146–10176 (2014)

[7] Lin, C.-Y., Marculescu, R.: Model personalization for human activity recognition. In: 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops) (2020)

[8] Lin, Y.-P., Jung, T.-P.: Improving EEG-based emotion classification using conditional transfer learning. Frontiers in Human Neuroscience **11**, 334 (2017)

[9] Wu, D., Xu, Y., Lu, B.-L.: Transfer learning for EEG-based brain-computer interfaces: A review of progress made since 2016. IEEE

Transactions on Cognitive and Developmental Systems, 4–19 (2020)

[10] Carlini, N., Liu, C., Erlingsson, Ú., Kos, J., Song, D.: The secret sharer: Evaluating and testing unintended memorization in neural networks. In: 28th USENIX Security Symposium, pp. 267–284 (2019)

[11] Balli, S., Sağbaş, E.A., Peker, M.: 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)

[12] Cramer, R., Damgard, I., Nielsen, J.: Secure Multiparty Computation and Secret Sharing. Cambridge University Press Print, New York (2015)

[13] Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R.G.L., Eichner, H., Rouayheb, S.E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P.B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konecný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Qi, H., Ramage, D., Raskar, R., Raykova, M., Song, D., Song, W., Stich, S.U., Sun, Z., Suresh, A.T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F.X., Yu, H., Zhao, S.: Advances and open problems in federated learning. Foundations and Trends in Machine Learning **14**(1–2), 1–210 (2021)

[14] Presotto, R., Civitarese, G., Bettini, C.: FedCLAR: federated clustering for personalized sensor-based human activity recognition. In: 2022 IEEE International Conference on Pervasive Computing and Communications (PerCom), pp. 227–236 (2022)

[15] Chen, Y., Qin, X., Wang, J., Yu, C., Gao, W.: FedHealth: a federated transfer learning framework for wearable healthcare. IEEE Intelligent Systems **35**(4), 83–93 (2020)

[16] Hu, R., Guo, Y., Li, H., Pei, Q., Gong, Y.: Personalized federated learning with differential privacy. IEEE Internet of Things Journal **7**(10), 9530–9539 (2020)

[17] Liu, S., Wang, J., Zhang, W.: Federated personalized random forest for human activity recognition. Mathematical Biosciences and Engineering **19**, 953–971 (2021)

[18] Dwork, C., Roth, A.: The algorithmic foundations of differential privacy. Foundations and Trends in Theoretical Computer Science **9**(3-4), 211–407 (2014)

[19] Hashemian, M., Razzazi, F., Zarrabi, H., Moin, M.S.: A privacy-preserving distributed transfer learning in activity recognition. Telecommunication Systems: Modelling, Analysis, Design and Management **72**(1), 69–79 (2019)

[20] Hashemian, M., Razzazi, F., Zarrabi, H., Moin, M.: Semi-supervised and unsupervised privacy-preserving distributed transfer learning approach in HAR systems. Wireless Personal Communications **117**, 1–18 (2021)

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

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

[23] De Cock, M., Dowsley, R., Horst, C., Katti, R., Nascimento, A., Poon, W.-S., Truex, S.: 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)

[24] Fritchman, K., Saminathan, K., Dowsley, R., Hughes, T., De Cock, M., Nascimento, A., Teredesai, A.: Privacy-preserving scoring of tree ensembles: A novel framework for AI in healthcare. In: Proceedings of 2018 IEEE BigData, pp. 2412–2421 (2018)

[25] Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A low latency framework for secure neural network inference. In: 27th USENIX Security Symposium, pp. 1651–1669 (2018)

[26] Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: CrypTFlow: Secure TensorFlow inference. In: 41st IEEE Symposium on Security and Privacy, pp. 336–353 (2020)

[27] Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via MiniONN transformations. In: ACM SIGSAC Conference on Computer and Communications Security, pp. 619–631 (2017)

[28] Reich, D., Todoki, A., Dowsley, R., De Cock, M., Nascimento, A.: Privacy-preserving classification of personal text messages with secure multi-party computation. In: Advances in Neural Information Processing Systems (NeurIPS), vol. 32, pp. 3752–3764 (2019)

[29] Riazi, S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: A hybrid secure computation framework for

machine learning applications. In: Asia Conference on Computer and Communications Security, pp. 707–721 (2018). ACM

[30] Rouhani, B.D., Riazi, M.S., Koushanfar, F.: DeepSecure: Scalable provably-secure deep learning. In: 55th Annual Design Automation Conference (DAC) (2018)

[31] Agarwal, A., Dowsley, R., McKinney, N.D., Wu, D., Lin, C.-T., De Cock, M., Nascimento, A.: Protecting privacy of users in brain-computer interface applications. IEEE Transactions on Neural Systems and Rehabilitation Engineering **27**(8), 1546–1555 (2019)

[32] De Cock, M., Dowsley, R., Nascimento, A.C.A., Newman, S.C.: 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, pp. 3–14 (2015)

[33] Nikolaenko, V., Weinsberg, U., Ioannidis, S., Joye, M., Boneh, D., Taft, N.: Privacy-preserving ridge regression on hundreds of millions of records. In: 2013 IEEE Symposium on Security and Privacy (SP), pp. 334–348 (2013)

[34] Abspoel, M., Escudero, D., Volgushev, N.: Secure training of decision trees with continuous attributes. Proceedings on Privacy Enhancing Technologies **2021**(1), 167–187 (2021)

[35] de Hoogh, S., Schoenmakers, B., Chen, P., op den Akker, H.: Practical secure decision tree learning in a teletreatment application. In: International Conference on Financial Cryptography and Data Security, pp. 179–194 (2014)

[36] Lindell, Y., Pinkas, B.: Privacy preserving data mining. In: Annual International Cryptology Conference, pp. 36–54 (2000)

[37] Adams, S., Choudhary, C., De Cock, M., Dowsley, R., Melanson, D., Nascimento, A.C., Railsback, D., Shen, J.: Privacy-preserving training of tree ensembles over continuous data. Proceedings on Privacy Enhancing Technologies (2), 205–226 (2022)

[38] Agrawal, N., Shahin Shamsabadi, A., Kusner, M.J., Gascón, A.: QUOTIENT: two-party secure neural network training and prediction. In: ACM SIGSAC Conference on Computer and Communications Security, pp. 1231–1247 (2019)

[39] De Cock, M., Dowsley, R., Nascimento, A.C.A., Railsback, D., Shen, J., Todoki, A.: High performance logistic regression for privacy-preserving genome analysis. BMC Medical Genomics **14**(1), 23 (2021)

[40] Guo, C., Hannun, A., Knott, B., van der Maaten, L., Tygert, M., Zhu, R.: Secure multiparty computations in floating-point arithmetic. Information and Inference: A Journal of the IMA **11**(1), 103–135 (2021)

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

[42] Wagh, S., Gupta, D., Chandran, N.: SecureNN: 3-party secure computation for neural network training. Proceedings on Privacy Enhancing Technologies **2019**(3), 26–49 (2019)

[43] Beaver, D.: Commodity-based cryptography (extended abstract). In: Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing. STOC '97, pp. 446–455 (1997)

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

[45] Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: ACM SIGSAC Conference on Computer and Communications Security, pp. 805–817 (2016)

[46] Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 1575–1590 (2020)

[47] Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: A cryptographic inference service for neural networks. In: 29th USENIX Security Symposium, pp. 2505–2522 (2020)

[48] Adams, S., Melanson, D., De Cock, M.: Private text classification with convolutional neural networks. In: Proceedings of the Third Workshop on Privacy in Natural Language Processing (NAACL Workshops), pp. 53–58 (2021)

[49] Escudero, D., Ghosh, S., Keller, M., Rachuri, R., Scholl, P.: Improved primitives for MPC over mixed arithmetic-binary circuits. In: Annual International Cryptology Conference, pp. 823–852 (2020)

[50] Barshan, B., Yüksek, M.C.: 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)

[51] Zhang, A., Lipton, Z.C., Li, M., Smola, A.J.: Dive Into Deep Learning, (2022). https://d2l.ai

[52] Chollet, F., et al.: Keras. https://github.com/fchollet/keras