Towards Private Biometric Authentication and Identification

by

Jonathan Gold

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2023

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

I am the sole author of Chapters 1, 3, 4 and 6, written under the supervision of Dr. Koray Karabina and Dr. Alfred Menezes. Chapter 2 is based on a paper "Improving Accuracy and Explainability of Online Handwriting Recognition" by Hilda Azimi, Steven Chang, myself and Koray Karabina. Chapter 5 is based on a graduate class project, conducted by myself along with Anuradha Kulkarni and Vijay Ravi.

Abstract

Handwriting and speech are important parts of our everyday lives. Handwriting recognition is the task that allows the recognizing of written text, whether it be letters, words or equations, from given data. When analyzing handwriting, we can analyze static images or the recording of written text through sensors. Handwriting recognition algorithms can be used in many applications, including signature verification, electronic document processing, as well as e-security and e-health related tasks.

The OnHW datasets consists of a set of datasets which, through the use of various sensors, captures the writing of characters, words, symbols and equations, recorded in the form of multivariate time series. We begin by developing character recognition models, targeting letters (and later symbols), trained and tested using the OnHW-chars dataset (and later the split OnHW-equations dataset). Our models were able to improve upon the accuracy of the previous best results on both datasets explored. Using our machine learning (ML) models, we provide 11.3%-23.56% improvements over the previous best ML models. Using deep learning (DL), as well as ensemble techniques, we were able to improve on the best previous models by 3.08%-7.01%. In addition to the accuracy improvements, we aim to provide some level of explainability, using a specialized version of LIME for time series data. This explanation helps provide some rationale for why the models make sense for the data, as well as why ensemble methods may be useful to improve accuracy rates for this task. To verify the robustness of our models trained over the OnHW-chars dataset, we trained our DL models using the same model parameters over a more recently published OnHW-equations dataset. Our DL models with ensemble learning provide 0.05%-4.75% improvements over the previous best DL models.

While the character recognition task has many applications, when using it to provide a service, it is important to consider user privacy since handwriting is biometric data and contains private information. Next, we design a framework that uses multiparty computation (MPC) to provide users with privacy over their handwritten data, when providing a service for character recognition. We then implement the framework using the models trained on public data to provide private inference on hidden user data. This framework is implemented in the CrypTen MPC framework. We obtain results on the accuracy difference of the models when making inference using MPC, as well as the costs associated with performing this inference. We found a $0.55\% - 1.42\%$ accuracy difference between plaintext inference and inference with MPC.

Next, we pivot to explore writer identification, which involves identifying the writer of some handwritten text. We use the OnHW-equations dataset for our analysis, which at

the time of writing has not been used for this task before. We first analyze and reformat the data to fit the writer identification task, as well as remove bias. Using DL models, we obtain accuracy results of up to 91.57% in identifying the writer using their handwriting. As with private inference in the character recognition task, it is important to account for user privacy when training writer identification models and making inference. We design and implement a framework for private training and inference for the writer recognition task, using the CrypTen MPC framework. Since training these models is very costly, we use simpler CNN's for private writer recognition. The chosen CNN trained privately in MPC obtained an accuracy of 77.45%. Next, we analyze the costs associated with privately training the CNN and other CNN's with altered model architectures.

Finally, we switch to explore voice as a biometric in the speaker verification task. As with handwriting, a person's voice contains unique characteristics which can be used to determine the speaker. Not only can voice be analyzed similarly with handwriting, in that we can explore the speech recognition and speaker identification tasks, it comes with similar privacy risks for users. We design and implement a unique framework for private speaker verification using the MP-SPDZ MPC framework. We analyze the costs associated with training the model and making inferences, with our main goal being to determine the time it takes to make private inference. We then used these times as part of a survey conducted to determine how much people value the privacy of their biometrics and how long they were willing to wait for the increased privacy. We found that people were willing to tolerate significant time delays in order to privately authenticate themselves, when primed with the benefits of using MPC for privacy.

Acknowledgements

The research presented in this thesis was supported by NSERC and National Research Council of Canada's Ideation Fund, New Beginnings Initiative.

I would like to express my sincere appreciation to my supervisors Dr. Alfred Menezes and Dr. Koray Karabina for their invaluable support throughout the past two years. It was a privilege to work under their direction. I am grateful to Alfred for his guidance and wise council, which I will take with me beyond my degree. Our meetings were always interesting and inspiring. I am grateful to Koray for his dedication to my learning, his mentorship and for putting his trust in me. I appreciate his time and energy in making me feel welcome in Waterloo and creating a strong bond with our research group. Many thanks to readers Dr. Douglas Stebila and Dr. David Jao, for taking their time to read and provide feedback on my thesis.

I thank my friends for always being there for me and providing a distraction when they knew I needed one.

Finally, thanks to my parents and my sister for their constant commitment to my success and their unconditional patience and love.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work will explore biometric analysis and authentication and the desire and feasibility of making these frameworks private. We will explore handwriting and speech analysis and how incorporating a privacy preserving component, through the use of secure multiparty computation, can and should be used with biometric analyses. From a handwriting perspective, in Chapter 2, we will explore the character recognition task and how we can use advanced machine and deep learning algorithms to determine a written character. In Chapter 3, we will analyze why we would want to add a privacy component to character recognition and how to incorporate multiparty computation to increase user privacy. In Chapter 4, we switch our focus from character recognition to writer identification. We will explore how writer identification can be used for authentication purposes, and how the training of these models can be accomplished privately using multiparty computation. Finally, in Chapter 5 we will briefly explore speaker authentication models and the potential for using multiparty computation with user specific speaker authentication models.

1.1 Online Character Recognition

Before discussing specific handwriting recognition tasks such as online character recognition, we must first discuss handwriting analysis generally. We define *handwriting* as the writing characteristics of an individual [119, 14].

Handwriting analysis is classified into one of two categories based on the collection processes to obtain the data. Generally, handwritten identification or recognition research can be classified as either *online* or *offline*. Offline handwriting refers to a scanned static

image of some handwriting [125]. Typically, a writer will write a piece of text (character, sentence, paragraph, etc.), before the written surface is scanned. The goal here is to analyze the spatial attributes of the image, to make out characters, paragraphs or general text. Online handwriting is captured at the time of writing by some sensor. This can be known as the dynamic method [125]. The sensor can be in the writer's pen [110], or in a device such as a tablet or digitizer [82, 88], as it tracks how the writing instrument moves through time and space [153, 148]. Online handwriting can expect better performance in recognition or identification tasks due to the increased information that is able to be acquired, such as position, velocity, pressure, altitude, angle, cadence or other unique stylizations [148, 153, 125, 14]. While online analysis does expect better results, it is important to note that there is still room for more exploration into offline methods as it can be more useful (and only possible) in certain instances, such as when analyzing already written text. For the rest of this thesis, when we explore handwriting data, we will be looking at online handwritten information.

The task of *handwritten character recognition* is the task of assigning a writing sample (of a single character) into a symbol class [119]. Applications for online handwriting recognition can include signature verification [50] and e-security and e-health related tasks [51].

Datasets There are many datasets available for handwritten character recognition in the online and offline settings. In the offline settings, some datasets include NIST [146], MNIST [83], EMNIST [37], CEDAR [67], IAM [91], HWDB [133], CASIA [86], IBM-UB-1 [132]. For the online setting, some datasets include OnHW [110], UNIPEN [61], PenDigits [8], LaViola [82], UJIpenchars [89], IAM-OnDB [88], IME-OnDB [21], CROHME 2014 [98], VOHTR2018 [101], IBM-UB-1 [132].

1.2 Online Writer Identification

Writer identification is the task of identifying a specific writer from a group of writers, using their handwriting [31]. This task can be seen as a "one-to-many" comparison of writing samples to determine the likeliest single writer out of a group of known writers [23].

Many available datasets and writer identification analyses are performed on data using $[x, y]$ coordinates (and potentially a pressure sensor) of the pentip (or the user's finger) as it writes [153, 148, 88]. For much of the analysis in the remainder of the thesis, we will be using the OnHW datasets collected with the DigiPen [110]. This data consists of four different sensors measuring $[x, y, z]$ coordinates as well as a one-dimensional force sensor.

This should yield much more information than just the $[x, y]$ coordinates of the pentip. At the time of writing, we are not aware of these datasets being used for writer identification.

Writer identification is further divided into the *text independent* and *text dependent* [6, 129] categorizations of the task. The text dependent approach requires the writer to write identical text to be able to identify, whereas the text independent approach allows the writer to write any text [6, 148, 100]. The text-dependent approach would be similar to a signature verification technique, where a writer would have to repeat the same word or phrase as part of the verification process. On the user's end, text-independent approaches would likely be preferable as there is generally less need for human intervention [23]. Theoretically, a text-independent model could be obtained by tracking a user writing as they normally would, whether that be tracking the writing of a paragraph or an essay or something of a similar style. On the other hand, for text-dependent writer recognition, the user would be required to write the same text repeatedly until there are enough samples to generate a model. This would likely need some sort of specialized enrollment process for the user into the system, making text-dependent solutions inapplicable in many settings [64]. Generally however, text-dependent solutions obtain better identification results [64]. Both text-dependent and text-independent solutions can be explored in the online setting as was done in [63], [7], [56] in the offline setting, albeit much more uncommon as [7] states that no text-dependent online dataset was found in the literature prior to their work. In the offline setting, there is extensive exploration into both text-dependent approaches as in [128] and text-independent approaches as in [136].

Like writer identification, *writer verification* is another task to determine writers, which can be used in an authentication framework. It involves a "one-to-one" comparison between two samples to decide whether or not they were written by the same person [23]. Along with verification, *closed-set* and *open-set* identification tasks can be used in biometric authentication frameworks [28]. Identification as described above, would fall into the closed-set identification task, where there is a closed set of writers and we are trying to identify which writer a given sample belongs to. The final task is open-set identification, which involves determining if a writing sample was written by someone belonging to the set of individuals, or not. This is not commonly used with handwriting biometrics, but can be used with other types of biometrics, such as speaker recognition, which we see next.

1.3 Speaker Recognition

Speaker recognition is the task of identifying someone based on their voice [77, 28]. Speaker recognition systems depend on individuals having both physical differences in their vocal

tract and other "voice producing organs", as well as non-physical characteristics such as accent, rhythm and intonation [77].

Speaker recognition can be viewed as similar to the handwriter recognition task in that we are trying to identify or authenticate a person through their biometric. Additionally, like in handwriter recognition, it is important to distinguish between text-independent and text-dependent speaker recognition. Text-dependent systems involve the user repeating a specific passphrase or prompted phrase. This only requires short utterances, but will require significant amounts of them in order to achieve high accuracy [28]. It is suggested in [65] that one application would be to have a user repeat the digits from $1 - 10$ in the enrollment phase, before being prompted with a random sequence of these digits in the testing phase. Text-independent speaker recognition systems require much longer utterances in comparison, however according to [28] it is much more versatile and practical in applications. Beigi [18] claims that there are not many practical applications of text-dependent solutions (even for speaker verification) and goes as far as to call it "comedic to ask someone to say a specific phrase to be able to identify him/her". In [134], the authors claim that text-independent speaker recognition systems are more commercially practical.

Speaker recognition is often divided into the verification task and the identification task. The verification task is to determine whether or not the speaker of an audio sample belongs to the person they are claiming to be [134]. According to [28], this is the only task appropriate for text-dependent solutions. We can divide text-dependent solutions into text-prompted or knowledge based solutions [18], however in most cases text-independent solutions are preferable. The identification task is further split into a closed-set identification task and an open-set identification task, as was described in Section 1.2. The open-set identification task involves determining whether an input voice is uttered by a speaker in a set of users (one of the enrolled users), or not. Closed-set identification involves determining which speaker in the set of enrolled users, is the speaker of an utterance. In closed-set identification, we assume the utterance in question was spoken by someone in the set.

Some applications of speaker recognition include authentication, forensics, surveillance, security, multi-speaker tracking and personalized user interfaces [134]. As with handwriter identification in Section 1.2, we will primarily use the speaker identification task in Chapter 5 in a biometric authentication scheme.

1.4 Multiparty Computation

Multiparty Computation (MPC) is an important cryptographic problem [45], which involves enabling multiple parties to securely perform distributed computing tasks [85]. The

inputs and outputs of these tasks should remain private to the computing parties. This is accomplished with the help of secret shares, which are to be distributed as inputs amongst the computing parties. MPC assumes that n parties each have private inputs x_i, where they would like to compute some function $f(x_1, ..., x_n)$, and all obtain the output, without other parties learning the inputs [45].

Three major properties which MPC protocols require are *privacy*, *correctness* and *independence of inputs* [85, 13]. Privacy implies that computing parties should not learn anything beyond what is necessary for computation [85]. Lindell states that this means that computing parties should only learn the output of the computation process and nothing more [85]. During the computation phase of the MPC protocol, the parties will (typically) be able to perform addition, multiplication, comparison, and other operations on the private data. The parties will be able to run a circuit for f, where the outcome remains private until the parties come together to reveal their outputs in the end.

In the relevant chapters and sections below, we will define what privacy means in terms of the specific tasks being performed and how to achieve these goals.

1.4.1 Adversarial Powers

Through MPC, our privacy goals stem from keeping the inputs private in the face of corrupted parties. The strength of privacy obtained by the scheme is dependent on the strength of the adversaries corrupting the computing parties, as well as the number of computing parties that are corruptible.

We call an adversary *semi-honest* (or *honest but curious*) if the corrupted parties will follow the provided protocol, however the adversary might attempt to reconstruct the inputs (or intermediary information) by combining all state information including messages sent/received and all calculations of all corrupted parties [85]. A *malicious* adversary has all of the powers of a semi-honest adversary, but the corrupted parties can also arbitrarily deviate from the intended protocol [85]. We can think of malicious adversaries as being able to act as they please and are not obligated to follow any instruction [13]. MPC schemes protecting against malicious adversaries provide a very strong level of security for the honest parties. Historically, these were the two main adversarial types until *covert* adversaries were introduced to provide an intermediate level [127] of security, in [13]. According to [13, 127, 85], covert adversaries can act maliciously, but with some probability of getting caught. This probability ϵ will be known as a deterrence factor [13]. The idea here is that the consequences for cheating (or being caught cheating), will outweigh the benefits of cheating itself.

In addition to how adversaries can act during the run of a protocol, the strength of security is also dependent on how and when adversaries are able to corrupt computing parties. Adversaries can act *statically*, *adaptively*, or *proactively* [85]. Static adversaries assume that the parties corrupted by the adversary are fixed before the protocol begins. This assumes that honest parties remain honest, and corrupt parties remain corrupt, throughout the run of the protocol [85]. Adaptive adversaries can decide which parties to corrupt and when to corrupt them. In this corruption strategy, parties who become corrupted are assumed to remain corrupted for the remainder of the protocol. Finally, proactive corruption models assume that parties can be corrupted after the protocol has begun, like in the adaptive model, however parties can subsequently become honest again. We note that the adaptive model is a special case of the proactive model where the parties do not become honest again. The proactive (and adaptive) corruption models aim to model an external hacker type of adversary, where one (or more) computing parties are subjected to the hack.

The strength of the MPC protocol is also dependent on the number of parties that the adversary can corrupt. Many protocols consider the honest majority case, where more than half of the computing parties are guaranteed to be honest. Additionally, there are protocols on the other end of the spectrum, that require only one party to remain honest for the protocol to be secure. When discussing the MPC protocols used in the remainder of the thesis, we will specify which of these security properties are met.

One additional point to note about these properties is that the stronger properties — malicious adaptive adversaries in the dishonest majority case — will be preferred from a security point of view. However, this comes at the expense of significant computational, time, and communication costs [13]. This trade-off must be explored when choosing an MPC protocol.

1.4.2 Client/Server Model

The client/server model utilizes a group of servers that can collectively act as a trusted third party for the users (clients) [41]. In this setup, the servers will act as the computing parties in place of the data holder. For authentication approaches, it can be useful for the client and the server it is trying to authenticate itself to, to serve as the two computing parties in an MPC protocol [22]. However, this method requires the client to perform computation, which in many cases (especially when using MPC approaches) can be costly [33, 38, 42, 43]. It can be particularly fitting to adopt the client/server approach when it is not feasible for the clients to perform the computation and instead leave the computation to a set of servers [38]. In this model, we can have three sets of participants: the *input*

parties (who provide the data), *computation parties* (generally servers who perform the computation), and *output parties* (who receive outputs) [42]. The computation proceeds in three stages. The input stage is where the clients divide their data into shares and send them to the relevant servers. The servers then compute the relevant function in the computation stage before returning the output share of their computation. The output shares are then combined by the output parties to determine the final output value.

1.5 CrypTen

Now that we have discussed what MPC is and some basic properties surrounding its use, we will look at CrypTen [79, 80], a python package designed to help developers use MPC in their projects. CrypTen is built to mimic the PyTorch [115] API to make it easier and more accessible for machine learning (ML) developers to have access to secure computing. PyTorch has become common in the deep learning community due to its ease of use and efficient implementation. According to [103], PyTorch along with TensorFlow [2], are the two top libraries in use by the artificial intelligence community.

After the "Machine-Learning first API", the other main design principle of CrypTen is "Eager Execution" [80]. The authors claim that they avoided implementing compilers for their own domain specific language, as seen in other cryptographic ML packages, like MP-SPDZ [71] which will be explored in Chapter 5. From my personal experience, I believe this helps make CrypTen much easier to use and debug compared to MP-SPDZ. While MP-SPDZ does well to provide a framework and compiler that closely resembles python, it can still feel quite restrictive and difficult to debug due to it passing through this extra compiler. One of the main reasons for these changes in CrypTen was for improved developer experience, and I believe this was done well.

Design CrypTen also can use GPU's to help perform computation, which is not always common amongst the most popular MPC packages. CrypTen makes use of CUDA libraries (cuBLAS and cuDNN). This is not trivial due to these packages not providing integer support. For integer multiplication of $a \cdot b$ for example, CrypTen makes a floating point representation of the 64 bit integers, by splitting the 64 bits into four 16-bit components before summing the pairwise products.

By default, CrypTen will assume that all participating parties are involved in the computation. This means that natively, CrypTen does not utilize the client/server model for computation as described in Section 1.4.2. However, it is possible (see CrypTen 2pc demo

and CrypTen GitHub issues request) to create a framework where not all parties participate in computation. I believe that this kind of framework workaround could allow for a client/server model approach. Additionally, the secret sharing and private computation processes used in CrypTen should allow for a client/server model approach. There is also some discussion in the issues section in GitHub (CrypTen GitHub issue on client/server model) that they may want to add this functionality at some point, however it does not appear to be implemented yet.

The secret sharing procedure in CrypTen will be described in greater detail in Section 1.5.1 and the procedure for implementing secure computation will be explored in Section 1.5.2. Finally, we will discuss the security of CrypTen's MPC secret sharing and computation protocols in Section 1.5.3.

1.5.1 Secret Sharing

CrypTen's MPC protocol makes use of both arithmetic and binary secret sharing. Certain ML functions are easier to compute on arithmetic secret shares while others are easier on binary secret shares [79]. According to [95], it may be advantageous to use arithmetic secret sharing when performing functions involving, or that can be translated to, modular addition and multiplication. Otherwise, it may be preferable to use binary secret sharing. For this reason, CrypTen makes use of both types of secret sharing. Below we will discuss both of these methods as well as the conversion between them.

Arithmetic Secret Shares

We denote P to be the set of parties involved in the protocol. Arithmetic secret sharing of a value $x \in \mathbb{Z}/Q\mathbb{Z}$ involves dividing x into shares $[x]_p$ such that:

$$x = \sum_{p \in P} [x]_p \mod Q.$$

Note that all $[x]_p$ are selected uniformly at random, except for the last one ($[x]_{|P|}$) which is set to the sum of all other shares, subtracted from x (alternatively $x - \sum_{p=1}^{|P|-1} [x]_p \mod Q$).

Each party $p \in P$ will hold a share $[x]_p$. We will define $[x]$ to be the ordered sequence of all $[x]_p$

$$[x] = \{[x]_p\}_{p \in P}.$$

Arithmetic secret sharing is used in many MPC protocols including [95, 12, 47].

Binary Secret Shares

The CrypTen protocol uses binary secret sharing as a special case of arithmetic secret sharing in the field $\mathbb{Z}/2\mathbb{Z}$. We denote a binary secret share of $x \in \mathbb{Z}/Q\mathbb{Z}$ belonging to party p as $\langle x \rangle_p$. We define $\vec{x}$ to be the binary representation of x. Note that each $\langle x \rangle_p$ is a bit-string (represented as an integer). Additionally, note that we use the symbol $\oplus$ to denote the bitwise XOR operation. The binary shares of x are created so that

$$\vec{x} = \oplus_{p \in P} \langle x \rangle_p.$$

As before, with arithmetic secret shares, we define $\langle x \rangle$ to be the ordered sequence of all $\langle x \rangle_p$

$$\langle x \rangle = \{\langle x \rangle_p\}_{p \in P}.$$

CrypTen notes that AND gates are particularly inefficient when performed in MPC, which makes binary secret sharing infeasible except when making comparisons [80]. In [95], the authors note that binary secret sharing (or Yao secret sharing, which is not used in CrypTen) can be more efficient when computing non-arithmetic functions such as ReLU or approximate activation functions, both of which are commonly used in deep learning computation. For this reason, binary secret sharing will be used for comparisons, where we see a conversion to binary where some computation is performed, before a conversion back to arithmetic as described in greater detail in Section 1.5.2. These comparisons and thus binary secret sharing are also used in other non-arithmetic functions (such as ReLU). We see that overall, throughout most computation, the hidden data will spend most of its time in arithmetic secret share form.

Arithmetic to Binary Secret Share Conversion

Due to the use of both arithmetic and binary secret shares in the CrypTen protocol, there must be a way to convert between these two formats. To perform this conversion, each party will divide their arithmetic secret share $[x]_p$ into $|P|$ binary secret shares

$$\langle [x]_p \rangle = \{\langle [x]_p] \rangle_i\}_{i \in P},$$

and send these binary shares to the other parties. Now, note that each party has a binary secret share of every other parties arithmetic share. For example, party i will now hold $\langle [x]_p \rangle_i$ for each $p \in P$. The parties then add each of these binary secret shares using a carry-lookahead adder [44]. The parties compute $\langle x \rangle = \sum_{p \in P} \langle [x_p] \rangle$ using the carry-lookahead adder.

This additional communication adds up when performing many computations on a lot of data. For this reason, it is beneficial to limit the number of conversions, except of course when the alternative method is more costly.

Binary to Arithmetic Secret Share Conversion

In order to convert from binary secret shares $\langle x \rangle$ to arithmetic secret shares $[x]$, we must first compute $[\langle x \rangle^b]$ for each bit b in $\langle x \rangle$, where $\langle x \rangle^b$ refers to the "b'th" bit of the bitstring. To assist with this computation, each party will obtain a pair of shares of a random bit $r \in \{0, 1\}$ (for each bit b being converted) from a trusted third party. The pair of shares received are $(\langle r \rangle, [r])$, so each party will receive both an arithmetic secret share and a binary secret share of r.

Next, the following steps are performed:

1. Compute $\langle z \rangle = \langle r \rangle \oplus \langle x \rangle^b$ in order to mask $\langle x \rangle^b$.
2. Next, reveal z. Since z is shared between the parties as binary secret shares $\langle z \rangle$, each party will send their share $\langle z \rangle_p$ to all other parties. Once all parties have the shares, they can obtain z by computing $z = \oplus_{p \in P} \langle z \rangle_p$. Note that z will be a single bit (not a bitstring).
3. Compute $[\langle x \rangle^b] = [r] + z - 2[r]z$.
4. Repeat steps 1-3 for each bit b before finally converting these bits into an integer, $[x] = \sum_{b=0}^{B} 2^b [\langle x \rangle^b]$.

From a security perspective, the idea is that the random bit $\langle r \rangle$ will mask $\langle x \rangle^b$, so that when revealing z, the parties will not learn anything about x.

As of [79], CrypTen generates the random bits r using a trusted third party. However, they claim that they would like to implement a version which generates these random pairs offline using oblivious transfer or additive homomorphic encryption. As of the time of writing this, I have not found evidence to suggest that these have been implemented.

1.5.2 Secure Computation

In this section we will discuss how to perform computation on the secret shares. We will discuss addition, multiplication and comparison. Almost all of the ML computation that

CrypTen uses will be a combination of these computations or creating linear approximations of non-linear functions, so that they can be computed with addition, multiplication and comparison.

Addition

To add two additive secret shares $[x]$ and $[y]$, each party will perform $[z]_p = [x]_p + [y]_p$. Since the sum of all $[x]_p$ is x and the sum of all $[y]_p$ is y, we get that the sum of all $[z]_p$ is $z = x + y$. From a computational perspective, addition can be seen as the easiest to perform, requiring no additional communication between parties. This will keep costs low and introduce no additional potential security concerns as nothing is being revealed and no communication is taking place.

Multiplication

Multiplication in MPC is often quite complex (at least in comparison to addition) [80, 95]. When using additive secret shares it can be common among MPC protocols to make use of Beaver Triples to assist with multiplication [80, 45, 17]. There are also continuous advancements in literature surrounding the creation of Beaver Triples [156, 155]. Currently, in the CrypTen framework, Beaver Triples are generated by a trusted third party. The inclusion of a trusted third party in the framework introduces additional trust (clearly) that this third party acts honestly with the generation of the Beaver Triples. Other possibilities for generating Beaver Triples include offline through Oblivious Transfer [72, 55] or additive Homomorphic Encryption [111]. According to [157], generating these triples is difficult and other protocols, such as MASCOT [72] which provides among the most efficient implementation of this, use Oblivious Transfer.

A Beaver Triple, is a triple $([a], [b], [c])$, where a and b are randomly selected and $ab = c$. To perform multiplication of x and y, parties begin by computing $[\epsilon] = [x] - [a]$ and $[\delta] = [y] - [b]$. The parties then reveal ϵ and δ, by sharing their secret shares of ϵ and δ with all parties and reconstructing the true values. The parties finally compute $[x][y] = [c] + \epsilon[b] + [a]\delta + \epsilon\delta$. In order to maintain security, it is important that each triple is only used once.

First, we can show that this is correct as

$$\begin{aligned}
[c] + \epsilon[b] + [a]\delta + \epsilon\delta &= [a][b] + ([x] - [a])[b] + [a]([y] - [b]) + ([x] - [a])([y] - [b]) \\
&= [a][b] + [x][b] - [a][b] + [a][y] - [a][b] + [x][y] - [x][b] - [a][y] + [a][b] \\
&= [x][y].
\end{aligned}$$

To note, the order of multiplication may not matter when multiplying integers (or floats), but a very similar process is used for matrix multiplication as well, where the order of multiplication would matter. In fact, according to [80], the result will hold for any linear function $f(x, y)$, for any two variables $[x]$ and $[y]$, so long as we set $c = f(a, b)$ when generating the Beaver Triples.

For multiplication to be secure, we must assume that a and b are drawn uniformly at random from $\mathbb{Z}/Q\mathbb{Z}$, in order to properly mask $[x]$ and $[y]$ through decryption.

In regards to communication costs, the communication occurring during the multiplication process is in the revealing of ϵ and δ, where there is a round of communication between each pair of parties for each variable.

Comparison

Any comparison of two integers x and y can be reduced down to the comparison of z with 0, so long as z is chosen correctly. This will be shown below. To make the comparison $[z < 0]$, we must convert the arithmetic secret share $[z]$ of z into a binary share $\langle z \rangle$, using the arithmetic to binary conversion protocol in Section 1.5.1. Let L represent the length (number of bits in the bitstring) of $\langle z \rangle$. Next, each party bit shifts $\langle z \rangle_p$ to the right by L bits to obtain the most significant bit $\langle b \rangle$ of z. The most significant bit represents the sign of z, with a 1 representing a negative value and 0 representing a positive value. Finally, we convert the bit $\langle b \rangle$ back to an arithmetic secret share $[b]$, using the binary to arithmetic conversion described above. At the end of this process, each party will hold an arithmetic share $[b]_p$ (an integer), so that $\sum_{p \in P} [b]_p$ equals either a 1 if $z < 0$ or 0 if $z \geq 0$.

We note that bit shifting is trivial since each bit in a binary secret share $\langle z \rangle$ is seen as an independent secret shared bit. To obtain the most significant bit, we shift $\langle z \rangle$ to the right by $L - 1$ where L is the length of the bitstring.

Comparison Reductions The following comparisons equivalences are all from [80]. For $[x < y]$, we take $[z] = [x] - [y]$ and evaluate $[z < 0]$. Below, we show that any other comparison can be reduced to $[x < y]$, which we have shown we can compute.

- $[x > y] = [y < x]$
- $[x \leq y] = 1 - [y < x]$
- $[x \geq y] = 1 - [x < y]$

- $[x = y] = [x \leq y] - [x < y] = 1 - [x < y] - [y < x]$
- $[x \neq y] = 1 - [x = y] = [x < y] + [y < x]$

Using each of these expressions, we can perform any comparison between two values in the CrypTen framework.

Deep Learning Computations

All other functions used in our computations will be a combination of the protocols described above. Linear computations will be performed using addition and multiplication. Non-linear functions will be computed using linear approximations.

1.5.3 Security

The CrypTen paper [80] claims that the protocol is secure against information leakage against a static passive adversary corrupting up to $|P|-1$ of the $|P|$ parties involved in the protocol. As mentioned in Section 1.4, a static adversary is one that picks the corrupted parties before the run of the protocol. A passive (or semi-honest) adversary means that the corrupted parties will not deviate from the protocol, however they will try to learn as much information as possible during the run. While passive adversaries and static adversaries are not as strong as they can be (malicious and adaptive respectively), an adversary able to corrupt $|P| - 1$ parties is ideal.

Chapter 2

Character Recognition

This chapter has been posted on arxiv and submitted for publication. The work was performed with Hilda Azimi (National Research Council of Canada), Steven Chang (University of Waterloo), myself (University of Waterloo) and Koray Karabina (National Research Council of Canada and University of Waterloo).

The authors of the paper are listed alphabetically and their major contributions are as follows. Hilda Azimi: data preprocessing; optimization of DL models. Steven Chang: data preprocessing; feature extraction; implementation of ML models (Strategy 1). Jonathan Gold: implementation of ML models (Strategy 2) and ensemble learning; explainability of DL models. Koray Karabina: supervision of the project; feature extraction; implementation of ML models (Strategy 1); implementation and optimization of DL models. All of the authors contributed to the writing, reviewing and editing the original and subsequent drafts of the paper.

2.1 Introduction

2.1.1 Problem Statement and Related Work

In this chapter we will explore the online handwriting recognition task, as discussed in Section 1.1. Recall that handwriting is defined to be the writing characteristics of an individual [119] and handwriting recognition is the process of converting written text (whether that be letters, numbers, words, symbols, etc.) into a form which can be interpreted and

recognized by a computer. Online handwriting data is captured at run time, by some sort of sensor.

Over the years, there has been an increasing interest to experiment with different types of technology to collect handwriting data, to create datasets, and to develop algorithms to perform recognition of characters and symbols [8, 140, 62]. More recently, the OnHW-chars dataset [110] has been published that contains multivariate time series data of the English alphabet collected using a ballpoint pen fitted with sensors. The pen used to collect this data was the STABILO DigiPen, which is fitted with various sensors: 2 accelerometers, 1 gyroscopes, 1 magnetometers and a force sensor. Each letter in OnHW-chars is an $n \times 14$ vector, where n is the number of timesteps collected. For each timestep, 14 features were collected as follows: for each of the two accelerometers, gyroscopes and magnetometers, x, y and z values were collected, as well as one feature representing time; the final feature is force, which is represented on a scale of 0 to 4096 (5.32 N) [110]. The dataset OnHW-chars provides a basis to develop algorithms to correctly identify which letter is being written based on sensor data. While publishing a dataset, the authors of [110] also provided some baseline results through their machine learning (ML) and deep learning (DL) classifiers. DL classifiers, trained on the same dataset OnHW-chars, were improved in [106]. In addition, [106] reports on the accuracy of classifiers trained on other new datasets, such as OnHW-equations, OnHW-symbols and OnHW-words. There has been a sequence of other papers published exploring other topics and models including [109, 108, 107, 78], however we will focus on [110, 106] as these are the most accurate reported results.

All code used for our analysis is publicly available [27].

A brief description of OnHW-chars Being the most recent and publicly available online handwriting dataset with some state-of-the-art classifiers trained on it, we focus on developing classifiers for OnHW-chars. OnHW-chars contains multiple versions of each letter from a to z, collected from 119 unique users. Left handed writers were excluded from the dataset.

There is a separate dataset for lowercase and uppercase letters as well as a combined dataset of both lowercase and uppercase letters. Additionally, the data was pre-split in [110] into 5 folds of train and test split for each of the lowercase, uppercase and combined datasets. For each of these, a writer independent (WI) and a writer dependent (WD) version of the split is provided. In WI, writer sets of the train and the test split of a fold are disjoint. If a writer appears in the training (test) data, they are not in the test (train). However, in WD, a writer's data will be split between the training and test data. All in all, this gives 6 datasets (3 cases and 2 writer dependency) further split into 5 folds each.

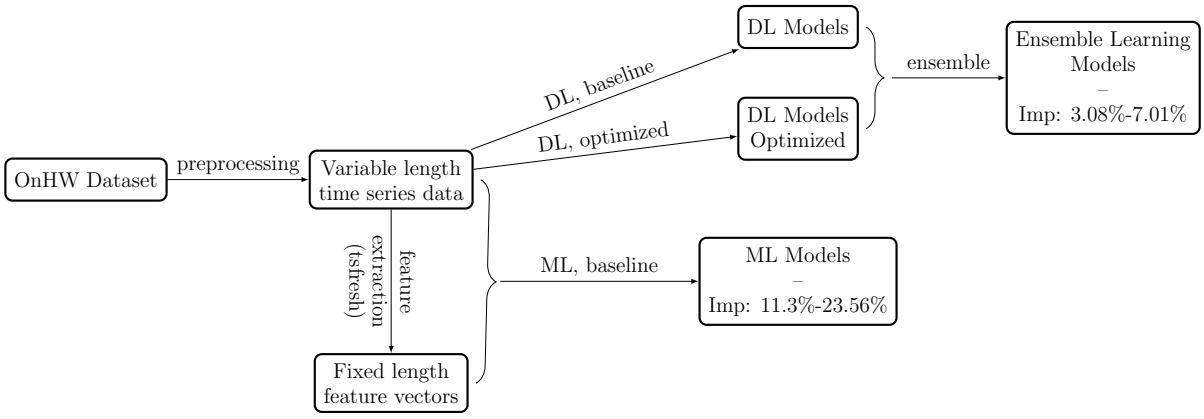


Figure 2.1: Our workflow for training machine learning (ML) and deep learning (DL) models, and utilizing the ensemble method. Feature extraction is performed using tsfresh [34]. Average accuracy improvements (Imp) are reported with respect to the previous ML and DL models in [110, 106].

In all this yields 30 datasets. When reporting on the accuracy of OnHW-chars classifiers in [110, 106], the authors average over 5 folds. Thus, they present classifier accuracies on 6 datasets, which we denote by lowercase-WD (L-WD), lowercase-WI (L-WI), uppercase-WD (U-WD), uppercase-WI (U-WI), combined-WD (C-WD), combined-WI (C-WI).

2.1.2 Contributions

Our contributions in this chapter are twofold:

Improving on the accuracy of classifiers on OnHW-chars and OnHW-equations To our knowledge, the best accuracy results of ML classifiers and DL classifiers for the OnHW-chars dataset are the ones reported in [110] and [106], respectively. As mentioned before, [106] improves DL classifiers in [110]. Our ML and DL models yield, respectively, 11.3%-23.56% and 2.17%-4.34%[1] improvements over the accuracy of the best ML and DL

[1]When working on improving upon accuracy results, mostly in the $80-90\%$ range, the improvements seen here are significant.

models reported in [110, 106]. We obtain these improvements thanks to the use of state-of-the-art feature extraction algorithms and optimizations of the models. We further utilize the ensemble learning method to achieve 3.08%-7.01%[2] improvements over the best results reported in [110, 106] for the OnHW-chars dataset; and 0.05%-4.75%[3] improvements over the best results reported in [106] for the OnHW-equations dataset. We refer the reader to Figure 2.1 for an overview of our workflow, Tables 2.2 and 2.3 for a summary of our results, and to Sections 2.3, 2.4, 2.5 and 2.6 for more details about the models, explanations, and workflows used.

Providing verifiable and reproducible results with some level of explainability In addition to providing improvements to accuracy across the spectrum, this chapter aims to provide some level of explainability of these models so as to provide more rationale as to why these methods are chosen and why the models are suitable for the type of data in the dataset. In addition to prediction accuracy, it is important to assess ML models on how they come to their decisions [52]. One goal is to explain the reason as to why it makes sense to utilize the models used for the specific data at hand. For the DL models we extend the local interpretable model-agnostic explanations (LIME) [126] architecture to add explainability to our multivariable time series (MTS) data. We refer the reader to Sections 2.4 and 2.5 for more details.

Finally, we would like to add some transparency to the process. A public repository is provided containing all the preprocessing and code for each model, so that it is reproducible and verifiable. Some models, specifically some DL models contain a level of randomness, which will alter the results slightly from one run of the model to the next. For these situations, a loadable encoded representation of the model will be provided as well. With this, we would like to make the model generation process public, so that others can have access to these models, as well as verify, reproduce, and improve our results.

[2]These improvements are significant results. Here we are showing how our results can improve even more, with the use of ensemble methods.

[3]These results show the robustness of the models used when trained and tested on other datasets. We see that the models are robust to new data.

2.2 Processing the OnHW-Chars Dataset

2.2.1 Preprocessing and Filtering

The OnHW-chars dataset is provided publicly in its raw form. In [110], detailed explanations are provided for preprocessing OnHW-chars. However, to our knowledge, the pre-processed versions are not publicly available, which introduces some challenge for reproducing results and providing fair comparisons. According to [110], a high pass filter is applied to the data with a cutoff frequency of 1 (Hz) to remove the gravitational acceleration from the accelerometer recordings. Also, a moving filter with a window of size 11 is used and that acts as a low pass filter, allowing high-frequency noise removal from the data. It is noted in [110] that the filtering is used when applying ML models and only trimming is applied for the DL models. Our pre-processing steps on raw data also vary depending on the choice of the classification algorithm. We will provide the details of changes in pre-processing steps for each methodology.

Table 2.1: Statistical characteristics of the OnHW-chars dataset subsets in terms of mean (μ) and standard deviation (σ) of the total number of timestamps for each sample of data.

Subset	μ	σ
Lowercase (L)	44.05	29.93
Uppercase (U)	52.85	42.82
Combined (C)	48.45	37.20

As mentioned in Section 2.1, OnHW-chars contains 6 datasets: L-WD, L-WI, U-WD, U-WI, C-WD, and C-WI with different characteristics. Table 2.1 presents statistical characteristics of the OnHW-chars subsets in terms of mean (μ) and standard deviation (σ) of the total number of samples for each instance of data. The σ for each subset in relation to their respected μ indicates that the number of samples for each instance in OnHW-chars is notably spread out and confirms the presence of outliers. In order to reduce the impact of outliers on the data and make the data more coherent, we discard instances based on the statistical characteristics of the lowercase subset. This is because only between 2%–4% of normal English literature characters are written in capitals depending on the text and the genre. Therefore, considering real-life applications, the lowercase subset can be a better candidate for detecting and removing outliers. Using μ and σ of the lowercase subset as given in Table 2.1, we remove any data sample with sequence length greater than

$\mu + 2\sigma \approx 104$. Considering that $\mu - 2\sigma$ yields a negative value, we determine, based on inspecting the dataset, that any sequence of length smaller than 10 should be discarded.

2.2.2 Extracting Statistical Features From OnHW-Chars

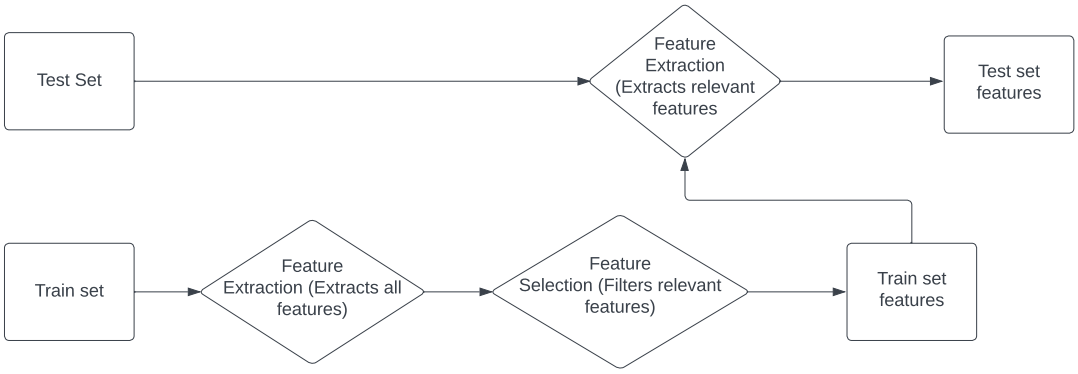
OnHW-chars contains variable length time-series data points. On the other hand, ML algorithms commonly accept fixed length data points for their training. Therefore, for each variable-length data point in OnHW-chars, we are motivated to extract a fixed number of features that carry as much information as possible about the data.

We first pre-process OnHW-chars as mentioned in Section 2.2.1, which is then provided as input to the tsfresh software package [34]. Tsfresh is the main component used in our feature extraction process. Tsfresh applies the feature extraction and scalable hypothesis testing (FRESH) algorithm [35] to extract features from variable length time series data, and further selects (and filters) relevant features based on their significance to the classification task at hand [34]. It computes a total of 794 time series features, obtained from the combination of 63 different time series characterization methods. The features range from simple statistical features such as the mean and standard deviation, to more complex ones including Fourier coefficients.

After applying tsfresh's feature extraction algorithm to OnHW-chars, there were a total of 10231 features extracted for every letter recording. This comes as a result of 787 features extraction per dimension for the 13-dimensional OnHW-chars dataset. These features are then further filtered using tsfresh's feature selection step, which conducts a series of scalable hypothesis tests and significantly drops the number of features. For example, 1571 features are extracted from the first fold of L-WI in the OnHW-chars dataset. We should emphasize that, in order to control leakage of information from the test set, we applied the feature selection step only on the training set. Once we have the selected features from the training set, those same features are directly extracted from the test set. The reader can find a flowchart depicting this in Figure 2.2.

In our experiments, we observe that accuracy of ML classifiers is significantly affected by the choice of the parameter n_significant in tsfresh. The parameter n_significant represents the number of classes a features contribution should be significant in its ability to predict. That parameter can be set between 1 and 26, which is where the trade-off arises. When selecting n_significant to be high, only a very small set of features will be selected, which tends to lead to low classification accuracy due to the high amount of information loss. Counter to that, when selecting n_significant to be low, we get many features but they may not predict the majority of class labels. This trade-off is later visualized in Figure A.1.

Figure 2.2: Feature extraction flowchart.



Through experimentation with the k-nearest neighbors (KNN) model, we chose to use n_significant = 17 in our feature extraction step. We caution that this choice might not be optimal and a better choice may exist. Once feature extraction and selection steps are completed, we use the extracted features to train our ML models. During training, other transformations may be applied to data, which we explain next.

2.3 Developing Models on the OnHW-Chars Dataset

2.3.1 Machine Learning (ML) Algorithms

For developing ML models, we follow two strategies. In both strategies, the OnHW-chars dataset is preprocessed and filtered as described in Section 2.2.1. In our first strategy, we extract fixed-length features from the preprocessed and filtered OnHW-chars dataset as explained in Section 2.2.2, and then these features are provided as input to the training of ML models where we use Decision Tree, Random Forest, Extra Trees, Logistic Regression, k-Nearest Neighbours, and Support Vector Machines. In our second strategy, we investigate elastic similarity measures for classification, by using the preprocessed and filtered OnHW-chars for training KNN using Dynamic Time Warping (DTW). In Section A.1, we provide details about our ML models, and we refer the reader to Table 2.2 for our results and a comparison with the results from [110]. In particular, we obtain 11.3%-23.56% improvements over the ML models in [110].

2.3.2 Deep Learning (DL) Algorithms

In order to provide a comparison with the (baseline) results in [110, 106], we first train DL models considered in [110, 106]. Our trained models include fully convolutional network (FCN), residual neural network (ResNet), long short-term memory (LSTM), bidirectional LSTM (BiLSTM), inception time (InceptionTime), xception time (XceptionTime), and explainable convolutional neural network (CNN) for multivariate time series (XCM). We explain each DL methodology that we train on OnHW-chars in Section A.2. We refer the reader to Table 2.2 for our results and a comparison with the best results from [110, 106]. Our baseline models yield 2.17%-3.91% improvements over the DL models in [110, 106]. Our DL models have been developed using the software package tsai [104], where we used the default architectural parameters in tsai [104] while setting the epoch number to 50 and the learning rate to 0.001.

2.4 Optimizations

In the following sections, we discuss architectural and hyperparameter optimization of some of our DL models and also the use of ensemble learning to further improve the accuracy of our DL models. Even though we observed similar accuracy after parameter optimizations, the use of ensemble learning provided up to 2.98% improvements over our DL models, whence 3.08%-7.01% improvements over the results reported in [110, 106]; see Table 2.2. For DL baseline methods in [110, 106], as indicated in Table 2.2, CNN-BiLSTM had the best performance among all other DL models. However, we could not implement the same model due to the lack of detailed information about the CNN architecture, such as the total number of layers, the total number of filters in each layer, and filter sizes. CNN, as a class of artificial neural networks, can take different structures leading to different performances and dramatically affecting the results for comparison purposes. Nevertheless, compared to CNN-BiLSTM, our InceptionTime implementation had a better performance overall.

2.4.1 Optimizing Architectural and Hyper Parameters

Since the InceptionTime model did better overall among our base DL models, we decided to optimize it. Due to a large choice of architectural parameters in LSTM, we also decided to optimize LSTM, LSTM-FCN, and MLSTM-FCN. Of course, other models could be optimized but we leave this for future work. We used the optuna framework [5] with 100 trials in our study. Across all of the optimization studies, our search space for the

Table 2.2: A summary of our results in comparison with [110, 106]. Columns labeled with "OnHW" show the best results from Table 4 in [110] and Table 6 in [106]. Values in bold font indicate the best result of their column in their group in the table. Values in bold font that are also underlined indicate the best result in their groups in the table. There are 5 groups: ML Baseline, DL Baseline, DL Optimized, Ensemble Learning, and Best Overall.

		Lowercase				Uppercase				Combined			
		WD		WI		WD		WI		WD		WI	
		Proposed	OnHW	Proposed	OnHW	Proposed	OnHW	Proposed	OnHW	Proposed	OnHW	Proposed	OnHW
ML Baseline	5NN with NCA	77	-	68.35	-	81.51	-	74.72	-	64.96	-	55.73	-
	5NN with PCA	57.7	-	44.42	-	59.49	-	48.7	-	36.1	-	23.5	-
	DT	49.34	30.49	44.85	22.89	55.91	33.23	51.48	24.32	37.39	20.32	33.06	20.33
	ET	70.49	-	61.81	-	73.83	-	66.15	-	56.8	-	48.2	-
	KNN-DTW	65.41	-	54.62	-	72.08	-	62.72	-	50.69	-	41.14	-
	LogReg	71.98	56.16	66.54	49.6	77.36	62.59	73.55	**53.26**	60.68	43.95	54.87	41.66
	RFC	71.45	58.02	64.97	45.55	75.32	63.19	69.25	45.96	58.33	43.6	51.65	43.62
	Linear SVM	75.64	**62.09**	68.1	**51.8**	80.75	**70.61**	74.67	54	66.24	**48.77**	58.04	**46.56**
	RBF SVM	**78.42**	-	**71.08**	-	**81.91**	-	**76.82**	-	**66.89**	-	**59.87**	-
	KNN	67.61	49.17	55.96	34.09	70.29	57.49	61.42	36.68	50.33	38.3	39.57	33.08
ML Improvements		16.33		19.28		11.3		23.56		18.12		13.31	
DL Baseline	FCN	86.93	81.62	73.67	71.48	88.16	85.37	78.71	77.24	77.39	67.41	62.15	58
	InceptionTime	**92.74**	84.14	**83.44**	75.28	**94.54**	87.8	**88.43**	81.62	83.17	70.43	70.49	61.68
	BiLSTM-FCN	86.53	-	73.74	-	88.11	-	79.54	-	77.89	-	63.31	-
	LSTM-FCN	86.5	81.43	74.36	71.41	88.28	85.43	79.91	77.07	77.92	67.34	63.26	57.93
	LSTM	88.33	79.83	78.33	73.03	90.83	88.68	84.59	81.91	79.15	67.83	67.61	60.29
	BiLSTM	88.69	82.43	78.5	75.72	91.3	89.15	84.37	81.09	79.42	69.37	67.5	63.38
	(Bi)MLSTM-FCN	86.7	-	74.49	-	89.2	-	80.74	-	79.15	-	65.1	-
	MLSTM-FCN	86.63	80.21	74.15	71.9	89.16	85.25	80.89	77.44	79.12	69.33	64.82	60.14
	ResCNN	90.23	82.52	78.26	72	91.53	86.91	82.41	78.64	80.22	67.55	65.43	58.67
	ResNet	92.48	83.01	81.64	71.93	94.11	86.41	86.28	78.03	82.84	68.56	68.65	58.74
	XCM	81.94	74.39	72.36	68.12	84.11	81.67	76.41	74.32	70.99	58.18	61.82	51.99
	XceptionTime	91.95	81.41	82.86	70.76	94.02	85.94	87.93	78.23	**83.32**	66.7	**71.8**	56.92
	CNN-BiLSTM	-	**89.66**	-	**80**	-	**92.58**	-	**85.64**	-	**78.98**	-	**68.44**
DL Optimized	InceptionTime	**92.79**	-	**83.91**	-	**94.75**	-	**88.74**	-	**82.71**	-	**71.82**	-
	LSTM-FCN	85.27	-	75.87	-	89.44	-	81.82	-	77.72	-	65.8	-
	LSTM	89.49	-	80.86	-	91.32	-	85.26	-	79.16	-	70.51	-
	MLSTM-FCN	87.35	-	76.36	-	87.35	-	80.65	-	79.19	-	67.83	-
DL Improvements		3.13		3.91		2.17		3.1		4.34		3.38	
Ensemble Learning	Plurality (top 3)	93.47	-	84.95	-	94.98	-	89.27	-	85.08	-	74.13	-
	Soft (top 2 + opt.)	93.66	-	85.39	-	95.14	-	89.74	-	85.36	-	74.56	-
	Weighted Soft	**94.18**	-	**86.12**	-	**95.66**	-	**90.34**	-	**85.99**	-	**74.8**	-
	Soft (all)	93.34	-	84.52	-	94.76	-	89.01	-	85.49	-	73.77	-
Best Overall		**94.18**	**89.66**	**86.12**	**80**	**95.66**	**92.58**	**90.34**	**85.64**	**85.99**	**78.98**	**74.8**	**68.44**
Overall Improvements		4.52		6.12		3.08		4.7		7.01		6.36	

"learning rate" is set between 0.00001 and 0.01 with logarithmic increments and the "epoch number" is exhausted from 25 to 100 with increments of 25. In our study, search space for InceptionTime parameters are as follows: "nf" takes $4, 8, 16, 32, 40, 48, 56, 64, 128$; "depth" takes values from 1 to 15 with increments of 1 and "fc_dropout" takes values from 0 to 0.9 with increments of 0.1 The search space for LSTM parameters are as follows: "n_layers" takes values from 1 to 5 with increments of 1; "rnn_dropout" and "fc_dropout" take values from 0 to 0.9 with increments of 0.1; "bidirectional" takes True or False. The search space for LSTM-FCN and MLSTM-FCN parameters are as follows: "rnn_layers" takes values from 1 to 5 with increments of 1; "rnn_dropout" and "fc_dropout" takes values from 0 to 0.9 with increments of 0.1; "bidirectional" takes True or False. We first optimized the model parameters independently on the L-WI, U-WI, and C-WI datasets. We used the same parameter sets when training our models on the respective WD datasets. Therefore, for a small number of cases, our models trained on WD underperform in comparison with the base DL models where the most significant drop is observed for MLSTM-FCN trained on the U-WD dataset. One could ideally optimize the parameters independently on WD as well, but we do not pursue this path mainly because of the high cost of optimizing parameters and the minor performance gains due to parameter optimization over our base models (e.g. compare the accuracy of InceptionTime in Table 2.2 for DL Baseline vs. DL Optimized). More details, including the optimized models and their parameters will be publicly available and the accuracy of the optimized results are presented in Table 2.2. Our optimization efforts resulted in similar accuracy in comparison with our base DL models.

2.4.2 Optimizing via the Use of Ensemble Learning

While the models above outperform the previously best published results and the optimizations of these models yield even more accurate results, we wanted to find a way to increase the accuracy even more. We begin by analyzing how the models make predictions, and specifically how each model differs from each other when making predictions. We examine the data where (the best) models make incorrect predictions and see how other models perform in these points. This analysis leads us to believe that combining the predictions of these models in an ensemble method may lead us to increased accuracy [49]. We present the findings of our analysis, as well as results below.

Failure Space

We will define the failure space of a classification model as the space of input data where the model makes an incorrect predictive classification. We examined how the failure spaces of

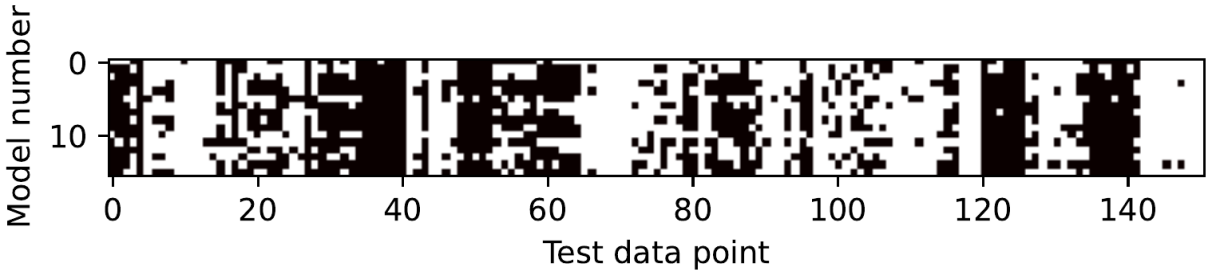


Figure 2.3: The prediction space analysis in the both independent fold 0 dataset. The models are numbered as follows: 0:XceptionTime,1:InceptionTime, 2: ResNet, 3: LSTM FCN, 4: BiLSTM FCN, 5:LSTM, 6:BiLSTM, 7:MLSTM FCN, 8:BiMLSTM FCN, 9:FCN, 10:ResCNN, 11:XCM, 12: Optimized InceptionTime, 13: Optimized LSTM, 14: Optimized LSTM FCN, 15:Optimized MLSTM FCN.

each of the 16 models trained above (12 base models and the 4 optimized models) compared to one another. It was hypothesized that the different architectures and even parameters (on a small scale) would cause different models to fail in different places. In other words, the intersection of the failure spaces of these models does not fully overlap. If this were to be the case, then there would be reason to believe that the models may be focusing on different aspects of the MTS and that there is potential for a combination of models to be used to collectively produce an output. Below, are two separate analyses exploring the failure spaces of the 16 models.

First, we explore how each of the models perform in comparison to one another. The dataset is fixed to the C-WI, fold 0 dataset. We then fix a letter, say "Z", and examine all test data points that are actually "Z"s. For each piece of test data, we determine whether each model predicted it correctly or incorrectly. The results are displayed in a heatmap in Figure 2.3. Each row corresponds to the prediction of a different model, and each column corresponds to a unique test data. The colour represents a correct or incorrect prediction, with a correct prediction being white and an incorrect prediction being black. We call this type of an analysis the prediction space analysis. We notice that the map looks very spotted. There are a few columns that are all black, but there are also many columns that are split between correct and incorrect predictions. This tells us that it might be beneficial to combine these models to diminish the incorrect predictions. The prediction space analysis shows us that oftentimes, when even the best performing models are incorrect, other models may predict the data point correctly.

After examining the failure spaces of each of the models, it was found that the models

were failing in different places. Hence, for a given test case, a portion of the 16 models can fail, but oftentimes others will succeed. From here we hypothesized that there may exist some combination of models which can work together to make predictions more accurately than individual models.

Plurality Voting

Through our failure space analysis, we wanted to try to find a way to combine multiple models to make one single prediction. The first attempt at this is plurality voting, where each model will make a prediction on a piece of data and vote on the predicted class. The final output class is the one that receives the most votes [154]. A tie in plurality voting is broken arbitrarily. This method produced results slightly worse than our best performing models. This is not unexpected since the best models and worst models all have an equal say in the output. One way to resolve this will be weighted voting and will be explored next. Another resolution is to drop the poorly performing models and only consider the top tier (best 3 models) for the plurality voting. This resulted in an increase of up to 1% on the best performing algorithm for each dataset.

Weighted Voting

In cases where the individual classifiers are of unequal performance, it can be valuable to give more voting power to the stronger performing models and less voting power to the weaker models [154]. As can be seen in Table 2.2, we trained 16 models with varying levels of performance and accuracy. In the Plurality voting method, the top 4 best models were considered in the voting process for the best results. For better accuracy, we tried to include more models in a weighted voting method, with lower weights. The models were divided into 3 categories: top, middle, and bottom tier. Out of the 16 models considered, 4 belonged in the top tier (InceptionTime, Optimized InceptionTime, XceptionTime and ResNet), 4 in the middle tier (ResCNN, Optimized LSTM, Optimized MLSTM_FCN and Optimized LSTM_FCN) and 8 in the bottom tier (LSTM, BILSTM, MLSTM_FCN, BIMLSTM_FCN, FCN, BILSTM_FCN, LSTM_FCN,XCM). The weights were established so that the top tier models are considered first and lower tier models are considered only in the event of a tie. The bottom tier of models get a weight of 1 applied to their vote, the middle tier gets a weight of 9 applied to their vote, and the top tier gets a weight of 45 applied to their vote. If all 4 bottom tier models agree on a class, their vote is still outweighed by a single middle tier model. Similarly, if all middle tier models agree on a class, their vote is outweighed by a single top tier model.

Soft Voting

The models trained above provide a probability that a given input belongs to each of the letter classes. The class with the highest probability becomes the predicted class from the model and up until now is all that has been considered. In soft voting, for each class, the probabilities of each prediction class are averaged. The class with the highest average is outputted. Soft voting was implemented, with an improvement in both plurality voting and weighted voting.

Weighted Soft Voting

The most promising approach we have found is the weighted soft voting model. This combines the benefits of soft voting with those of weighted voting. Provided here is a soft model that gives weights to the probabilities depending on the quality of model. This model, using the same weights as above, was the best performing of the ensemble methods tried and yielded between a 0.91% and 2.98% increase.

Ensemble Learning in Practice

As seen above in Table 2.2, the accuracy of the ensemble learning techniques is competitive with or improved upon existing methods. An increase in accuracy when using ensemble methods is consistent with the literature [147, 149, 123, 84]. However, we should explore the impact of implementing a protocol like this in a practical application.

It is important to examine the accuracy vs. training-time trade-off [143, 147] of using ensemble learning models. Not only do ensemble learning models take longer to train, but they are also larger in size and require more time to infer predictions. The size of the ensemble learning model and time it takes to run are both combinations of the models used to build them, plus some small overhead. The time required to make an inference and the size of the models must both be considered when using ensemble models in practice. We hypothesize that user-device based prediction may not be practical with ensemble methods. To use ensemble methods, it may be useful or even required to use server-based predictions as the models can be housed on a server with sufficient storage and computing power.

2.5 Further Explaining DL Models

It is important to not treat these DL models as black box, but to gain some intuition as to why predictions are being made in these ways. We already provided some explanations regarding failure spaces of the models, which motivated the use of ensemble learning. In this section, we use an interpretation of LIME [126] for time series data, lime-for-time [94]. In this interpretation, the time series is divided into 20 "slices" and the importance of each slice is determined by the LIME algorithm. Additionally, since we are working with MTS data, each of the 13 signals (channels) will be analyzed separately.

The 13 channels are listed below. Channels $0-2$ correspond to the first accelerometer sensor in the x, y and z axes respectively. Channels $3-5$ correspond to the second accelerometer sensor in the x, y and z axes respectively. Channels $6-8$ correspond to the gyroscope sensor, again in the x, y and z axes. Channels $9-11$ correspond to the gyroscope sensor, again in the x, y and z axes. Finally, the 12th channel corresponds to the force sensor.

The top 30 signals and slices, in terms of importance, will be examined. A green bar, indicating a score greater than 0, signifies that this section of the data has a positive impact on the model output and contributes to the prediction, while a red bar, indicating a score less than 0, signifies that this section of the data has a negative impact on the model output and gives evidence against the prediction [126]. Figure 2.4 shows all 13 channels of the raw data with an importance bar overlaid on top of it. The darkness of colour indicates how much influence that slice of the data has on the classification.

We show two examples of this explanation analysis in order to give some backing and explanation to the predictions being made. The first is an example of where a model predicts correctly and is fairly certain about its prediction. This is shown in Figure 2.4 regarding the explanation of the Optimized LSTM model correctly predicting a letter "B" in the C-WI fold 0 dataset. As we can see, the green and red bars are primarily on the 6th, 8th and 12th channel. This means that these channels have more influence (or importance) on the overall prediction. Each channel corresponds to a signal from a sensor and thus different dimension of the MTS.

The second example provides the explanation of two different (yet both strong) models, which disagree about the output class. This is shown in Figure 2.5. As we can see both of these models inherit influence from different places. One example is that XceptionTime, which exerts influence into the first and 11th channel for this prediction whereas InceptionTime does not, while InceptionTime exerts influence into the 4th and 7th channel and XceptionTime does not. Additionally, we see different sections of the 6th, 8th and 12th

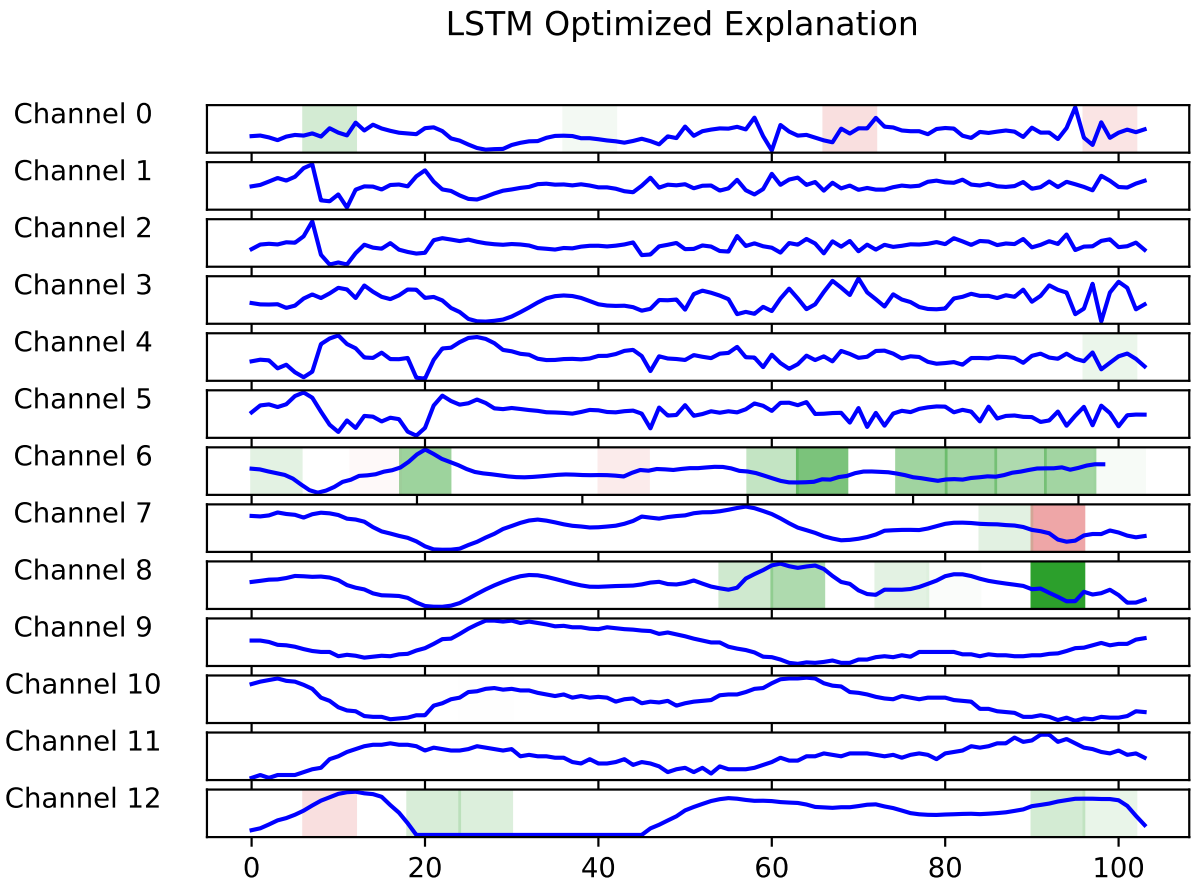


Figure 2.4: Explanation of LSTM Optimized on the both independent fold 0 dataset, for a test letter "B". Blue lines represent 13-dimensional data with respect to time.

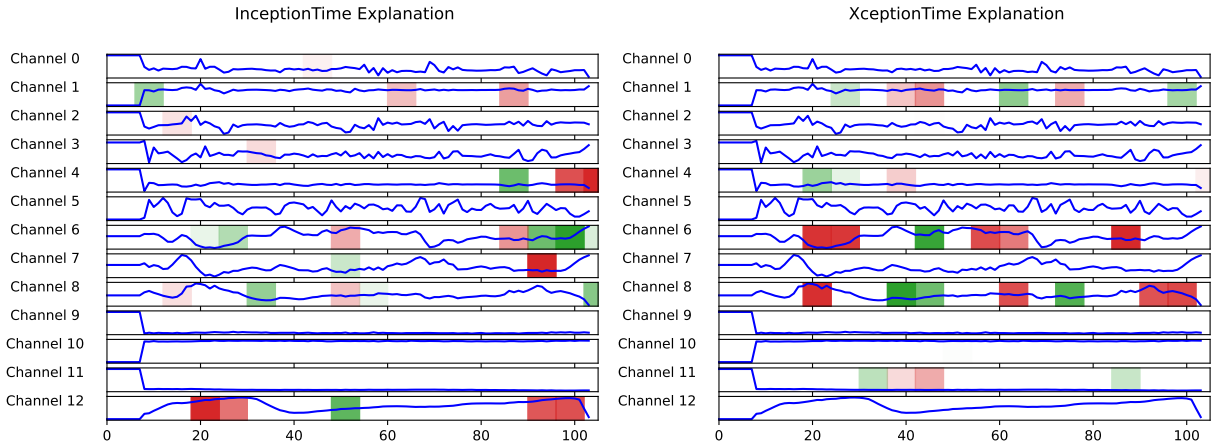


Figure 2.5: The first image on the left is the explanation of a correct InceptionTime prediction of the letter "K". The second image on the right is the explanation of an incorrect XceptionTime prediction of the same letter. XceptionTime predicts "R". Blue lines represent 13-dimensional data with respect to time.

channels being highlighted. This tells us that not only are different models making different predictions from each other, but they are deriving importance from different parts of the MTS. This leads us to believe that by combining these models, we may be able to take advantage of different models inheriting importance from different areas, thus improving the overall "catchment" of the models chosen. Along with the failure space analysis showing that the 16 models fail in different places, these explanations show that the models also value different parts of the signal to have different levels of importance. These two pieces of analysis provide even more motivation for ensemble learning, as this provides reason to believe that there may be benefit in combining these models to produce a signal output, with better success than any individual model.

2.6 Extended Analysis Over the OnHW-Equations Dataset

In addition to the OnHW-chars dataset as analyzed above, the split OnHW-equations dataset has recently been published in [106]. This dataset is the OnHW-equations dataset divided and split on a character basis, so that our task remains a character recognition task.

For this analysis we will use the same models used in the analysis in Section 2.3.2 in

order to show the robustness of these models on new datasets. We have only looked at the deep learning models and not the machine learning models, since the deep learning models provided the best results. These models will be freshly trained on the new data (as equation characters are different from letters), with model parameters remaining the same across the baseline and optimized DL models. One difference in this set was that we ran the baseline models for 25 epochs instead of 50 as above.

Additionally, for the analysis presented, we include only the results from the fold-0 dataset, as this is what is presented in the benchmarking paper [106].

The only preprocessing performed on the data was to standardize the length of the time series across all data points. We chose a standard length of 133 timesteps as this is equal to the mean plus two times the variance. If a piece of data has fewer timesteps, we append zeroes to the end to reach 133 and if a piece of data has more timesteps, we truncate the data to 133 timesteps.

The results of this analysis are presented in Table 2.3. We see that XceptionTime is the best performing model on this dataset, performing 1.29% better than the best performing existing model in the writer independent case. However, it performs 0.32% worse in the writer dependent case.

We also see that the optimized models provide similar performance to their baseline counterparts.

Finally, we see an improvement on all the best results from all ensemble methods. At the highest, we see an improvement of 4.75% on the highest accuracy from [106] in the writer independent case and 0.05% in the writer dependent case.

2.7 Concluding Remarks

We developed various handwriting recognition models on the OnHW-chars and OnHW-equations datasets [110, 106]. Our ML models improved the accuracy of the previously known ML models [110] up to 23.56% and our DL models (coupled with ensemble methods) improved the accuracy of the previously known DL models [106] up to 7.01%. We also provided some level of explainability for our models. Our explanations motivated the use of ensemble learning to boost the accuracy of our models and justified its success. Being the most recent and publicly available online handwriting dataset with some state-of-the-art classifiers trained on it, we have chosen to develop our models on the OnHW-chars and OnHW-equations datasets. We expect that our techniques are applicable to other datasets in a more general context.

Table 2.3: A summary of our results in comparison with [106] for the OnHW-equations CTC split dataset. Columns labeled with "OnHW" show the best of best results from Table 6 in [106]. Values in bold font indicate the best result of their column in their group in the table. Values in bold font that are also underlined indicate the best result in their groups in the table. There are 4 groups: DL Baseline, DL Optimized, Ensemble Learning, and Best Overall.

		OnHW-equations and symbols			
		WD		WI	
		Proposed	OnHW	Proposed	OnHW
DL Baseline	FCN	95.10	94.03	81.17	81.46
	InceptionTime	96.01	94.87	87.78	84.35
	BiLSTM-FCN	94.98	-	80.55	-
	LSTM-FCN	95.08	93.95	81.54	81.47
	LSTM	94.74	-	82.43	-
	BiLSTM	94.39	-	85.09	-
	(Bi)MLSTM-FCN	95.28	-	84.48	-
	MLSTM-FCN	95.21	-	83.46	-
	ResCNN	95.40	94.58	83.18	80.95
	ResNet	96.03	94.68	83.18	83.45
	XCM	90.74	-	81.45	-
	XceptionTime	**96.39**	94.03	**88.48**	82.24
	CNN+TCN	-	**96.70**	-	84.91
	CNN+BiLSTM + Label Smoothing	-	95.86	-	**87.09**
DL Optimized	InceptionTime	96.18	-	88.08	-
	LSTM-FCN	94.77	-	79.81	-
	LSTM	95.03	-	**89.18**	-
	MLSTM-FCN	95.05	-	84.67	-
DL Improvements		-0.31		2.09	
Ensemble Learning	Plurality (top 3)	96.60	-	90.15	-
	Soft (top 2 + opt.)	96.63	-	**91.84**	-
	Weighted Soft	**96.75**	-	91.69	-
	Soft (all)	96.55	-	91.44	-
Best Overall		**96.75**	**96.70**	**91.84**	**87.09**
Overall Improvements		0.05		4.75	

An interesting future work would be to develop better ensemble methods. For example, one could try to optimize the weights in the voting stage using meta learners or combinatorial approaches [40]. Another approach would be to consider Bayesian model averaging [97]. Finally, it would be interesting to run a deeper analysis on the explainability of models, and to understand the importance of features as this would provide some insight to improve the performance and efficiency of the models.

Chapter 3

Private Character Recognition

3.1 Introduction

We have spent Chapter 2 discussing handwriting character recognition and how to use machine and deep learning techniques to obtain the best possible accuracy in the given task. Before we move on to the theme of this chapter, which will be adding user privacy to this task, we must explore applications of character recognition in Section 3.1.1. After exploring these applications, we will motivate why these applications may benefit from introducing a privacy component in Section 3.1.2.

The remainder of the chapter will explore how we design and implement a protocol for private character recognition, before considering the effects that this framework has on accuracy and costs. First, in Section 3.2, we will outline the design of the protocol. Next, in Section 3.3, we will describe the implementation of this design. Finally, in Section 3.4 we will explore the implementation, and what kind of costs are associated with introducing a privacy component to the character recognition task.

3.1.1 Applications of Online Character Recognition

As described in Section 1.1, the handwritten character recognition task aims to recognize a character written with a pen containing sensors to track its movement. Some applications of online handwriting recognition includes pen-based computers [119], signature verification [119], home safety [119] and school exam analysis.

Our particular area of interest for the character recognition task is in analyzing examinations (school). We will explore this more in Chapter 4 in regards to proctoring students. However, while recording student's handwriting for proctoring we can process what is being written as well using character recognition.

Additionally, although moreso in the offline setting, optical character recognition can also be used in reading and processing forms or documents. Similar ideas to those presented in the rest of the chapter should be applicable in the offline setting as well (with minor modifications).

3.1.2 Motivation

We have discussed online handwriting in various forms for much of the thesis so far (Chapter 2 and Sections 1.1 and 1.2). Particularly, Chapter 2 explores the task of online character recognition. As seen in Section 3.1.1, there are many applications of character recognition. Some of these applications involve a local device collecting and analyzing the handwriting data. This can be seen in pen based computing or certain applications of document analysis. Other types of applications involve collecting the handwritten data and sending the information to a third party or service provider. This can be observed in signature verification or exam analysis systems.

Such a system involves sending raw biometric data (in the form of handwriting), to these third party service providers. We must note that such biometric data is viewed as sensitive personal information and should be treated as such when sent to a third party. Transferring the raw data can create serious security and privacy threats [138, 75].

Handwriting data in the wrong hands can be problematic. The information that we are writing about may be private and users may not want the content of their writing to be public. Even if a user may be comfortable sharing the content of their writing, it can still be an invasion of privacy to share their handwriting itself. Kindt [75] claims that exposing biological characteristics such as handwriting to the public can increase the risk of being subject to "tracking and tracing" and surveillance. Kindt cites the secret use of facial recognition at the 2001 Super Bowl, to find criminals in the crowd, and indicates that similar kinds of surveillance can be used with handwriting. In addition to surveillance, it was briefly mentioned in Section 1.1 that online handwriting analysis has been used in security and e-health applications [51]. As discussed in [51], handwriting analysis can help with diagnosis of diseases such as Alzheimer's [117], ADHD [81] and Parkinson's [26]. Due to the potential use in health diagnosis, surveillance and keeping the content of the writing private, the importance for keeping handwriting data private is clear.

Now that we have described why it is important to keep handwriting data private, as well as there being different systems which may require a user to send their handwriting data to a third party service provider, it is important to design handwriting systems where privacy can be maintained throughout the service being provided. In this chapter we explore the use of MPC to help a third party provide a service for character recognition, in a private manner.

3.2 Protocol

In Section 3.2.1, we will discuss the proposed framework for private character recognition. In Section 3.2.2, we explore the threat model used for this analysis.

3.2.1 Private Character Recognition Framework

When designing a framework, first we must establish the goals of the framework. The real world situation we are looking to model is where we have some service provider, who is aiming to provide a service involving character recognition. As highlighted above, it is important for the service provided to take into account user privacy, so the user does not have to share their raw handwriting data to the service provider. To help with the privacy of the user's data, we will make use of MPC.

First, we assume that the user has recorded their handwritten biometric data. Second, we assume that the service provider has a character recognition model, like the one in Chapter 2. We make the assumption here that the service provider has successfully trained the character recognition model without the need for private data. This is not an assumption that we can make in all situations, as we will see in Chapters 4 and 5. In the character recognition applications, however, we believe this to be a reasonable assumption, since we do not need user specific data to train a model for this specific task. There are existing public databases of handwriting which can be used to train the model, as we have shown in Chapter 2.

We aim to design a protocol where the service provider can use their model to make a prediction about the user's data without seeing the data itself. To accomplish this, we will propose two methods. The first is a two-party communication model, which we will use for our implementation in Section 3.3. The second is a client/server model, which we will not implement directly, however can be useful in certain situations and should yield similar results.

Two-Party Communications Model

First, we explore the two-party communications model. In this model, we assume that the two parties involved in the computation are the model holder and the data holder (in our case these are the service provider and user respectively). In other words, there are no external computing parties needed for the calculation. In the following explanation, we will use CrypTen's MPC framework. Other frameworks can be used, with the general idea holding in other MPC models as well.

The user divides their data into two (sets of) shares, keeping one for themselves and sending the other to the service provider. In turn, the server hides their model, by dividing the model parameters into two (sets of) shares, keeping one for themselves and sending the other to the user.

Next, the parties will perform the required computation using the hidden model to make a prediction on the hidden data. This data will still be in hidden form, until the service provider and user come together to reveal the output. The output can be revealed to either party individually or to both, depending on what is needed for the application.

Client/Server Model

As mentioned in Section 1.5, the client/server model is not natively supported in the CrypTen MPC framework, so we will not be using it for our implementation. Even though we will not implement this framework, we wanted to introduce it as a possibility as there are benefits to using this approach.

In the client/server model, we assume that the user is not one of the computing parties (cluster of servers treated as a trusted third party). This takes computational stress away from the user, which is beneficial in many (most) situations due to the client not needing to perform any heavy computation, as well as being consistently available for communication. The service provider can be one of the computing parties (part of the consortium of servers), or it can delegate the server role to others. In either design, the user will divide their data into shares as before and send one set of shares to each computing party. The server will also divide its model into secret shares and send these shares to the computing parties. Once the computing parties have the shares, they will compute the specified function and send their output shares to either party individually or to both, depending on what is needed for the application. The client/server model theoretically works with any number of computing parties.

3.2.2 Threat Model

The threat model we will use for our analysis is slightly different for each of the frameworks described above. In both cases, however, we will use a static semi-honest adversary who is able to corrupt up to $|P| - 1$ of the computing parties. Recall from Section 1.5.3, that this is the security guarantee in the CrypTen framework.

For the two party communication model, we have two computing parties, the user and the service provider. For this to be secure, we assume that both parties follow the protocol instructions by running the given circuit and communicating correctly when required. However, while they are performing their computation, either party can try to learn information about the other parties inputs. We assume that the parties do not collude with each other. This is a natural assumption since if the parties were colluding, they would not need to hide their data from anyone and the use of MPC becomes redundant. Additionally, if we add more computing parties to the scheme, we assume that an adversary can corrupt all of these additional computing parties, as well as one of the client or the service provider, and the adversary should not learn anything about the other party's inputs.

For the client/server model, we have a cluster of servers acting as the computing parties. In this case for our threat model, we first assume that all parties follow the instructions of the protocol. Next, we must assume that at least one of the computing servers remains honest through the run of the protocol. The final assumption used is that the adversary must decide on the corrupted servers before the protocol begins, which means we assume that the adversary cannot switch which parties it is corrupting.

3.3 Implementation

We have implemented the two-party computation framework as described above using the CrypTen python package. This will be described in Section 3.3.3. First, in Section 3.3.1, we will describe the dataset used for our analysis. In Section 3.3.2, we will discuss the models used in the implementation. Finally in Section 3.3.4, we will describe the hardware used.

3.3.1 Dataset

The dataset used in this analysis is the same dataset used in Chapter 2. For this analysis, however, we will only use the first fold of the data. Since we are focused on two main

results — the accuracy difference in plaintext vs. MPC inference, as well as the costs associated with making these predictions — it is not necessary to run all five folds of data. We will run our models on the writer independent lowercase dataset. We also use the same preprocessing as used in Chapter 2. Due to the costs associated with using MPC, as we will see in Section 3.4, we round each datapoint to five decimal places as we find that this keeps the accuracy consistent, while limiting the size of the data. Each datapoint is 10816 bytes (104 timesteps $\times$ 13 dimensions $\times$ 8 bytes per element).

3.3.2 PyTorch Models in CrypTen

The models used in this analysis are FCN, ResNet, ResCNN and InceptionTime. We considered all the same models used in Chapter 2, however there were issues with the implementation of some of these models as described below.

To convert a model into a CrypTen model, which is needed for MPC inference in CrypTen, we must first create the model in PyTorch. Since tsai is based on a PyTorch backing, we were able to use the same models as we used previously. This time however, we had to adapt the models into a PyTorch learning environment. We trained each model for 100 epochs using batch size 64, learning rate of 0.001 and the Adam optimizer. We trained the models for 100 epochs, as opposed to 50 as in Section 2, as we found this to be necessary to achieve similar results when training the models in PyTorch [115] compared with tsai [104].

With these trained models, we used CrypTen's "from_pytorch()" function to transform the model from a PyTorch model to a CrypTen model. This function makes use of the Open Neural Network Exchange (ONNX) [105] python framework to help with the conversion process. With this CrypTen model, we must note that the nonlinear functions used in the model are approximated using linear approximations. This will affect accuracy.

We attempted to convert all of the models used in Chapter 2 into CrypTen models, however we ran into some implementation errors along the way. There was an ONNX related error with the implementation of XceptionTime (in regards to the avg_pool_1d layer). Additionally, for all of the LSTM based models, the conversion from PyTorch to CrypTen, had issues with the dropout layers.

Additionally, we must note that ResCNN by default did not work either. This is because alternative versions of the ReLU function, such as Leaky ReLU, PReLU and ELU, are not supported in CrypTen. To work around this issue, we replaced these alternative functions with ReLU. As well, we found that by default there was an issue with the squeeze function in CrypTen; a small alteration to the CrypTen package fixed this issue.

3.3.3 CrypTen Inference

We now have the data used to make inference, and the models we use for predictions. Once we have our model loaded into the CrypTen framework, we encrypt it on behalf of the service provider. We next load in our data and encrypt it on behalf of the user. With a now encrypted model and encrypted data, we are able to make a prediction. In our implementation we make the predictions one datapoint at a time.

3.3.4 Hardware

For this work, we use a Dell PowerEdge R840 server with four Intel Xeon Gold 6230 20-core 2.1 GHz (Cascade Lake) cpus. All experiments were run on a local network, using CrypTen's distributed launcher to run and simulate the communication between multiple parties.

3.4 Results

We have explained how we make character recognition inferences using private data and a (potentially) private model. We say "potentially" here, as the model can be known to the other parties, but is not required to be. As mentioned before, nonlinear computation is not performed in CrypTen and instead it makes use of linear approximations. Due to this we expect slightly different results when performing private computation in comparison to plaintext computation. We analyze the difference in accuracy when performing private inference compared with plaintext inference in Section 3.4.1. In Section 3.4.2, we explore the computational costs associated with making these predictions. We are making inferences on 3946 datapoints. Each of the datapoints are 13×104 element matrices. Additionally, all of the analysis is using two parties for computation, in the two-party communications model described in Section 3.2.

3.4.1 Accuracy

As shown in Table 3.1, we trained 4 different models and obtained the prediction accuracy in the plaintext space and the MPC space. We see that FCN predicted with an accuracy of 70.76% in plaintext and 69.33% in MPC. We see that InceptionTime predicted with an accuracy of 77.52% and only 3.42% in the MPC space. 3.42% is approximately random

Table 3.1: Accuracy of encrypted vs. plaintext inference (%)

	Plaintext	MPC (CrypTen)
FCN	70.76	69.34
InceptionTime	77.52	3.42
ResCNN	74.66	73.80
ResNet	79.62	79.07

Table 3.2: Communication costs of encrypted inference (per prediction)

	Rounds	Data transferred (MB)	Time (seconds)
FCN	31	29.80	0.035
InceptionTime	579	972.72	2.571
ResCNN	62	39.20	0.056
ResNet	112	56.10	0.108

prediction, so we can only assume there is an error here. ResCNN makes plaintext prediction with an accuracy of 74.66% and MPC prediction with an accuracy of 73.80%. Finally, ResNet makes a plaintext prediction with an accuracy of 79.62% and an MPC prediction with an accuracy of 79.07%. As we can see, in all cases (other than InceptionTime) the accuracy cost of making a character recognition prediction is between 0.55% and 1.42%.

3.4.2 Communication Costs

We analyzed the number of rounds of communication, the amount of data sent between parties during this communication, and the time it took to make the prediction. All costs listed are an average over the 3946 predictions made.

First, we look at the rounds of communication needed to make a prediction using each model. For FCN, it takes 31 rounds of communication between the parties, InceptionTime takes 579, ResCNN takes 62 and ResNet takes 112.

Next we look at the data transferred between parties, during their communication rounds to make a prediction. We see the parties send a total of 29.80 MB when predicting using FCN, 972.72 MB when predicting using InceptionTime, 39.20 MB when predicting using ResCNN and 56.10 MB using ResNet.

Finally, we look at the time it takes to make a prediction in seconds. It takes 0.035 seconds to make a prediction with FCN, 2.571 seconds with InceptionTime, 0.056 seconds with ResCNN, and 0.108 seconds with ResNet.

As we see again, InceptionTime appears to have some error associated with it, as each metric is an order of magnitude greater than all of the other models. We also see that FCN has the least amount of costs associated with it, followed by ResNet and then ResCNN.

3.5 Concluding Remarks

In this chapter we explored the use of MPC in making predictions for the character recognition task. We began by motivating why this task would need privacy. Next we described a protocol and our implementation of the protocol. We finished the chapter with some results of our implementation regarding the costs associated with performing character recognition predictions using MPC.

In the future, it would be interesting to implement the client/server model. While we believe that both models described above are applicable in different situations, the client/server model may be more useful in many applications.

Additionally, it would be interesting to explore how more complex writing structures, such as words, paragraphs or longer texts, can be analyzed privately. The motivation explored earlier in this chapter holds for longer forms of text as well, and presumably the costs would grow with the length of the text being analyzed.

Chapter 4

Writer Identification

In this chapter, we will explore what writer identification is and how we can use it for our authentication use case. After that, we will explore and explain why it can be important to have the writer identification task remain private.

4.1 Introduction

We have spent the previous two chapters exploring handwriting analysis and then incorporating user privacy into the analysis procedure. In this chapter, we will shift our focus from the character recognition task to the writer identification task. As mentioned in Section 1.2, writer identification is the task of identifying the writer of a piece of text by their handwriting. In this section we will explore text-independent identification as this is more applicable than text-dependent identification in most scenarios.

Writer identification can be used as the beginning of the framework for an authentication scheme involving handwriting. Ultimately this will be our goal and part of our motivation for this chapter, which will be explored in Section 4.1.1.

Writer identification is part of (along with writer verification) the more generic task of writer recognition, which is a branch of biometrics aiming to authenticate users by their handwriting [125].

In Section 4.1.1, we will explore why we may want to use writer identification in an authentication scheme, before exploring why we would want to incorporate privacy into this task. The remainder of the chapter will explore an implementation of writer identification

on a dataset that has not been used for this task before (at the time of writing this). This will include an explanation of the dataset in Section 4.2 and a deeper analysis of the dataset and its use in the writer identification task in Section 4.3. In Section 4.4 we present the results of the writer identification task. Next, we explore the use of MPC on the writer identification task in Section 4.5 and a brief implementation explanation in Section 4.6. In Section 4.7 we explore the differences in accuracy for the plaintext models and the MPC models. In Section 4.8, we explore the costs associated with training models using MPC. We end with some concluding remarks in Section 4.9.

4.1.1 Motivation

As mentioned in Section 1.2, the writer identification is a "one to many" type of comparison. This task can be used to authenticate users in a closed set. The assumption needed is that if we are given a writing sample, we know that the writing sample belongs to someone in a closed group of writers.

One application we have considered is authenticating a piece of writing in a classroom setting. We may know that the writing belongs to someone in the class, but we want to identify and authenticate the specific writer. This works for an examination, where students must identify themselves as a member of the class during writing, however at a later time, we can use writer identification to confirm that the writing belongs to the student that they are claiming to be. Similarly, in a workplace setting, we may know that the writers work in the office, however we may want to confirm that the writer is the specific writer they claim to be.

In addition to using writer identification to identify people, we must still remember the motivation for the previous chapter in Section 3.1.2, which discussed the need for privacy when users share their handwriting data. As mentioned, handwriting is a biometric which can unwillingly contain personal information about a person. In the last chapter, we only needed to make inference on private data since the models can be trained on public data. As we will see in this chapter, training a writer identification model requires handwriting samples of the users in the closed set. For all of the reasons as to why a user should not want to share their raw handwriting data in the last chapter for inference, we must also consider that the user should not want to share their raw handwriting data for the training of the writer identification model. In this chapter we will first look at writer identification models, before showing how we can train such a model while keeping user data private.

4.2 ONHW Equations Data

As with the OnHW-Chars dataset discussed in Chapter 2, the OnHW-Equations [106] dataset is part of the group of OnHW datasets. The OnHW-Equations dataset contains 55 users writing equations on a piece of paper. Each user writes a total of approximately 195 equations for a total of $10,713$ equations in the dataset. The users write with the STABILO DigiPen, which contains 5 sensors measuring a total of 13 datapoints per timestep. These sensors include 2 accelerometers, 1 gyroscope and 1 magnetometer all measuring in the x, y and z dimensions. The final sensor is a force sensor, which measures the force applied to the tip of the pen. Each equation consists of a sequence of up to 15 symbols. The symbols can be digits from $0-9$, as well as $+, -, \cdot, :, =$.

We chose this specific dataset from OnHW [106] for two main reasons. First, we wanted to use a sequence based dataset (instead of a character based dataset), since we would like to perform text-independent character recognition. For text-independent character recognition, we should be looking at someones writing over time and not only one specific letter, as there should be more information for the model to learn off, per sample. We chose the equations dataset instead of the words based datasets since our ultimate goal is analyzing students handwriting in math.

This dataset only has one fold, split into a training and testing split. Our analysis will be performed using this split.

Note that for our analysis, we round each datapoint to 5 decimal places.

4.3 Analysis of Timesteps and Data

Before using the OnHW-Equations dataset for writer identification, we must perform a timestep analysis of the data. In this dataset, there are an arbitrary number of individual characters written per sample. For our writer identification, we want to truncate each sample at a given number of timesteps to provide consistency. The aim of this analysis is to determine how much time is needed per sample, to obtain data which can be used to reasonably successfully identify the writer.

4.3.1 Initial Results

We trained the models used in the character recognition task in Chapter 2 and described in Appendix A for the writer identification task. We believe that these models should be

appropriate for this task as well. We trained each model for 25 epochs. The results are presented in Table 4.1.

Table 4.1: Timestep analysis

Model	Timesteps			
	1	25	50	100
LSTM FCN	89.31	96.28	96.43	95.81
BILSTM FCN	89.35	96.17	96.28	96.25
LSTM	73.61	90.15	91.37	90.96
BILSTM	80.10	92.26	93.54	93.49
MLSTM FCN	89.17	96.28	96.55	96.21
BILSTM FCN	89.43	96.27	96.52	95.97
FCN	88.93	96.22	96.49	95.96
ResNet	94.21	97.11	97.05	97.22
ResCNN	93.35	96.74	96.72	96.78
InceptionTime	93.87	96.92	97.33	97.18
XceptionTime	92.17	96.93	97.30	97.44
XCM	55.26	79.60	82.44	82.90

As we can see, the results in this table are problematic. Reading the table from right to left, we see that with 100 timesteps per sample, our models obtain accuracies between 90.96% and 97.44% (not including XCM, as this model clearly struggles). This starts off very promising, however as we look to the left, the accuracies remain very promising. In fact, looking at the left hand column of the table, with timestep 1, we see that the accuracy of the models still reaches into the 90's and even gets as high as 94.21%. This is too good to be true. With a single point in time, using sensors that track movement over time, we should not obtain an accuracy this high unless there is inherent biases in the data

In addition to running these deep learning models, we also tested the accuracies of a simple ML algorithm KNN. The results of this analysis are in Table 4.2. We see that even a very simple linear model such as KNN is able to predict the correct user at an accuracy of 87.55% with a single timestep. As we can see, this is clearly problematic and shows that there is some bias in the dataset.

Our first thoughts surrounding this bias was that there was some indicator in the first datapoint of each sample. This was not the case. We performed an analysis fixing the starting point at some value greater than 0. We also performed analysis choosing a random start value for each sample. Neither of these analyses had a substantial effect on the accuracy.

Table 4.2: Timestep analysis KNN

Model	Timesteps						
	1	5	10	25	50	100	200
5NN	87.55	92.74	93.63	94.57	92.92	89.09	81.05
1NN	87.41	92.17	94.72	95.28	95.14	91.17	84.60

We spend the next few subsections describing our process in trying to find why this is happening, and how we fixed the issue.

We begin by analyzing the pen id's that the different users write with, as well as the dates that data was collected. We found some patterns in the dates that the data was collected, as the earlier the data was collected, the lower the user's id tended to be. Similarly, the pens used were consistent with a few user id's before changing pens. However, these patterns were not significant enough to cause the issues we are seeing. Multiple users provided handwritten data on the same days and used the same pens. This cannot explain the issue we are having. Next, we must look at the data itself.

4.3.2 Dimension Reduction part 1

Now that we have diagnosed that there is a problem, we begin with our investigation into the cause of the problem.

Our first attempt is to determine if any individual dimension is the cause behind this error. To determine this, we restricted the data to only be a single dimension. We used 200 timesteps per sample and a single dimension at a time, for data of size 1×200.

We see the results of the analysis in Table 4.3. Each model trained is in this table. If we fix a given row, and look across the columns, we look to see if any accuracy amount is significantly higher than the others. As we can see, for each model, the accuracies remain consistent, no matter which of the dimensions is chosen.

Thus, we find that no single dimension is the cause of our problem.

4.3.3 Dimension Reduction part 2 - with Timestep Analysis

Next, we plotted the time series data of each dimension individually. We only looked at the first 50 timesteps for this analysis because of visual constraints. For this analysis, for visual purposes, we only included a single sample data point per user. We do this to try

Table 4.3: Single dimension writer identification accuracies (%)

Model	Dimension												
	0	1	2	3	4	5	6	7	8	9	10	11	12
LSTM FCN	24.66	24.30	22.81	24.23	22.64	24.49	24.16	25.06	23.64	24.32	25.06	23.73	25.15
BILSTM FCN	25.30	23.85	24.14	23.54	23.09	24.21	23.61	25.68	24.42	25.32	25.15	24.37	25.51
LSTM	16.82	16.91	16.86	16.94	16.49	16.18	16.60	16.77	16.75	17.01	16.25	17.20	16.94
BILSTM	19.14	19.04	16.89	18.52	18.43	17.95	17.93	18.57	17.72	17.81	18.62	18.36	18.76
MLSTM FCN	25.01	23.97	23.14	24.09	23.28	25.06	23.24	24.49	23.83	24.87	24.44	24.99	24.28
BILSTM FCN	25.11	25.08	23.73	24.59	23.64	24.25	23.31	25.65	23.80	25.53	24.68	24.25	24.96
FCN	23.71	24.47	22.69	23.40	22.24	24.89	23.12	24.28	22.95	23.61	24.14	24.18	24.18
ResNet	24.96	22.90	22.90	24.25	23.97	24.16	23.21	23.76	22.31	23.59	23.99	24.09	23.50
ResCNN	23.71	23.21	23.19	23.66	23.26	24.09	22.52	24.30	22.90	24.25	23.87	23.76	23.99
InceptionTime	24.21	24.66	23.40	23.42	22.67	23.57	23.21	24.51	23.00	25.72	24.23	23.16	24.30
XceptionTime	23.54	23.05	21.96	23.16	22.05	22.24	22.26	23.21	22.19	23.52	23.40	22.76	23.38
XCM	14.28	15.28	15.80	16.53	16.30	13.31	15.18	16.96	18.02	16.08	17.88	17.34	16.32

to find any irregularities. Additionally, we have the results of performing a KNN on this single dimension with these 50 timesteps, and a single sample data point per user.

Table 4.4: KNN dimension analysis accuracy (%)

Model	Dimension												
	0	1	2	3	4	5	6	7	8	9	10	11	12
5NN	9.49	6.65	8.50	11.75	10.62	6.75	5.90	7.17	5.95	22.09	30.01	27.8	8.31

We can see that most of the figures in Figure 4.1 do not tell us much information. For the most part, they are signals on a graph and appear to be randomly distributed. That is, until we see dimensions 9, 10 and 11. These dimensions have a clear pattern as the values in these dimensions remain roughly consistent throughout the 50 timesteps in all three dimensions. Additionally, these consistent values do not appear to overlap between users. Additionally, in the KNN analysis, we see a clear difference in the accuracy in dimensions $9, 10$ and 11, compared with the rest of the dimensions.

4.3.4 What This Means for the Data

The above analysis gives us a good indication that dimensions $9, 10, 11$ (when calling the first dimension, dimension 0) are the problematic dimensions for our writer identification task. These three dimensions appear to hold some bias between users, that is independent of the user's writing. These three dimensions belong to the magnetometer sensor on the

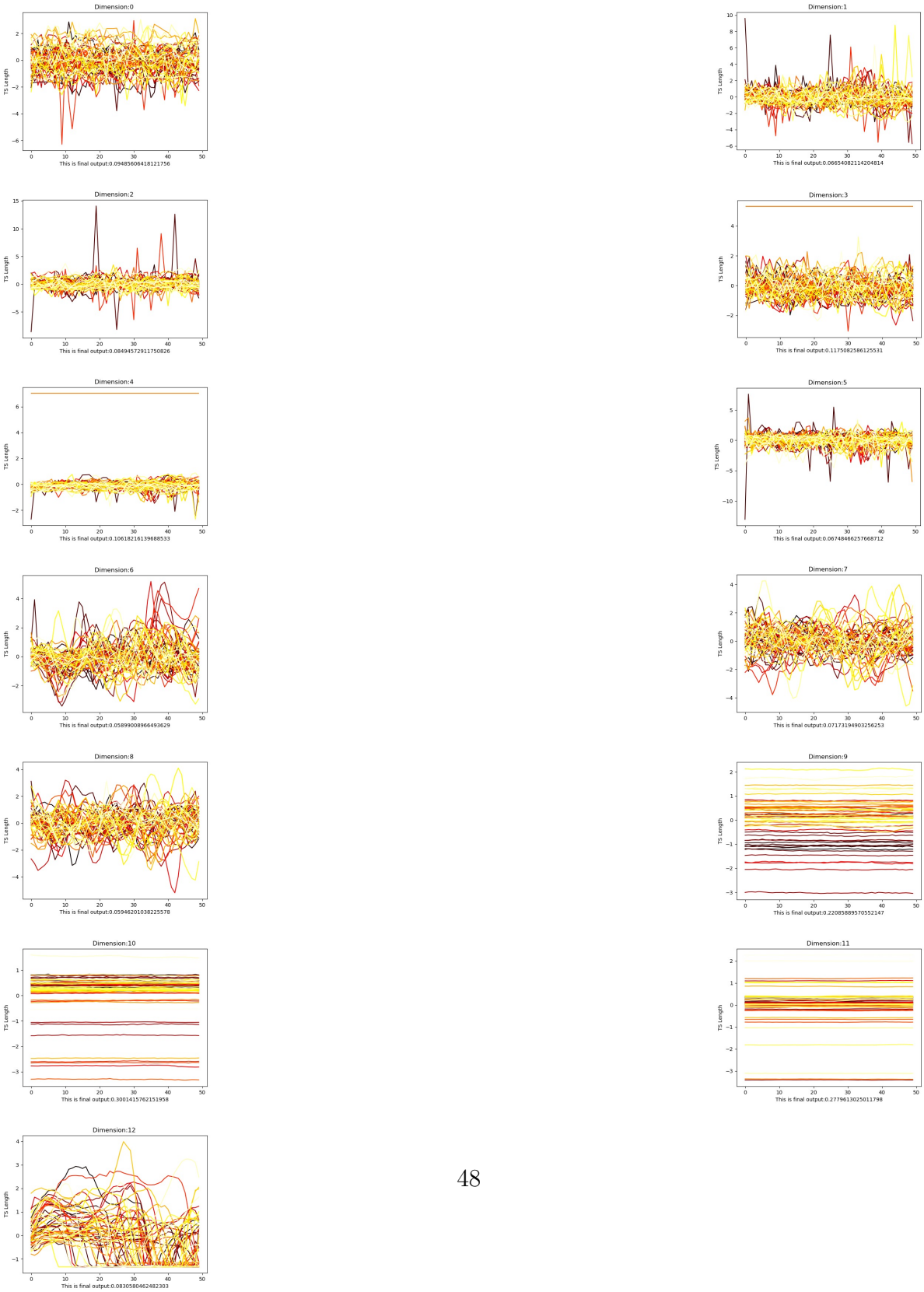


Figure 4.1: KNN dimension analysis

pen. This gives us reason to believe that the magnetometer has some bias in it. For the rest of our analysis with the OnHW-Equations dataset, we will remove these three dimensions (or the magnetometer sensor).

To confirm that these dimensions are problematic, we first reduce our data to only look at these three dimensions. We then reduce the data further to have 200 timesteps per sample and train our writer identification models. Next, we reduce the data even further, to contain 1 timestep per sample, and again train our writer identification models. We present the results in Table 4.5. We see that with a single timestep, these models are able to correctly predict a user, up to 89.92% of the time, which shows the significant bias in this sensor.

Table 4.5: Writer identification with only dimensions 9, 10 and 11

Model	Timesteps	
	1	200
LSTM FCN	88.02	90.86
BILSTM FCN	88.58	90.43
LSTM	83.49	89.39
BILSTM	86.56	88.73
MLSTM FCN	88.63	90.95
BILSTM FCN	88.44	91.09
FCN	88.02	90.86
ResNet	88.92	92.80
ResCNN	89.29	92.75
InceptionTime	89.43	92.37
XceptionTime	88.87	92.52
XCM	83.11	82.14
KNN	87.12	90.48

Now that we have removed the problematic dimensions, we can finally continue with our timestep analysis. Recall that the goal of this analysis is to determine how many timesteps we should include per datasample.

We see the results of our timestep analysis in Table 4.6. First we notice from this table that with more timesteps we see a higher accuracy, and as the timesteps decrease so does the accuracy. This is what we would expect from our data. Next we see that when using 200 timesteps the accuracy is high and when we lower the number of timesteps, the accuracy dips. We notice that when we increase the timesteps past 200, the accuracy seems

Table 4.6: Writer identification with dimensions 9, 10 and 11 removed (%)

Model	Timesteps						
	1	5	10	25	50	200	300
LSTM FCN	35.05	46.04	52.64	65.31	67.82	71.06	68.66
BILSTM FCN	35.57	45.66	54.72	64.46	68.29	69.07	69.69
LSTM	21.18	27.03	35.47	64.46	56.49	62.06	58.81
BILSTM	26.89	34.43	42.97	57.72	63.00	65.37	64.84
MLSTM FCN	36.04	46.56	57.74	63.10	68.00	70.68	71.25
BILSTM FCN	35.57	46.37	52.88	63.90	67.77	70.68	71.11
FCN	35.66	45.85	52.03	62.15	66.59	69.92	69.34
ResNet	35.42	47.41	54.29	64.61	71.59	79.06	79.68
ResCNN	34.48	43.21	49.53	62.77	64.42	74.04	72.67
InceptionTime	34.76	49.48	59.39	72.44	79.38	89.77	88.93
XceptionTime	37.03	53.82	64.06	77.39	84.43	90.10	88.49
XCM	23.63	28.87	34.91	45.02	46.15	44.39	50.69

to plateau. To note, 200 timesteps is two seconds worth of data. We wanted to keep the number of timesteps relatively low to limit the amount a user would need to write during the enrollment phase. For our models, each user has approximately 150 samples in the training set, which would equate to approximately 5 minutes of writing, sliced up into two second intervals. We believe this to be a reasonable amount of writing for a user to enroll in the program.

To note, with this datastructure, each data sample is 16000 bytes (200 timesteps $\times$ 10 dimensions $\times$ 8 bytes per element).

4.4 Plaintext Model Explanation and Results

Now that we have set the dimensions of the data for our analysis, we must explain our plaintext results. For these results, we have fixed our timesteps at 200 and have removed dimensions $9, 10$ and 11. We extend this training for 500 epochs, as this is where we find all of our models accuracy plateau.

We see that in the best case, we obtain a writer identification accuracy of 91.57%, with the XceptionTime model. Overall, we see that InceptionTime (89.77%) and XceptionTime (91.57%) are the best performing models, by quite a large margin. ResNet obtains an

Table 4.7: Writer identification plaintext results

Models	Accuracy
LSTM FCN	73.33
BILSTM FCN	74.14
LSTM	60.97
BILSTM	64.09
MLSTM FCN	74.47
BILSTM FCN	75.46
FCN	73.38
ResNet	86.26
ResCNN	83.89
InceptionTime	90.86
XceptionTime	91.57
XCM	48.60

accuracy of 86.26% and ResCNN obtains an accuracy of 83.89%. The LSTM based models are weaker, along with FCN, all having accuracy below 75.46%. Finally, XCM has an accuracy of only 48.60%. XCM has performed poorly consistently when attempting the writer identification task.

4.5 MPC Framework

As mentioned in Section 4.1.1, not only are we interested in the writer identification task, but in making the writer identification framework private.

In Chapter 3, we saw the use of MPC to make inference on private user data. This however is not sufficient for our writer identification task. When performing writer identification, it is important that given a sample, we can guarantee that the sample has been written by one of the closed set of writers used in the training set of the models. For this reason, we cannot train our models on public data and must train it on data specific to the group. Since we are motivated to keep these users handwriting data private, not only do we need the users data to be private when making inference, we must also train the model on private data.

We will look at two potential frameworks for the private writer identification task. In both settings we assume that there are n users in the closed set of writers. Next, we

assume there is a service provider who holds the model and would like to provide a service involving the writer identification task.

4.5.1 $n+1$ Party Communication Method

In the $n+1$ party communication method, we assume that each of the parties in the closed set, as well as the service provider, are all part of the computation.

We begin with each of the n users recording their raw handwriting data and dividing it into $n+1$ sets of shares. We also have the service provider divide their model parameters into $n+1$ sets of shares. These $n+1$ parties will act as the computing parties in the protocol. Each party will send a set of shares to each other party. Collectively, these $n+1$ parties will jointly (and privately) train the writer identification model. This model will never be revealed to any of the parties, as the trained model will remain in the form of the shares.

When it comes time to make a prediction using this model, the writing data in question will be divided into shares and distributed amongst the $n+1$ parties. The parties will then use the shares of the trained model that they already have, (and communicate) to make a prediction. The prediction will be in the form of these hidden shares. Finally, when it is time to reveal the output of this prediction, the parties share their output shares with whoever the protocol requires revealing the prediction to.

The model has significant weaknesses in that all $n+1$ parties must consistently be available for communication. As will be seen later, these models take a significant amount of time to train, which in the $n+1$ party communication method requires the closed set of writers to be available throughout. Additionally, these parties must be able to perform the computation on their device and send large amounts of information back and forth. This is not feasible in most practical settings.

4.5.2 Client/Server Model

The client/server model should be preferred for this task in many situations, because unlike the method above, it delegates the computation to third party servers.

The client/server model involves each of the n users acting solely as the input parties in the protocol. The service provider will also act as an input party, to provide the model parameters. Unlike the method described in Section 4.5.1, these input parties will not be involved in the computation, which takes a significant computational load off the users.

We will assume that there are p servers acting as the computing parties for the protocol. To run the protocol, each input party divides their data into p shares, and sends each share to a different server. The servers will perform private computation on the shares of data, to train the private model. This model will remain private, divided amongst shares between the computing parties. When it comes time to make a prediction, the relevant user (writer), will divide their handwriting data into p shares and send each share to a different computing party. Again, the computing parties will provide inference before sending their output prediction share to the required party or parties.

4.5.3 Threat Model

The threat model for this section is as described in Section 3.2.2.

4.6 Implementation

When implementing this MPC training in CrypTen, we simulate the client/server model. However as mentioned in Section 1.5, CrypTen does not natively support the client/server model. For this reason, we simulate this by providing one of the computing parties with the raw data and having this party divide the data into shares and send it to the remaining parties. While this is not ideal from an implementation perspective, our goal with this analysis is to determine the costs associated with training models in the MPC space. While this simulation may not accurately simulate the privacy aspect of the protocol, it will accurately simulate the costs. We can assume that few additional costs would be incurred by each party individually dividing their own data into shares and sharing it amongst the computing parties, as mentioned in Section 4.5.2.

After the data (and the model parameters) have been shared, we train the model using CrypTen's model training framework [80]. CrypTen has support for most functionality we require for training, however the only optimization function supported is Stochastic Gradient Descent (SGD), so this will be used for our implementation.

4.7 Taking Plaintext Model to MPC

As explored in Chapter 3 and as we will see later in Section 4.8, the addition of privacy (through MPC), comes with significant costs. These costs come in the form of time and

data size of the communication between parties. Unlike in Chapter 3, where we were focused on MPC inference, now we are focused on MPC training of models. When training these models, these costs are much larger than in inference. For this reason, we will not be able to use sophisticated models. We will perform an analysis of the models trained in Section 4.7.1, as well as some analysis of the number of epochs required.

To measure time, data sent between parties, and the rounds of communication, we use CrypTen's built-in functions [80].

4.7.1 Plaintext vs. CrypTen Accuracy Results

We will be using a Convolutional Neural Network (CNN) as the base architecture for our models. We use a CNN due to its flexibility with model parameters and simple nature, which should make it more transferable to MPC than LSTM's or other complex models [4]. With this architecture, we experiment with different parameters, to try to obtain the best quality results whilst keeping the parameters and layers relatively lightweight.

When exploring CNN's, we will consider the following base architecture: a CNN with one convolutional layer with 4 filters and a kernel size of 3 and a ReLU activation, followed by a max pooling layer with kernel size 2. This layer is followed by a linear layer with 100 output classes and another ReLU activation function, followed by a final linear layer with 55 output classes. These 55 output classes represent the 55 writers we are attempting to predict in the writer identification task.

All results below are with a SGD optimizer, a learning rate of 0.001, and a batch size of 128. Each model was trained for 1000 epochs as we found that the models required this high number of epochs in order to converge using SGD. When using other optimizers such as Adam, we found that the models were able to converge quicker, however we wanted to replicate the model parameters which would be used when training the models privately using CrypTen. We must note that every time a model is trained, there is a level of randomness involved in the training process, which leads to different model weights in the trained model. Thus each time a model is trained, the final trained model is unique and can result in different accuracies.

The results of the plaintext training are recorded in Table 4.8. We begin with the base model, and name each subsequent model according to the change in architecture or model parameters, from the base model.

As we can see in Table 4.8, the best accuracy occurs in the model without the max pooling layer obtaining an accuracy of 81%. We find that our base model obtains an

Table 4.8: Plaintext training of models

Model	Accuracy (%)
Base model	74.28
8 filters in convolutional layer	76.74
16 filters in convolutional layer	78.64
Kernel size 1	68.40
Kernel size 5	69.78
Kernel size 7	66.89
Kernel size 9	46.47
Without max pooling layer	81.00
Without ReLU activations	71.34
Without Dropout Layer	74.23
2 convolutional Layers	66.18
3 convolutional Layers	50.83
4 convolutional Layers	27.62

accuracy of 74.28%. Changing the kernel size appears to have a negative effect on the accuracy. Additionally, we find that increasing the number of filters in the convolutional layer increases the accuracy of the models. Finally, we find that adding more convolutional layers, while keeping other parameters consistent, drastically decreases the accuracy. I believe we would need to alter other parameters to get better results when increasing the number of convolutional layers.

After training these models in the plaintext space, we wanted to take one of these models and train it in the MPC space, in order to get a sense of the accuracy difference in training models privately. We train the model privately for 500 epochs.

To note, there was one major issue we ran into when training these models. After approximately 70-100 epochs of training the model privately, the test loss values would suddenly spike. Exploding gradients appears to be a problem in other research using CrypTen, and work has gone into trying to limit this from happening [48, 10]. For our research however, we save the trained model in plaintext every 50 epochs. We then load in and encrypt the saved model and continue training. This is not an ideal solution, however it is a reasonable workaround, since we are aiming to obtain an accuracy result from our model. We can think of this approach as having a trusted third party responsible for this model reset.

Due to time constraints, we were only able to train one model in MPC. For this reason,

Table 4.9: Plaintext training vs. CrypTen training

Model	Accuracy
Plaintext Model	74.28
MPC Model	77.45

we chose to train the base model as described above.

The results are presented in Table 4.9. We see that when training the base model in MPC for 500 epochs we obtain an accuracy of 77.45%. Interestingly, this is a better accuracy than we found when training the model in the plaintext space. While we do generally expect lower accuracies when training models in MPC due to the approximation of functions in the model, there is still an element of randomness that goes into training a model. As mentioned, due to time constraints, it was not possible to attempt to train this model again in MPC to see if we obtain a better or worse accuracy.

4.8 CrypTen: Communication and Time Analysis

We next explore the costs associated with training our models in CrypTen's MPC framework. We begin with the model chosen from Section 4.7.1.

In this section, we will analyze the costs associated with changing different parameters when training our model in MPC. We begin with changing the number of computing parties involved in our protocol in Section 4.8.1. This will change the amount of communication between parties, and is the main MPC parameter we can change. In Section 4.8.2, we explore the effect of changing the batch size. Next, in Section 4.8.3, we explore the effect that changing the number of filters in our convolutional layer has on the computational costs. In Section 4.8.4, we look at the effect that changing the size of the kernel has on the costs. After that, in Section 4.8.5, we explore the costs associated with increasing the number of convolutional layers. The final two sections involve exploring the effect of including the max pooling layer in Section 4.8.6 and including the ReLU activations in Section 4.8.7.

For each analysis, we train our models for 5 epochs and record the average results through these 5 epochs. We will be using a learning rate of 0.001 and, unless otherwise specified, a batch size of 128. Additionally, unless otherwise specified, we will be using two computing parties. When measuring time, data sent between parties, and the rounds of communication, we use CrypTen's built-in functions [80].

4.8.1 Number of Parties

We begin our cost analysis by determining the costs associated with having different numbers of computing parties involved in the protocol. Recall that the threat model in CrypTen is designed to be secure against an adversary corrupting $n-1$ computing parties. Thus, the more computing parties we have, the less reliant we are on any one party to be honest.

Table 4.10: Costs associated with the number of computing parties (per epoch)

Number of parties	Data sent (GB)	Rounds	Time (s)
2	142.9	20296	103
3	348.6	46266	278
4	507.5	46266	501
5	666.3	56801	1484
6	825.1	56801	2479
7	984.0	56801	3916
8	1142.8	56801	2957

Table 4.10 lists the costs associated with training our model with $2, 3, 4, 5, 6, 7, 8$ computing parties. First, we notice that the data sent (even with only two parties), is very high. The data sent between parties begins at 142.9 GB per epoch of training, increasing linearly all the way to 1.1428 TB of data with 8 parties.

Interestingly, the number of rounds of communication increases at powers of 2 parties. 2 parties requires 20296 rounds of communication, 3 and 4 parties requires 46266, and $5, 6, 7$ and 8 parties requires 56801 rounds of communication. According to CrypTen [80], the pattern regarding the growth of the number of rounds increasing when the number of parties increases to 2^{k+1} from 2^k, is due to the implementation of the comparator.

Next, we see that the time taken to train each epoch appears to grow exponentially with the number of parties, until 8 parties. I am not sure what caused this issue.

4.8.2 Batch Analysis

Next, we analyse the effect that batch size has on the computational costs. The batch size represents the number of data samples that the model sees before updating its parameters.

We explore the costs associated with having a batch size of $8, 16, 32, 64, 128, 256$ and 512. From Table 4.11 we notice that the data sent appears to follow an exponential decay

Table 4.11: Costs associated with the batch size (per epoch)

Batch size	GB	Rounds	Time (s)
8	229.4	328252	311
16	184.1	164126	222
32	161.2	81918	149
64	149.4	40814	169
128	142.9	20296	106
256	140.1	10182	138
512	135.3	4980	117

type of distribution as we increase the batch size. With a batch size of 8, the computing parties send 229.4 GB of data between them, and with a batch size of 512, the computing parties send 135.3 GB of data between them. We also see that the number of rounds of communication appears to follow an exponential decay type of distribution, beginning with 328252 rounds with a batch size of 8 and ending with 4980 rounds with a batch size of 512.

4.8.3 Number of Filters in the Convolutional Layer

Here we analyze the costs associated with the number of filters in the convolutional layer of our CNN. The number of filters is one of two parameters in the convolutional layer, along with the size of the kernel. The number of filters represents the size of the output of the convolutional layer [115].

Table 4.12: Costs associated with the number of filters in the convolutional layer (per epoch)

Number of Filters	GB	Rounds	Time (s)
4	142.9	20296	127
8	283.3	20296	437
16	564.2	20296	502
32	1125.9	20296	2176

From Table 4.12 we see that the number of filters appears to have a linear relationship with the amount of data sent between parties. The amount of data sent between parties increases from 142.9 GB to 1.1259 TB, when increasing the number of filters from 4 to 32.

Additionally, we observe the number of rounds is unchanged with the number of filters. Finally, the number of seconds it takes to train an epoch grows quite quickly. For this reason, it quickly becomes infeasible to train models with a large number of filters. This is unfortunate since if we recall in Table 4.8, increasing the number of filters in the convolutional layer has a positive impact on the accuracy.

4.8.4 Kernel Size in the Convolutional Layer

In addition to the number of filters, the other parameter we can change is the kernel size of the convolutional layer. The change of the kernel size n changes the $n \times n$ convolving kernel used in the computation [115].

Table 4.13: Costs associated with the kernel size in the convolutional layer (per epoch)

Kernel size	GB	Rounds	Time (s)
1	179.8	20296	135
3	142.9	20296	127
5	106.7	20296	102
7	71.3	20296	63
9	36.5	20296	47

In Table 4.13, we see that there is an inverse linear relationship between the kernel size and the amount of data sent between parties. With a kernel size of 1, 179.8 GB of data is sent between parties, continuing down to 36.5 GB of data when using a kernel size of 9. Additionally, the number of rounds does not change with the kernel size. Finally, we see approximately an inverse linear relationship between the kernel size and the time it takes to train an epoch. With a kernel size of 1, it takes 135 seconds to train and with a kernel size of 9, it takes 47 seconds.

4.8.5 Number of Convolutional Layers

In this section, we explore the costs associated with increasing the number of convolutional layers in our model. For simplicity, we keep the number of filters and the kernal size consistent.

In Table 4.14, we see a linearly increasing number of rounds when increasing the number of convolutional layers. When there is one convolutional layer, there are 20296 rounds,

Table 4.14: Costs associated with the number of convolutional layers (per epoch)

Number of convolutional layers	GB	Rounds	Time (s)
1	142.9	20296	127
2	112.3	21831	158
3	80.8	23366	135
4	48.4	24901	81

compared to 24901 rounds with four convolutional layers. Additionally, we see a linearly decreasing amount of data sent between parties. With one convolutional layer, there is 142.9 GB of data sent between the parties, compared with 48.4 GB when there are 4 convolutional layers.

4.8.6 Max Pooling Layer

Next, we explore the costs associated with including a max pooling layer in our model. A max pooling layer takes the maximum value of a given neighborhood around a value and replaces the value with this maximum [58].

Table 4.15: Costs associated with including a Max Pooling Layer (per epoch)

Max Pooling Layer	GB	Rounds	Time (s)
Yes	142.9	20296	127
No	60.4	16412	76

In Table 4.15, we see that the inclusion of a max pooling layer, dramatically increases the costs. We see an increase in data communicated from 60.4 GB to 142.9 GB. We also see an increase in rounds from 16412 to 20296. Finally, we see an increase in the time it takes to train the model, from 76 seconds to 127 seconds.

4.8.7 ReLU Activation

The last part of the model we will be exploring is the inclusion of the ReLU activation function. This activation function takes the maximum of every datapoint and 0, and replaces the datapoint with the result.

Table 4.16: Costs associated with including a ReLU activation (per epoch)

ReLU activation	GB	Rounds	Time (s)
Yes	142.9	20296	127
No	109.6	18670	83

As we can see in Table 4.16, including the ReLU activation also increases the costs quite dramatically. The data sent between parties increases from 109.6 GB to 142.9 GB. Additionally, the number of rounds of communication increases from 18670 to 20296. Finally, the time increases from 83 seconds to 127 seconds.

4.9 Concluding Remarks

In this chapter, we began by discussing the OnHW-Equations dataset and its use in the writer identification framework. This proved trickier than originally anticipated as there were problems with the data. We identified the problem as being the dimensions of the data associated with the magnetometer sensor having some inherent bias associated with the writer who wrote a sample with this sensor.

Once we found the issue, we removed these dimensions from the data and proceeded with the writer identification task. First, we determined the number of timesteps needed per sample as 200.

Next, we trained a combination of complex deep learning and simpler CNN models to perform the writer identification task. We found that the complex deep learning models were very accurate and while the accuracy dropped when using the simpler CNN models, they still obtained a competitive accuracy.

We picked one of the better performing models, with a relatively simple architecture, and trained it in CrypTen's MPC framework.

Finally, we analyzed the computational and communication costs associated with different parameters in the model, as well as different number of computing parties.

4.9.1 Future Work

Ours is the first work on performing writer identification with the OnHW datasets. Further work in this area would be exploring this task with different models or approaches

regarding the data itself including the number of timesteps or the dimensions considered. Additionally, the biases with the magnetometer sensor should be explored further.

With more time and resources, it would have been interesting to see how a more complex model, such as including more filters in the convolutional layer, could train in the MPC space.

Finally, it would be interesting to see how these models translate into other MPC frameworks such as MP-SPDZ.

Chapter 5

Speaker Authentication

The goal in this chapter is to continue our analysis into a new form of biometrics, from handwriting authentication to speaker authentication. Additionally, we make use of a new MPC framework in MP-SPDZ [71]. In Chapter 4, we have shown that we can get strong prediction accuracy using simple CNN's and more complex models. In this chapter we would like to explore the costs associated with using different MPC protocols and slightly more complex models for training and inference. We conduct a brief survey to try and get a sense of what people think of the tradeoff between privacy and the costs associated with it. This chapter comes from a course project done with Anuradha Kulkarni and Vijay Ravi from the University of Waterloo.

5.1 Motivation

Voice or Speaker authentication as a form of biometric authentication is growing in use [137]. In a Speaker authentication framework, a user will attempt to authenticate themselves to a device or service, using their voice. One advantage of voice authentication is that it is relatively low cost and very usable compared to some other authentication types. For example, passwords often require memorization or at least a password manager, so it may be useful and more usable to perform phone based authentication using voice instead of a password [150].

Voice authentication has been implemented and used in mobile devices for authentication purposes. Google has used voice authentication to unlock Android phones [150, 90] and Tencent has used voice authentication to log in to WeChat [150, 141, 90].

At the heart of this type of biometric authentication is the speaker recognition task as described in Section 1.3. Recall that speaker recognition is divided into three tasks: speaker verification, closed set identification, and open set identification. All three of these tasks can be transformed into an authentication scheme depending on the particular use case. We will specifically be exploring the speaker verification task. In this task, a user will attempt to confirm their identity as a specific user. Their voice sample will be analyzed along with the enrolled voices in the system to determine if it is the correct speaker.

Voice authentication schemes, while very usable for the user, can be subject to various attacks. They can be vulnerable to replay attacks [150], where an adversary tries to replay a recorded sample of the users voice in order to spoof the authentication system. This is clearly problematic and there are many proposed solutions to this problem, including liveness detection [150, 90]. In addition to replay attacks, there are attacks aimed at perturbing a random person's voice (unrelated to the initial speaker) until the new noisy voice signal gets accepted by the speaker recognition system [28]. The aim of this specific attack was that the voice would still sound like a normal voice to a bystander and thus would be hard to detect. It has also been shown in [145] that with sufficient recorded samples of someone's voice, a synthetic voice can be created to mimic the original speaker voice and fool speaker recognition systems.

While there are constantly attempts to thwart these attacks using more sophisticated speaker recognition modelling [30, 29], we believe that another important defence is to keep the user's voice out of the hands of adversaries as much as possible. For this we propose incorporating MPC into the voice authentication framework. When authenticating yourself using your voice, naturally the user must send their voice to the authenticating service. We believe that this process should be made as secure as possible, so that the user's raw voice signal is never shared outside the local device. To do this we will use MPC as part of our training and inference procedures as described below.

The rest of the chapter is as follows. Section 5.2 is the background, Section 5.3 presents the protocol, Section 5.4 discusses the implementation details, Section 5.5 focuses on a survey surrounding people's thoughts on privacy and the costs associated with it, and Section 5.6 presents the conclusion along with the limitations and future work.

5.2 Background

In order to build our implementation in Section 5.4 and the survey in Section 5.5, some background information is necessary.

In Section 5.2.1 we will provide the relevant background information for voice verification and recognition, as well as provide a framework for voice authentication. In Section 1.4 we have discussed MPC and some relevant background information, however we will expand on this in Section 5.2.3 where we explain the main MPC protocol for speaker authentication explored in this chapter. Additionally, in Section 5.2.4 we will describe a second protocol which is to be used for parts of our analysis. Finally in Section 5.2.5 we describe the main framework which will be used for implementing the protocol.

5.2.1 Voice Authentication Framework

Voice authentication is the process of authenticating a user through biometrics, using their voice. The key task involved in voice authentication is the speaker verification (or recognition) task as described in Section 1.3. This is the task of identifying whether an unknown speaker is a specific (enrolled) speaker, or not [28, 68]. Figure 5.1 is a flowchart of how voice authentication could work. We split voice authentication into two phases: the enrollment phase where a user enrolls with the service, and the verification phase where an unknown user tries to authenticate themselves as the enrolled user. The unknown user should only succeed with authentication if they are in fact the enrolled user.

Enrollment Phase

The enrollment phase begins with the user recording their voice as a raw audio signal. The user will preprocess their data into a format that is more readable into a model, before they send it to the server. Once the server receives sufficiently many preprocessed signals, they will train a user specific verification model. The more audio signals the server receives in the enrollment phase, the stronger the user specific verification model should perform.

Verification Phase

The verification phase involves an unknown user, claiming to be the enrolled user. We begin with the unknown user's recorded voice as a raw audio signal. The user preprocesses their voice and sends it to the server, as before. Now, the server will use the model created in the enrollment phase to provide a score on the likelihood that the unknown user is the enrolled user. This score, along with some predetermined threshold value, will lead to a decision about the identity of the user.

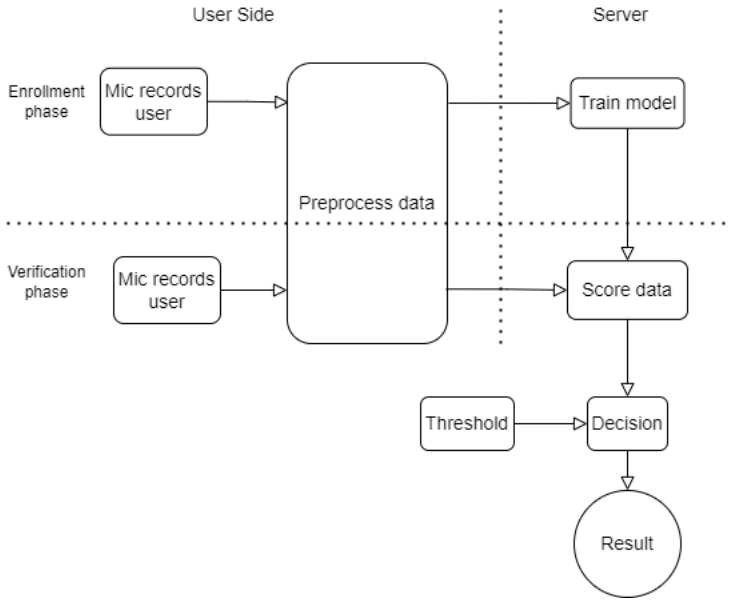


Figure 5.1: This figure shows an example of the voice authentication framework

5.2.2 Multiparty Computation

For the study in this chapter, we will be exploring the case of a single party with private inputs and a second party with a private model. We will make use of external servers to help with the computation of the function, as described in Section 1.4.2 on the client/server model in MPC.

We will make use of the protocols described in Sections 5.2.3 and 5.2.4 to help us achieve these goals. Before moving on to the implementation, results and protocols used in our analysis, we must define how we would like MPC to add privacy to our specific problem.

For our framework, privacy means that the computing parties (or servers) do not learn the authenticating users' voice (or be able to derive any information about their voice), through the creation of the verification model (enrollment phase) and the inference itself (verification phase). We would also like our MPC protocol to be correct. In this case, this means that through the MPC process, we are able to obtain the correct desired output, or in other words, the same (or close the same) prediction that a plaintext model would generate. We recognize that the prediction will not be identical in every scenario, as some functions used to make these predictions are approximations when performed through MPC.

5.2.3 Semi-Honest Three-Party MPC Protocol

For the implementation of our voice authentication framework in the MPC setting, we will make use of both protocols secure against a semi-honest adversary and a malicious adversary corrupting one of the computing servers. The protocol used that is secure against a semi-honest adversary is described in [9]. This protocol runs in the client/server model with three computing parties acting as servers. Recall that a semi-honest adversary will follow the protocol, but might try to learn information about the inputs.

The protocol supports Boolean or arithmetic operations. It is based on replicated secret sharing and works in the ring of integers modulo 2^k [71, 9, 16].

In the protocol, the secret shares are dependent on each other in such a way that knowing any two shares are sufficient to determine the third. This is why the protocol requires an honest majority. However, with only a single share, the adversary is unable to learn anything about the input data.

5.2.4 Malicious Four-Party MPC Protocol

We will also explore a four-party protocol secure against a malicious adversary. The protocol used will be as the one described in [41]. This protocol will also employ the client/server model for computation. As with the protocol described in Section 5.2.3, it is also based on replicated secret sharing and works in the ring of integers modulo 2^k. At a high level these protocols look similar, however they differ in some fundamental structures which allow this protocol to be secure in the presence of a single malicious computing party, as opposed to a single semi-honest computing party as above. For more details, please see [41].

One main difference, besides working with four parties as opposed to three, is the cheating identification step which allows for security against a malicious computing party. The cheating identification will detect when one of the parties deviates from the protocol. The honest parties will be able to make use of this identification step and abort the protocol run to prevent the adversary from learning unwanted information. This is one of the ways to incorporate security against a malicious adversary who, as mentioned in Section 1.4.1, is allowed to deviate from the run of the protocol without compromising security.

5.2.5 MP-SPDZ

To help us implement our voice authentication framework with MPC, we make use of the MP-SPDZ framework [71]. This framework extended the SPDZ-2 implementation to

include more than 30 protocols [71]. One main contribution of MP-SPDZ is the ability to easily swap between these protocols. The framework divides its protocols into the types of adversarial strengths mentioned above (semi-honest and malicious) as well as the number of dishonest parties (honest majority and dishonest majority). As mentioned above, each of these protocols, with their differing adversarial strengths, comes with different costs associated with them. The MP-SPDZ framework allows for an easy transition between these protocols, in order to facilitate choosing the most appropriate protocol for a given task.

MP-SPDZ uses a python-like language to assist with the creation of MPC functions. It provides a high-level, easy-to-use, interface to help with the addition, multiplication, comparison, and access of secret values. The implementation also includes simple datatypes such as vectors and arrays of these secret values. This high-level interface is built upon a low-level efficient implementation of the protocols.

In addition to these basic computation methods, the framework offers more complex matrix based computation such as inner-product, matrix multiplication, and 2D convolution [71]. We will make use of these specialized computations as well as the simpler computation listed above to build our voice authentication framework.

MP-SPDZ is similar to CrypTen in that they are both python based programs to help programmers incorporate MPC into their programs. MP-SPDZ is fundamentally different in design in multiple ways. First, as mentioned in Section 1.5, CrypTen was designed to not use a compiler for their own domain specific language. MP-SPDZ on the other hand does have this. MP-SPDZ naturally supports TensorFlow [2], with some support for PyTorch, whereas CrypTen is designed the other way. Both of these approaches are valid and both MP-SPDZ and CrypTen have their own pros and cons.

5.3 Protocol

In this section, we explore the threat model, voice authentication framework, and speaker recognition models used in our version of voice authentication with MPC.

5.3.1 Threat Model

The strength of the MPC protocol's threat model is determined by how strong we make the adversaries and by how many computing parties (or servers in this case) that the adversary

is allowed to corrupt. For our protocol, we consider two threat models. In both of these cases, we assume the client/server model as described in Section 1.4.2.

The first threat model, and the primary model used in our implementation and results below, is described in Section 5.2.3. This is a three-party protocol in the client/server model. In this case, as mentioned, we will have a semi-honest adversary who can corrupt one of the three computing parties.

The second threat model we consider is the four-party malicious threat model with an honest majority, as explored in Section 5.2.4. This model supports four computing parties in the client/server model. We assume that one of these four parties can be maliciously corrupted. This second threat model considered is clearly stronger than the one mentioned above, however, due to computational and time costs, the first threat model was selected.

Stronger threat models were also experimented with and considered. However, due to high computational costs, these additional threat models were not used for our final experiments. Other threat models to be explored could include dishonest majority models, where the adversary can corrupt more servers. Some more advanced MPC protocols which are too computationally intensive, but allow for strong adversaries include [39, 72].

5.3.2 Voice Authentication Framework

Voice authentication with MPC uses a similar framework as voice authentication described in Section 5.2.1, with some additional steps to keep the users' inputs private throughout the entire process. As before, the protocol will still contain two phases, the enrollment phase and the verification phase as shown in Figure 5.2. First, we detail the enrollment phase where a user will enroll with the service and then we detail the verification phase where a user will try to authenticate themselves to the service. In this case, the authentication service provider will be outsourcing the computation to computing servers.

Enrollment Phase

The enrollment phase is where the model will be trained. However, we would like to keep both the inputs private throughout the training of the model, as well as keeping the trained model itself private. It is important that the model also remains private as otherwise an adversary with access to the model might be able to retrieve some information about the user's voice with access to the model.

The enrollment phase begins on the user side, where the enrolling user speaks some prespecified amount, and their voice is recorded using a microphone. The raw data signal

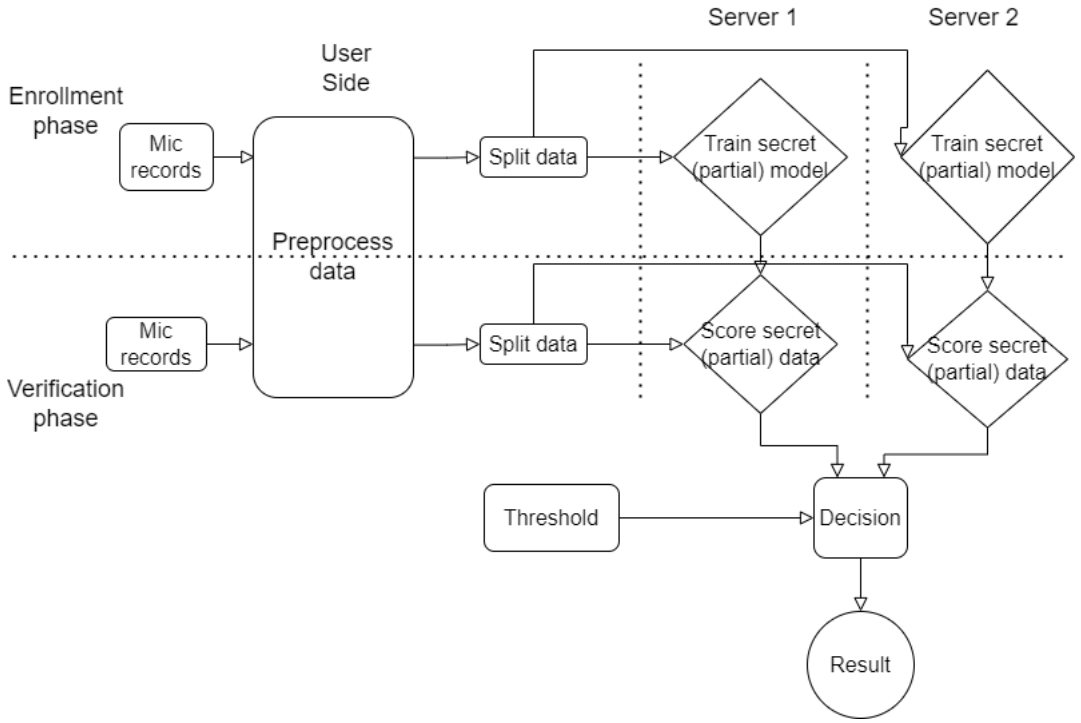


Figure 5.2: This figure shows an example of the voice authentication framework using MPC in the client/server model

from their recorded voice is then divided into utterances to be used as enrollment voices by the system. After these utterances are recorded, the user is responsible for preprocessing the data, which will be explained in greater detail in Section 5.4.2.

After preprocessing the data, the user will split their preprocessed voice data into shares using the protocols specified in Section 5.2 (either the semi-honest three-party protocol in Section 5.2.3 or the malicious four-party protocol in Section 5.2.4). These shares are then sent to the respective computing servers to train the user-specific model. Figure 5.2 shows the user sending the shares to two servers. We chose two servers for visual purposes — the number of servers in practice would depend on the MPC protocol chosen.

After each server receives the share of the user's voice data, they will train a private user-specific model. The parameters of this user-specific model will be specified by the authenticating service. Once trained, these models will have private weights that are unknown by the servers.

One note on these user specific models is that this approach to authentication is not

standard in the literature. This was a decision made for this specific project. However, with more time, it would be interesting to extend these results into more state-of-the-art models.

Verification Phase

Once a user has enrolled in the service and the servers have each trained their private, user-specific models, a user will now be able to privately verify or authenticate themselves to the authentication service.

Similarly to the enrollment phase, a user begins the verification process by recording their voice utterance and preprocessing this raw input signal. Again, the preprocessed signal is split into shares as before and sent to the respective servers for verification.

Each server will use its privately trained model to make an inference about the private share they have just received. This inference will be a score, still private and in the form of a share. These scores will be used to make a decision about whether to accept or reject the user.

Once the servers have the scores, they will compare them with some predetermined threshold value. The servers will send their decision share to the authentication service to reveal the decision.

To note, once the preprocessed input data is sent from the user, every step the servers take remains private until the final decision is revealed in the end. When this final decision is revealed, only the decision becomes known and not the score or the trained model. In the semi-honest threat model, for this to hold we assume that the servers follow the protocol and do not collude with each other.

5.3.3 Models

At the heart of voice authentication lies the speaker verification task, as outlined in Sections 1.3 and 5.2.1. This is the task determining whether a voice input belongs to a specific speaker or not [68, 28]. For this task, we — logistic regression model and two neural networks.

After logistic regression, two neural networks were considered — Neural Networks A from [96, 139] and B from [87, 95, 139]. Neural Network A is a deep neural network. It consists of 3 fully connected layers, utilizing ReLU as the activation [139]. Normalization is applied at the end to convert the output to the proper format. Neural Network B is a 4

layer convolutional neural network. The first two layers are two-dimensional convolutional layers using ReLU as the activation followed by a max pooling layer. The third layer is a fully connected layer, again with RelU activation. Finally, we have a linear layer, with a similar normalization as that in Neural Network A [139].

5.4 Implementation

In this section we will explore how we implement our voice authentication, before presenting the results of our analysis in Section 5.4.5.

5.4.1 Dataset

For our implementation and subsequent analysis, we use the LibriSpeech dataset [112] for speech data. This dataset is English language based containing 1000 hours of speech from audiobooks. The dataset is split into 7 subsets for various purposes, with each subset containing a similar number of male and female speakers. There are clean datasets, which comprise mostly of clean voice data, while the other data contains noisy data as well. The LibriSpeech dataset is commonly used for speaker recognition [122] as well as speech recognition tasks [114, 113].

While LibriSpeech is regarded as a clean, good quality corpus [135] and being created from audiobooks may make it less similar to real life speech, it can still be appropriate for evaluating speaker recognition methods [135]. For our main goal of determining inference time in an MPC setting, this dataset is appropriate.

The voices in the dataset are split up by speaker, allowing for implementation into speaker recognition models. The audio data comes in a flac file. We will explore in Section 5.4.2 how to convert these files into data that can be used in our models. In Section 5.4.3, we will show how we divide the speakers data into enrolling data for the building of the model and verifying data for the testing of the model.

5.4.2 Preprocessing

As mentioned above, LibriSpeech voice data comes in flac files. Before being able to build models or verify voices, this data must be loaded and preprocessed. To accomplish this, we make use of the Librosa python package [92, 93].

First, through the Librosa package, we can load a flac file into an array representation of an audio time series. For this, we use a target sampling rate of 16,000. After that, we apply a Short-Time Fourier Transform to the time series data. By computing discrete Fourier Transforms over short windows, this will represent the signal in the time-frequency domain [93, 99] and convert the signal into a spectrogram that can be better interpreted by the models.

After we have our spectrogram, we standardize the data so that the magnitudes of all signals belong within the same scale. This will be the inputs to our models. Through this process we obtain a standard 257×250 dimension matrix for each audio signal.

5.4.3 Building the Models

Above, we have explained the dataset and the methods for preprocessing the data. We have also shown in Section 5.3.3 the models that we will be using for our analysis.

In order to implement this all together, the final step is to determine which data we will build our models upon. For each user we will begin by dividing the data into enrollment data and verifying data. The enrollment data will be used to train the models and the verifying data will be used for testing purposes. We split each user's data so that 60% of the data is enrollment data and 40% is testing data.

However, one single user's data is not enough to train a verification model on. If you recall, a verification model should accept users who are similar and reject users who are different. This requires both the initial genuine enrolling user with an accept label, as well as other users acting as imposters with a reject label.

Our goal is to match the amount of imposter data as genuine data for our model, To create this imposter data, we randomly select another user from the dataset, and then randomly select an utterance. We repeat this procedure until we have sufficient imposter data. To create our train data, we combine our genuine train data with imposter train data. Similarly, to create our test data we combine our genuine test data with imposter test data. For both cases we shuffle the data before putting it into the model, so the model is not influenced in any way.

Additionally, we must note that the data will be inputted into the models in a private manner via our MPC protocol. We must specify where the data is coming from to confirm that this would be valid in a real world application. For the genuine data, this is the voice data coming directly from the user enrolling. For the imposter data, specifically the imposter training data, this can be obtained by the computing servers in a variety of ways.

One option is to take publicly available speech data and split it into shares for the servers to use.

5.4.4 Hardware

To run our experiments we used a machine with the following specifications: Ubuntu 22.04.1, Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz, with the x86_64 architecture.

5.4.5 Results

We now present the results of our implementation. Through our analysis, we are trying to determine the costs associated with including MPC in our voice authentication framework. We will begin by exploring how long it takes to make a single verification. This is the most important information obtained from our implementation as this is what is needed for the survey in Section 5.5. In addition, we still provide additional information such as training time, inference communication costs, and total communication costs of the entire protocol.

In each of the sections we show a table with two columns and three rows. The rows represent the results from the three different models described in Section 5.3.3. The first column in the table represents results from running the protocol in the semi-honest three-server model as described in Section 5.2.3. The second column represents results from running the protocol in the malicious four-server model as described in Section 5.2.4. It is worth noting that running the neural networks for the malicious case was too computationally intensive for any of our machines to handle, so we note N/A in these cases.

Inference Time

We begin our results with the time it takes, in seconds, to make an inference. To begin, we train each of the models on the user's enrollment data. We then begin the timer and send the user's input shares to the respective servers. Once received, the servers privately score the input and come to a private decision. They then exchange communication to receive and reveal the output of the protocol, before we stop the timer. The results of this timing is in Table 5.1. As we can see, inference using the logistic regression model is very quick, with inference taking under one tenth of a second in the semi-honest case and one and a half tenths in the malicious case. These are not the best quality of models however, so the neural networks time is more important. We see neural network A taking 29.71 seconds to

Table 5.1: Time (seconds) taken to make inference

Model	MPC Protocol	
	Semi-Honest 3 Servers	Malicious 4 Servers
Logistic Regression	0.075	0.147
Neural Network A	29.71	N/A
Neural Network B	9.24	N/A

Table 5.2: Time (seconds per epoch) taken to train the model

Model	MPC Protocol	
	Semi-Honest 3 Servers	Malicious 4 Servers
Logistic Regression	5.64	6.81
Neural Network A	44.5	N/A
Neural Network B	25.75	N/A

make inference and neural network B taking 9.24 seconds. Neural network A takes longer than B as expected, since the fully connected layers involved in A are more complex than the convolutional layers in B.

Training Time

While inference time above is our most relevant analysis, we have chosen to include other results as well such as training time. We measure training time here as the time it takes the model to run through a single epoch of training as seen in Table 5.2. For logistic regression, there is only a single epoch to train a model, so these values are simply the time it takes to train. We see that logistic regression takes 5.64 seconds in the semi-honest setting and 6.81 seconds in the malicious setting. Neural network A takes 44.5 seconds per epoch to train and neural network B takes 25.75 seconds.

Table 5.3: Data exchanged (MB) when making inference

Model	MPC Protocol	
	Semi-Honest 3 Servers	Malicious 4 Servers
Logistic Regression	0.30	0.924
Neural Network A	1.48	N/A
Neural Network B	63.68	N/A

Table 5.4: Total data exchanged (MB)

Model	MPC Protocol	
	Semi-Honest 3 Servers	Malicious 4 Servers
Logistic Regression	166	172.1
Neural Network A	1330.62	N/A
Neural Network B	474.42	N/A

Inference Communication Costs

In addition to time, we also measured how much data is sent between the servers in order to make inference. The amount of data sent is directly related to the MPC protocol used. We measure inference the same way as Section 5.4.5, however in this case we measure data sent. The results can be seen in Table 5.3. We see logistic regression sends 0.3 MB in the semi-honest setting and 0.924 MB in the malicious setting. Neural network A sends 1.48 MB of data and neural network B sends 63.68 MB of data.

Total Communication Costs

Finally we measure the entire communication costs of training and making inference. For training, we use 132 inputs and use 5 epochs for the neural networks. For testing, we use 90 testing elements. The results are in Table 5.4. Logistic regression costs 166 MB of data exchanged in the semi-honest case and 172.1 MB in the malicious case. Neural network A costs 1330.62 MB and neural network B costs 472.42 MB.

5.5 Project Survey

As part of the group project with Anuradha Kulkarni and Vijay Ravi, we conducted a survey to determine what people think of the privacy of their biometrics and what costs they are willing to incur in order to keep these biometrics private. Below I will present the questions we asked as part of this survey and provide a brief summary of the general thoughts of the people surveyed. With our survey, we wanted insight into four questions to be explored. The first question was "How do people feel about sharing their fingerprint, phone PIN, and face ID with a remote server". Second, we wanted to explore "How do people feel about authenticating locally with the voice in the pre-product description phase and the post-product description phase". Third, we wanted to know "How do people feel about remote voice authentication across different scenarios". Finally, "How long are people willing to wait to authenticate themselves by voice for improved privacy benefits in the MPC setting".

We designed our survey questions so as to answer the above questions. We begin by asking five questions, before describing a product using MPC (and explaining what MPC is), before asking another six questions.

The pre-product questions were as follows.

1. On a scale of $1-5$, with 1 being "not concerned" and 5 being "very concerned", how do you feel about sending your **fingerprint** to a remote server, like your bank, for authentication?
2. On a scale of $1-5$, how do you feel about sending your **phone PIN** to a remote server, like your bank, for authentication?
3. On a scale of $1-5$, how do you feel about sending your **face ID** to a remote server, like your bank, for authentication?
4. On a scale of $1-5$, how do you feel about unlocking your phone with your **voice**?
5. On a scale of $1-5$, how do you feel about sending your **voice** to a remote server, like your bank, for authentication?

We then described an MPC-based voice authentication framework. It was described in the scenario that a bank application intends to use voice authentication with MPC to provide privacy. A brief explanation of MPC was provided and the link [1] was presented if the person wanted more information. Next we asked the following 6 questions.

6. On a scale of $1-5$, how do you feel about unlocking your phone with your **voice**, **without MPC**?

7. On a scale of $1-5$, how do you feel about sending your **voice** to a remote server, like your bank, for authentication, **without MPC**?

8. On a scale of $1-5$, how do you feel about sending your **voice** to a remote server, like your bank, for authentication, **with MPC**?

9. Yes or No, Are you willing to wait an average of **45 seconds**, to securely authenticate your bank account at the cost of improved privacy in the MPC setting?

10. Yes or No, Are you willing to wait an average of **5 seconds**, to securely authenticate your bank account at the cost of improved privacy in the MPC setting?

11. Yes or No, Are you willing to wait an average of **1 seconds**, to securely authenticate your bank account at the cost of improved privacy in the MPC setting?

We had 20 respondents for our survey.

In response to each of our insight questions, first we wanted to learn "How do people feel about sharing their fingerprint, phone PIN, and face ID with a remote server". We found that people had a moderate level of concern with sharing these personal elements with a remote server. We surprisingly found that people felt more comfortable with sending their face or fingerprint to a remote server than their phone PIN.

Second, we wanted to learn "How do people feel about authenticating locally with the voice in the pre-product description phase and the post-product description phase". We found that people shared a similar level of comfort with sharing their voice, as with sharing their face and fingerprint with a remote server. Furthermore, people were more comfortable unlocking their phone (local case) using their voice than sending it to a remote server.

Third, we wanted to learn "How do people feel about remote voice authentication across different scenarios". In this case, we primed the people with the voice authentication idea and how MPC may be used for added privacy. We saw a significant increase in the level of concern with sending their voice to a remote server (in the non-MPC case) after priming the person with the idea of using MPC for added privacy. Next, we found that people were now more comfortable sharing their voice in a private manner (using MPC) with a remote server. We found people to be more comfortable with this than any previous question surrounding sharing voice, face, fingerprint or phone PIN.

Finally, we wanted to learn "How long are people willing to wait to authenticate themselves by voice for improved privacy benefits in the MPC setting". We found that most people were willing 45 seconds for the added privacy, almost all were willing to wait 5 seconds for the added privacy, and all were willing to wait 1 second for the added privacy.

While this summary of our course project is not precisely indicative of what the general public may or may not feel about including MPC in an authentication scheme, we feel that one main takeaway is that people do value privacy and (at least some) people are willing to provide some levels of concessions (in this case time) in order to improve their biometric's privacy.

While this was a brief survey for a course project, we believe that questions similar to these could be an interesting area of further research. We believe that accompanying practical applications of MPC, it could be important and interesting to get a sense of how people feel about the costs associated with this added privacy (on a larger scale).

5.6 Concluding Remarks

In Chapter 5, we introduced a privacy-preserving voice authentication framework using MPC techniques for the privacy component. For our voice authentication framework, we used user-specific neural networks for the speaker verification task. With this framework, our main goal was to obtain inference times in order to determine the costs of inference using MPC for a voice authentication task. Using these results, we surveyed university students to determine how much value people place in privacy for their biometrics, as well as whether or not they were willing to incur the costs associated with this privacy. Albeit this was only a survey, and a full user study would be needed for further analysis, we believe that people really value their privacy and are willing to incur costs associated with this privacy. In many scenarios, I believe it to be feasible to design a speaker verification system that can restrict the costs to within a reasonable manner for the users.

5.6.1 Future Work

This study opens the door to much future work which we will discuss from an implementation and survey standpoint.

Implementation

The implementation can be improved upon in a number of ways. The first main idea is through the framework itself. It is widespread in voice authentication to train a Universal Background Model [28] to distinguish between characteristics and properties that can differentiate people's voices. Such a framework could be advantageous from a privacy-preserving standpoint as this type of model can be trained upon publicly available data, so the training does not have to occur inside the MPC framework. We could utilize MPC strictly for inference as is shown in [32] and [137].

Additionally, while the purpose of this study was to determine the costs associated with training and making inference on user specific voice models and finding peoples opinions on this, we have not performed an accuracy analysis on our models. To perform a proper accuracy analysis we would need to create models and analyze the results in MPC for each user in the dataset. We did not have the time to perform this analysis.

Within our framework, future work includes implementing more speaker recognition models. In this chapter, we trained logistic regression and two neural networks. We can try more complex neural networks, decision trees, and other machine-learning models. Additionally, we can explore other types of MPC protocols, with varying (more complex) levels of adversarial threat.

However, this leads well to the biggest limitation of the study, which will be explored in Section 5.6.2. This limitation is the computational power available. All of the additional speaker recognition models and MPC models attempted, quickly became too computationally intensive for our machines to handle. Future work can explore cutting computational costs to make these options more viable.

One additional piece of future work could be applying our methods to other forms of biometrics for authentication such as brainwave [11], gait [54], or keystroke dynamics [3].

Survey

Due to the scope of this project, the survey conducted could not be enhanced into a full study. While this survey provided some background on peoples thoughts towards privacy, a larger study would be needed to properly make any claims about a population.

5.6.2 Limitations

The major limitation with this analysis is the computational power available. With more computational power, we would be able to test more advanced machine learning and deep learning models for the speaker recognition task. Additionally, with more computational power, we would be able to test the framework with MPC protocols which are secure against stronger adversaries.

Chapter 6

Conclusions

We explored the use of handwriting and voice biometrics to solve recognition, identification and authentication tasks. Due to the private nature of these biometrics, we used MPC to incorporate privacy components into our frameworks.

In Chapter 2, we developed handwriting character recognition models based on the OnHW-chars and OnHW-split-equations datasets [106, 110]. With the help of ensemble methods, our models were able to obtain stronger results than the best previously reported marks. In addition to using state-of-the-art techniques to obtain these results and ensemble methods to boost the results, we used LIME [94] to help explain what part of the data signal is contributing to the predictions that the models are making.

In Chapter 3, we utilized MPC to introduce privacy into the handwriting character recognition task. We designed a framework for private handwritten character recognition and implemented it into the CrypTen [79] MPC framework. With this implementation, we obtain results on how the accuracy of the models differ when making private comparison against plaintext comparison and find a difference in accuracy of between 0.55% and 1.42%. Unfortunately, private inference comes with substantial costs, and we reported on the costs associated with making private inference on the handwriting character recognition task.

In Chapter 4, we began to explore using biometrics to identify or authenticate people. We developed writer identification models based on the OnHW-equations dataset. First however, we discarded the data from the magnetometer sensor as we believe there is some bias in this data which can easily identify the writer. We performed a timestep analysis on the data to determine that we should use two seconds of data per sample. Next, we obtained accuracies up to 91.57% in identifying the writer of a piece of text, out of a group of writers. With these high accuracy models, we pivoted to exploring how to incorporate privacy, into

the training and inference of writer identification models. With private training, came heavy costs, so we had to simplify our models (CNN's). After implementing these models, we obtained an MPC training accuracy of 77.45%. Finally, we reported on the costs associated with training simple CNN's under various model architectures.

In Chapter 5, we switched from handwriting to voice as we explored a private voice authentication framework. This chapter was part of a class project, where we designed and implemented a private voice authentication framework using the MP-SPDZ [71] MPC framework. We estimated the costs associated with this implementation. Finally, we reported on a survey conducted as part of this class project, to determine how much people value privacy and whether they are willing to incur costs for this privacy. While no definitive conclusions can be drawn from this analysis, we believe it to be an important area for future research. That is, when using techniques such as MPC to incorporate privacy into a service for a consumer, studies should be conducted in order to gauge user interest in these types of ideas. These accompanying studies do not appear in the literature as often as they should. We leave proper user studies for each of Chapters 3, 4 and 5 for future research.

6.1 MP-SPDZ vs. CrypTen

In this section, I will provide some of my thoughts and experiences based on using CrypTen [79, 80] and MP-SPDZ [71]. For more details on CrypTen and MP-SPDZ, see Sections 1.5 and 5.2.5, respectively.

The CrypTen and MP-SPDZ frameworks differ dramatically in structure. First, as mentioned in Section 1.5, CrypTen was designed specifically not to include its own domain specific language. On the other hand, MP-SPDZ does have its own domain specific language, along with a compiler implemented to compile python-like code. Due to this overall design structure, I found CrypTen much easier to use and debug. I found that the code written in CrypTen was very similar to PyTorch code, making it easy to follow and begin coding. One major design goal of CrypTen was to be accessible to developers who do not have strong cryptographic backgrounds. Additionally, I believe that errors were easier to debug when using CrypTen. One reason was that in MP-SPDZ, parsing through the inner workings of the compiler to find the source of the error often proved difficult.

While I found CrypTen to be the easier framework to use, MP-SPDZ offers much more flexibility surrounding its MPC protocols. CrypTen offers only the single protocol as described in Section 1.5. This protocol is flexible in the sense that it accommodates a dynamic

number of computing parties, however the protocol always uses the same secret sharing scheme, semi-honest adversaries, and does not natively support the client/server model. MP-SPDZ on the other hand offers many protocols, which are (mostly) interchangeable at runtime. These protocols differ in the number of parties involved in the protocol, strength of adversary (including malicious and covert adversaries) and number of corruptable computing parties. In addition, the different schemes implemented in MP-SPDZ have a variety of secret sharing methods. Some of these protocols implemented include MASCOT [72], SPDZ2K [39], LowGear [73], CowGear (covert adaptation of LowGear), ATLAS [59], to name a few. This allows a lot of flexibility in MP-SPDZ in terms of choosing the best and most efficient protocol for the given task.

As mentioned in Section 4.7.1, CrypTen does not have a native implementation of the Adam algorithm [76] for optimization. On the other hand, Adam and the amsgrad variant [124] are both implemented natively in MP-SPDZ. From the experiments run through this thesis, I found that when using Adam as the optimization function, the model often converged much sooner than when using SGD. This can play a major role when training models in the MPC space since there are high costs per epoch. If the model takes fewer epochs to run, that would be beneficial.

Overall, my experience with CrypTen and MP-SPDZ was that CrypTen was much easier to use, but MP-SPDZ offers much more flexibility when it comes to creating and optimizing the framework.

6.2 Semi-Honest vs. Malicious Security for Handwriting

Throughout Chapters 3 and 4 we explore the use of CrypTen's MPC framework for incorporating privacy in handwritten character recognition and writer identification. As mentioned in Section 1.5.3, the CrypTen protocol provides security against information leakage against a static passive adversary corrupting up to $|P| - 1$ of the $|P|$ parties. Ideally for the frameworks, we would be able to provide security against malicious computing parties. It has been noted in [80], that possible extensions for CrypTen involve incorporating support for malicious security and alternative methods for Beaver Triple generation (in order to remove the need for a trusted third party).

However, when providing security against malicious adversaries, there is a significant increase in the costs incurred. We have made the assumption that security against a semi-honest adversary is sufficient, and while this may be true in certain instances, when

biometric data is being considered, it may often be preferable to incorporate a protocol providing security against stronger adversaries. We leave the exploration of malicious security for future work.

References

[1] What is Secure Multiparty Computation? https://inpher.io/technology/what-is-secure-multiparty-computation/, 2022. Accessed: 2022-12-05.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.

[3] A. Acar, S. Ali, K. Karabina, C. Kaygusuz, H. Aksu, K. Akkaya, and S. Uluagac. A Lightweight Privacy-Aware Continuous Authentication Protocol-PACA. *ACM Trans. Priv. Secur.*, 24(4):1–28, 2021.

[4] S. Adams, D. Melanson, and M. De Cock. Private Text Classification with Convolutional Neural Networks. In *Proceedings of the Third Workshop on Privacy in Natural Language Processing*, PrivateNLP 2021, pages 53–58, 2021.

[5] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'19)*, pages 2623–2631, 2019.

[6] S. Al-Maadeed. Text-Dependent Writer Identification for Arabic Handwriting. *Journal of Electrical and Computer Engineering (JECE)*, pages 1–8, 2012.

[7] M. Z. Al-Shamaileh, A. B. Hassanat, A. S. Tarawneh, M. Sohel R., C. Celik, and M. Jawthari. New Online/Offline text-dependent Arabic Handwriting dataset for

Writer Authentication and Identification. In *2019 10th International Conference on Information and Communication Systems (ICICS)*, pages 116–121, 2019.

[8] F. Alimoglu and E. Alpaydin. Combining Multiple Representations and Classifiers for Pen-based Handwritten Digit Recognition. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition (ICDAR'97)*, volume 2, pages 637–640, 1997.

[9] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 805–817, 2016.

[10] T. Aremu and K. Nandakumar. PolyKervNets: Activation-free Neural Networks For Efficient Private Inference. In *First IEEE Conference on Secure and Trustworthy Machine Learning*, SATML 2023, 2023.

[11] P. Arias-Cabarcos, T. Habrich, K. Becker, C. Becker, and T. Strufe. Inexpensive Brainwave Authentication: New Techniques and Insights on User Acceptance. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 55–72, 2021.

[12] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private Collaborative Forecasting and Benchmarking. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, pages 103–114, 2004.

[13] Y. Aumann and Y. Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In *Theory of Cryptography Conference — TCC 2007, LNCS 4392*, pages 137–156, 2007.

[14] H. Azimi, S. Chang, J. Gold, and K. Karabina. Improving Accuracy and Explainability of Online Handwriting Recognition. In *arXiv 2209.09102*, 2022.

[15] H. Azimi, P. Xi, M. Bouchard, R. Goubran, and F. Knoefel. Machine Learning-Based Automatic Detection of Central Sleep Apnea Events From a Pressure Sensitive Mat. *IEEE Access*, 8:173428–173439, 2020.

[16] A. Baccarini, M. Blanton, and C. Yuan. Multi-Party Replicated Secret Sharing over a Ring with Applications to Privacy-Preserving Machine Learning. Cryptology ePrint Archive, Paper 2020/1577, 2020. https://eprint.iacr.org/2020/1577.

[17] D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology — Crypto '91, LNCS 576*, pages 420–432, 1992.

[18] H. Beigi. *Fundamentals of Speaker Recognition*. 2011.

[19] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New Ensemble Methods for Evolving Data Streams. *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD '09)*, pages 139–148, 2009.

[20] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *Computer Security — ESORICS 2008*, pages 192–206, 2008.

[21] F. Bouteruche, S. Macé, and E. Anquetil. Fuzzy Relative Positioning for On-Line Handwritten Stroke Analysis. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Université de Rennes 1, 2006. http://www.suvisoft.com.

[22] J. Bringer, H. Chabanne, and A. Patey. Privacy-Preserving Biometric Identification Using Secure Multiparty Computation: An Overview and Recent Trends. *IEEE Signal Processing Magazine*, 30(2):42–52, 2013.

[23] M. Bulacu and L. Schomaker. Text-Independent Writer Identification and Verification Using Textural and Allographic Features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(4):701–717, 2007.

[24] A. Bulling, U. Blanke, and B. Schiele. A Tutorial on Human Activity Recognition Using Body-worn Inertial Sensors. *ACM Computing Surveys*, 46(3):1–33, 2014.

[25] J. Caiado, N. Crato, and D. Pena. Comparison of Times Series with Unequal Length in the Frequency Domain. *Communications in Statistics — Simulation and Computation*, 38(3):527–540, 2009.

[26] R. Castrillón, A. Acien, J.R. Orozco-Arroyave, A. Morales, J.F. Vargas, R. Vera-Rodríguez, J. Fierrez, J. Ortega-Garcia, and A. Villegas. Characterization of the Handwriting Skills as a Biomarker for Parkinson's Disease. In *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*, pages 1–5, 2019.

[27] S. Chang, J. Gold, and K. Karabina. ImpAcX_OnHW: Improving Accuracy and Explainability of Handwriting Recognition Models. https://github.com/KorayKarabina/ImpAcX_OnHW, 2022.

[28] G. Chen, S. Chenb, L. Fan, X. Du, Z. Zhao, F. Song, and Y. Liu. Who is Real Bob? Adversarial Attacks on Speaker Recognition Systems. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 694–711, 2021.

[29] Z. Chen. On the Detection of Adaptive Adversarial Attacks in Speaker Verification Systems. *IEEE Internet of Things Journal (Early Access)*, 2023.

[30] Z. Chen, L. C. Chang, C. Chen, G. Wang, and Z. Bi. Defending against FakeBob Adversarial Attacks in Speaker Verification Systems with Noise-Adding. *Algorithms*, 15(8), 2022.

[31] Z. Chen, H. X. Yu, A. Wu, and W. S. Zheng. Letter-Level Online Writer Identification. *International Journal of Computer Vision*, 129(5):1394–1409, 2021.

[32] O. Chouchane, B. Brossier, J. E. G. Gamboa, T. Lardy, H. Tak, O. Ermis, M. R. Kamble, J. Patino, N. W.D. Evans, M. Önen, et al. Privacy-Preserving Voice Anti-Spoofing Using Secure Multi-Party Computation. In *Proc. Interspeech 2021*, pages 856–860, 2021.

[33] A. R. Choudhuri, A. Goel, M. Green, A. Jain, and G. Kaptchuk. Fluid MPC: Secure Multiparty Computation with Dynamic Participants. In *Advances in Cryptology — CRYPTO 2021, LNCS 12826*, pages 94–123, 2021.

[34] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr. Time Series Feature Extraction on Basis of Scalable Hypothesis Tests (tsfresh — A Python Package). *Neurocomputing*, 307:72–77, 2018.

[35] M. Christ, A. W. Kempa-Liehr, and M. Feindt. Distributed and Parallel Time Series Feature Extraction for Industrial Big Data Applications. *Asian Machine Learning Conference. Workshop on Learning on Big Data*, 2016. https://arxiv.org/abs/1610.07717v1.

[36] R Cleve. Limits on the Security of Coin Flips When Half the Processors Are Faulty. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, page 364–369, 1986.

[37] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. EMNIST: Extending MNIST to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926, 2017.

[38] R. Cramer, I. Damgård, and Y. Ishai. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *Theory of Cryptography — TCC 2005, LNCS 3378*, pages 342–362, 2005.

[39] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^K for Dishonest Majority. *Advances in Cryptology — CRYPTO 2018, LNCS 10992*, pages 769–798, 2018.

[40] R. Cruz, R. Sabourin, and G. Cavalcanti. On Meta-Learning for Dynamic Ensemble Selection. *2014 22nd International Conference on Pattern Recognition*, pages 1230–1235, 2014.

[41] A. Dalskov, D. Escudero, and M. Keller. Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2183–2200, 2021.

[42] I. Damgård and Y. Ishai. Constant-Round Multiparty Computation Using a Black-Box Pseudorandom Generator. In *Advances in Cryptology — CRYPTO 2005, LNCS 3621*, pages 378–394, 2005.

[43] I. Damgård and Y. Ishai. Scalable Secure Multiparty Computation. In *Advances in Cryptology — CRYPTO 2006, LNCS 4117*, pages 501–520, 2006.

[44] I. Damgård, M. Fitzi, E. Kiltz, J.B. Nielsen, and T. Toft. Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation. *Theory of Cryptography — TCC 2006, LNCS 3876*, pages 285–304, 2006.

[45] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. *Advances in Cryptology — CRYPTO 2012, LNCS 7417*, page 643–662, 2012.

[46] M. De Cock, R. Dowsley, A. C. A. Nascimento, D. Railsback, J. Shen, and A. Todoki. High Performance Logistic Regression for Privacy-Preserving Genome Analysis. *BMC Medical Genomics*, 14(1):23, 2021.

[47] D. Demmler, T. Schneider, and M. Zohner. ABY-A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *Network and Distributed System Security Symposium*, NDSS 2015, 2015.

[48] A. Diaa, L. Fenaux, T. Humphries, M. Dietz, F. Ebrahimianghazani, B. Kacsmar, X. Li, N. Lukas, R. A. Mahdavi, S. Oya, E. Amjadian, and F. Kerschbaum. Fast and Private Inference of Deep Neural Networks by Co-designing Activation Functions. In *arXiv 2306.08538*, 2023.

[49] T. Dietterich. Ensemble Methods in Machine Learning. *MCS 2000: Multiple Classifier Systems — First International Workshop*, 1857:1–15, 2000.

[50] M. Fahmy. Online Handwritten Signature Verification System based on DWT Features Extraction and Neural Network Classification. *Ain Shams Engineering Journal*, 1(1):59–70, 2010.

[51] M. Faundez-Zanuy, J. Fierrez, M. A. Ferrer, M. Diaz, R. Tolosana, and R. Plamondon. Handwriting Biometrics: Applications and Future Trends in e-Security and e-Health. *Cognitive Computation*, 12(5):940–953, 2020.

[52] K. Fauvel, T. Lin, V. Masson, È. Fromont, and A. Termier. XCM: An Explainable Convolutional Neural Network for Multivariate Time Series Classification. In *arXiv 2009.04796*, 2020. https://arxiv.org/abs/2009.04796.

[53] H.I Fawaz, B. Lucas, G. Forestier, C. Pelletier, D.F. Schmidt, J. Weber, G.I. Webb, L. Idoumghar, P. Muller, and F. Petitjean. InceptionTime: Finding AlexNet for Time Series Classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, 2020.

[54] A. Ferlini, D. Ma, R. Harle, and C. Mascolo. EarGate: Gait-Based User Identification with in-Ear Microphones. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, page 337–349, 2021.

[55] T. K. Frederiksen, M. Keller, E. Orsini, and P. Scholl. A Unified Approach to MPC with Preprocessing Using OT. In *Advances in Cryptology — ASIACRYPT 2015, LNCS 9452*, pages 711–735, 2015.

[56] M. Gargouri, S. Kanoun, and J. M. Ogier. Text-Independent Writer Identification on Online Arabic Handwriting. In *2013 12th International Conference on Document Analysis and Recognition (ICDAR)*, pages 428–432, 2013.

[57] P. Geurts, D. Ernst, and L. Wehenkel. Extremely Randomized Trees. *Machine Learning*, 63(1):3–42, 2006.

[58] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. 2016. http://www.deeplearningbook.org.

[59] V. Goyal, H. Li, R. Ostrovsky, A. Polychroniadou, and Y. Song. ATLAS: Efficient and Scalable MPC in the Honest Majority Setting. Cryptology ePrint Archive, Paper 2021/833, 2021. https://eprint.iacr.org/2021/833.

[60] A. Graves and J. Schmidhuber. Framewise Phoneme Classification with Bidirectional LSTM Networks. *Proceedings of IEEE International Joint Conference on Neural Networks, 2005*, 4:2047–2052, 2005.

[61] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet. UNIPEN Project of On-Line Data Exchange and Recognizer Benchmarks. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 — Conference C: Signal Processing (Cat. No.94CH3440-5)*, volume 2, pages 29–33, 1994.

[62] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet. UNIPEN Project of Online Data Exchange and Recognizer Benchmarks. *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, 2:29–33, 1994.

[63] Md. Al Mehedi Hasan, J. Shin, and Md. Maniruzzaman. Online Kanji Characters Based Writer Identification Using Sequential Forward Floating Selection and Support Vector Machine. *Applied Sciences*, 12(20), 2022.

[64] Z. He, B. Fang, J. Du, Y. Y. Tang, and X. You. A Novel Method for Off-line Handwriting-based Writer Identification. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, volume 1, pages 242–246, 2005.

[65] M. Hébert. Text-Dependent Speaker Recognition. *Springer Handbook of Speech Processing*, pages 743–762, 2008.

[66] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[67] J.J. Hull. A Database for Handwritten Text Recognition Research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.

[68] M. M. Kabir, M. F. Mridha, J. Shin, I. Jahan, and A. Q. Ohi. A Survey of Speaker Recognition: Fundamental Theories, Recognition Methods and Opportunities. *IEEE Access*, 9:79236–79263, 2021.

[69] F. Karim, S. Majumdar, H. Darabi, and S. Chen. LSTM Fully Convolutional Networks for Time Series Classification. *IEEE Access*, 6:1662–1669, 2018.

[70] F. Karim, S. Majumdar, H. Darabi, and S. Harford. Multivariate LSTM-FCNs for Time Series Classification. *Neural Networks*, 116:237–245, 2019.

[71] M. Keller. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, pages 1575–1590, 2020.

[72] M. Keller, E. Orsini, and P. Scholl. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 830–842, 2016.

[73] M. Keller, V. Pastro, and D. Rotaru. Overdrive: Making SPDZ Great Again. Cryptology ePrint Archive, Paper 2017/1230, 2017. https://eprint.iacr.org/2017/1230.

[74] E. Keogh and C. Ratanamahatana. Exact Indexing of Dynamic Time Warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[75] E. J. Kindt. *Privacy and Data Protection Issues of Biometric Applications*. 2016.

[76] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *arXiv 1412.6980*, 2017.

[77] T. Kinnunen and H. Li. An Overview of Text-Independent Speaker Recognition: from Features to Supervectors. *Speech Communication*, 52(1):12–40, 2010.

[78] A. Klaß, S. Lorenz, M. Lauer-Schmaltz, D. Rügamer, B. Bischl, C. Mutschler, and F. Ott. Uncertainty-aware Evaluation of Time-Series Classification for Online Handwriting Recognition with Domain Shift. In *arXiv 2206.08640*, 2022.

[79] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten. CrypTen: Secure Multi-Party Computation Meets Machine Learning. In *Advances in Neural Information Processing Systems*, volume 34 of *NeurIPS 2021*, pages 4961–4973, 2021.

[80] B. Knott, S. Venkataraman, A.Y. Hannun, S. Sengupta, M. Ibrahim, and L van der Maaten. CrypTen: Secure Multi-Party Computation Meets Machine Learning. In *arXiv 2109.00984*, 2021.

[81] P. Laniel, N. Faci, R. Plamondon, M. H. Beauchamp, and B. Gauthier. Kinematic Analysis of Fast Pen Strokes in Children with ADHD. *Appl Neuropsychol Child*, 9(2):125–140, 2019.

[82] J. LaViola and R. Zeleznik. A Practical Approach for Writer-Dependent Symbol Recognition Using a Writer-Independent Symbol Recognizer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1917–1926, 2007.

[83] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[84] J. Li, N. Song, G. Yang, M. Li, and Q. Cai. Improving Positioning Accuracy of Vehicular Navigation System During GPS Outages Utilizing Ensemble Learning Algorithm. *Information Fusion*, 35:1–10, 2017.

[85] Y. Lindell. Secure Multiparty Computation. *Commun. ACM*, 64(1):86–96, 2020.

[86] C. L. Liu, F. Yin, D. H. Wang, and Q. F. Wang. CASIA Online and Offline Chinese Handwriting Databases. In *2011 International Conference on Document Analysis and Recognition (ICDAR)*, pages 37–41, 2011.

[87] J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 619–631, 2017.

[88] M. Liwicki and H. Bunke. IAM-OnDB - an on-line English sentence database acquired from handwritten text on a whiteboard. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, volume 2, pages 956–961, 2005.

[89] David Llorens, Federico Prat, Andrés Marzal, Juan Miguel Vilar, M. José Castro, J. C. Amengual, S. Barrachina, A. Castellanos, S. E. Boquera, J. A. Gómez, et al. The UJIpenchars Database: A Pen-Based Database of Isolated Handwritten Characters. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, 2008.

[90] L. Lu, J. Yu, Y. Chen, and Y. Wang. VocalLock: Sensing Vocal Tract for Passphrase-Independent User Authentication Leveraging Acoustic Signals on Smartphones. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 4(2), 2020.

[91] U.-V. Marti and H. Bunke. The IAM-database: an English sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition (IJDAR)*, 5(1):39–46, 2002.

[92] B. McFee, A. Metsai, M. McVicar, S. Balke, C. Thomé, C. Raffel, F. Zalkow, A. Malek, Dana, K. Lee, O. Nieto, D. Ellis, J. Mason, E. Battenberg, S. Seyfarth, R. Yamamoto, Viktor, r. Morozov, K. Choi, J. Moore, R. Bittner, S. Hidaka, Z. Wei, nullmightybofo, A. Weiss, D. Hereñú, F.-R. Stöter, L. Nickel, P. Friesch, M. Vollrath, and T. Kim. librosa/librosa: 0.9.2, 2022.

[93] B. McFee, C. Raffel, D. Liang, D. P. W. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and Music Signal Analysis in Python. In *Proceedings of the 14th Python in Science Conference*, SciPy 2015, pages 18–24, 2015.

[94] E. Metzenthin. LIME For Time, 2020. https://github.com/emanuel-metzenthin/Lime-For-Time.

[95] P. Mohassel and P. Rindal. ABY3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 35–52, 2018.

[96] P. Mohassel and Y. Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38. IEEE, 2017.

[97] K. Monteith, J. Carroll, K. Seppi, and T. Martinez. Turning Bayesian Model Averaging into Bayesian Model Combination. *The 2011 International Joint Conference on Neural Networks*, pages 2657–2663, 2011.

[98] H. Mouchère, C. Viard-Gaudin, R. Zanibbi, and U. Garain. ICFHR 2014 competition on recognition of on-line handwritten mathematical expressions (CROHME 2014). In *2014 14th International Conference on Frontiers in Handwriting Recognition*, pages 791–796, 2014.

[99] K. Nasser. CHAPTER 7 — Frequency Domain Processing. In *Digital Signal Processing System Design (Second Edition)*, pages 175–196. Second edition, 2008.

[100] H. T. Nguyen, C. T. Nguyen, T. Ino, B. Indurkhya, and M. Nakagawa. Text-Independent Writer Identification using Convolutional Neural Network. *Pattern Recognition Letters*, 121:104–112, 2019.

[101] H. T. Nguyen, C. T. Nguyen, and M. Nakagawa. ICFHR 2018 – Competition on Vietnamese Online Handwritten Text Recognition using HANDS-VNOnDB (VOHTR2018). In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 494–499, 2018.

[102] T. Nishide and K. Ohta. Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol. *Public Key Cryptography – PKC 2007, LNCS 4450*, pages 343–360, 2007.

[103] O. C. Novac, M. C. Chirodea, C. M. Novac, N. Bizon, M. Oproescu, O. P. Stan, and C. E. Gordan. Analysis of the Application Efficiency of TensorFlow and PyTorch in Convolutional Neural Network. *Sensors*, 22(22), 2022.

[104] I. Oguiza. tsai - A State-of-the-art Deep Learning Library for Time Series and Sequential Data, 2020. https://github.com/timeseriesAI/tsai.

[105] Onnx. Onnx/onnx: Open Standard for Machine Learning Interoperability.

[106] F. Ott, D. Rügamer, L. Heublein, T. Hamann, J. Barth, B. Bischl, and C. Mutschler. Benchmarking Online Sequence-to-Sequence and Character-based Handwriting Recognition from IMU-Enhanced Pens. *International Journal on Document Analysis and Recognition (IJDAR)*, 25(4):385–414, 2022.

[107] F. Ott, D. Rügamer, L. Heublein, B. Bischl, and C. Mutschler. Cross-Modal Common Representation Learning with Triplet Loss Functions. In *arXiv 2202.07901*, 2022.

[108] F. Ott, D. Rügamer, L. Heublein, B. Bischl, and C. Mutschler. Domain Adaptation for Time-Series Classification to Mitigate Covariate Shift. In *arXiv 2204.03342*, 2022.

[109] F. Ott, D. Rügamer, L. Heublein, B. Bischl, and C. Mutschler. Joint Classification and Trajectory Regression of Online Handwriting using a Multi-Task Learning Approach. *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1244–1254, 2022.

[110] F. Ott, M. Wehbi, T. Hamann, J. Barth, B. M. Eskofier, and C. Mutschler. The OnHW Dataset: Online Handwriting Recognition from IMU-Enhanced Ballpoint Pens with Machine Learning. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(3):1–20, 2020.

[111] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology — EUROCRYPT'99, LNCS 1592*, pages 223–238, 1999.

[112] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015.

[113] D. S. Park, W. Chan, Y. Zhang, C. C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. *arXiv preprint arXiv:1904.08779*, 2019.

[114] D. S. Park, Y. Zhang, Y. Jia, W. Han, C. C. Chiu, B. Li, Yonghui Wu, and Quoc V Le. Improved Noisy Student Training for Automatic Speech Recognition. *arXiv preprint arXiv:2005.09629*, 2020.

[115] A. Paszke, S. Gross, F. Massa, A. Lerer, G. Bradbury, J.and Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, NeurIPS 2019, pages 8024–8035, 2019.

[116] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[117] G. Pirlo, M. Diaz, M. A. Ferrer, D. Impedovo, F. Occhionero, and U. Zurlo. Early Diagnosis of Neurodegenerative Diseases by Handwritten Signature Analysis. In *New Trends in Image Analysis and Processing–ICIAP 2015, Proceedings 18*, pages 290–297, 2015.

[118] A. Porwal, E. Carranza, and M. Hale. A Hybrid Fuzzy Weights-of-evidence Model for Mineral Potential Mapping. *Natural Resources Research*, 15(1):1–14, 2006.

[119] A. H. Priya, S. Mishra, S. A. Raj, S. Mandal, and S. Datta. Online and Offline Character Recognition: A Survey. *2016 International Conference on Communication and Signal Processing (ICCSP)*, pages 0967–0970, 2016.

[120] P. Radivojac, Z. Obradovic, A. K. Dunker, and S. Vucetic. Feature Selection Filters Based on the Permutation Test. In *15th European Conference on Machine Learning*, ECML'04, 2004.

[121] E. Rahimian, S. Zabihi, S.F. Atashzar, A. Asif, and A. Mohammadi. XceptionTime: A Novel Deep Architecture based on Depthwise Separable Convolutions for Hand Gesture Classification. In *arXiv 1911.03803*, 2019. http://arxiv.org/abs/1911.03803.

[122] M. Ravanelli and Y. Bengio. Speaker Recognition from Raw Waveform with SincNet. In *2018 IEEE Spoken Language Technology Workshop*, SLT, pages 1021–1028, 2018.

[123] K. Raza. Chapter 8 — Improving the Prediction Accuracy of Heart Disease with Ensemble Learning and Majority Voting Rule. *U-Healthcare Monitoring Systems*, pages 179–196, 2019.

[124] S. J. Reddi, S. Kale, and S. Kumar. On the Convergence of Adam and Beyond. In *International Conference on Learning Representations*, 2018.

[125] A. Rehman, S. Naz, and M. I. Razzak. Writer Identification using Machine Learning Approaches: A Comprehensive Review. *Multimedia Tools and Applications*, 78(8):10889–10931, 2019.

[126] M. Ribeiro, S. Singh, and C. Guestrin. "Why Should I Trust You?": Explaining the Predictions of any Classifier. *Proceedings of the Demonstrations Session, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT 2016)*, pages 97–101, 2016.

[127] Peter S., Mark S., and Luisa S. Multiparty Computation with Covert Security and Public Verifiability. Cryptology ePrint Archive, Paper 2021/366, 2021. https://eprint.iacr.org/2021/366.

[128] K. Saranya and M. S. Vijaya. An interactive Tool for Writer Identification based on Offline Text Dependent Approach. *International Journal of Advanced Research in Artificial Intelligence (IJARAI)*, 2(1), 2013.

[129] K. Saranya and M. S. Vijaya. Text Dependent Writer Identification using Support Vector Machine. *International Journal of Computer Applications*, 65(2), 2013.

[130] U. Schlegel, H. Arnout, M. El-Assady, D. Oelke, and D. Keim. Towards a Rigorous Evaluation of XAI Methods on Time Series. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 4197–4201, 2019.

[131] S. Seto, W. Zhang, and Y. Zhou. Multivariate Time Series Classification Using Dynamic Time Warping Template Selection for Human Activity Recognition. In *arXiv 1512.06747*, 2015. https://arxiv.org/abs/1512.06747.

[132] A. Shivram, C. Ramaiah, S. Setlur, and V. Govindaraju. IBM_UB_1: A Dual Mode Unconstrained English Handwriting Dataset. In *2013 12th International Conference on Document Analysis and Recognition (ICDAR)*, pages 13–17, 2013.

[133] H. Singh, R. K. Sharma, and V. P. Singh. Online Handwriting Recognition Systems for Indic and Non-Indic Scripts: A Review. *Artificial Intelligence Review*, 54(2):1525–1579, 2021.

[134] N. Singh, R.A. Khan, and R. Shree. Applications of Speaker Recognition. *Procedia Engineering*, 38:3122–3126, 2012.

[135] D. Sztahó, G. Szaszák, and A. Beke. Deep Learning Methods in Speaker Recognition: A Review. *Periodica Polytechnica Electrical Engineering and Computer Science*, 65(4):310–328, 2021.

[136] G. J. Tan, G. Sulong, and M. S. M. Rahim. Writer identification: A comparative study across three world major languages. *Forensic Science International*, 279:41–52, 2017.

[137] F. Teixeira, A. Abad, B. Raj, and I. Trancoso. Towards End-to-End Private Automatic Speaker Recognition. *arXiv preprint arXiv:2206.11750*, 2022.

[138] C. Toli, A. Aly, and B. Preneel. Privacy-preserving multibiometric authentication in cloud with untrusted database providers. *Cryptology ePrint Archive Paper 2018/359*, 2018.

[139] D. Wagh, S.and Gupta and N. Chandran. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proceedings on Privacy Enhancing Technologies Symposium*, 2019(3):26–49, 2019.

[140] J. Wang and F. Chuang. An Accelerometer-based Digital Pen with a Trajectory Recognition Algorithm for Handwritten Digit and Gesture Recognition. *IEEE Transactions on Industrial Electronics*, 59(7):2998–3007, 2012.

[141] Q. Wang, X. Lin, M. Zhou, Y. Chen, C. Wang, Q. Li, and X. Luo. VoicePop: A Pop Noise based Anti-spoofing System for Voice Authentication on Smartphones. In *IEEE Conference on Computer Communications*, INFOCOM 2019, pages 2062–2070, 2019.

[142] Z. Wang, W. Yan, and T. Oates. Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline. In *arXiv 1611.06455*, 2016. http://arxiv.org/abs/1611.06455.

[143] A. Wasay, B. Hentschel, Y. Liao, S. Chen, and S. Idreos. MotherNets: Rapid Deep Ensemble Learning. *Proceedings of Machine Learning and Systems*, 2:199–215, 2020.

[144] K. Weinberger and L. Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *Journal of Machine Learning Research*, 10(2):207–244, 2009.

[145] E. Wenger, M. Bronckers, C. Cianfarani, J. Cryan, A. Sha, H. Zheng, and B. Y. Zhao. "Hello, It's Me": Deep Learning-Based Speech Synthesis Attacks in the Real World. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 235–251, 2021.

[146] R. A. Wilkinson, J. Geist, S. Janet, P. J. Grother, C. J. C. Burges, R. Creecy, B. Hammond, J. J. Hull, N. Larsen, T. P. Vogl, et al. *The First Census Optical Character Recognition System Conference*, volume 184. 1992.

[147] L. Yang. Classifiers Selection for Ensemble Learning based on Accuracy and Diversity. *Procedia Engineering*, 15:4266–4270, 2011.

[148] W. Yang, L. Jin, and M. Liu. DeepWriterID: An End-to-End Online Text-Independent Writer Identification System. *IEEE Intelligent Systems*, 31(2):45–53, 2016.

[149] S. Yerima, S. Sezer, and I. Muttik. High Accuracy Android Malware Detection Using Ensemble Learning. *IET Information Security*, 9(6):313–320, 2015.

[150] L. Zhang, S. Tan, Y. Chen, and J. Yang. A Phoneme Localization Based Liveness Detection for Text-independent Speaker Verification. *IEEE Transactions on Mobile Computing*, pages 1–14, 2022.

[151] L. Zhang, S. Tan, J. Yang, and Y. Chen. VoiceLive: A Phoneme Localization Based Liveness Detection for Voice Authentication on Smartphones. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1080–1091, 2016.

[152] X. Zhang, Y. Zhuang, H. Hu, and W. Wang. 3-D Laser-Based Multiclass and Multiview Object Detection in Cluttered Indoor Scenes. *IEEE Transactions on Neural Networks and Learning Systems*, 28(1):177–190, 2017.

[153] X. Y. Zhang, G. S. Xie, C. L. Liu, and Y. Bengio. End-to-End Online Writer Identification With Recurrent Neural Network. *IEEE Transactions on Human-Machine Systems*, 47(2):285–292, 2017.

[154] Z. Zhou. *Ensemble Methods: Foundations and Algorithms*. 2012.

[155] H. Zhu and W. K. Ng. Beaver Triple Generator from Multiplicatively Homomorphic Key Management Protocol. In *Advanced Information Networking and Applications*, AINA 2022 LNNS 451, pages 492–503, 2022.

[156] H. Zhu and W. K. Ng. Highly Scalable Beaver Triple Generator from Additively Homomorphic Encryption. In *Advanced Information Networking and Applications*, AINA 2022 LNNS 451, pages 504–514, 2022.

[157] R. S. Zhu, H.and Mong Goh and W. K. Ng. Privacy-Preserving Weighted Federated Learning Within the Secret Sharing Framework. *IEEE Access*, 8:198275–198284, 2020.

[158] X. Zou, Z. Wang, Q. Li, and W. Sheng. Integration of Residual Network and Convolutional Neural Network along with Various Activation Functions and Global Pooling for Time Series Classification. *Neurocomputing*, 367:39–45, 2019.

Appendix A

Models Trained

Appendix A was part of the paper that resulted in Chapter 2.

A.1 Machine Learning Algorithms

A.1.1 Strategy 1: ML Models Trained on Fixed-Length Feature Vectors

As explained in Section 2.2.2, we utilize tsfresh [34] to extract fixed-length feature vectors, which are then used as input in training some ML models. To our knowledge this is the first instance in which tsfresh has been applied to the OnHW-chars dataset. In order to provide a fair comparison with the accuracy results as reported in Table 4 in [110], we train Random Forest, Decision Tree, Logistic Regression, Linear Support Vector Machines, and k-Nearest Neighbor algorithms. To further improve accuracy, we implement metric learning as described in [144] for KNN. Additionally, we train the Extra-Trees classifier implemented using scikit-learn [116], based off of [57], and SVM with nonlinear kernels. Our implementation mainly uses the software package scikit-learn [116]. In the following, we provide an overview and explanation for each algorithm we implement following Strategy 1.

Decision Tree: Decision Trees are tree-like structures where each branch represents a test on a feature, and the leaf nodes represent classes. Starting at the root node, we perform a sequence of tests to move along the tree. The prediction is based on the leaf node we

end up on. To establish a baseline comparison with the results in [110], we use decision trees with default parameters as in scikit-learn [116], and no additional preproccessing is applied on the extracted feature set.

Random Forest: Random Forests are a collection of individual decision trees. This collection operates together as an ensemble where each tree gets a vote for the end label/class. The class with the most votes is the one that is chosen in the end. As before, we would like a baseline comparison with [110] to outline the effects tsfresh had on the end classification accuracy. We choose scikit-learn's default parameters: 100 trees, no defined max depth, minimal sample split of 2, and minimal samples leaf of 1.

Extra Trees: Extra Trees outlined in [57] is implemented in scikit-learn [116], and they are similar to a Random Forests with a few key differences. These are not implemented in [110]. We implement them with the default parameters of scikit-learn, similiar to the above mentioned models. This gives us 100 trees, no defined maximum depth, a minimum sample split of 2 and no specified max leaf nodes.

Logistic Regression: Logistic Regression is also a baseline we would like to establish to further outline the effects tsfresh has on the end classification accuracy. First the QuantileTransformer [116] is fitted to the selected feature set with 1000 quantiles and a uniform output distribution. After fitting the transformer it is applied to both the test and train sets to scale each of the features. When running logistic regression we choose default parameters as in scikit-learn [116], except that we changed the default maximum iterations parameter from 100 to 1000.

KNN: KNN is a simple non-parametric algorithm which classifies new observations based on the distance from known observations. We use scikit-learn's default parameter $k = 5$ to show the impact of our feature extraction method in comparison with the one in [110]. Using the selected features from Section 2.2.2, we scale the feature set using QuantileTransformer [116] in the same way we did previously for logistic regression. No dimensionality reduction is performed. To find an appropriate level of n_significant, we construct a series of models with varying levels of n_significant, as seen in Figure A.1. Here we take n_significant = 17 for the feature selection step of all our models. We note that this choice may not be optimal and a better choice may exist.

Supervised Metric Learning with KNN: As an extension to the previously mentioned KNN algorithm, we apply metric learning in a supervised fashion to improve accuracy scores. The pipeline remains identical, but instead of using QuantileTransformer, neighborhood component analysis (NCA) from scikit-learn [116] is implemented to learn a distance metric and reduce the dimensionality of the test and train sets. NCA improves KNN accuracy by directly maximizing the stochastic variant of the leave-one-out KNN score on the training set. When in use, NCA requires standardized data, so StandardScalar [116] was first applied to the selected features. For NCA, we specify the parameters "init = LDA", so that LDA (i.e., linear discriminant analysis) is used to initialize the linear transformation, and "n_components = 20", reducing the inputted feature space down to 20 features. This choice of n_components is obtained from the accuracy of KNN models over different levels of n_components, as seen in Figure A.2.

SVM: Similiar to KNN, SVM aims to group observations based on their distance from known groups. QuantileTransformer is applied to the feature set in the same fashion as we used it before. When running SVM, we used the same parameters as in [110]. Furthermore, we implement SVM with both a linear and Gaussian kernel.

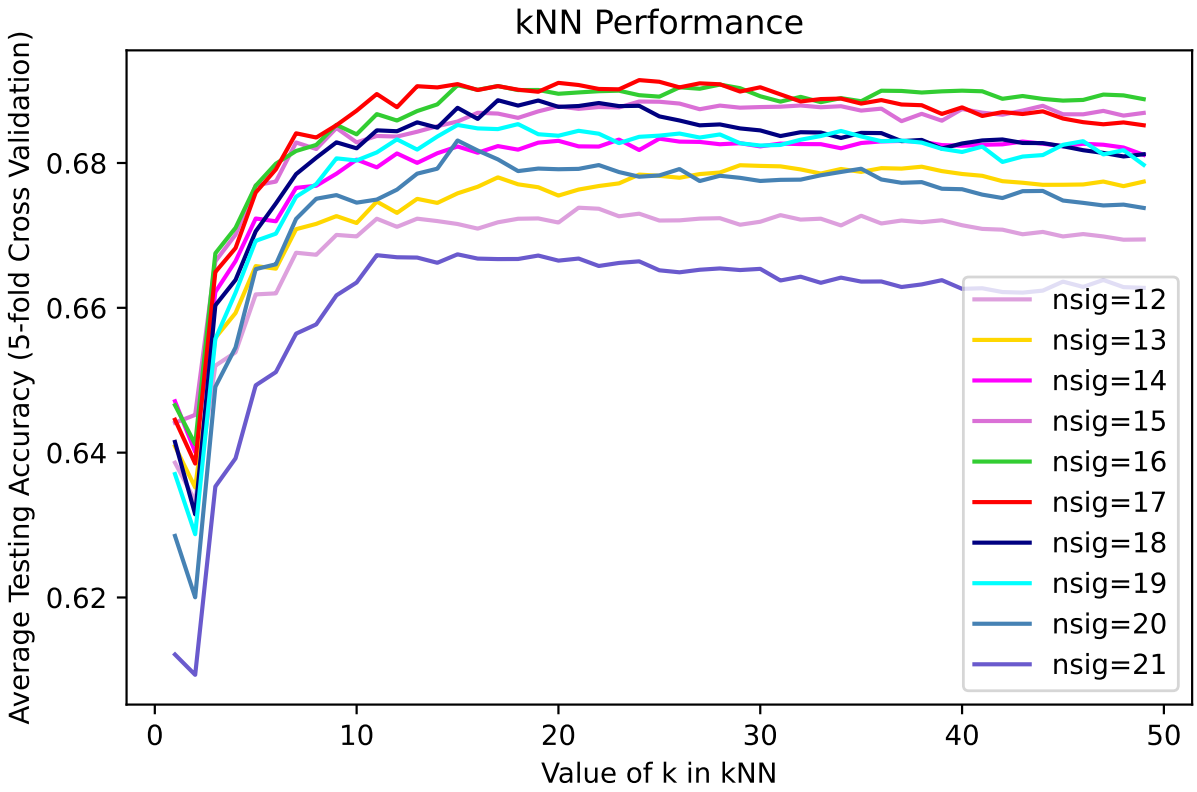
A.1.2 Strategy 2: ML Models Trained on Variable-Length Time Series Data

As opposed to some ML models that receive fixed-length feature vectors as input, some ML models can directly work with variable length time-series data. In our second strategy, we train dynamic time warping (DTW) models based on the preprocessed and filtered OnHW-chars dataset as explained in Section 2.2.1.

DTW: DTW is a technique to measure the optimal alignment between 2 time series. [74]. One main advantage to using DTW as a similarity measure, as opposed to Euclidean distance, is that DTW supports vectors of differing lengths. One downside to DTW, however, is that it is not a distance metric as it does not satisfy the triangle inequality and it is not positive definite [74]. Despite this, it is still often used to measure (or approximate) the distance between two time series. DTW has grown in use for time series data clustering [131].

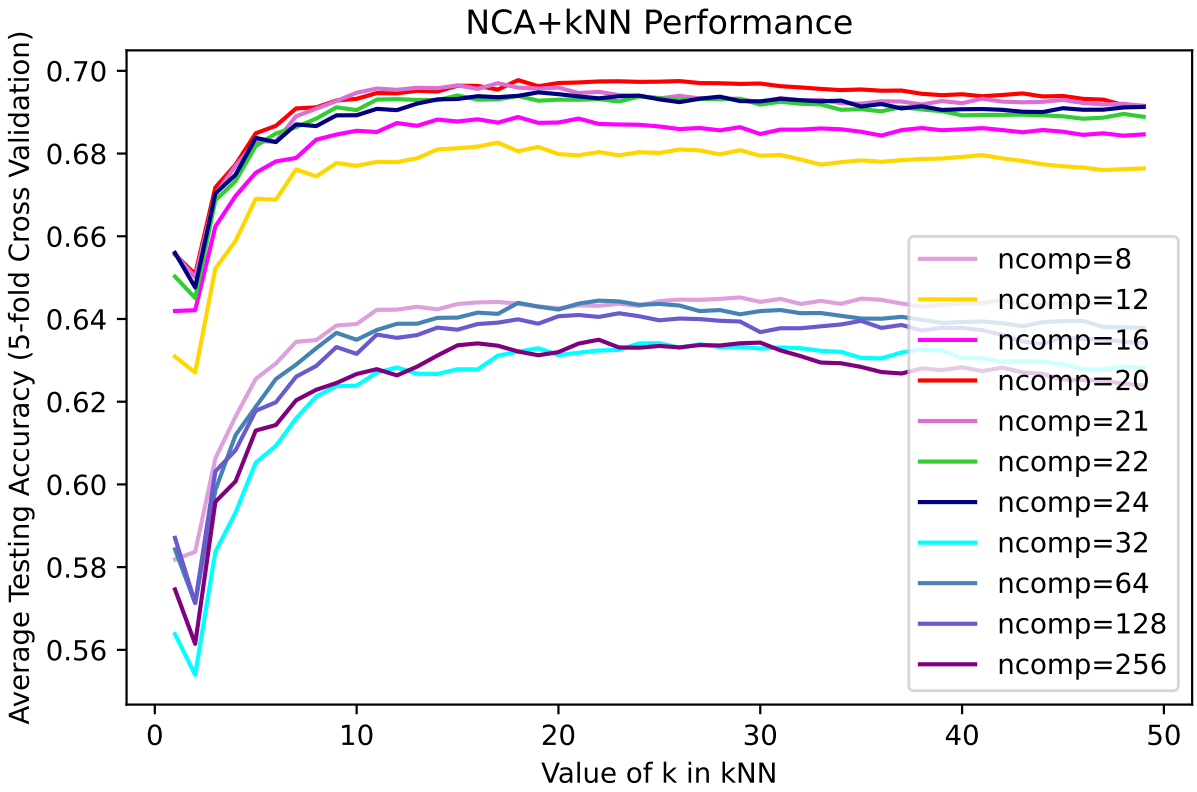
Having support for differing length vectors is important for time series data and particularly crucial for sensor data such as the OnHW-chars dataset [110], since these vectors

Figure A.1: KNN accuracy with various levels of n_significant, using lowercase writer-independent (L-WI) data.



will often have different lengths. One approach is to zero-pad the time series and determine the distance between these zero-padded vectors using Euclidean distance, however Euclidean distance compares the points in each vector in sequence [25]. This is problematic for the OnHW-chars dataset because the actual length it takes to write each letter is inconsistent and slight inconsistencies in the timing will massively affect the similarity and particularly problematic when there are large differences in lengths. Variable length can also be attributed to issues with the sensor. Similarity measures which are flexible to variable length vectors, such as DTW, are preferred in this case. For our purposes, these similarity measures will be used in a KNN algorithm to classify the vectors.

Figure A.2: KNN accuracy over various levels of n_components, using lowercase writer-independent (L-WI) data.



A.2 Deep Learning Algorithms

FCN: FCN has been shown to achieve state-of-the-art performance on the task of classifying time series sequences [142]. The basic block of FCN is a convolutional layer followed by a batch normalization layer and a ReLU activation layer. The final network is built by stacking three convolution blocks. Like ResNet, the FCN architecture excludes pooling operations to prevent overfitting [142]. In order to improve generalization, batch normalization is applied to speed up convergence speed. After the convolution blocks, the features are fed into a global average pooling layer instead of a fully connected layer, reducing the number of weights. The final label is produced by a softmax layer [142].

LSTM: An LSTM is a type of RNN that can learn long-term dependencies between time steps of sequential data. Contrary to CNN, an LSTM can remember the state of the network between predictions. The essential components of an LSTM network are a sequence input layer to incorporate time-series data into the network and an LSTM layer to learn long-term dependencies between time steps of sequence data. The LSTM layer contains hidden units providing inputs to memory cells and their corresponding gate units. All units (except for gate units) have connections to all units in the next layer [66]. In our implementation, we set the LSTM parameter n_layers in tsai [104] to 2.

BiLSTM: The BiLSTM network extends the traditional LSTM networks. While the LSTM layer considers the time sequence in a forward direction, the BiLSTM layer considers it both backward and forwards [60]. Indeed, the BiLSTM network trains two LSTM networks on the input sequence. During this process, the first recurrent layer is replicated in the network, and therefore two layers are created side-by-side. The input sequence will be an input to the first layer; meanwhile, its reversed replica will be an input to the second layer. This approach adds additional context to the network, resulting in faster and better model learning.

LSTM-FCN & BiLSTM-FCN: It has been shown that the performance of FCN for time series classification can enhance by adding LSTM sub-modules [69]. The fully convolutional part of this architecture consists of three temporal convolutional blocks. Each block contains a temporal convolutional layer, accompanied by batch normalization followed by a ReLU activation function. Following the final convolution block, global average pooling is applied. In parallel, the time-series input is fed into a dimension shuffle layer. Next, the transformed time series as the output of the shuffle layer is passed into the LSTM block containing the LSTM layer, followed by a dropout. Finally, the output of the global pooling layer (from the fully convolutional part) and the LSTM block are concatenated and passed onto a softmax classification layer [69].

MLSTM-FCN & MBiLSTM-FCN: MLSTM-FCN and MBiLSTM-FCN are multivariate time series classification models whose architecture is based on the univariate time series classification models, including LSTM-FCN and Attention LSTM-FCN. The fully convolutional part in both groups (i.e., univariate and multivariate time series classification models) consists of three temporal convolutional blocks. However, compared to LSTM-FCN, the first two convolutional blocks conclude with a squeeze-and-excite block

that adaptively recalibrates the input feature maps. This process can be considered as a form of learned self-attention on the output feature maps of prior layers [70].

ResNet: Similar to FCN, the architecture of ResNet consists of three residual blocks, followed by a global average pooling layer and a softmax layer. The presence of a shortcut connection in each residual block makes the structure of ResNet very deep and enables the gradient to flow directly through the bottom layers. The filters in both architectures of FCN and ResNet are very similar. While the convolution extracts the local features in the temporal axis, the sliding filters take into account the dependencies among different time intervals and frequencies. Compared to FCN, ResNet is claimed to be a better candidate to be applied to larger and more complex data because it is more likely to strike a good trade-off between generalization and interpretability [142].

ResCNN: ResCNN is a hybrid scheme for time series classification that integrates a residual network with a CNN. In ResCNN, the strength of ResNet and CNN are combined. ResNet can learn highly complex patterns in the data due to the presence of a shortcut connection technique. However, this technique is computationally expensive and can easily cause overfitting. On the other hand, although CNN is capable of learning the temporal and spatial patterns from raw data, it cannot recover the complex patterns in the data because of few levels of the network. The architecture of ResCNN is constructed by facilitating a residual learning block at the first three convolutional layers to incorporate the strength of both networks. Additionally, batch normalization and diverse activation functions are adopted in different layers of ResCNN to enhance the nonlinear abstraction capacity. Moreover, in order to avoid overfitting, the pooling operation is removed, and the features are fed into a global average pooling instead of a fully connected layer [158].

InceptionTime: Inspired by the Inception-v4 architecture, the InceptionTime is an ensemble of 5 Inception networks, with each prediction given an even weight. Each Inception network contains only two residual blocks compare to ResNet, with three residual blocks. Each block in the inception network comprises three Inception modules rather than traditional fully convolutional layers. The first principal component of the Inception module is the "bottleneck" layer which allows the Inception network to have much longer filters than ResNet (almost ten times), with roughly the same number of parameters to be learned. The second major component of the Inception module is sliding multiple filters of different lengths simultaneously on the same input time series [53].

XceptionTime: Inspired by InceptionTime, XceptionTime is designed to be independent of the time window. The use of adaptive average pooling in this architecture makes XceptionTime more robust to the temporal translation of the inputs as the temporal information will sum out. One key difference between the XceptionTime module and InceptionTime module previously proposed in [53], is adopting depthwise separable convolutions, which significantly mitigates the required number of parameters in the network and also can lead to higher accuracy [121].

XCM: Compared to typical CNN architectures, XCM extracts observed variables features (2D convolution filters) and time features (1D convolution filters) directly from the input data. Features related to time fully incorporate the timing information from the input data, not from the processed features related to observed variables (features maps from 2D convolution filters). Therefore, on average, this process can lead to a better classification performance than the 2D/1D sequential approach. XCM uses 1D global average pooling followed by a softmax layer for classification, which reduces the number of trainable parameters and improves the network's generalization ability [52].