ENNigma: Uma Biblioteca para Redes Neuronais Privadas

PEDRO MANUEL BARROS BARBOSA

Outubro de 2023

ENNigma: A Framework for Private Neural Networks

Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development

Pedro Manuel Barros Barbosa

1180592

Thesis submitted for:

Master's Degree in Artificial Intelligence Engineering

Supervised by:
Dr. Isabel Cecília Correia da Silva Praça Gomes Pereira

Co-Supervised by:
Dr. Ivone de Fátima da Cruz Amorim
Dr. Eva Catarina Gomes Maia

Porto, October 3, 2023

“The best way to predict your future is to create it.”

- P. Drucker & A. Lincoln

Abstract

The increasing concerns about data privacy and the stringent enforcement of data protection laws are placing growing pressure on organizations to secure large datasets. The challenge of ensuring data privacy becomes even more complex in the domains of Artificial Intelligence and Machine Learning due to their requirement for large amounts of data. While approaches like differential privacy and secure multi-party computation allow data to be used with some privacy guarantees, they often compromise data integrity or accessibility as a tradeoff. In contrast, when using encryption-based strategies, this is not the case. While basic encryption only protects data during transmission and storage, Homomorphic Encryption (HE) is able to preserve data privacy during its processing on a centralized server. Despite its advantages, the computational overhead HE introduces is notably challenging when integrated into Neural Networks (NNs), which are already computationally expensive.

In this work, we present a framework called ENNigma, which is a Private Neural Network (PNN) that uses HE for data privacy preservation. Unlike some state-of-the-art approaches, ENNigma guarantees data security throughout every operation, maintaining this guarantee even if the server is compromised. The impact of this privacy preservation layer on the NN performance is minimal, with the only major drawback being its computational cost. Several optimizations were implemented to maximize the efficiency of ENNigma, leading to occasional computational time reduction above 50%.

In the context of the Network Intrusion Detection System application domain, particularly within the sub-domain of Distributed Denial of Service attack detection, several models were developed and employed to assess ENNigma's performance in a real-world scenario. These models demonstrated comparable performance to non-private NNs while also achieving the two-and-a-half-minute inference latency mark. This suggests that our framework is approaching a state where it can be effectively utilized in real-time applications.

The key takeaway is that ENNigma represents a significant advancement in the field of PNN as it ensures data privacy with minimal impact on NN performance. While it is not yet ready for real-world deployment due to its computational complexity, this framework serves as a milestone toward realizing fully private and efficient NNs.

Keywords: Artificial Intelligence, Neural Networks, Data Privacy, Private Neural Networks, Homomorphic Encryption, TFHE

Resumo

As preocupações crescentes com a privacidade de dados e a implementação de leis que visam endereçar este problema, estão a pressionar as organizações para assegurar a segurança das suas bases de dados. Este desafio torna-se ainda mais complexo nos domínios da Inteligência Artificial e *Machine Learning*, que dependem do acesso a grandes volumes de dados para obterem bons resultados. As abordagens existentes, tal como *Differential Privacy* e *Secure Multi-party Computation*, já permitem o uso de dados com algumas garantias de privacidade. No entanto, na maioria das vezes, comprometem a integridade ou a acessibilidade aos mesmos. Por outro lado, ao usar estratégias baseadas em cifras, isso não ocorre. Ao contrário das cifras mais tradicionais, que apenas protegem os dados durante a transmissão e armazenamento, as cifras homomórficas são capazes de preservar a privacidade dos dados durante o seu processamento. Nomeadamente se o mesmo for centralizado num unico servidor. Apesar das suas vantagens, o custo computacional introduzido por este tipo de cifras é bastante desafiador quando integrado em Redes Neurais que, por natureza, já são computacionalmente pesadas.

Neste trabalho, apresentamos uma biblioteca chamada ENNigma, que é uma Rede Neural Privada construída usando cifras homomórficas para preservar a privacidade dos dados. Ao contrário de algumas abordagens estado-da-arte, a ENNigma garante a segurança dos dados em todas as operações, mantendo essa garantia mesmo que o servidor seja comprometido. O impacto da introdução desta camada de segurança, no desempenho da rede neural, é mínimo, sendo a sua única grande desvantagem o seu custo computacional. Foram ainda implementadas diversas otimizações para maximizar a eficiência da biblioteca apresentada, levando a reduções ocasionais no tempo computacional acima de 50%.

No contexto do domínio de aplicação de Sistemas de Detecção de Intrusão em Redes de Computadores, em particular dentro do subdomínio de detecção de ataques do tipo *Distributed Denial of Service*, vários modelos foram desenvolvidos para avaliar o desempenho da ENNigma num cenário real. Estes modelos demonstraram desempenho comparável às redes neurais não privadas, ao mesmo tempo que alcançaram uma latência de inferência de dois minutos e meio. Isso sugere que a biblioteca apresentada está a aproximar-se de um estado em que pode ser utilizada em aplicações em tempo real.

A principal conclusão é que a biblioteca ENNigma representa um avanço significativo na área das Redes Neurais Privadas, pois assegura a privacidade dos dados com um impacto mínimo no desempenho da rede neural. Embora esta ferramenta ainda não esteja pronta para utilização no mundo real, devido à sua complexidade computacional, serve como um marco importante para o desenvolvimento de redes neurais totalmente privadas e eficientes.

Palavras-chave: Artificial Intelligence, Neural Networks, Data Privacy, Private Neural Networks, Homomorphic Encryption, TFHE

Acknowledgement

I would like to express my sincere thanks to the Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development (GECAD) for not only proposing an intriguing research topic but also providing me with the essential resources necessary to make this work a reality. I am also thankful to my colleagues at GECAD for promoting an environment of dialogue and collaboration, which has immeasurably enhanced this journey.

To my supervisors, Dr. Isabel Praça, Dr. Ivone Amorim, and Dr. Eva Maia, I am grateful for your advice and expertise. It has been indispensable and invaluable in guiding this thesis on the right path and, particularly, helping me get better at writing.

Lastly, my family and friends have been a solid foundation throughout this journey. Their support has been instrumental in my ability to persist and succeed.

This work was partially supported by the Norte Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (ERDF) within the project "Cybers SeC IP" (NORTE-01-0145-FEDER-000044). This work has also received funding from the project UIDB/00760/2020.

Contents

List of Figures

List of Tables

List of Acronyms

AF	Activation Function.
AI	Artificial Intelligence.
API	Application Programming Interface.
DDoS	Distributed Denial of Service.
DL	Deep Learning.
DNN	Deep Neural Network.
FHE	Fully Homomorphic Encryption.
HE	Homomorphic Encryption.
IDS	Intrusion Detection System.
LF	Loss Function.
ML	Machine Learning.
MSE	Mean-Squared Error.
NIDS	Network Intrusion Detection System.
NN	Neural Network.
PNN	Private Neural Network.
SIMD	Single Instruction, Multiple Data.
SotA	State-of-the-Art.

Chapter 1

Introduction

This chapter serves as an introduction to this thesis. It starts by presenting the context of this work, with a particular focus on the data privacy aspect of Neural Networks (NNs) and the role that Homomorphic Encryption (HE) plays in that context. Following that will be a section detailing and explaining the problem found and the objectives defined for this thesis. Then, our main contributions are also presented, as well as a description of the structure of this document.

1.1 Context

As the Internet expands, with an estimated 500 billion connected devices projected by 2030 [1], the importance of safeguarding data is increasingly apparent. While the concept of privacy may not have a universal definition [2], there is a growing agreement among organizations and governments that safeguarding data is crucial, especially sensitive information. Due to this increasing awareness, governments have started to introduce legislation aimed at imposing strict limitations on the use and handling of such data.

Encryption appears as a promising solution to mitigate data privacy concerns by allowing data to be transmitted and stored in an encoding format. This is done in a way that only people who know a secret decryption key can decode it into a legible form [3]. Even though this solves some problems, the processing of the data still has to be done using it in its non-encrypted form. With the rise of methods that are able to handle large bulks of data at once, like Artificial Intelligence (AI), it is more common to find hundreds or thousands of records in a system being processed simultaneously. Due to the lack of practical alternatives, this data is often unencrypted, which means that if a malicious attacker compromises one of these systems, it could have access to large volumes of sensitive data [4]. Therefore, developing methods to process data while limiting this risk of exposure is paramount.

There are already some strategies to perform this data processing in NN privately, but all come at a significant cost [5]. For example, federated learning and model splitting rely on decentralized data processing to ensure that no single involved party knows anything about the whole data. In this approach, each party may know an insignificant part of the whole data, but they cannot guess anything from it [6]. On the other hand, differential privacy limits the ability to see the data as-is. This is because this approach adds noise to the data in an attempt to prevent anyone from extracting information about any individual entry [7].

HE is an encryption technique that does not impose limitations on the data itself. In fact, its limitations are confined to the computational tasks, as it only supports a subset of operations and adds significant computational overhead. If these limitations are mitigated, the HE strategy could become more practical. This, in turn, could lead to the creation of better-performing models due to better access to the actual data. That is due to the fact that decentralized data processing makes the process of generalization more difficult as each distributed group will have unique features and the fact that adding noise directly affects the final model performance [8].

At the same time, besides protecting the privacy of the data, it is also essential to protect the connected devices themselves from external threats to ensure their proper functioning and security. There are already various types of malicious attacks being used to target these devices, such as Distributed Denial of Service (DDoS), which can disrupt accesses and temporarily make them inoperable. Looking into this specific type of attack, Kaspersky reported that in Q3 of 2022, there was an increase in DDoS attacks of 47.87% from Q3 of 2021 [9]. Some of these attacks can be very impactful, like the one in February 2021, where Crypto Currency Exchange EXMO was taken offline for 2 hours following a massive DDoS attack that reached 30 gigabytes per second [10], or the one in September 2022, where Activision Blizzard was targeted by a DDoS attack which resulted in an outage that lasted three-and-a-half hours counting from the first public acknowledgment of the attack [11]. This shows that creating solutions capable of accurately identifying these attacks is critical.

DDoS attack detection models are used to detect and mitigate the impact of such attacks. However, most of the modern DDoS attack detection is made using NNs [12], which requires large amounts of high-quality data to function correctly [13]. Yet, due to privacy concerns and regulations, this private data, which is often the most useful for NN training [14], is omitted or obfuscated in the final published datasets. Consequently, employing techniques such as using HE for data privacy in NNs could enable the use of data that was previously unshareable. This, in turn, could lead to the creation of better-performing models.

That said, not all organizations have the capability to create and utilize such Network Intrusion Detection System (NIDS). Therefore, organizations often outsource their security to third-party entities that offer this service, commonly referred to as Managed Security Services [15]. This outsourcing is itself a possible privacy vulnerability, as the data sent for security analysis needs to be decrypted before being processed. However, when using HE, it becomes possible for these providers to use data privately, even in NN models. Therefore, making NNs private using HE would also enhance the data privacy of organizations that outsource their security.

In summary, examining the current reality of Private Neural Network (PNN), which are NNs that preserve the privacy of the data (also known as privacy-preserving NNs), it is clear that the usage of HE has a significant potential. Yet, computational overhead can be a limitation for real-world usage. Besides that, the creation of DDoS attack detection NN models is possibly being impacted by privacy concerns and regulations regarding data privacy. Thus, creating them using PNNs could prove to be an effective solution. Moreover, DDoS attack detection could benefit from these added privacy guarantees since it can be improved by using data that is now not available due to privacy concerns. As such, using the data privacy guarantees offered by HE makes the training and inference of NN models possible without privacy concerns. Consequently, by providing additional privacy guarantees, organizations may securely outsource their data.

1.2 Problem

The integration of HE in NNs faces a significant limitation due to computational costs, and joining these two areas is not currently applicable in real-world scenarios. This is clearly an existing challenge that needs to be addressed to enable the development of better and more secure NN models, not just in DDoS attack detection but across various domains.

There are already some works that address the usage of HE in NNs for data privacy preservation. The works of Pulido-Gaytan et al. [16] and Podschwadt et al. [17] review the State-of-the-Art (SotA) of those solutions and conclude that the vast majority of the proposed solutions only use HE to protect data privacy during inference. Furthermore, even though few works use HE for data privacy preservation in NN training and inference, there is no comprehensive review of such solutions. This is essential to allow researchers to quickly understand the current state, challenges, limitations, and many other aspects connected to the use of HE in NN training **and** inference. Moreover, in this research area, there is a lack of a ready-to-use PNN library to allow newcomers to get started.

Due to the significant computational overhead introduced by HE, this approach is currently computationally prohibitive. So, analyzing the evolution of HE and choosing the most promising HE scheme is essential to create a solution whose computational overhead is as small as possible. This, of course, does not exclude the necessity for architectural and implementation-level optimizations.

Applying this privacy preservation approach to the NIDS domain is also essential to evaluate its feasibility and kickstart the adoption of such solutions in this field. Due to the fact that the NIDS domain is very large, it may be necessary to start on a specific subdomain and only then expand to cover the whole domain.

Nevertheless, while a no-compromise solution may be far beyond current knowledge, future solutions to this problem must enhance the current works in reducing the computational overhead. In the immediate future, the priority should be to establish a baseline framework that can serve as a foundation for further optimizations and improvements. This baseline would enable a standardized approach to evaluating new strategies for reducing computational costs, thereby accelerating advancements in this critical research area.

1.3 Objectives

Considering the problem stated, the main objectives of this work are: developing a PNN that uses HE for data privacy preservation, creating a DDoS attack detection model using that same NN, and comparing their results with SotA research. Each of these main objectives was then subdivided into more specific ones, namely the following:

- **O1 - Develop an optimized PNN that preserves data privacy using HE**
  - **O1.1** - Analyse the evolution and current applications of data privacy preservation in NN using HE.
  - **O1.2** - Evaluate current challenges and limitations related to the usage of HE for data privacy preservation in NN.
  - **O1.3** - Identify the most promising HE scheme for the task of data privacy preservation in NN.

  - **O1.4** - Conceptualize an optimized PNN architecture that uses the HE scheme identified in O1.3.
  - **O1.5** - Implement a computationally efficient PNN using the architecture defined in O1.4.
  - **O1.6** - Test and validate the architecture and resulting implementation, defined, respectively, in O1.4 and O1.5.
- **O2 - Create a PNN model for DDoS attack detection using the PNN implemented in O1**
  - **O2.1** - Analyse the evolution of NIDS and current solutions, specifically, DDoS attack detection.
  - **O2.2** - Select the current top-performing DDoS attack detection models and respective datasets.
  - **O2.3** - Use the PNN implemented in O1 to create a model for DDoS attack detection.
  - **O2.4** - Evaluate the performance of the model created in O2.3.
- **O3 - Compare the performance of the NN implemented in O1 and the obtained model in O2 with the SotA research.**
  - **O3.1** - Compare the PNN created in O1 with SotA PNN.
  - **O3.2** - Compare the model created in O2 with SotA DDoS attack detection NN models.

1.4 Contributions

The main contributions of this thesis are:

- **C1 - SotA Review** - Review of the SotA related to the usage of HE for data privacy preservation in NN training and inference.
- **C2 - PNN** - This thesis proposes a novel PNN that is capable of learning and inferring without compromising the privacy of the data at any point. All this while having a minimal impact on classification metrics. Additionally, the computational complexity of HE was minimized through the implementation of several optimizations
  - **C3 - Optimized Number Operations** - Efficiently implementing numeric operations while fulfilling the restrictions imposed by HE schemes like TFHE is a complex topic. In this thesis, the algorithms proposed by Song et al. [18] were optimized, which resulted in a significant performance increase.
  - **C4 - Optimized Gradient Calculation** - Calculating the gradient during backpropagation is the most expensive process in NN training. So, the gradient multiplication by the learning rate was exchanged by a shift operation, which reduced the cost of this operation by more than 99%. Even though it did not significantly reduce the final training time, it shows that this is a method that can be used for significant optimizations if applied in critical computational steps.

  - **C5 - Optimized Sigmoid Approximation** - Similarly to the gradient calculation, an optimized version of the Sigmoid approximation proposed by Myers and Hutchinson [19] was also presented. This optimization was done by changing multiplications by shifts and provided an increase between 33.96% and 70.43%.

- **C6 - Study on the impact of a specific HE on NN performance** - A study analyzing the impact of the limitations imposed by the TFHE scheme in the performance of the NN models. This includes metrics and learning pace variations.
- **C7 - PNN DDoS attack detection model** - Using the proposed PNN, a DDoS detection model was proposed that guarantees robust data privacy from start to finish with minimal classification performance impact, as verified by using the standard classification metrics of this domain.

The work done for this thesis also resulted in two scientific publications:

- **Ivone Amorim, Eva Maia, Pedro Barbosa, Isabel Praça [20]** - Performed a detailed SotA review of the usage of HE in NN for data privacy preservation.
- **Ivone Amorim, Pedro Barbosa, Eva Maia, Isabel Praça [21]** - Studied the impacts of the restrictions imposed by the TFHE scheme on NN classification performance applied to the DDoS attack detection domain.

1.5 Document Structure

This document is organized as follows.

The first chapter, named Introduction, is where this section is located. Its objective is to provide the reader with the motivations for this work and the problem that is addressed. Besides that, it also defines our objectives and main scientific contributions.

The second chapter, Background, explains the three leading concepts that are considered the most important to allow non-specialist readers to be able to understand and critically examine the proposed solution. These concepts are: Privacy, Homomorphic Encryption, and Neural Networks.

The third chapter is formed by the SotA review of these three main areas. It starts with the application domain, which is NIDS. Then, it explores the PNN solutions. Finally, it reviews the usage of HE for data privacy preservation in NNs.

The fourth chapter defines the architecture of the proposed PNN using a well-defined methodology. This was created using two architectural visualization models, which allows the architecture to be presented from different perspectives and levels of detail.

The fifth chapter implements the PNN defined in Chapter 4. This includes detailing the custom number operations algorithms required by the usage of the TFHE scheme, as well as several optimization strategies made to ensure the maximum efficiency of the proposed solution. At the end of this chapter, there is a section dedicated to explaining the procedures used to validate the implemented solution.

The sixth chapter uses the implementation provided in Chapter 5 to create an NN model for DDoS attack detection. The objectives of this chapter are two-fold. Firstly, it helps validate the proposed NN architecture and respective implementation while also allowing the computational complexity of using HE to be evaluated in a real-world scenario. It starts by first presenting the dataset and pre-processing steps. It then details the NN hyper-parameters and ends by presenting the results of the creation and execution of such an NN model.

The seventh and last chapter overviews all the work done and presented in this document. It discusses the results achieved, explores the challenges and limitations of the proposed solution, and revisits the objectives defined in Chapter 1. The chapter concludes with recommendations for future work and final remarks from the author.

Chapter 2

Background

This chapter will present the most important concepts referred to in this thesis to allow any non-specialist reader to be able to understand the solution proposed. Firstly, the concept of data privacy will be presented with a particular focus on the usage of encryption and its application in AI. Then, HE will be presented from a not-to-technical standpoint. The final concept that is presented is NN and, contrary to the previous one, this one will be comprehensively detailed from a technical viewpoint.

2.1 Data Privacy

Privacy is an ancient concept that existed long before the current era, but it is generally accepted as only really starting to be well known in the 19th-20th century [2]. Even then, there was no universal definition for privacy, with its concrete form differing according to social, economic, and cultural environments [2]. Privacy is a complex concept that involves several parts that are, in some way, related to each other, with Solove [22] identifying six of them: the right to be let alone, limited access to the self, secrecy, control of personal information, personhood, and intimacy. In the 20th century, privacy started to become acknowledged as a fundamental human right in some countries, and so it began to become legally enforced [2].

With the introduction of computers and the internet, there was a dramatic shift in the flow of information, with small things that once disappeared in seconds now being stored forever in this new information space [23]. So, information that was once spread among friends, family, and neighbors is now distributed all over the internet and stored in remote databases [23]. Even if we want to get out of this data collection, it is nearly impossible to live without being part of it [24].

Moreover, the widespread adoption of digital devices and the significant rise in the number of data being collected and stored deepened existing privacy issues, making the protection of data one of the most important tasks when building new digital systems. In 2018, there was the introduction of the GDPR [25], and in 2020, the introduction of California's CCPA [26]. These two legislations, although following different baselines, transformed data privacy concerns from an ethical obligation to an enforced task whose violation can be punished. This risk of fines and other punitive measures was a significant step to making data privacy preservation a primary consideration for organizations.

2.1.1 Encryption

From the moment data is generated, privacy issues start to arise, namely when transmitting and storing it. Encryption is one of the tools that allow those two actions to be made with reduced risk of possible privacy breaches. It does this by transforming the original data, known as plaintext, into a cryptic form, known as ciphertext, that can only be transformed back to the respective plaintext using a secret key. This allows data to be sent over any network and stored anywhere without risking data breaches, assuming that the key is safely stored. The protection provided by encryption is fundamentally bounded by the ability to keep the key confidential. That is, safeguarding those keys is paramount to ensuring the privacy of encrypted data [27]. There are two main encryption methods: Symmetric and Asymmetric. While the former is faster, more efficient, and more robust [28], it relies on the same key for both encryption and decryption, which can be a problem when communicating between two parties without any pre-shared key [28]. Meanwhile, the latter, because it uses a different key for encryption and decryption, allows encrypted data to be sent between parties without having to debate on a secret key and without having to send a key to one another, avoiding the risk of the key being leaked.

As explained, the basic form of encryption already provides privacy guarantees for data in some scenarios. However, there are more complex encryption algorithms that allow the usage of encryption in different use cases. For example, proxy re-encryption is a type of encryption that allows a ciphertext to be converted from one encryption key to another without ever converting it to plaintext. That is, a ciphertext encrypted with key A can be directly converted to an equivalent ciphertext encrypted with key B. Threshold encryption is a type of encryption that requires, from a poll of keys, a minimum amount to be decrypted, which means that it is possible to encrypt the plaintext and only allow it to be decrypted if at least n% of the key holders agree to do so; finally, HE allows operations to be made directly on the ciphertext, which means that a third-party can compute a result and return it to the client without ever known the plaintext. This latter type is beneficial if the data owner requires some computational that he does not know how to do but is unwilling to share the data.

2.1.2 Data Privacy in AI

AI models require lots of data to be able to train and learn how to solve the problems they are given. However, the usage of data, whether in small or large quantities, can be problematic due to the fact that it can contain private information. So, the data privacy violation risk exists not only in the collection, transmission, and storage of this data but also in its usage for training, inference, and model sharing [5].

Although basic encryption can still be used to protect those models during storage and transmission, it is not that valuable for privacy preservation in AI because it does not provide privacy protection during data processing, which is the place where AI privacy risks are more prominent. However, the added flexibility of more complex encryption methods allows data privacy preservation during training and inference, which is the case of HE.

NN

NNs are a type of AI model that is inspired by the human brain [29] and is able to learn complex mappings on the data that is provided to it during training. How it works will be explained in a follow-up section (Section 2.3), but its privacy issues are similar to the ones faced by other AI models. Besides this fact, its need for more substantial quantities of data (when compared to more traditional AI models) exacerbates these privacy concerns. During non-private training and inference, the data needs to be in plaintext form, which means that if the system is compromised, this data can be accessed by a malicious actor. Besides that, during model storage and transmission, the complex mappings it learned could be reverse-engineered to obtain information about the training data. However, because NNs are a black-box model, this latter issue is less probable in this type of AI model.

As will be analyzed in Chapter 3, there are some strategies to preserve privacy in NN, some only during inference, some only during training, and others on both. Still, it is an issue of active research, and researching it is the focus of this thesis.

2.2 Homomorphic Encryption

This section will introduce HE. It will start by presenting the essential basics of cryptography. After laying this groundwork, the focus will shift to HE. Together with the explanation of HE, the various types and respective schemes of HE will be explored and analyzed. To wrap up, the section will survey known HE libraries, underlining the specific schemes each one implements.

2.2.1 Cryptography Basics

Cryptography is a fundamental part of everyday information security that allows individuals and organizations to protect their sensitive data and communications from unauthorized access and tampering. Encryption can be traced back to ancient civilizations, where it was used to protect military and political communications [30]. Nonetheless, it has become increasingly vital in today's digital age as the amount of data being transmitted and stored electronically continues to grow. The usage of cryptography has four main goals: Confidentiality, Data Integrity, Authentication, and Non-repudiation [3]. Cryptography tools vary in type, from unkeyed primitives like Random Sequences and One-way Permutations, Symmetric-Key primitives like symmetric-key ciphers and Hash Functions, and Public-key Primitives like public-key ciphers and signatures [3].

To explain cryptography, it is sometimes easier to do it using the very typical Alice and Bob scenario. In this scenario, Alice and Bob want to communicate, but they want their communications to be protected against eavesdropping from an external person, often called Eve. If Alice and Bob already have a shared private key, their communication can be done using that same key so that if Eve captures the ciphertext, it cannot decipher it. The type of cipher that allows this type of communication is called Symmetric Encryption. On the other hand, if Alice and Bob do not have already shared a private key, the presented solution is not possible. To solve this problem, Asymmetric encryption must be used. This type of encryption uses a separate key for encryption (public key) and decryption (private key). Alice and Bob generate a pair each and send the public one to the other while keeping the private key for themselves. In this scenario, when Alice wants to send a message to Bob, she encrypts the plaintext using Bob's public key, and Bob decrypts it using its private key, and the other way around, if Bob wants to send a message to Alice. Eve, even if in possession of the public keys, still cannot see the original plaintext due to the fact that it does not have the decryption key. Another important aspect is that if Eve changes the ciphertext, the recipient would be able to know due to the fact that the key would no longer work.

Due to the low efficiency and security of Asymmetric Encryption [28], a third scenario is also very common in the real world. This scenario uses Asymmetric Encryption to allow Alice and Bob to securely share a private key and then use Symmetric Encryption for the remainder of the communications[1]. This way, the flexibility provided by the former and the robustness and efficiency of the latter can be used together.

According to Kerckoffs' principle [31], the security of an encryption scheme must not rely on the secrecy of the encryption algorithm but exclusively on the secrecy of the secret key. Given that the privacy of the algorithm is not important and to facilitate communication between different entities, standard encryption schemes were defined. The Advanced Encryption Standard [32] is one of the go-to standards to use in symmetric encryption use cases. It provides robust security while being fast and efficient. For asymmetric encryption, the RSA algorithm [33] is the go-to choice. However, RSA alternatives are being researched for a post-quantum world where it is believed that RSA will not maintain its security grantees given the capability of quantum computers doing extremely heavy computation in fractions of the time (when compared to classical computers). Algorithms like the ones based on the learning with error problem are proving to be viable alternatives.

Besides the based encryption and decryption function, there are more advanced cryptographic systems that go beyond these basic capabilities with the objective of allowing their use in different environments [34]. Some were already referred to in the previous section (Section 2.1.1), but there is a wide variety of them. For example, Format-Preserving Encryption maintains the format of the original plaintext to a ciphertext with the same format, Searchable Encryption allows the searching of terms in databases without decrypting its data, Tweakable Encryption allows encryption results to be "tweaked" to create different versions of the same cipher. In this thesis, HE is the most relevant one and will be explained in more detail.

2.2.2 Homomorphic Encryption Schemes

Explaining HE can be done using the same Alice and Bob scenario. To do this, it is necessary to change the objective to Alice wanting Bob to perform a computation for her; in this case, let that computation be $2 + 3$. Besides that, the following condition is added: Bob cannot be able to eavesdrop on Alice's data, which means he can not know the numbers 2 and 3 nor on the result 5. So, in this updated scenario, Alice encrypts 2 and 3 using HE and sends them to Bob, which computes the result using the HE operation equivalent to plaintext addition. Bob then sends the result, which is a ciphertext containing the plaintext value 5, to Alice, who will, in turn, decipher it.

Although only recently gaining momentum, HE was first conceptualized by Rivest, Adleman, and Dertouzos [35] shortly after the development of the RSA cryptosystem. Unlike a non-homomorphic encrypted scenario, data would travel to the third-party server encrypted, remain encrypted during all the operations, and return encrypted to the client. This way, the server would never know the contents of the data while still being able to process it (Figure 2.1).

From a mathematical standpoint, the encryption function E can be defined as Homomorphic if, given a set of plaintexts M and an operation $\odot$, the equation in (2.1) is satisfied. The

[1]Due to privacy reasons, secure applications of this hybrid scenario require a timeout/threshold after which a new private key must be generated and shared

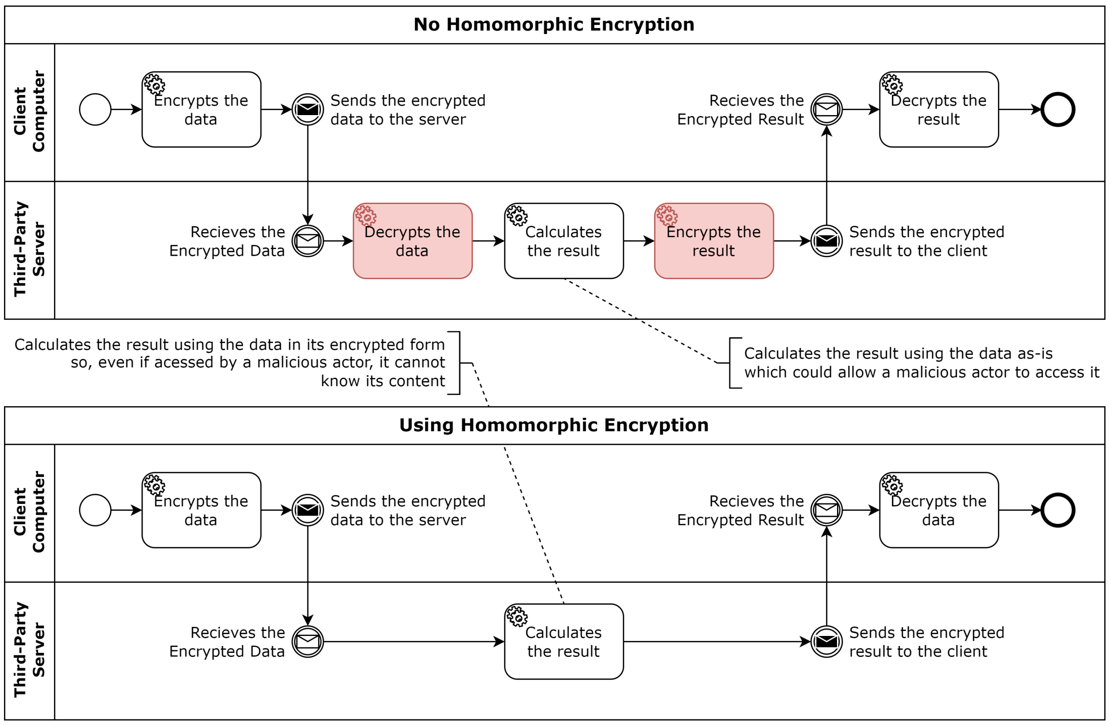
No Homomorphic Encryption
Client Computer
Third-Party Server
Encrypts the data
Sends the encrypted data to the server
Recieves the Encrypted Result
Decrypts the result
Recieves the Encrypted Data
Decrypts the data
Calculates the result
Encrypts the result
Sends the encrypted result to the client
Calculates the result using the data in its encrypted form so, even if acessed by a malicious actor, it cannot know its content
Calculates the result using the data as-is which could allow a malicious actor to access it
Using Homomorphic Encryption
Client Computer
Third-Party Server
Encrypts the data
Sends the encrypted data to the server
Recieves the Encrypted Result
Decrypts the data
Recieves the Encrypted Data
Calculates the result
Sends the encrypted result to the client

Figure 2.1: Non-Homomorphic and Homomorphic Encryption data flow

types of operations $\odot$ supported and the number of times they can be performed without decrypting the cyphertext are used to classify the HE schemes.

$$\forall m_1, m_2 \in M, \quad E(m_1 \odot m_2) = E(m_1) \odot E(m_2) \tag{2.1}$$

The first HE schemes were very limited and only allowed a single type of operation to be performed in ciphertext. These schemes of HE become known as Partial Homomorphic Encryption schemes. Some, like the one by Paillier [36], only allowed additions, while the one suggested by Elgamal [37] only allowed multiplications. After that, a new type of HE called Somewhat Homomorphic Encryption was created, and it extended the previous one by allowing an additional operation type. However, this added operation was limited and could only be completed a limited number of times. For example, the scheme proposed by Boneh, Goh, and Nissim [38] allowed an arbitrary number of additions but only one multiplication.

Defying the assumed boundaries of HE, Gentry [39] presented the first HE scheme that allowed unlimited operations on multiple operation types at once, in this specific case, addition and multiplication. This type of scheme is called Fully Homomorphic Encryption (FHE), and it presents a breakthrough in HE capabilities by opening doors for a wide range of new applications.

According to Minelli [40], there were roughly three generations. The algorithms from the first generation are the ones derived directly from the work of Gentry [39], which is based on ideal lattices in a polynomial ring. The second generation refers to all works proposed in the sequence of the work presented by Brakerski and Vaikuntanathan [41, 42], which is based on assumptions of worst-case hardness. Finally, the third generation includes all works based on the GSW FHE scheme proposed by Gentry, Sahai, and Waters [43] based on the approximate eigenvector problem.

These advances in HE research resulted in several HE schemes that can be used in the real world. The Modular Arithmetic Approach, especially the BGV and BFV schemes, set the stage with its usage of modular arithmetic over finite fields. This allows fast evaluation of ciphertext without requiring the usage of the bootstrapping operation. It supports fast scalar multiplication and Single Instruction, Multiple Data (SIMD) operations. Its real-world applicability was proven by its usage in CryptoNets [44].

Next, the Boolean Circuit Approach resulted in mainly two very efficient HE schemes. These schemes, the Fully Homomorphic Encryption with Bootstrapping (FHEW) and the Fully Homomorphic Encryption Library over the Torus (TFHE) achieved incredible operation speed by using the GSW technique. Contrary to previous schemes, these ones operate on the bit level and only provide boolean operations. Although these bit-by-bit operations can seem slow when compared to computing the whole number, their highly efficient sub-second bootstrap operations make them competitive. One of the major drawbacks of these schemes is the fact that they do not support SIMD, which makes them fall behind other schemes in parallelized computations [45].

The Approximate Number Arithmetic Approach, known mainly due to the CKKS scheme [46], offers a solution where the number representation inside the ciphertext is floating-point. By doing so, it allows efficient computations on vectors of real numbers using techniques such as batching and fast polynomial approximation, making it highly efficient for certain applications like logistic regression learning. Although it does not guarantee exact values when decrypting, it provides approximate ones that are sufficient in most applications. If its SIMD capabilities are fully utilized, it can be the best performing of the schemes mentioned [45].

The above-mentioned approaches' main advantages and disadvantages are summarized in Table 2.1.

Approach	Advantages	Disadvantages
BGV/BFV	Efficient for integer arithmetic Bootstrapping not required Exact results after decryption Supports SIMD	No direct support for real numbers Expensive Bootstrapping
CKKS	Efficient real number arithmetic Efficient vectors computations Fast polynomial approximation Supports SIMD	Approximate results after decryption Expensive bootstrapping Less efficient for integer-based tasks
TFHE	Efficient bit-by-bit operations Fast bootstrapping Exact results after decryption	Multiple ciphertexts per number No support for SIMD

Table 2.1: Fully Homomorphic Encryption Aproaches

2.2.3 Software Libraries

Implementing HE schemes can be a complex task. Due to that, it is important to know which libraries already exist to avoid expending resources by finding one where a library already provides a more efficient and more robust implementation. Even then, it is important to understand the scope and support of each library before using them because, while some are mere concepts, others are fully functional and maintained libraries.

The oldest library that will be examined is the HElib [47] by IBM. It started by only supporting the BGV scheme, with the support for CKKS being added later. It uses the Smart-Vercauteren ciphertext packing technique [48] and the Gentry-Halevi-Smart optimizations [49] to achieve fast and efficient HE operations. It is written mainly in C++ and uses the NTL Mathematical Library[2].

The Microsoft SEAL [50] library was introduced by Microsoft at the end of 2018. It currently supports the BGV, BFV, and CKKS schemes, but it previously supported the YASHE scheme [51]. Their authors acknowledge the steep learning curve to learn it, but taking into account the complexity of HE, it aims to be a user-friendly library. It is written in C++ and can easily be used with .NET using wrappers.

The HEAAN library [52] was the release library of the CKKS scheme. While the first versions did not support bootstrapping, the most recent ones do. This library is currently deprecated, with the last code change being registered in January 2022. Similar to HElib, it is written in C++ and uses the NTL Mathematical Library.

The FHEW (Fastest Homomorphic Encryption in the West) [53] and TFHE (Fully Homomorphic Encryption over Torus) [54] libraries were also released as implementation of the schemes their authors proposed in their publications. Both operate at bit-level ciphertexts, and the first introduced the capacity of sub-second bootstrap operations. The second worked upon the efficiency gains of the first and achieved a remarkable 14-millisecond single-core latency for binary gate evaluation. Despite their exceptional achievements in performance, both are already deprecated. TFHE-rs [55] (previously called concrete) by Zama further optimized the TFHE scheme. Their library is written in Rust and is a replacement for the discontinued TFHE library while offering an 8-millisecond gate evaluation time. Besides the bit encryption support, Zama also added several Integer capabilities to this library.

Finally, the OpenFHE [56] (previously called Paradise) is a special case because it results from the combination of various HE libraries in one. This fact, combined with the fact that they recruited previous contributors from other libraries, makes it one of the most complete HE libraries. It supports all major schemes and several bootstrap designs. Derived from the deprecated Paradise library, it has since incorporated several features from other libraries. It is written in C++ and is supported by the Defense Advanced Research Projects Agency of the United States Government.

The functionalities of each library are summarized in Table 2.2.

2.3 Neural Networks

AI is a term that refers to a broad range of techniques and algorithms that try to replicate human intelligence, while Machine Learning (ML) is the part of AI that is capable of learning from the data it is provided with. There are a vast amount of traditional ML algorithms, like Linear Regression, Naive Bayes, Support Vector Machines, and others, but the introduction of NNs was one of the breakthroughs in the ML space [13].

[2]https://libntl.org/

Library	Owner	Language	BGV	BFV	CKKS	FHEW	TFHE
SEAL	Microsoft	C++	x	x	x		
HElib	IBM	C++	x		x		
HEAAN	CryptoLab	C++			x		
FHEW	-	C++				x	
TFHE	-	C++					x
TFHE-rs	Zama	Rust					x
OpenFHE	-	C++	x	x	x	x	x

Table 2.2: Fully Homomorphic Encryption Library features

As already referred to, NNs were created by trying to simulate the way the human brain works [29]. They are, in essence, directed graphs where the nodes are often referred to as "neurons" and their task is to compute the output value with the given inputs. They do this by making a weighted sum of their inputs to condense them into a single dimension that is then fed into an Activation Function (AF), which will define the final output.

From an outside perspective, NN is composed of layers where the one in which the input values are injected is called the input layer, the one where the final output is received is called the output layer, and the remaining layers are referred to as hidden layers. The training and inference of these NN is made by injecting data in different places and using different data flow directions. These two processes will be explained in the following subsections: Section 2.3.1 and 2.3.2, respectively.

With the correct combination of AF, neuron count, layer count, and other hyper-parameters, NN can have the ability to learn complex problems, which means that they are useful in a diverse domain range, all the way from image analysis to full-on chat-bots. There are more specific types of NN that excel in specific areas, such as the exceptional performance of Convolutional Neural Networks in image analysis. Nevertheless, in this thesis, only the most basic form of NNs will be used.

There are two modes of operation for NN: inference and training. The first mode is performed by inserting the inputs on the first layer and obtaining the output on the last (forward propagation). It is used to classify the data using the mappings learned by the NN in the second mode of operation. The training mode is where the NN is fed with data whose expected output is already known. Through the propagation of error in the reverse direction of inference (backpropagation), the NN is able to converge into that data slowly.

Next, the forward propagation and backpropagation steps will be explained in detail.

2.3.1 Forward propagation

As just mentioned, to use NN, it is necessary to insert data into it. The data is inserted in the input layer, and then, layer after layer, that data will be processed, and once the final layer is reached, the values will correspond to the NN prediction. Each layer executes all of its neurons in parallel using the input of the previous layers and sends the outputs of all neurons to the next one. Each neuron performs a pre-defined operation, which is the result of the AF in the point that corresponds to the weighted sum of the neuron inputs (given *n* inputs, the weighted sum is given in (2.2)). Usually, these neurons use all of the inputs of the layer and decide on the importance of each depending on the weight to which they have been attributed. As will be explained later (Section 2.3.2), these weights will be updated during training to make the output of the NN as close as possible to the expected output. To make it possible for neurons to have biases, there is an additional "imaginary" input in each neuron that has a constant input value of 1.0 and a non-constant weight (often called b) that can be updated during training.

$$z = \sum_{i=0}^{n} w_i * x_i + (1.0) * b \tag{2.2}$$

The fact that the neurons are executed in parallel qualifies them to be calculated using matrices. This way of calculating allows specific matrix optimization to be employed, namely, using SIMD. Considering the two-dimensional weight matrix W (with each row representing the heights of one neuron), the input array I, the bias array B, and the result array Z, the weighted sum can be calculated using (2.3). Nevertheless, even though the weighted sum can be calculated using these matrix operations, the final output of each neuron still needs to be calculated using the AF. If it cannot be represented using matrix operations, each output must be inserted into the AF individually (the matrix calculation of the weighted sum can still be used).

$$Z = W * I + B \tag{2.3}$$

From a less technical standpoint, the forward propagation process gives inputs to the neurons so that they can determine if the relation they learned to identify exists or not. When this is applied to all of the layers, the NN basically determines the existence of a broader relation using the consecutive smaller relation conclusions.

Next, a brief explanation of AFs will be presented, as well as some commonly used AFs.

Activation Functions

As mentioned, AFs are used by neurons to determine their output given a weighted input, which means that they are the main factor that determines if the neuron fires or not, or, in other words, if the relation that neurons were trained to identify exists or not. The main objective of AFs is to non-linearly transform the weighted sum of the inputs into an output, which is crucial to allow them to understand complex data patterns. Besides that, some AFs can have a different, less known, but also important, objective, which is normalizing the output of the neuron to ensure the NN values maintain a manageable magnitude.

There is a large variety of AFs, each with its advantages and disadvantages, with the ones mentioned in this thesis represented in Table 2.3. The identity activation function, which maps the input to the output, can be seen as the simplest one, but a more complex and non-linear function provides more advantages. A very popular AF is the sigmoid, which uses the logistic regression to map the input into a probability value between 0 and 1, with the values closest to 0 approximating 1/2 and $-\infty$ and ∞ values approximating, respectively, to 0 and 1. The softmax is often viewed as an alternative to the sigmoid in multi-class classification, as it ensures that the sum of all outputs sums to 1 so that they can be viewed as probabilities.

Another common AF is the Rectified Linear Unit (ReLU) and its variants, which are the identity function (or with similar behavior) for values higher than 0 and 0 (or next to zero) for negative input values. These functions are computationally cheaper, but they do not place an upper limit on their outputs, which means that the NN is not upper-bounded. This non-normalization can cause some problems when using the NN. Still, due to the fact that non-private NN works with 64-bit floating point numbers, this problem is rare, which means its computational efficiency often offsets this problem.

The choice of an AF does have implications for the inference phase; however, the primary considerations for selecting an AF are most relevant during the training phase. For this reason, a more detailed discussion of the advantages and disadvantages of different activation functions will be presented when referring to the usage of AF during training.

AF	**Equation**	**Derivative**	**Output Range**
Identity	$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
ReLU	$f(x) = \begin{cases} 0 & ifx < 0 \\ x & ifx >= 0 \end{cases}$	$f'(x) = \begin{cases} 0 & ifx < 0 \\ 1 & ifx > 0 \\ undefined & ifx = 0 \end{cases}$	$(0, \infty)$

Table 2.3: Examples of activation functions, their derivatives, and output ranges

2.3.2 Backpropagation

Training an NN involves two significant steps: the first one is the forward propagation step, which is also executed when classifying data using the NN, and the second is the backpropagation step. While the first is responsible for the classification of the input, the latter is responsible for propagating the classification error backward through the NN to update it in a way that minimizes that error. In summary, the primary purpose of the backpropagation step is to minimize the error of the NN for the latest forward propagation step, which means that, after updating the NN, the repetition of the same forward propagation step should return an output closer to the expected one.

To modify the NN into a state closer to the desired one, a gradient is used. This gradient is made up using the loss of the sample in question together with the derivatives of the different operations used to reach the NN output. The loss is calculated using a Loss Function (LF) into which the predicted and expected output are input. A NN is considered trained when a global minimum is reached. How the gradient is calculated and navigated will be further explained further ahead

NNs are initialized at a random state, and training them brings them closer to the chosen domain. However, reaching global minims is not guaranteed and, even if possible, may not be justifiable due to the computational cost. Furthermore, the data sample used for training may not be sufficient for optimal learning, requiring multiple passes over the dataset. The number of epochs quantifies this, and it is a hyperparameter that affects the NN's performance and training times. Too few epochs can not be enough to train the NN, while too much can be computationally expensive and can hurt the NN's ability to generalize [57].

Optimizers

The loss function, if viewed as a function of the input (gradient), can be viewed as mountains and lowlands where the objective of the training process is to descend those mountains into the lowest lowland possible [13]. One can visualize this as a ball rolling downhill on a loss landscape. The problem is that it is not possible to see the whole gradient at once, but only the tiny section right next to it, so there needs to be a strategy to try to find the global minimum while not stopping at a local minimum. The concept of NN Optimizer emerges to solve this problem.

Mathematically, given a loss (error) E, and AF σ (derivative $\frac{\partial \sigma(z_j)}{\partial z_j}$), the output value of the neuron j as o_j, and the weighted sum of the neuron j as z_j, the calculation of the error derivative of the input i of the neuron j can be calculated using (2.4).

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial z_j} o_i = \frac{\partial E}{\partial o_j}\frac{\partial \sigma(z_j)}{\partial z_j} o_i = \delta_j o_i \tag{2.4}$$

The derivative of the error is not the same in all layers because the error is calculated using the LF in the last layer and using the weighted error derivative of the previous layer (which corresponds to the next layer because the errors are propagated backward). This means that the derivative of the error for the output layer is given by (2.5) (LS represented as L and expected output of the neuron j represented as exp_j), and by (2.6) (the weight of the neuron j of the previous layer neuron l represented by $w_{j,l}$, error delta of the previous layer neuron l represented by δ_l, and n representing the number of neurons in the previous layer) for the remaining layers.

$$\delta_j = \frac{\partial L(o_j, exp_j)}{\partial o_j}\frac{\partial \sigma(z_j)}{\partial z_j} o_i \tag{2.5}$$

$$\delta_j = \sum_{l=0}^{n}(w_{jl}\delta_l)\frac{\partial \sigma(z_j)}{\partial z_j} o_i \tag{2.6}$$

Using a value of learning rate η, it is possible to calculate the updates of the weight and bias using the formulas in, respectively, (2.7) and (2.8). The formula to calculate the update of the bias is more straightforward than the one of the weights because the "imaginary" input value is always one. To reduce the number of operations required, it is possible to rewrite the formula for the weight update based on the update of the bias and reuse the calculations, as shown in (2.9).

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta o_i \delta_j = -\eta \frac{\partial E}{\partial o_j}\frac{\partial \sigma(z_j)}{\partial z_j} o_i \tag{2.7}$$

$$\Delta b_j = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta o_i \delta_j = -\eta \frac{\partial E}{\partial o_j}\frac{\partial \sigma(z_j)}{\partial z_j}(1) = -\eta \frac{\partial E}{\partial o_j}\frac{\partial \sigma(z_j)}{\partial z_j} \tag{2.8}$$

$$\Delta w_{ij} = \Delta b_j o_i \tag{2.9}$$

In its most foundational form, gradient descent calculates this gradient using the entire dataset. However, for massive datasets, this method can become computationally burdensome. On the other end of the spectrum, Stochastic Gradient Descent leverages just a single data point at each iteration to estimate the gradient. While this latest one adds speed and randomness that can assist in evading local minimums, it also can lead to less stable convergence. Mini-batch gradient Descent is often chosen as a middle ground as it benefits from computational efficiencies due to the fact that it uses a subset of the data (called batch) to compute the gradient.

As the complexities of models and datasets grew, more sophisticated optimization techniques appeared. The concept of Momentum emerged from the analogy of a ball rolling down the hill, gaining speed as it goes. Using this strategy, it is possible to remove oscillations in the convergence and speed up the descent. This principle was further developed in methods like Adagrad, which adjusts the learning rate individually for each parameter. RMSprop further refined this by modifying the learning rate based on a moving average of the squared gradients, addressing some of the challenges observed with Adagrad. Finally, the Adam optimizer combined the principles of Momentum and RMSprop, keeping moving averages of both the gradient and its square.

Choosing the right optimizer can significantly influence the efficiency and effectiveness of the training process. For instance, while adaptive methods like Adam might be the go-to for deep learning tasks, there can be some cases where more straightforward methods like SGD can result in models that generalize better to unseen data.

Loss Calculation

The error between predicted and obtained values is referred to as a loss, with higher values representing a higher disparity between predicted and expected. The calculation of this loss is done using a LF with a common one being the Mean-Squared Error (MSE) LF for regression tasks and the Cross-Entropy LF for multi-class classification tasks. As seen in the previous section, the LF is only used in the output layer, but its outputs influence all the layers. As such, it is essential to correctly choose the LF because, if not chosen correctly, it can make the NN unstable, as it can make the backpropagation step minimize the loss in the opposite direction or scale of the expected result.

Trainable Activation Functions

To reduce the loss of an NN during the backpropagation phase, a relation must exist between the output of each neuron and its output. This calculation is direct on the weighted sum of the inputs, but it is not direct on the AF. As seen in (2.4), in the backpropagation, it is necessary to calculate the derivative of the AF. This means that it is a requirement that all AFs that are used during the training phase must have a derivative.

The derivatives of the AFs are all unique, so they will also directly impact the training performance of the NN. One typical example of this is the vanishing gradient problem of the sigmoid AF, as, for high absolute values of the input, its derivative tends to 0, which means that the changes of the backpropagation will disappear, so the NN will not be able to train. ReLU, on the other hand, does not have this problem because, for positive values of x, its derivative is always 1, but for negative input values, it is 0, which means that, once in that state, that neuron will also no longer change for that data. Regardless, there are already some variations of the ReLU that resolve/minimize this problem by introducing variation on the negative part of the AF.

Chapter 3

State-of-the-Art

This chapter reviews the SotA of the three domains that the work of this thesis is related to. It starts by going through the history of NIDS to then go in-depth on current solutions, datasets, and data privacy on NIDS. Due to the fact that the presented PNN models will be DDoS attack detection models, the SotA of this specific NIDS sub-domain will also be reviewed. After that, it switches over to data privacy preservation in NN, where it reviews the different techniques used to protect the privacy of data during its use in NNs. This chapter ends by focusing specifically on the SotA of the usage of HE for data privacy preservation in NNs.

3.1 Network Intrusion Detection

The increasing amount of information stored digitally created the need for systems capable of protecting data by detecting intrusion attempts. Lunt and Jagannathan [58], in 1988, introduced one of the first potential solutions to address this problem. Their work proposes a prototype capable of recognizing attacks by identifying deviations from the user's normal behavior. Shortly after, instead of using user behavior as a baseline, Denning [59] presented an expert system that detected intrusions based on the rules defined by an expert. In 2000, Axelsson [60] reviewed the SotA of Intrusion Detection Systems (IDSs), which they found to be mainly composed of very manual and expert-driven systems. Although attempts to automate this were developed, the majority still proved to be significantly manual or based on limited semi-automatic capabilities of knowledge acquisition. This report also identified two types of IDSs classifications: one by its location, either host-based intrusion detection systems when installed in the target computer or Network-based Intrusion Detection Systems when installed in dedicated hardware to analyze network packages; or by its detection action strategy, either by emitting alerts (passive) or by taking actions to stop the intrusion (active).

More recently, in 2019, Khraisat et al. [61] presented a completely different reality compared to the previous ones. Since those first approaches, the usage of computers grew exponentially, which increased the interest and investment in such systems. Besides that, starting in the early 2010s, there was a boom in the usage of AI that made new solutions with significantly better performance being released. This increase in traffic and computational performance opened a new door for innovations in the IDS space. This work of Khraisat et al. [61] identified two main types of IDSs: the Signature-Based Intrusion Detection System and the Anomaly-Based Intrusion Detection System. The Signature-Based type uses information gathered from previous attacks to create a fingerprint of each attack to compare it to the traffic it is analyzing. Although this strategy provides reasonable accuracy for known attacks, it cannot detect new, not yet documented attacks (0-day attacks) or known attacks whose signature has been altered. On the other hand, the second type (Anomaly-Based) learns what is usual behavior and considers significant deviations from this behavior as an attack. This strategy gives it the capability to detect non-documented attacks. Yet, this latter approach comes at the cost of high false-positive identifications while being challenging to adapt to highly dynamic systems.

In 2022, Yang et al. [62] published a systematic literature review that presented some improvements in relation to the previously mentioned one. The main difference is a significant increase in the proportion of the usage of Deep Learning (DL) based IDS, such as Deep Neural Networks, Recurrent Neural Networks, and Convolutional Neural Networks. Although their usage is increasing, their superiority when compared to more traditional ML methods still needs to be proven. Finally, in the same year, Abdallah, Eleisah, and Otoom [63] surveyed the SotA techniques. The most recurring top performer was the random forest method, with DL only beating it in some specific cases. The application of AI in IDSs is analyzed more in-depth in the next section.

3.1.1 Network Intrusion Detection and Artificial Intelligence

As referred above, pioneer IDSs were manual and expert-driven, which means that the ability to detect intrusions relied mainly on rules manually defined by experts in that area. As the complexity of attacks and their variations increased, there was a need for IDSs to be able to learn what is an intrusion or not without the help of an expert. As AI methods can infer information from data automatically, their application in IDSs removed the need for manual rule definition that used expensive man-hours. This resulted in various solutions that go beyond the performance of the manual rule definition approaches.

Regarding ML approaches, the most used methods are the Support Vector Machines and the Decision Trees. In Anthi et al. [64], authors report a 90% F1-Score for the identification of an attack while maintaining sub-seconds response time using a J48 decision tree model and a custom dataset. In Teng et al. [65], authors report an accuracy of 89.02% on the KDD CUP1999 dataset combining Support Vector Machines and a decision tree where each level of the latter uses one or more Support Vector Machine model results to choose the next path (3 levels, total of 4 Support Vector Machines in configuration 1-1-2). Although these are the most used methods, other approaches also have their advantages. For example, Casas, Mazel, and Owezarski [66] used an unsupervised k-means algorithm in conjunction with the Evidence Accumulation Clustering technique to detect outliers in the traffic analyses, which allows it to detect 0-day attacks while maintaining a reported accuracy of 90%. Koc, Mazzuchi, and Sarkani [67] used a Hidden Naïve Bayes method that reported an accuracy of 93.72% in the KDD'99 dataset for overall classifier performance and 99.60% for denial of service Classifier performance. Singh and Vigila [68] used fuzzy logic together with the Principal Component Analysis that resulted in a 99.08% accuracy using the KDD99 dataset with a train-test split of approximately 10%-90%.

When analyzing DL approaches, it is possible to find the use of Deep Neural Networks. For example, in a publication by Vinayakumar et al. [69] that describes a multi-layer neural network that, in its three-layer configuration, reported an accuracy of 93.5% in the CICIDS 2017 dataset (although not the best accuracy, it does it with less number of parameters, that is, with less computational resources). Other DL approaches are also used, like in Althubiti et al. [70] where authors describe a solution that reports an accuracy of 99.44% in the binary classification of intrusions using a Long Short Term Memory Recurring Neural Network or in Ashiku and Dagli [71] and Dong, Wang, and He [72] that present solutions using

Convolutional Neural Networks with a reported accuracy of, respectively, 95.6% on user-defined UNSW-NB15 dataset and 94.32% on the KDD-99 dataset. Altunay and Albayrak [73] proposed a hybrid solution based on CNN and LSTM where the CNN is used to extract features of the raw network traffic data, and then these features are inputted into the LSTM to identify patterns in these features, which results in an accuracy of 93.21% and 99.84% for binary classification in UNSW-NB15 and X-IIoTID datasets, respectively.

For the more traditional methods, more recent papers proposed alternatives to fix some of the issues limiting those solutions. Kannari, Chowdary, and Biradar [74] used the Random Forest classifier but added the use of the technique called "Recursive Feature Elimination", which reduces the number of features by identifying and picking the most relevant ones (a total of 21 features selected). This resulted in a detection accuracy of 99.83% for binary attack detection on the NSL-KDD dataset. With a similar approach, Waskle, Parashar, and Singh [75] used the Principal Component Analysis technique to reduce the dimension of the data, which resulted in an accuracy of 96.78% when detecting the attack type in the same dataset. Zhang, Song, and Wang [76] proposed an optimized Support Vector Machines algorithm that uses the particle swarm optimizer that increased accuracy and reduction of training and inference time when compared to Support Vector Machines in the tests performed by authors.

For deep learning-based methods, some of the more recent papers proposed combinations of strategies to increase overall performance. [77] proposed an optimized feature extraction technique using auto-encoders, that is, algorithms that use the original data and encode it according to rules they infer during training. This approach resulted in a precision of 86.02% and 95.38% in the Deep Auto Encoder + Deep Neural Network configuration in the datasets NSL-KDD and CSE-CIC-IDS2018, respectively. Even then, some authors do apply optimizations to the neural networks like in Sheikhi and Kostakos [78], which used a genetic algorithm to select the best features and then feed them into a NN whose initial weights were optimized employing Particle Swarm Optimization and Grew Wolf Optimization. This resulted in an accuracy of 94.2% in the binary classification of the NSL-KDD dataset.

Datasets

The introduction of AI and its ability to extract valuable information from data has made the latter one of the most valuable assets of the 21st century, sometimes even being referred to as the "New Gold" in analogy to the importance of the metal gold in society. As such, soon after IDSs started using AI, data capable of training such systems began to appear, collected either through simulated or real environments. This subsection summarizes the data used to train these models (dataset) in the reviewed literature to understand their popularity. The papers that mention each dataset are identified in Table 3.1.

All the datasets mentioned above can be used for the binary classification of attacks, but they vary in the attacks they can identify. The NSL-KDD and related datasets are able to identify twenty-two different attacks that belong to four different attack classes plus normal traffic. The CSIC 2010 dataset was collected in an e-commerce application running in Spain, and it has more than sixty thousand entries, from which more than 25,000 are from attacks. Such attacks include SQL injection, buffer overflow, information gathering, file disclosure, CRLF injection, XSS, and parameter tampering. The UNSW-NB15 dataset contains raw network packets captured using the *tcpdump* tool, and it identifies nine types of attacks: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms. WSN-DS is a dataset for intrusion detection in wireless networks, and it includes

Table 3.1: Datasets used in the reviewed literature

Dataset	Release	Articles
KDDCUP 99	1999	[65–69, 72, 79]
Kyoto2006+	2006	[69]
NSL-KDD	2009	[64, 69, 74, 75, 77–80]
CSIC 2010	2010	[70]
UNSW-NB15	2015	[69, 71, 73, 79]
WSN-DS	2016	[69]
CIC-IDS 2017	2017	[69]
CSE-CIC-IDS 2018	2018	[77]
Bot-IoT	2018	[81]
X-IIoTID	2021	[73]

Denial of Service attacks in wireless sensor networks: Blackhole, Grayhole, Flooding, and Scheduling attacks.

CIC-IDS2017 [82] and CSE-CIC-IDS 2018 [83] datasets try to solve the problem of outdated and unreliable datasets for Intrusion Detection. They contain various attacks as well as normal behavior. The latter was created using a system to profile the abstract behavior of 25 users on the HTTP, HTTPS, FTP, SSH, and email protocols. Although their objective is similar, the 2018 dataset has more records and adds some attack types. Finally, the Bot-IoT [84] and the X-IIoTID [85] datasets are two datasets focused on intrusion detection on Internet of Things networks. They were both generated in a laboratory setting, but the latter is specially focused on the Industrial Internet of Things segment. They contain various attack types like Distributed Denial of Service or malware infections.

According to Table 3.1, the KDDCUP 99 [86] dataset is the second most refereed dataset in the literature, but its mentions are, on average, before the year 2019. This happens because it was one of the first available datasets to train IDSs and, at the time, one of the best. As the years went on, flaws in its data were becoming increasingly more visible, which made authors search for alternatives. Due to its popularity, some authors considered it relevant to include it even when using other datasets to allow comparisons to previous works.

The most used dataset is the NSL-KDD [87]. It is based on the KDDCUP 99 dataset. It fixed some problems of its predecessor, mainly the presence of redundant records and its unreasonable size [87], to provide a more balanced dataset and facilitate the comparison of results between scientific publications. It is important to mention that, despite fixing the mentioned problems, it still suffers from some of the issues of the original dataset. Even though it was published in 2009, recent works still use it to present their results due to its ease of use, its popularity, and its pre-balanced status.

After excluding the last two datasets (Bot-IoT and the X-IIoTID) because of their focus on the Internet of Things, there are other less-used alternatives to the NSL-KDD ones. Although the number of papers that mention them is not enough to be considered a viable alternative to present comparable results to the other work, they can still be used for comparisons with specific papers or to validate certain characteristics of the proposed solution.

3.1.2 Network Intrusion Detection and Data Privacy

As will be mentioned in Section 3.2, there is more than one solution for privacy preservation in NNs. As such, there are different approaches to address privacy concerns in IDS.

The solution proposed by Sarhan et al. [81] uses Federated Learning to allow the sharing of important information for models to be trained with better accuracy without centralization. Using this approach, in terms of accuracy, federated learning was always better performing than localized learning (only using data from that organization). Still, it lagged behind centralized learning, but considering the advantages of the non-centralized structure of federated learning, this could be an appealing alternative for most organizations, especially for those currently using localized knowledge. However, this above-mentioned work lacks a solution for the privacy issue as a whole, solving only the data governance and ownership issues. This partial privacy guarantee is a common trend, as all papers examined in the work of Fedorchenko, Novikova, and Shulepov [14] use Federated Learning to solve only some specified issues. Moreover, only a few offer solutions to all the privacy issues. Even then, the number of papers found in the query of this paper was 18 (of which only 12 were considered relevant), which shows how little literature there is tackling this problem.

Another approach was proposed by Tabassum et al. [79] that used generators to learn using the local traffic data, and to then send this model gradients to the central model for processing. The use of Federated Learning in this situation allowed the generators, in this case, Generative Adversarial Networks, to be able to learn the data while not sending it to the central server. Various tests were done on this solution, and their accuracy was 99.1%, 99.29%, and 99.4% for the datasets KDD99, NSL-KDD, and UNSW-NB15, respectively.

Other works also used federated learning to try to solve this issue, like Anastasakis et al. [80] that proposed a solution capable of identifying Denial-of-Service attacks using a Fully Connected Neural Network while maintaining the privacy of the data at a cost of approximately 7% accuracy when compared to others works that do not guarantee privacy. Yang et al. [88] proposed an unsupervised machine learning alternative using an ensemble of Auto-encoders that, they claim, is 40% lower in complexity and 19% higher in accuracy. Still, their non-compromise (1% accuracy reduction) privacy guarantee is not sufficient given the high value of the privacy parameter (15).

As identified by Mokry et al. [89], the increase of the privacy parameter (which is correlated to the amount of noise added) does impact the model's ability to learn, which shows that adding noise as a guarantee for data privacy affects the model performance. As noted by the same authors, "privacy protection does incur extra execution time and communication cost" [89] and that extra time needed to use such data can be a limiting factor for more complex models with large amounts of parameters.

Spathoulas, Theodoridis, and Damiris [90] tried to solve a more specific task inside the IDS domain. This task was to create a way to privately share intrusion detection alerts because of their capability to allow organizations to efficiently protect their networks from attacks that began in other networks. To achieve this, they used a clustering technique and homomorphic encryption. Their use of encryption guarantees data privacy, but their honest but curious participant approach makes their solution vulnerable to malicious clients and does not guarantee data integrity.

DDoS Attack Detection

DDoSs attacks occupy a critical niche within the vast field of NIDS. As one of the most powerful threats to modern networking systems, DDoS attack detection is essential. For this reason, there is a substantial research community focused on creating models capable of detecting this specific type of attack.

To look at the current state of DDoS detection, it is essential to compile the most recent works. By analyzing the comprehensive systematic review on the application of ML techniques in detecting DDoS attacks within Software-Defined Networking made by Ali, Chong, and Manickam [12], and the DL focused study of Mittal, Kumar, and Behal [91], it is possible to take several conclusions. First of all, NNs stand out as the most used, with them being used in 29.41% of the works analyzed by the study, while in the systematic literature review, it is possible to see that NNs are the most common in recent years. Secondly, the DDoS-specific CICDDoS2019 dataset [92] is the predominant choice from 2020 onwards, probably because it offers the most up-to-date and is from the same authors as the previous most-used dataset: the CICIDS2017. Finally, research papers published post-2020 have consistently reported accuracy rates exceeding 99%.

Focusing on specific works, the one by Sbai and boukhari [93] presents an IDS for DDoS attacks expressly targeting data flooding. This IDS was created using DNNs and trained with the CICDDoS2019 dataset. The reported binary classification accuracy was 99.997% for UDP-based attacks. Converting network traffic into images to extract the maximum information out of CNN models has also been explored by Hussain et al. [94]. Their tests showcased a remarkable 99.99% accuracy in binary classification for detecting these attacks using the CICDDoS2019 dataset.

The study by Assis et al. [95] explored the capabilities of a Gated Recurrent Unit to detect DDoS attacks in Software-Defined Networking. The performance of their GRU model was evaluated using two datasets: CICDDoS2019 and CICIDS2018. The results using the CICDDoS2019 revealed that their solution achieved top performance with an accuracy value of 99.6%, only being outperformed by the kNN approach, which scored 99.7%. In the same year, Cil, Yildiz, and Buldu [96] experimented with a DNN model aimed explicitly at detecting DDoS attacks from network packets. They claimed remarkable success rates with the CICDDoS2019 dataset, detecting DDoS attacks with a staggering 99.99% accuracy and classifying the type of attacks with 94.57% accuracy. Amaizu et al. [97] employed a dual DNN framework in an attempt to increase the model performance. One network was dedicated to identifying the attacks, while the other focused on categorizing the type of attacks. Their approach achieved a 99.66% accuracy on the CICDDoS2019 dataset.

In summary, the accuracies for DDoS attack detection of recent models are above 99%, and almost all of them are based on NNs. Furthermore, the dataset of choice for recent works is the CICDDoS2019 dataset.

3.2 Private Neural Networks

As referred to in Chapter 2, the privacy of data used in NN training and inference is not preserved by default. However, some strategies can be used to add privacy preservation to NN. Independent of the chosen strategy, their use comes at some cost in terms of the performance or accuracy of the final model, with the right alternative being dependent on the use case. Next, each one of the significant approaches will be presented and explained.

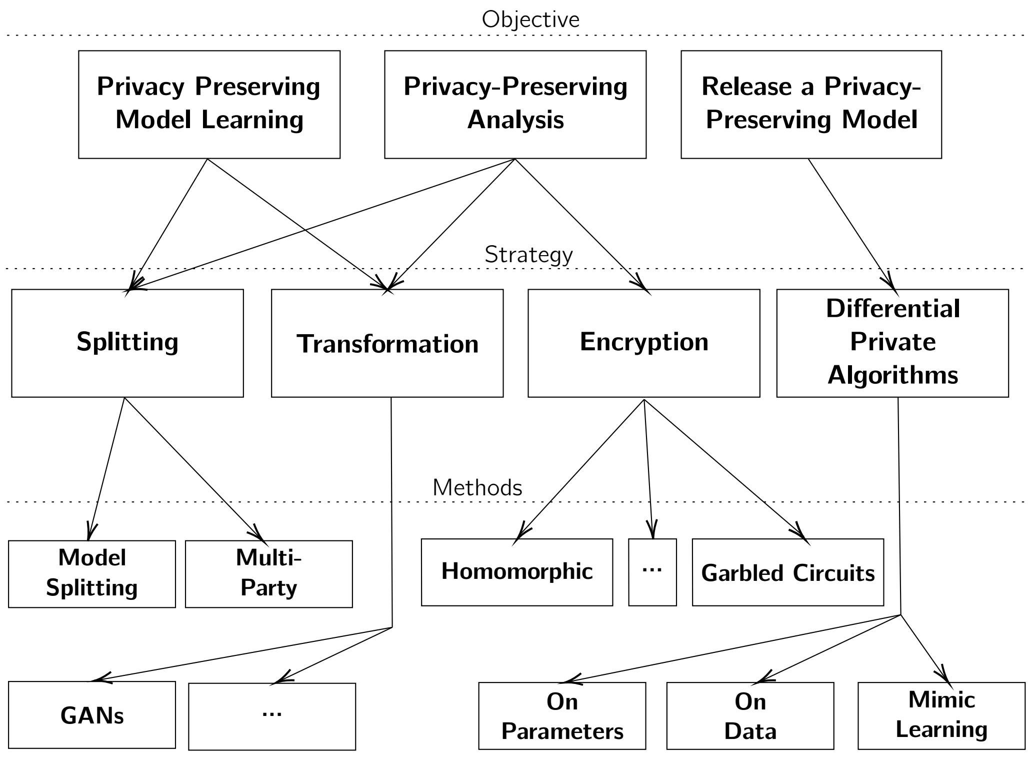

Figure 3.1: Privacy Preserving AI

3.2.1 Privacy Preservation Approaches

As previously mentioned, the use of data to train AI models and their usage to classify data creates a problem regarding the security and privacy of the data. Nonetheless, high-quality, large-scale datasets are essential for creating effective AI models [13]. To reconcile this with privacy concerns, it is necessary to ensure the security and privacy of that data. There are several ways to ensure data privacy, with the most basic one being to use generic data. Nevertheless, this approach would limit the capabilities of the final model because most of the relevant data, in other words, the data that leads to meaningful conclusions, is often behind that privacy barrier [14]. Therefore, other approaches must be identified. Boulemtafes, Derhab, and Challal [5] identified three types of privacy preservation in AI: privacy preservation during model learning (Privacy Preserving Model Learning), privacy preservation during inference of results (Privacy-Preserving Analysis), and privacy preservation during trained model sharing (Release a Privacy-Preserving Model). These types are summarized in Figure 3.1.

Starting with the last type of privacy preservation listed (Release a Privacy-Preserving Model), the strategy for the accomplishment of such task has been done using algorithms that apply noise to the data to give a guarantee of privacy proportional to the amount of noise added (Differential Private Algorithms). This is because, otherwise, "sharing a trained model, either as a service (black-box) or as a model with its internals (white-box), exposes its original training data to leakage risks" [5]. This application of deferentially private algorithms can be done in its parameters, in its input data, or by using mimic learning [5].

The second kind (Privacy-Preserving Analysis) can be done using three distinct strategies: by splitting the model into smaller parts (Model splitting or Multi-party computation), by using AI to obfuscate the data (Transformation), or by using encryption. The first type (Privacy Preserving Model Learning) shares almost all the strategies (excluding encryption) with the second (Privacy-Preserving Analysis) while adding an additional sub-type possibility that is to use partial sharing in the splitting strategy (not represented in Figure 3.1). The model splitting strategy, the multi-party computation strategy, and the partial sharing rely on the belief that if someone has only incomplete data, they cannot infer significant information from their users. However, even a tiny fraction of gradients could be exploited to leak helpful information. On top of that, these strategies suffer from difficulties when trying to find global/local optimums [8]. The transformation strategy can be applied using differential privacy to add noise to the original data or by using Generative Adversarial Networks (GANs) to learn the data and then create a random but near-equivalent version of that same data. The only strategy that uses data in its original form (without partition or noise) is the one named encryption.

The ability of the Encryption strategy to use the original data allows better accuracy and ease of use (no need for complex inter-system interaction configuration). Still, it comes at the cost of computational resources. Even then, it is possible to see that, in the review made by Boulemtafes, Derhab, and Challal [5], the most used technique used for privacy-preserving is encryption, more specifically Homomorphic Encryption (HE) with a total of 13 mentions (of the 19 that were about encryption) which is almost half of the approaches used for Privacy-Preserving Model and Privacy-Preserving Analysis. The next sub-section goes more in-depth about the usage of HE for data preservation in NN for training and inference.

3.2.2 Activation Function Approximations

Some of the approaches for privacy preservation place limitations on the type of operations supported, which is the case of the usage of HE. This means that some functions cannot be calculated under that approach. As an alternative, it is possible to replace these functions with similar output counterparts that do not use unsupported operations or, if supported, use pre-computed values stored in a lookup table. The addition, subtraction, and multiplication operations are commonly supported by HE schemes, which means that all the operations in forward propagation and backpropagation are supported. However, AFs and LFs can introduce unsupported operations. If so, one of the mentioned alternatives must be used.

There are several options to approximate AFs. The sigmoid AF, for example, has various alternatives: some using the Taylor Series as a baseline, and others using piecewise functions. Gilad-Bachrach et al. [44] replaced the sigmoid AF with the square function, which does not resemble the typical sigmoid shape. For higher accuracy approximations, some adopted higher-degree polynomials as polynomial approximations. For instance, Onoufriou, Mayfield, and Leontidis [98] replaced it with a polynomial of degree 3 (as in (3.1)), which, according to the authors, correctly follows the sigmoid between the ranges of -5 and 5. Yuan and Yu [99] approximated the sigmoid activation function using Maclaurin series expansion, and it resulted in (3.2). Ghimes et al. [100] opted for the simplest AF possible, the identity function, to reduce the cost of computing over encrypted data.

$$\sigma(x) \approx 0.5 + 0.197x - 0.004x^3, \tag{3.1}$$

$$\sigma(x) = \frac{1}{2} + \frac{x}{4} + \frac{x^2}{48} + \frac{x^5}{480} + O(x^6). \tag{3.2}$$

In relation to piecewise approximations, the one by Myers and Hutchinson [19] which was used in the work of Chen and Zhong [101] and in the work of Bansal, Chen, and Zhong [102], combines nine linear functions, as defined in (3.3) and present a good approximation for all input ranges without needing exponentially increasing polynomial degrees. Furthermore, the fact that it only requires the computation of linear functions makes it a good alternative for resource-limited environments.

$$\begin{cases} 1 & x > 0 \\ 0.015625x + 0.875 & 4 < x \leq 8 \\ 0.03125x + 0.8125 & 2 < x \leq 4 \\ 0.125x + 0.625 & 1 < x \leq 2 \\ 0.25x + 0.5 & -1 < x \leq 1 \\ 0.125x + 0.375 & -2 < x \leq -1 \\ 0.03125x + 0.1875 & -4 < x \leq -2 \\ 0.015625x + 0.125 & -8 < x \leq -4 \\ 0 & x \leq -8 \end{cases} \tag{3.3}$$

As mentioned by Amorim et al. [20], the usage of approximations for AFs is a common practice when using HE for data privacy preservation in NN. The case of HE is not unique, as this is also the problem in the other approaches. In another work, Amorim et al. [21] evaluated the impact caused by the usage of approximation in the accuracy performance of private NN and concluded that the difference is not significant.

3.3 Private Neural Networks using Homomorphic Encryption

Using encryption for data privacy preservation in NN is one of the more common options in literature, yet it is still an area of active research. As already mentioned, this popularity can be attributed to its ability to process data in a central server without added noise, which could allow private models to match the non-private ones in terms of accuracy closely. However, because there is a gap in the literature on solutions that use HE for data privacy in NN training **and** inference, it is essential to make one before jumping into the proposed solutions.

Although several works are already using HE for NN, the vast majority of them only use HE for data privacy preservation during the inference phase. Yet, as the focus was to create a PNN that preserved data privacy during training inference, our search was mainly limited to those which resulted in only six publications being found. The following subsection will describe each of these works and analyze the types of NN and HE schemes they employed. Then, there is a subsection that will outline the challenges, limitations, and workarounds proposed in existing literature. Finally, this section ends with a subsection summarizing the key findings of this literature review, where future prospects are also analyzed.

3.3.1 PNN for Training and Inference

The first work that proposed a solution to the data privacy problem during training and inference of NN models using HE was the one by Hesamifard et al. [103]. It was published in 2017, and the approach used a replacement of the Sigmoid and ReLU AFs by low-degree polynomial approximation to allow them to be calculated in a HE encrypted context. To solve the problem of ciphertext noise, which increases with each multiplication and can reach a critical point, they proposed that, given a threshold (lower than the critical point), the ciphertext would be sent back to the client so that it could return a fresh ciphertext. While this approach adds considerable complexity to the overall system, the authors claim it outperforms bootstrapping in efficiency. To validate their proposed solution, tests were executed using three different datasets across five distinct NN configurations, achieving reported accuracies above 99%. At the time, no other solutions were using HE for data privacy preservation in NN training and inference for comparison. However, when compared with the usage of HE for data privacy preservation only during inference and with the use of other SotA approaches, namely, Multi-Party Computation, the proposed solution showcased superior performance. Focusing on the comparison with Multi-Party Computation alternatives, the number of communication needed is referred to as an advantage due to its significant reduction in the number of interactions required. The proposed solution uses the BGV scheme implemented by the HElib library.

Not mentioning the previous work by Hesamifard et al. [103], Ghimes et al. [100] published a work in 2018 proposing a solution where no server-client interaction was needed. This no-interaction approach aligned with their primary goal, which was to allow the outsourcing of NN to third parties without compromising data security and privacy. To address the constraints introduced by HE in the computation of AFs and mitigate the computational overhead of calculating them, they advocated for using the identity function as AF. Yet, as acknowledged by the authors, the outcomes were suboptimal due to their high error values and long computational times. An important observation is that, during their tests, they used the SEAL library's implementation of the now obsolete YASHE scheme.

In 2019, Nandakumar et al. [104] gave another try to the BGV scheme, but, contrary to the work of Hesamifard et al. [103], it did so without using server-client communication to refresh the ciphertext. The main challenges identified were the computation of the AF and LF as well as the computation of their derivatives. Contrary to the previous works, and based on the solution proposed by Crawford et al. [105], the authors of this paper choose to pre-compute these functions and store their values in a table that would then be looked up in a homomorphic context. This solution calculates such functions faster which may result in less total ciphertext noise and lower overall computational time. However, it can only provide a low-precision approximation of these functions. To avoid the need for excessive table lookups, the authors choose the sigmoid AF and quadratic LF because of their simple derivatives. To take advantage of the BGV support of SMID and to reduce computational complexity, a data-packing strategy was used. The experiments of the proposed solution were conducted on the MNIST dataset in two NN configurations. Using optimization and multiple CPU threads, the less complex four-layer NN reported an accuracy of 96% with an execution time of 40 minutes. Even with those results, the authors acknowledge that the training is still four to five times slower than in plaintext and that further research is required to improve efficiency.

Recognizing the major limitation of the BGV for table lookups, Lou et al. [106] decided to propose a solution called "Glyph" that used a HE scheme switching technique. The usage of this switching technique allowed for the continued use of the BGV scheme while delegating table lookups to the TFHE scheme. This HE scheme switching is possible because it is possible to map the plaintext spaces of BGV and TFHE to a common algebraic structure using natural algebraic homomorphisms. This leverage of the fast multiplication-accumulation operations of the BGV scheme, with the quick table lookups provided by the TFHE, comes at the cost of the scheme switching. This cost, as mentioned by the authors, is easily overcome if the switching is only done where the computational complexity justifies it. The experiments executed showed that the switching did indeed bring a significant improvement to the computational time, with Ghyph being able to perform training in 8 days that was estimated to take 13.4 years using the solution proposed by Nandakumar et al. [104] while having a final accuracy 0.8% higher. Overall, their experiments showed that Glyph is able to achieve SotA accuracies while reducing training latency by 69% to 99% over prior HE-based privacy-preserving techniques.

In contrast to previous methodologies, Onoufriou, Mayfield, and Leontidis [98] introduced a significantly different approach in 2021. With the focus on the domain of milk yield forecasting for the agri-food sector, the authors choose to ensure data privacy using HE in CNN. While Lou et al. [106] used CNNs, they refrained from training them and focused instead on training the DNN part of the NN. This makes this wort to be the first one to tackle the training of CNN, yet it requires plaintext data processing. While modifications to the CNN structure can overcome some of the challenges during inference, the backpropagation phase introduces significantly more challenging problems. Faced with these limitations, the authors decided that it was better to do the backpropagation phase in plaintext. The replacement of the average with a scaled sum allowed the usage of the division operation (not supported by the CKKS scheme) to be avoided. The replacement of the Sigmoid AF with the polynomial approximation proposed by Chen et al. [107] solved the same problem and the usage of the exponential operation. In the end, the tests reported an accuracy of 87.6% and a Mean Absolute Percentage Error of 12.4% on a dataset with 30 years of breeding, feeding, and milk yield data using the SEAL library's implementation of the CKKS. Independent of the results, the usage of plaintext data for backpropagation is a severe limitation of this research, which means that further research is needed to address this challenge.

At last, Yoo and Yoon [108] unveiled a framework called "t-BMPNet". The objective of this framework is to facilitate the design of private and secure deep learning models, namely, Multi-layer Perceptron Neural Networks. Using the TFHE scheme, this framework aims to allow the training and inference of NN within the encrypted domain while not requiring the replacement of AF by polynomial approximations or table lookups. The authors claim that this way, it is possible to obtain more accurate NN. A notable characteristic of the TFHE scheme is that it encrypts individual bits as opposed to numbers, which gives more flexibility to the user but also requires the manual definition of the number types and their accompanying operations, utilizing the provided ciphertext boolean functions. Despite this challenge, this unlocks the potential to carry out operations that other schemes do not support, namely, the division, shift, and comparison operations and the exponential function. Setting aside an anomaly observed in the reproduction of their exponential function, which will be presented later in this thesis (Section 5.3.1), the theoretical implications are profound as it means that it is possible to execute the NN with the Sigmoid AF without any approximations - an accomplishment that none of the preceding research was able to achieve. The results from the authors' experiments confirmed the high accuracy of their approach. Yet, they acknowledge the significant computational time required, demonstrating the need for continued research and optimization.

Used Private Neural Network Types

The types of NNs used in the reviewed literature are not very diverse, with only two types being used: DNN and CNN. Even then, only two of the selected publications used CNNs, while the remaining used DNNs. Interestingly, the work by Lou et al. [106] was the only one to compare both DNN and CNN. Considering that CNNs typically consist of a "Convolution" component followed by a DNN, it becomes clear that the establishment of trainable private DNNs is a prerequisite for developing trainable private CNNs. Essentially, the latter can be viewed as an expanded version of the former with added complexity. Excluding certain studies that have prioritized specific metrics, as Yoo and Yoon [108] did by prioritizing accuracy as a performance measure over computational time, it is possible to verify that there is a positive trend in increased overall performance over the years. However, it is also clear that CNNs have several challenges that make it difficult to implement using HE. This is corroborated by the fact that Lou et al. [106] opted instead for leveraging a pre-trained CNN and concentrating only on training the DNN component of the neural network and that Onoufriou, Mayfield, and Leontidis [98] opted for plaintext backpropagation. From this, it becomes evident that training CNNs using homomorphically encrypted data remains a significant challenge and limitation to its widespread use.

Used Homomorphic Encryption Schemes

In terms of HE schemes, it is clear that most works use FHE schemes, with only one work not using an FHE scheme and instead using an SMHE scheme. Within the FHE schemes, the BGV scheme was the most used, with half the selected works using it. In the hierarchy of preference, the BGV scheme is succeeded by the TFHE scheme, which is used in two publications. Concluding the list is the CKKS scheme with an appearance in just one of the reviewed publications. With the SMHE schemes, the only one used was the YASHE scheme. The results are somewhat related to the scheme used, with the work that employed the deprecated YASHE scheme being unable to attain satisfactory results. The ones that use still relevant schemes have positive results. Noteworthy is the fact that the work that used CKKS did achieve acceptable results, but its usage of plaintext propagation somewhat diminishes the significance of its results. Studies using the BGV scheme saw a consistent uptrend in overall performance, reaching its peak when combined with the TFHE scheme. The TFHE scheme, notable for its excellent performance in conjunction with BGV, also made it achievable to create very accurate private NN by not using approximations at the cost of superior computational overhead.

3.3.2 Identified Challenges and Limitations

Upon reviewing the existing literature, it was found that regardless of the NN type and HE scheme used, some transversal challenges and limitations impact the creation of every PNN using HE. Next, each major challenge and limitation identified with the respective workarounds will be listed and discussed.

Most HE schemes support, at least, addition and multiplication. However, NN often requires more types of operations for their usage. Even for schemes that do support other operations, those often come at a significantly higher cost, which means that their use must be made with caution. All this means that there are limitations that must be considered when building NN with HE, with different types of NN introducing specific restrictions. In DNNs, unsupported operations often appear in the AFs and LFs, as some require comparisons, division, or exponentiation. CNNs add to AFs and LFs the convolution-related operations, namely, the Average Pooling, which involves division, or the Max Pooling, which requires comparisons.

The solution to this limited operation problem differs according to the group of HE schemes used. Schemes that encrypt whole numbers, such as the BGV, CKKS, and YASHE, do not offer great flexibility to use other options, as manipulating the numbers is limited by the operations provided by the scheme. In this type of scheme, there are primarily two approaches utilized to address the limitations on available operations, which are the usage of polynomial approximations or the usage of table lookup strategies. Most authors choose to use polynomial approximations [98, 100, 103], while some opted for the second [104, 106]. While the first can produce more accurate results, this accuracy comes at the cost of several expensive multiplication operations that are mandatory when using high-degree polynomials. The second one avoids this by using pre-calculated values. However, its restricted ability to only provide low-precision approximations can affect the accuracy of the NN.

Schemes that only encrypt individual bits are more flexible, as they provide all the basic boolean operations to allow most operations to be manually implemented. Lou et al. [106] used this flexibility together with the speed of the TFHE scheme to create a fast table lookup. In contrast, Yoo and Yoon [108] used this flexibility to implement the whole NN using numbers represented using these bits. Formidably, according to their authors, the latter approach allows all the required operations to be implemented, which includes division, comparison, and exponentiation. As will be possible to see in Chapter 5, these results could not be fully replicated. However, if proven correct, it means that, when using such schemes, the first significant burden disappears, and time becomes the predominant issue when using HE in NN.

Also common in all the reviewed literature is the computational overhead added by HE. NNs are already complex by themselves, with some of them requiring entire data centers to achieve results in practical time, but adding HE only exacerbates this problem. While base encryption is already expensive, adding homomorphic characteristics to it only adds more complexity, which means that an operation over HE encrypted ciphertext takes several orders of magnitude more than the plaintext one, with Yoo and Yoon [108] reporting that it takes 1 minute and 39 seconds to calculate the result of a single multiplication of two 32-bit numbers while a plaintext equivalent on a modern computer takes mere nanoseconds. Consequently, this limitation can make NN, once trained in seconds, untrainable.

The primary strategy for mitigating this problem is to find a balance between the complexity of the neural network and computational speed. Still, NN complexity involves several related aspects that must be carefully managed to ensure the best relation between the NN's ability to learn and infer correctly and the computational time. The basis for every NN is the number representation, which has a direct impact on every operation performed inside the NN, with a lower number of bits affecting the precision of the NN outputs. In comparison, a higher number of bits may make the NN too expensive to compute. Additionally, the number of layers and neurons of an NN is also very influential in its computation complexity because its growth increases exponentially the number of numeric operations required. However, too few layers or neurons can hinder the ability of the NN to hold information and make it unable to learn complex mapping. Due to this fact, most of the reviewed works used only a maximum of three layers [100, 104, 108], with the work of Hesamifard et al. [103] reaching a maximum of five hidden layers. AFs can also have a significant impact on the NN complexity, as their computational cost can be significantly high, especially when using high-degree polynomial approximations.

In the same computational overhead problem, some HE schemes allow the usage of a strategy, commonly referred to as "ciphertext packing", that allows the computational overhead of the HE function to be shared across multiple ciphertexts at once. As mentioned by Hesamifard et al. [103], increasing the number of ciphertexts packed together also increases the memory required, so although it can have a significant positive impact on the final NN performance, it is important not to over-do it.

Finally, the last major limitation identified was related to the representation of numbers in HE encrypted ciphertext. Most HE schemes represent numbers using the fixed-point representation instead of the traditional floating-point representation, which affects the representable number space and precision. Furthermore, the inability of fixed-point representation to adapt to changes in number magnitude means that unbounded functions are significantly more prone to overflows, which, in turn, can cause numerical instability. This lack of numerical stability can cause catastrophic loss of accuracy of the NN, rendering it unusable. In this last limitation, it is essential to note that even though BGV uses an integer representation of numbers, its inner workings are identical to the ones of fixed-point representation, so most of the problems identified can be applied to both.

In relation to this number representation limitation, there is not much that can be done, especially in number encrypting HE schemes. Nevertheless, encoding strategies can be used to represent fractional numbers in integer-only schemes; however, this comes at the cost of reducing the range of representable numbers. Bit by bit, HE schemes offer a bit more flexibility, but, as mentioned by Yoo and Yoon [108], even though it is supported, floating-point operations are too expensive for HE applications when compared to fixed-point.

3.3.3 Key Findings and Future Prospects

From the reviewed literature, it is clear that CNNs are still a far cry from full PNN using HE. Nonetheless, DNNs have already progressed significantly towards a point where real-world usage is possible. Likewise, it is also possible to see that the FHE schemes are the most used, with the BGV and TFHE schemes emerging as the most promising prospects. Intriguingly, a combination of these two could potentially cancel out each other weaknesses, so further research is essential. Focusing on the TFHE scheme, if additional optimizations are made, it possesses the potential to be viable as a natural alternative for PNN that supports inference and training while not compromising accuracy. This was the scheme chosen for the PNN proposed in this thesis.

Regardless, the usage of HE for data privacy protection in NN is still defined by several limitations that are still to be overcome. If the results from [108] are replicated, the first significant limitation may be about to be solved thanks to bit-by-bit HE schemes, and its solution can be brought to other schemes using HE switching techniques. Nevertheless, using approximations has proven to be a viable option as they are now able to allow NN to achieve comparable performance to non-approximated NN. On the other hand, problems related to computational overhead are still very deeply rooted inside all HE operations. Due to this fact, overcoming this limitation is still a balancing act between speed and the different components of the NN, with the bits used for number representation and the numbers of neurons/layers being the most impactful hyper-parameters. Number representation is also considerably connected to this topic, with fixed-point being the most used type due to its lower computational cost when compared to floating-point.

Chapter 4

Private Neural Network Architecture

In light of the limitations identified in the existing SotA, this chapter introduces a PNN architecture designed to address or mitigate those shortcomings. By doing so, the aim is to offer a PNN that has less impact on the model's performance while achieving the maximum possible computational efficiency. To do this, this chapter presents the architecture of such a solution using a well-defined methodology. This methodology will be presented in the second section, and the following sections will explain the architecture itself, in descending order of detail, as defined in the methodology. The problem and respective system requirements are presented in the first section.

4.1 System Requirements

Guaranteeing that the requirements of a system are well-defined before defining its architecture is essential. Well-defined requirements are necessary to ensure that the resulting solution is capable of correctly performing all the required functionalities of the initial problem. Firstly, to do so, the problem must be clearly defined. Given the objectives of this thesis and the SotA review made in Chapter 3, there is a need for further optimizations of the HE-based NN to reduce their computational overhead. The work of Yoo and Yoon [108] presents a framework for building multi-layer perceptron networks using the TFHE scheme, but the computational complexity needs to be reduced. In this chapter, the architecture of a PNN library will be presented. The objective of this library is to provide a PNN that supports both inference **and** training while also improving computational efficiency and maintaining acceptable NN performance.

To address this issue, the library must meet specific prerequisites. Upon studying the problem, it becomes clear that two primary processes need to be supported: training and inference. These processes are well-defined in the context of a standard NN. Importantly, due to the way HE works, incorporating HE in NNs does not require changes to the underlying mathematical formulas. However, approximations may be necessary when the operations required are computationally expensive or unsupported by the selected HE scheme. To make this PNN solution practical in real-world scenarios, it should also provide a procedure to define the structure of each NN, which includes the number of layers, the number of neurons for each layer, the chosen AFs, and many others. This ability to create NN models is essential to avoid constant code changes, as the structure of NNs can change regularly.

Examining each process individually, it is clear that inference is straightforward and does not require to be configurable. In contrast, the training process needs to be highly configurable. This requirement of high configuration is due to the presence of hyper-parameters — such as the number of epochs, batch size, and learning rate — that must be defined on a per-training basis and cannot be hard-coded.

From a non-functional point of view, the main requirement of the solution is for it to be efficient. Besides that, due to the fact that HE is a rapidly changing field, the architecture should allow the exchange for a newer scheme with the minimum amount of changes required.

4.2 Methodology

To present a clear definition of the PNN architecture, a combination of two architectural visualization models will be used: 4+1 [109] and C4 [110].

The first model emerged in 1995 and determined that a system must be defined using four different views that complement each other and one general aggregating view. These views are the following:

1. **Logical View** - This view focuses on the functional requirement of the system;
2. **Process View** - This view represents the behavior of the system during execution, namely, the communication between each software module required to fulfill a use case;
3. **Implementation View** - This view illustrates the organization of the software during its development;
4. **Deployment View** - This view depicts the hardware disposition of the different software entities in an execution environment;
5. **Scenarios** - This view is an aggregation of the other views, and it portrays an overall perspective of the four views of the use cases.

Due to the redundancy of the +1 View (Scenarios), it is not used in this chapter. Furthermore, because this methodology will be used to describe the architecture of a library, the Deployment View will also not be used. The remaining views are used according to the needs of each section.

The second model defines four levels of architectural representation based on their level of abstraction. That is, the upper level represents the system from a very abstract view, and the subsequent views go progressively into more detail. The four levels defined by this model are:

1. **Context** - Provides a starting point, displaying how the software system fits into the world around it;
2. **Container** - Shows the high-level technical building blocks of the software system;
3. **Component** - Explains the individual containers, displaying the components inside them;
4. **Code** - Details the implementation of the components;

The authors of the C4 model do not recommend the usage of the fourth level, and given that it is not essential to this particular use case, it will not be used in this document.

In summary, this chapter will present the architecture of the system using a combination of two architectural visualization models: the 4+1 model specifies the "ways" to look at the problem, and the C4 model specifies which levels of "zoom" are needed to present them. All of the diagrams present in this chapter will be in the **U**nified **M**odeling **L**anguage [111] so that all of them are in a standard notation and to provide consistency. Furthermore, any software entity external to the system, which includes external applications and libraries, will be colored gray to allow easy differentiation from the ones introduced by this architecture.

4.3 Level 1 - Proposed Solution Context

This section explains the proposed solution's architecture in its higher level of abstraction defined in the C4 model. At this level, only the logical and process views will be presented. The implementation view is not present because, at this level of abstraction, their added details are insufficient to justify their inclusion.

4.3.1 Logical View

From a logical standpoint, it was defined that this solution is composed of a single software system, which will be called "ENNigma", and it must be able to perform all of the use cases mentioned. Any external Software System will then be able to use this library using the provided ENNigma API to create, train, and use PNN models. The representation of this view is present in Figure 4.1.

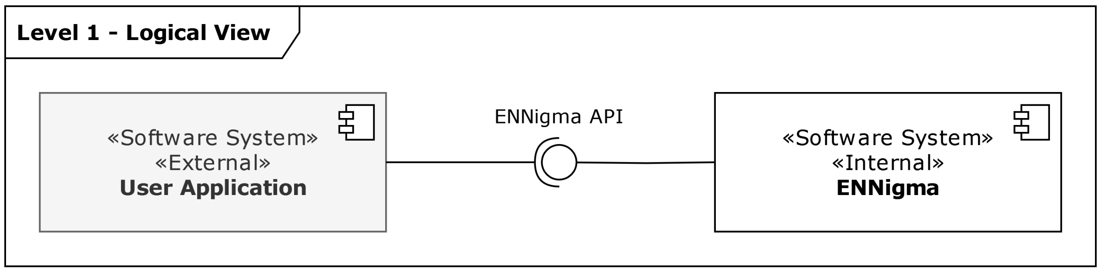

Figure 4.1: Level 1 - Logical View

4.3.2 Process View

To fulfill the requirements, the ENNigma Software System will provide the three main functionalities identified, and in this section, each of them will be described at the most abstract level. They will be revisited in the next levels of abstraction for a more detailed description.

The first main functionality is the creation of a new PNN model. The Application Programming Interface (API) will provide an interface that allows an external application to define the model structure and obtain an instance of the newly created model. This model-creation process is essential because these model instances are the ones that will later be trained and used for inference.

The second main functionality is the training of a PNN model. That is, given a model instance created using the previously described functionality, the external application can train it using labeled data. To do this, this external application must use the API to send the model instance and the encrypted training data to ENNigma, which will, once the training is finished, return the trained model and the training metrics.

The final main functionality is the ability to classify data using a model instance. This data classification process can be done on any model instance, yet its utility comes from its usage on a trained model. To classify data, the external application must use the API to send the trained model instance and the data to ENNigma, which will then return the classification results.

These three processes are depicted in Figure 4.2.

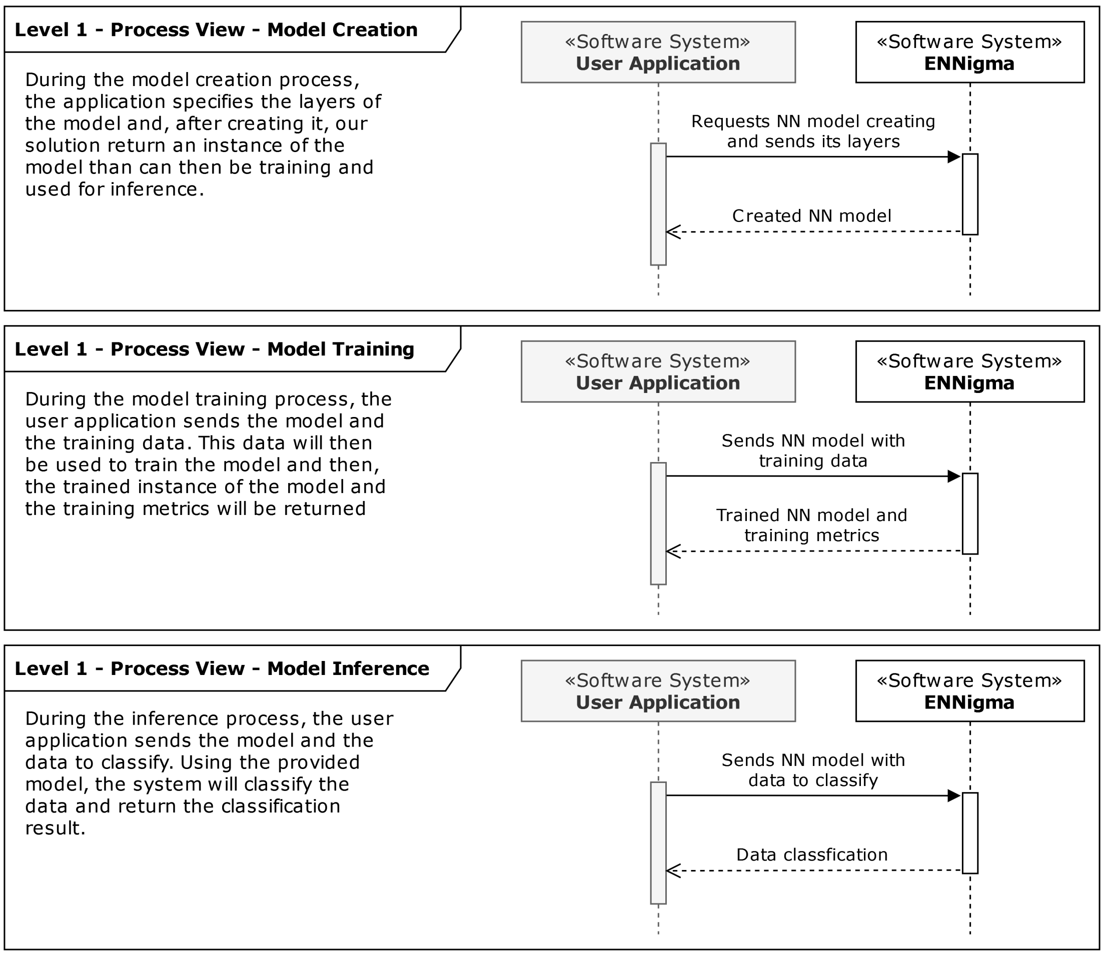
Level 1 - Process View - Model Creation
During the model creation process, the application specifies the layers of the model and, after creating it, our solution return an instance of the model than can then be training and used for inference.
«Software System» User Application
«Software System» ENNigma
Requests NN model creating and sends its layers
Created NN model
Level 1 - Process View - Model Training
During the model training process, the user application sends the model and the training data. This data will then be used to train the model and then, the trained instance of the model and the training metrics will be returned
«Software System» User Application
«Software System» ENNigma
Sends NN model with training data
Trained NN model and training metrics
Level 1 - Process View - Model Inference
During the inference process, the user application sends the model and the data to classify. Using the provided model, the system will classify the data and return the classification result.
«Software System» User Application
«Software System» ENNigma
Sends NN model with data to classify
Data classfication

Figure 4.2: Level 1 - Process View

4.4 Level 2 - Intrusion Classifier Software System

Having the most abstract level explained in the previous section, this one goes down to the Container level of the C4 model. The three views will be used in this chapter. As there is only one software system in this architecture, all of the information in this section is related to that one. At this level, a significant split of the Software System was made to create four well-defined containers. This section will start by explaining those and the thought process that led to their definition, and only after will it present the different views.

4.4.1 Division of Responsabilities

Designing a PNN is a task that involves several modifications to the architecture of a NN. Due to the usage of HE, these changes are more contained to the data processing than the data itself, so the architecture must be able to support those changes.

As seen in Chapter 3, one of the significant limitations of using HE with NN is the need to use a custom number representation instead of the standard floating-point representation provided by the system. Due to the fact that number representation is the foundation of any NN, it should be the first thing to be defined. To do this, a container called "Numbers" was defined. Its responsibility is to handle all number-related tasks, such as memory representation, operations, and type conversion.

The encryption, decryption, and homomorphic operations will be done using the TFHE scheme because, as reviewed in Chapter 3, it is a promising HE scheme. The usage of HE could be directly injected into this Numbers container, but to reduce coupling to the chosen TFHE library, this responsibility will be delegated to a new container called "Crypto". This means that if a better scheme is developed, the abstraction of the Numbers allows a new implementation of Crypto to be used without significant code change.

With the number representation well defined, it is time to add a new container called "NeuralNetworks" that, using the numbers provided, will implement all NN logic. The reduced number of supported operations will be handled case by case, namely when implementing AFs. The remaining NN implementation will use the traditional NN equations.

Finally, to present a single interface to the outside world, a fourth container, called "Aggregator" was defined. It will join the NeuralNetworks with the respective Crypto container (injecting it to the NeuralNetworks as an implementation of the Numbers container) and provide an abstract interface to the outside that supports all the required use cases while hiding the internal division of the Software System.

Next, each container will be briefly summarized.

Numbers

The Numbers container is, at its core, the one responsible for the definition of all the number types and operations used in all of the remaining containers. Besides basic number operations like addition and multiplication, it also defines matrix representation and operations that will be used to create the PNN.

Crypto

The Crypto container is the one responsible for the crypto implementation of the Numbers. The way the latter is implemented allows the types provided by this container to be used interchangeably with other implementations. This means that other containers can implement their functionalities without knowing the existence of this one and, even then, fully support it.

NeuralNetworks

The basis for a PNN is in the NN itself, and it is responsible for that exact task. To do that, it uses the number types and operations defined in the Numbers to implement the NN. This NN implementation requires, besides the basic NN, other components such as Metrics, AFs, and LFs, to name a few. More details about these components will be presented in the next level of abstraction dedicated to this container.

The only NN type supported will be the Deep Neural Network (DNN) variant, which means it will not support convolutions nor recurrence. However, this NN implementation must be configurable to allow it to accommodate different models and domains at once. This is very important because the model's structure and hyper-parameter can be subject to change, and using a configurable implementation, changing it does not require code changes in the library.

Aggregator

The final container is called Aggregator, and its responsibility is to provide a unified interface that joins together the NN implementation, provided by the NeuralNetworks, with the encryption number implementation, provided by the Crypto. This replacement of the Numbers can be done thanks to the abstracted number types. As already mentioned, if needed, the TFHE-based Crypto container can easily be replaced by another that uses a different HE scheme.

It is important to reiterate that the objective of this one is to provide a unified interface. Therefore, it must not implement any new features because these are the responsibilities of the other containers. Furthermore, the existence of this one reduces the details of the other, which minimizes the impact of breaking changes in the other containers.

4.4.2 Logical View

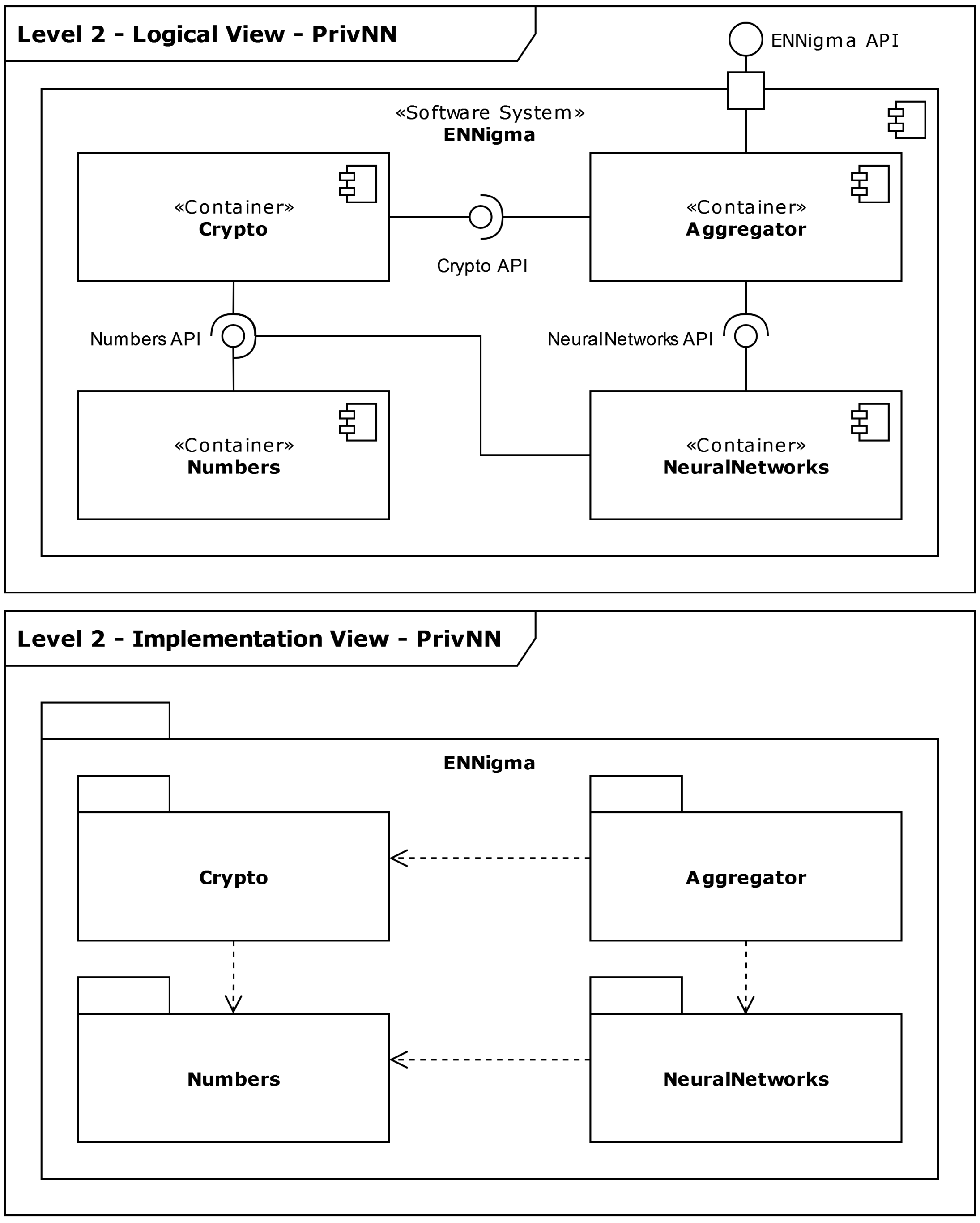

Figure 4.3: Level 2 - Logical and Implementation Views

As just mentioned, this Software System is subdivided into four containers. This internal division is translated one-to-one to the logical view. The logical view is represented in Figure 4.3.

4.4.3 Implementation View

To make implementation easier, faster, and more maintainable, the code base should be divided into smaller independent packages. Considering the very distinct objectives and structure of the containers identified, this software system can easily be divided into four packages, one for each container. This way, all the code base for each container is inside their own package, making implementation as independent as possible. The implementation view is illustrated in Figure 4.3.

The relations between the packages are directly related to their logical connections. This means that the Number package is independent. In contrast, the remaining packages depend on at least one other package.

4.4.4 Process View

Fulfilling the three main processes of this library requires the USA of all of the containers. As shown in the logic view, the PNN is the only one that is aware of the existence of the Crypto container; because of this, it is the one responsible for the injection of this container into the NeuralNetworks one.

Compared to the previous level of detail, this one only adds the communication between the different containers. These interactions are mainly between the Aggregator and the NeuralNetworks containers and between the NeuralNetworks and Crypto containers (with the last as an implementation of the Numbers containers).

The main difference between the three processes is that the first only encrypts numbers, and the other two perform operations on encrypted numbers. It is important to note that the implementation of the Crypto container ensures that all data used in this library is encrypted. This includes training data, data to classify, training metrics, and classification results. These three processes are illustrated in Figure 4.4.

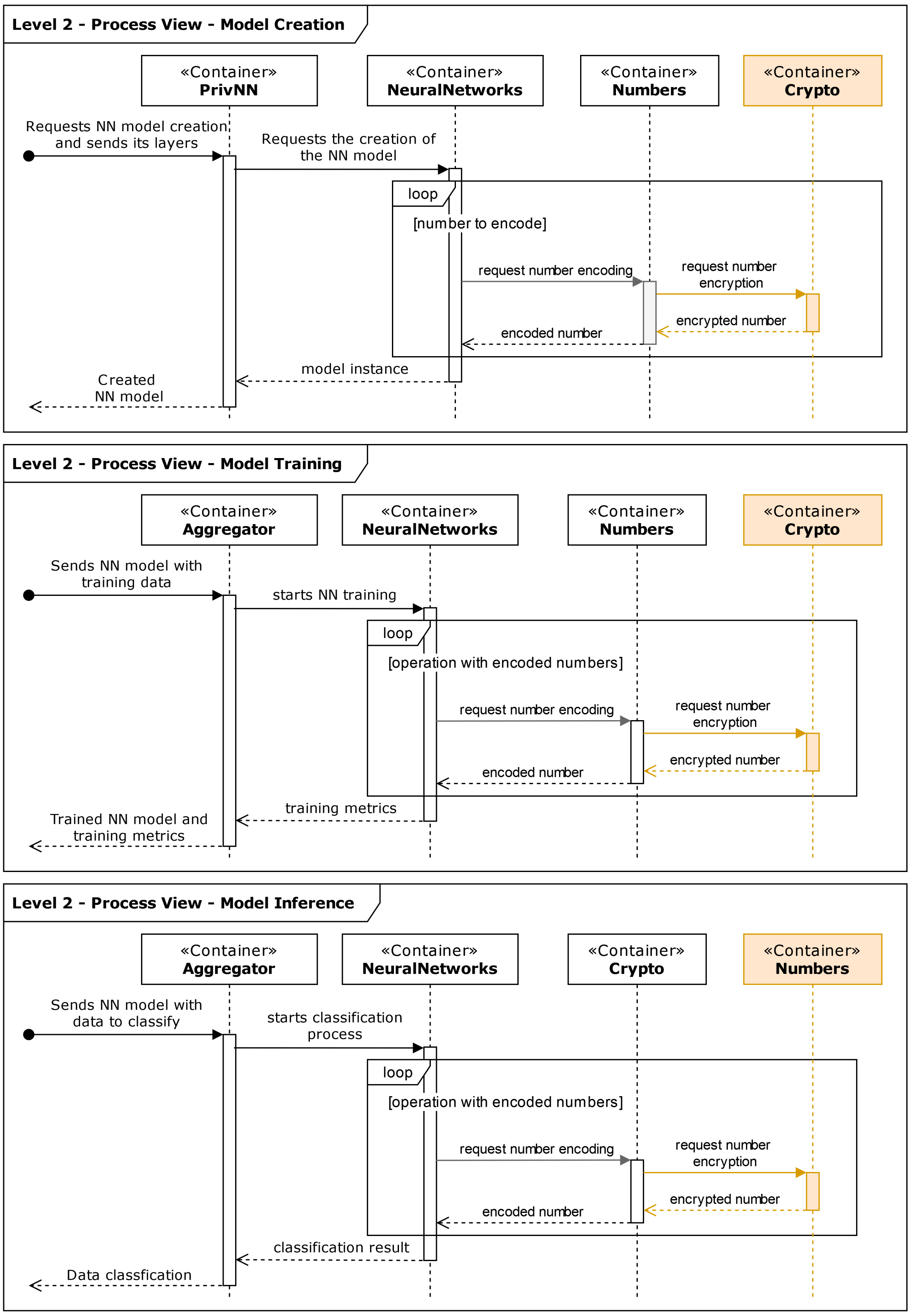
Level 2 - Process View - Model Creation
«Container» PrivNN
«Container» NeuralNetworks
«Container» Numbers
«Container» Crypto
Requests NN model creation and sends its layers
Requests the creation of the NN model
loop
[number to encode]
request number encoding
request number encryption
encrypted number
encoded number
model instance
Created NN model
Level 2 - Process View - Model Training
«Container» Aggregator
«Container» NeuralNetworks
«Container» Numbers
«Container» Crypto
Sends NN model with training data
starts NN training
loop
[operation with encoded numbers]
request number encoding
request number encryption
encrypted number
encoded number
training metrics
Trained NN model and training metrics
Level 2 - Process View - Model Inference
«Container» Aggregator
«Container» NeuralNetworks
«Container» Crypto
«Container» Numbers
Sends NN model with data to classify
starts classification process
loop
[operation with encoded numbers]
request number encoding
request number encryption
encrypted number
encoded number
classification result
Data classfication

Figure 4.4: Level 2 - Process View (Gray-colored containers represent an abstraction for the calling container)

4.5 Level 3 - Numbers Container

The numbers container is, as explained in the previous sections, responsible for the declaration of the numeric types and matrix representation types. In this level of detail, only the logical and implementation views will be presented. Nonetheless, and because the fourth level will not be presented, the algorithms of the custom number representation will only be explained in Section 5.1.

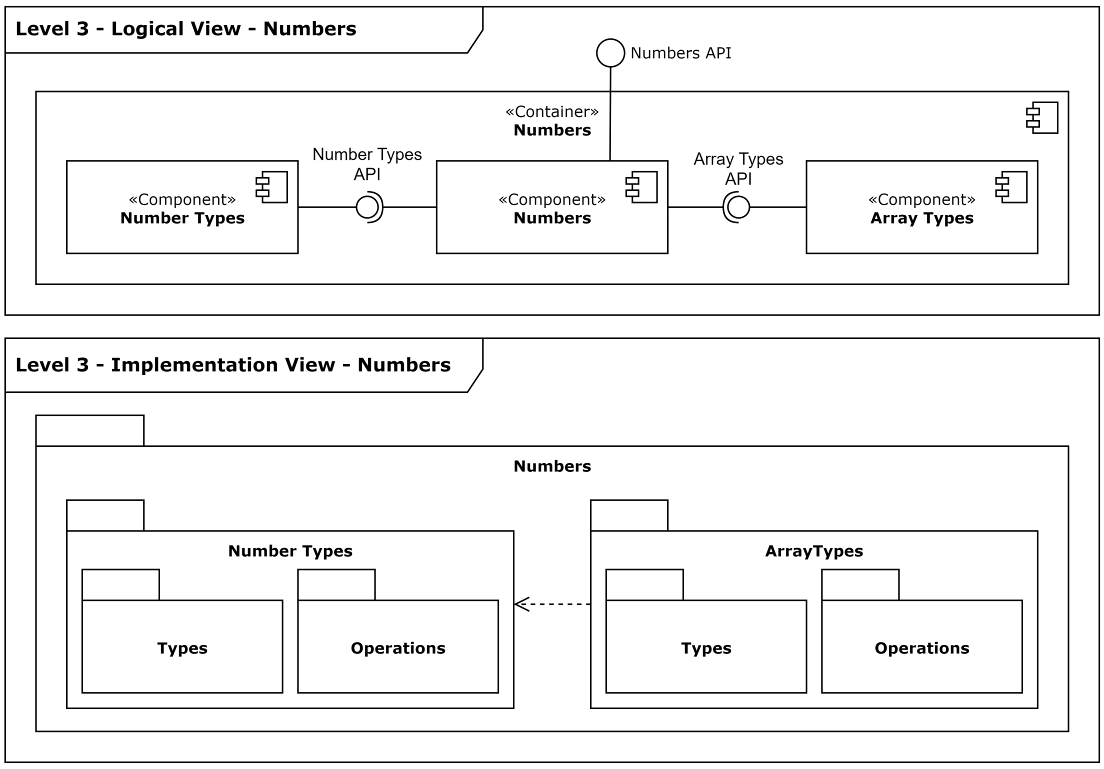

Figure 4.5: Level 3 - Numbers Container - Logical, and Implementation Views

4.5.1 Logical View

From a logical standpoint, this container is divided into two main components and one aggregator component. The first two components are responsible for the number type and matrix type definitions. At the same time, the third is responsible for providing a unified interface to the outside, which is, in this case, other containers. The logical view is presented in Figure 4.5.

4.5.2 Implementation View

The fact that the two main components are each responsible for a type definition means that they are also separated from an implementation point-of-view. Likewise, because they define types and operations, it is essential to separate those two into sub-packages. The implementation view is presented in Figure 4.5.

4.6 Level 3 - Crypto Container

This container is one of the most important because it is the one that ensures the privacy of the data in every operation performed in this library. To do this, it uses an external TFHE library, in this case, TFHE-rs [55], to represent the bits and implement the logical operations between those bits. This container will only be presented in two views: Logical and Implementation.

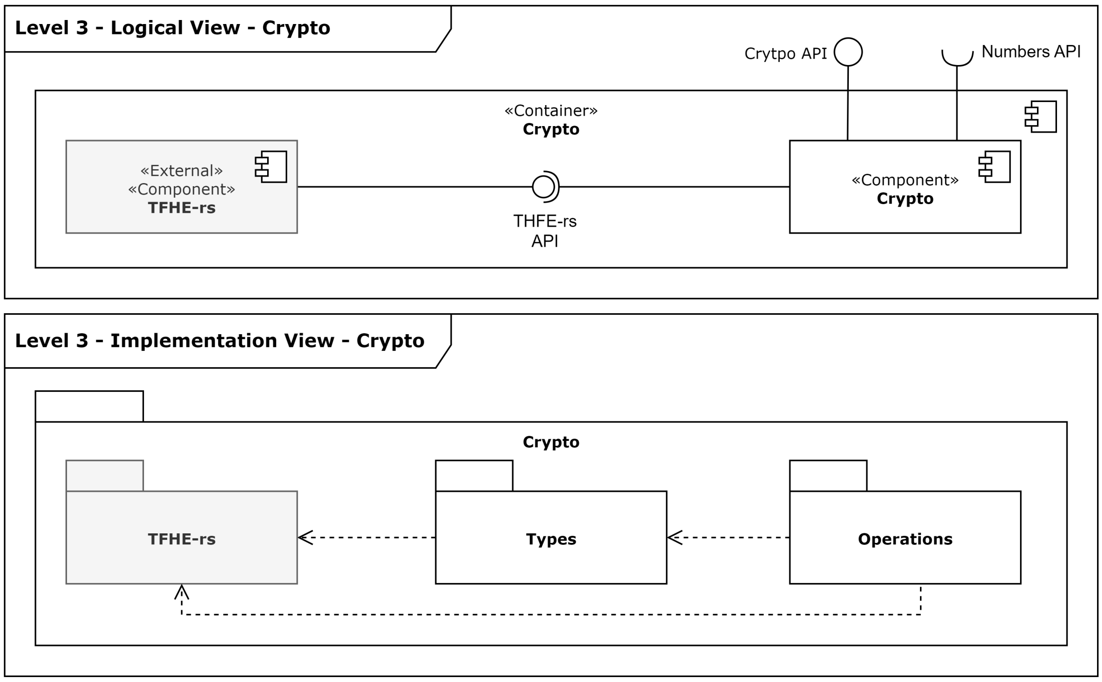

Figure 4.6: Level 3 - Crypto Container - Logical, and Implementation Views
(Gray-colored represent an external container/package)

4.6.1 Logical View

From a logical perspective, this one is only composed of one internal component that uses an external component to implement the HE number type. Besides that, it also depends on the Numbers container, which is self-explanatory given that the objective of this container is to implement the types defined in the former. The logical view is presented in Figure 4.6.

4.6.2 Implementation View

Implementation-wise, this container is divided into three parts: one external and two internal. The external package is the TFHE-rs library, and the internal packages are one for the type definition, namely the HE encrypted bit representation. The other is the implementation of the operations over that bit to allow it to be used in the number type defined in the Numbers container. The implementation view is presented in Figure 4.6.

4.7 Level 3 - NeuralNetworks Container

The NeuralNetworks container, by itself, is similar to any NN library with the exception that, instead of using the architecture-provided floating-point number representation, it uses a custom number representation, which is defined by the Numbers container. In this section, the Logical, Implementation, and Process views will be presented.

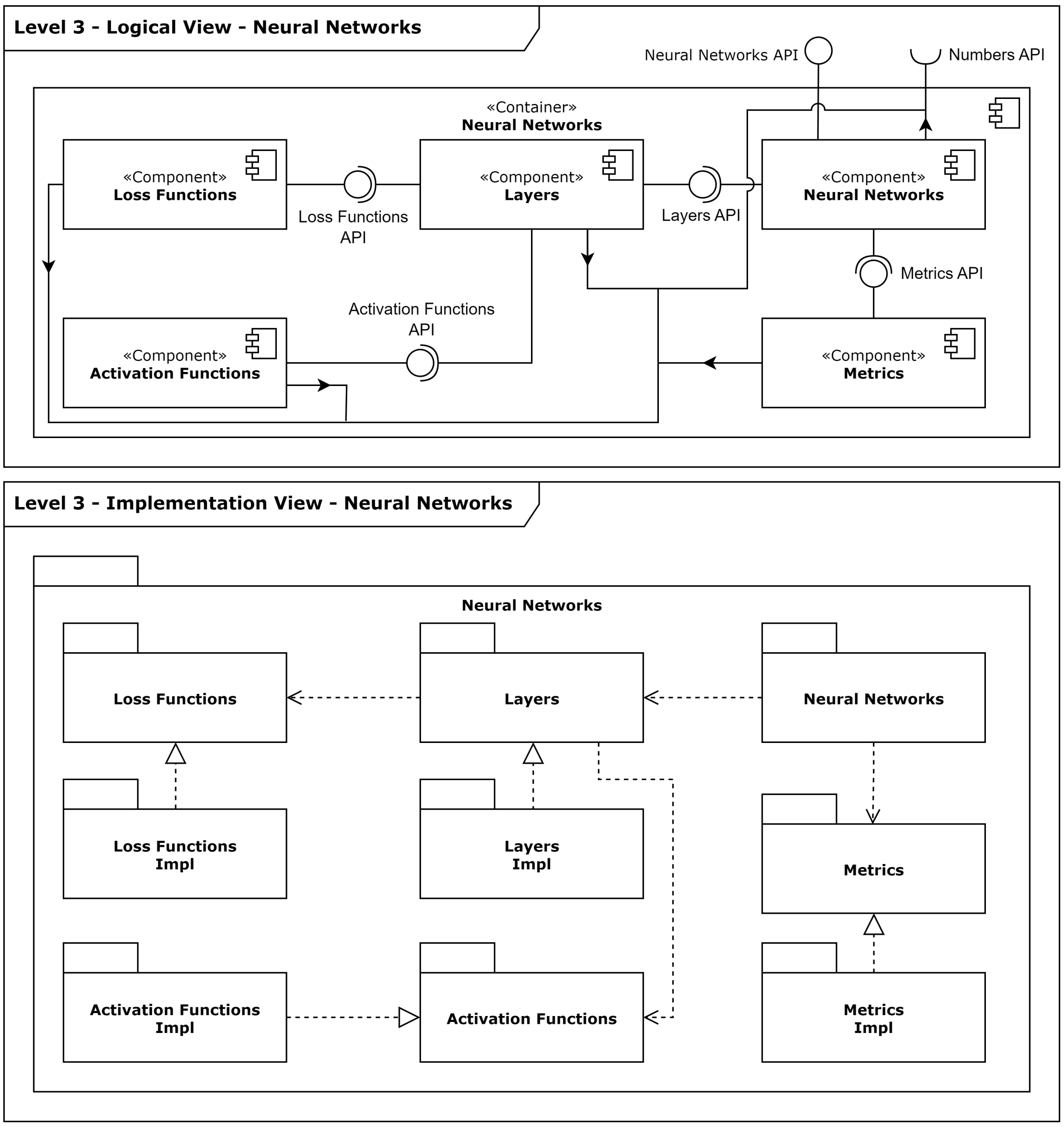

Figure 4.7: Level 3 - NeuralNetworks Container - Logical and Implementation Views (Black arrows indicate the direction of the flow and have no component relation meaning)

4.7.1 Logical View

Designing a NN library requires several different components to be created and linked together. In total, there were identified five different components, and they will now be enumerated and described:

- **Loss Functions** - LFs are an important part of every NN- They are responsible for the calculation of the error (loss) of the predicted output of the NN. There is more than one LF, so this component is responsible for the implementation of the required ones. In this thesis, only one was implemented, and it will be presented in the next chapter (Chapter 5).
- **Activation Functions** - AFs are used in every layer to define the output of a neuron. Similarly to the LF, there are several available alternative AFs, so this component is responsible for the implementation of the required ones. Only one will be used in this thesis, and it will be presented in the next chapter (Chapter 5).
- **Layers** - The NN layers are the main component of NNs. Their responsibilities are various, which include receiving the input from the previous layer/input of the model, calculating the weighted sum of those inputs, using the activation function, and, if training the NN, calculating the loss and subsequent weight updates. In this thesis, only the basic DNN layer type was used, but if more are supported, it is the responsibility of this component to implement them.
- **Metrics** - The role of this component is to provide a way to evaluate the model performance from different perspectives. These different perspectives can be analyzed using various metrics, but all return an overall score of the NN's ability to predict outputs correctly. As always, because there are several metrics, the implementation of these is the responsibility of this component. The ones used will be enumerated in Chapter 6.
- **Neural Networks** - This is the final component of this container, and it has several responsibilities that are broad because it needs to coordinate the usage of all the other components. First of all, this component must provide an interface for the creation of the model, which includes creating each layer, assigning AFs and LFs, and assigning the selected metrics. Then, in inference, it needs to inject the inputs to the first layer and return the outputs from the last layers. If in training, also inject the expected outputs in the last layer so that it can propagate the error backward and update the weights. Finally, at the end of the training, the metrics must also be calculated and returned to the caller.

The logic view of this container is depicted in Figure 4.7.

4.7.2 Implementation View

The implementation division of this container is based on the defined logic components. Still, the relation is, with the exception of the Neural Networks component, always 1 to 2, where the definition of the types and their implementation are separated in two different packages (the name of the last is equal to the first with the suffix "Impl"). The Neural Networks component only has one package because its implementation is unique, no matter what layers, LFs, AFs, and metrics are chosen.

The implementation view of this container is portrayed in Figure 4.7.

4.7.3 Process View

At this level of abstraction, it is possible to define the required interactions between the different components necessary to execute each process and better understand their relations.

In the model creation process, the Neural Networks component creates and adds each layer to the model and then creates and adds each metric. The first step requires the creation of the AF and LF instances to be used to create the layer instance, while the second only requires the creation of each metric instance. With this, the model can be created and returned to the user.

The model inference process is the process of feeding data into the first layer and obtaining the output on the final layer. Each step of the way, the weighted sum of the inputs is calculated and sent to the AF. After this recursive process is done, the result is returned.

The first part of the model training process is similar to the inference one. However, instead of returning the output of the final layer, it gives it to the LF together with the expected output. This kick-starts the backpropagation step, and these errors are then propagated backward until the first layer is reached. Finally, the metrics are calculated and returned to the caller.

These three processes are illustrated in Figure 4.8.

4.8 Level 3 - Aggregator Container

The final container of this library is the aggregator container, whose responsibility is to join the interfaces of the different containers in order to provide a unified interface that allows the creation and training of a PNN that can also be used to classify data. This abstracted view of this proposed solution allows internal modification to be isolated from the outside world, which allows modifications and upgrades in the future without requiring changes to the external API. No views will be provided for this container as, at this level of abstraction, no significant design aspect justifies their inclusion.

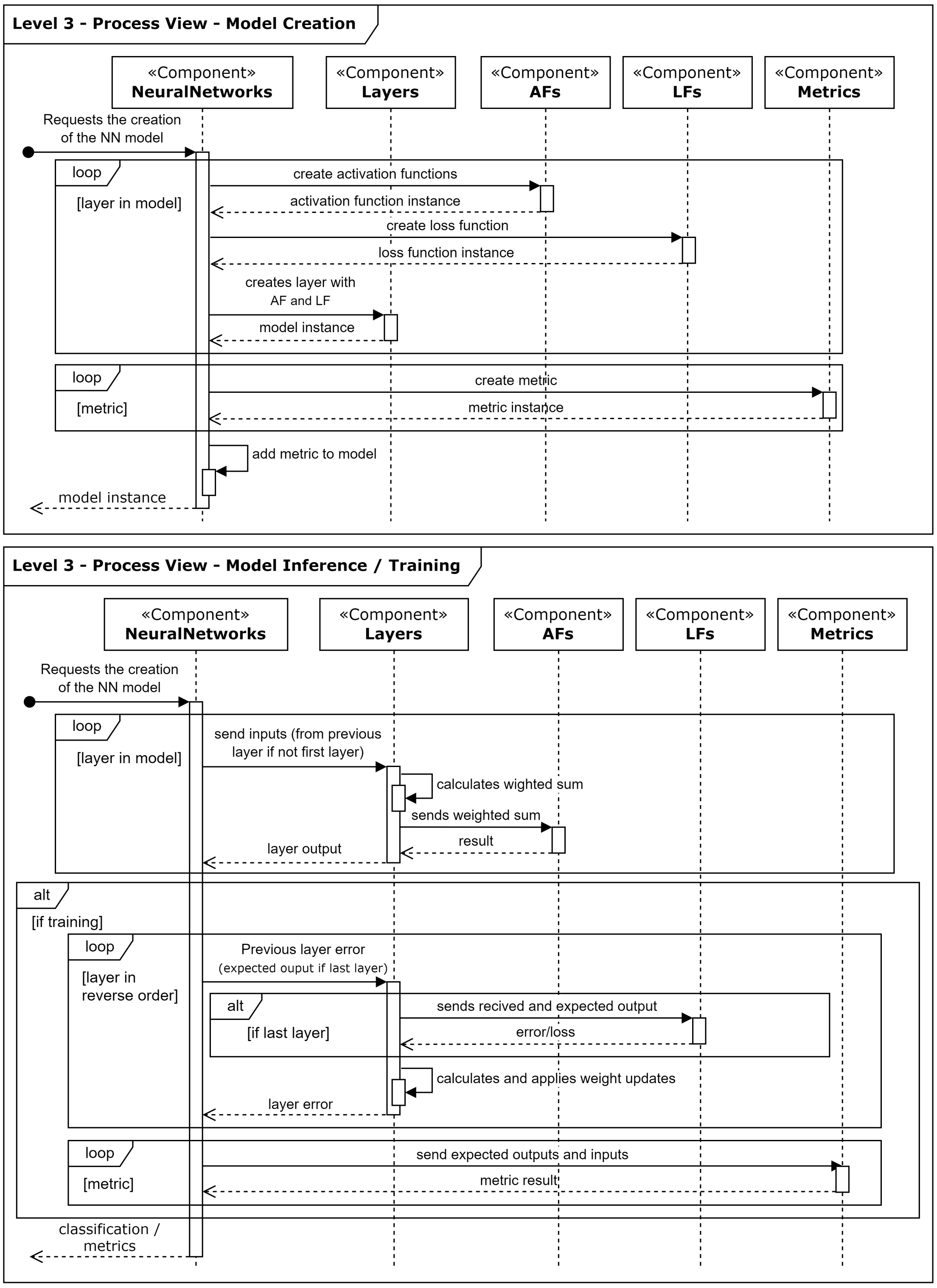
Level 3 - Process View - Model Creation
«Component» NeuralNetworks
«Component» Layers
«Component» AFs
«Component» LFs
«Component» Metrics
Requests the creation of the NN model
loop
[layer in model]
create activation functions
activation function instance
create loss function
loss function instance
creates layer with AF and LF
model instance
loop
[metric]
create metric
metric instance
add metric to model
model instance
Level 3 - Process View - Model Inference / Training
«Component» NeuralNetworks
«Component» Layers
«Component» AFs
«Component» LFs
«Component» Metrics
Requests the creation of the NN model
loop
[layer in model]
send inputs (from previous layer if not first layer)
calculates wighted sum
sends weighted sum
result
layer output
alt
[if training]
loop
[layer in reverse order]
Previous layer error (expected ouput if last layer)
alt
[if last layer]
sends recived and expected output
error/loss
calculates and applies weight updates
layer error
loop
[metric]
send expected outputs and inputs
metric result
classification / metrics

Figure 4.8: Level 3 - NeuralNetworks Container - Process View

Chapter 5

Private Neural Network Implementation

This chapter contains information about the implementation of the ENNigma PNN. First of all, the algorithms and optimizations of the implemented number operations will be presented. Then, the optimizations applied to matrix operation will be presented in the second section. After that, the implementations of the AFs and LFs are presented, with the respective selection process and, if any, optimizations. At the end, there is a section focused on explaining the Validation and Experimentation process to let the reader understand how the correct results are ensured. All of ENNigma was implemented in Rust and is publicly available[1].

5.1 Number Operations

The number representation introduced by the Numbers container must implement the basic number operations to be able to replace the hardware-implemented number representations. However, due to the fact that this new number representation must support encrypted bits, the algorithms for the number operations must be adapted. These adaptations can require significant modifications, as they can only use operations supported by the TFHE scheme.

One of the major limitations of making such algorithms is that it is not possible to branch based on the value of a bit, so if there are several branches, all of them must be calculated, and, using bit operations, only one of them can be chosen. Song et al. [18] proposed a series of algorithms considering this limitation and were able to prove that it is possible to implement several operations while meeting this criterion.

Next, each used algorithm will be presented, and if any, the optimizations applied will also be explained and justified. At the end, an analysis of the performance of the operations will be presented.

5.1.1 Addition Operation

The addition operation algorithm is constructed by applying a binary circuit for each bit of the number, from least significant to most significant. This circuit is made up of two XOR, two AND, and one OR gate. This circuit uses the bit of each number, *a* and *b*, and a carry bit, which means that the output of the *nth* bit is influenced by the $n-1$ bit, and, in this

[1] Avaliable at https://github.com/PedroMBB/ENNigma

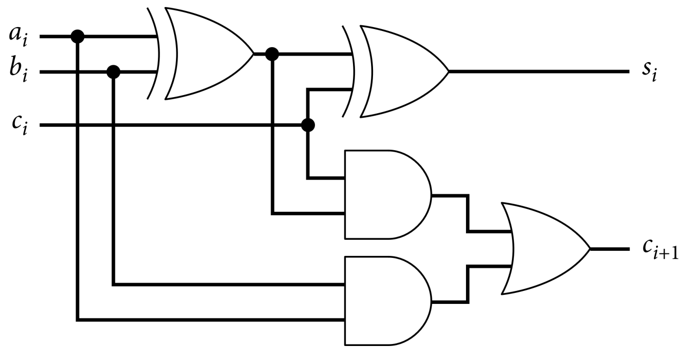

Figure 5.1: Addition operation logic circuit [18]

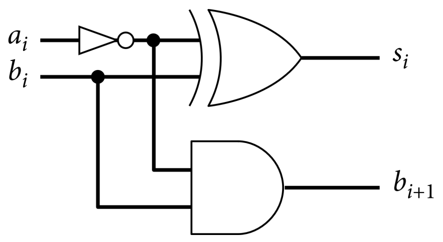

Figure 5.2: 2's complement operation logic circuit [18]

case, its value is initialized with the representation of the binary value "false". The circuit is illustrated in Figure 5.1.

This algorithm was slightly altered so that the last circuit does not calculate the carry bit, as it will be discarded. This change can seem insignificant, but a millisecond reduction can be noticeable in a highly complex system, like NN, due to the high amount of additions.

5.1.2 2's Complement Operation

The algorithm to get the 2's complement of a number (get it's symmetric) is also based on the repeated execution of a binary circuit. However, this time, because the operation is performed over a single number, there are only two input bits: the bit of the number and the carry bit. In this algorithm, the carry bit is also initialized with the representation of the value False. The circuit is depicted in Figure 5.2.

Similarly to the addition operation, the last circuit was also altered to remove the calculation of the last carry bit.

5.1.3 Subtraction Operation

The subtraction operation can be represented as the addition of a number with the complement of the second. However, implementing the subtraction operation directly can reduce the number of operations per bit pair from 8 to 7 while also avoiding iterating over the bits more than once. As such, the subtraction algorithm was implemented using two XOR, two AND, one OR, and two NOT gates, with the same repeating strategy as the previous operations. Similarly, the carry bit is initialized with the representation of the value False. The circuit is portrayed in Figure 5.3.

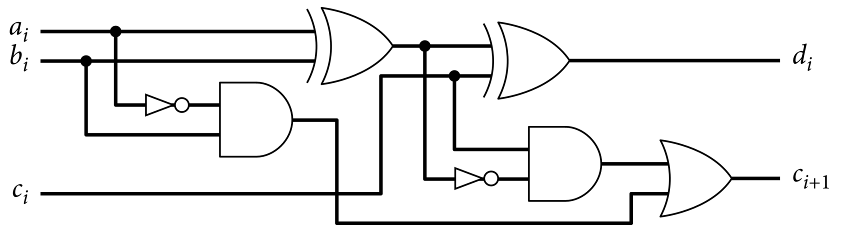

Figure 5.3: Subtraction operation logic circuit [18]

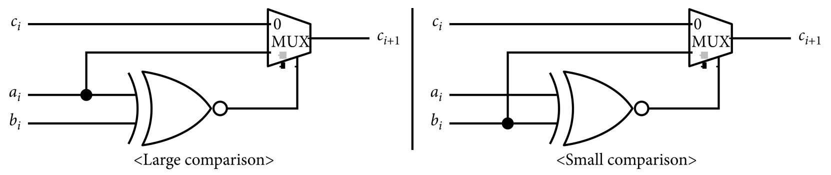

Figure 5.4: Comparison operation logic circuits [18]

As usual, the last circuit was altered to remove the calculation of the carry bit. However, in this case, the impact is more significant because it results in the reduction of the number of gates on the last circuit from seven to two.

5.1.4 Equivalent, Large, and Small Comparison Operations

The equivalent operation algorithm is the result of the AND gate between the results of the application of the XNOR gate between each bit pair. The output will be a single bit that will contain the binary value that will indicate if the two numbers are equal or not represented by the values True and False, respectively.

The large and small comparisons are more complex and require the application of the respective circuit for each bit pair of the numbers, with the result being returned in the carry bit of the last circuit. As usual, the execution of this circuit flows from least significant to most significant bits, and the carry bit is initialized as False. To make this operation work with negative numbers, the final result must be the XOR of the last carry bit with the result of the XOR of the signal bits. Considering the numbers A and B, their bits, respectively, a_i and b_i, and the carry bit c_i, the comparison operation circuits are as shown in Figure 5.4.

5.1.5 Shift Operations

The shift algorithms to perform shift operations do not require the execution of any binary gate since they only require the addition of bits on one end of the number and the removal of that same amount on the other end. The side from which bits are inserted and removed is defined by the direction of the shift, with the left shift requiring the addition of the bits on the side of the less significant bit, while on the right shift, the bits are inserted on the opposite side.

5.1.6 Absolute Value Operation

The absolute operation can be implemented using a simple if-then-else algorithm using the MUX gate. Using the signal bit, the result is either its own number or its 2's complement. Due to the fact that the signal is encrypted, it is not possible to choose whether it is negative

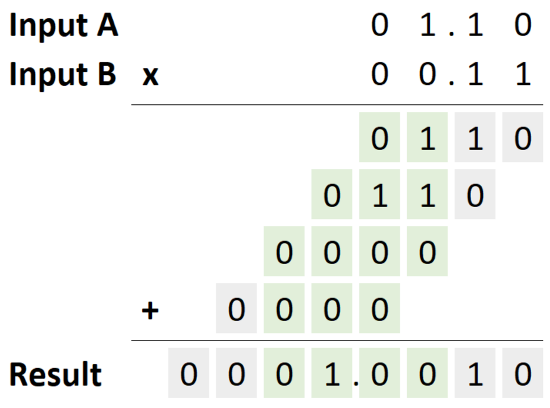

Figure 5.5: Unisgned multiplication of two unsigned four-bit numbers with two bits of precision (Green - Calculated values, Gray - Discarted values)

or not. So, it t is necessary to calculate both and let the MUX gate homomorphically choose the correct one.

5.1.7 Multiplication Operation

The final operation that will be detailed is multiplication. This operation is the most complex one because, besides the unsigned multiplication, it includes two Absolute, one 2's complement, and a MUX operation. The absolute and 2's complement operations have already been defined, and the MUX one is the application of the bit operation "MUX" on each of the number's bits. The objective of the MUX operation is to adjust the final value signal according to the signals of the original values. That is, make the result positive if the input values were both negative or both positive and negative otherwise.

The unsigned binary multiplication is the new component that needs to be implemented, and it is done by using the long multiplication method. This method indicates that each digit of the second number must be multiplied by the first number, and the final result is the sum of each of these results.

Given that the absolute and two complement operations are already optimized and that the MUX operation is a single call for every bit, the unsigned multiplication is the only place where optimizations can be applied. Usually, the number of bits in the result of a multiplication operation is the sum of the sizes of the input values. In other words, the sum of two N-bit numbers results in a 2N-bit number. Given that, in the ENNigma library, the number of bits used to represent numbers is pre-defined, it is required that the result of the multiplication has the same number of bits as the input values, so if the input values have n, n bits must be discarded. Seeing this, an optimization was applied where the calculation of those N bits is not performed, saving several computational times. An example of this optimization for two unsigned four-bit numbers with two precision bits is illustrated in Figure 5.5.

5.1.8 Memory usage and allocation

Allocation memory is a computationally expensive operation, and if considering the usage of HE, it is even more expensive due to the bigger memory requirements. Seeing this, and to optimize all operations, all memory required is, if possible, allocated at once and, if not,

	8 bits	16 bits	32 bits	64 bits
SHL	2.09E-06	9.79E-06	8.74E-06	3.12E-05
SHR	5.83E-06	9.97E-06	1.75E-05	4.51E-05
NEG	1.13E-01	2.60E-01	5.35E-01	1.10E+00
GRE	2.14E-01	4.24E-01	8.34E-01	1.65E+00
GRT	2.14E-01	4.21E-01	8.29E-01	1.67E+00
ABS	2.41E-01	5.23E-01	1.08E+00	2.17E+00
SUB	3.03E-01	6.51E-01	1.36E+00	2.76E+00
SUM	3.05E-01	6.51E-01	1.37E+00	2.75E+00
MUL	2.36E+00	9.81E+00	4.01E+01	1.62E+02

Table 5.1: Ciphertext operation duration (in seconds) with different quantity of bits using TFHE-rs (rows divided into three groups: constant, linear, and exponential bit-wise operations count)

in the least number of allocations possible. To further reduce memory usage, it was also defined that, if possible, the reuse of already allocated memory (but no longer used) should be preferred to reallocating new memory.

5.1.9 Operations Performance

As explained in Chapter 2, the computation of the various NN steps is the execution of well-defined mathematical operations. Due to this fact, the computational time of the ENNigma is directly related to the computational cost of each individual operation. So, it is essential to examine the computational cost of the operations using the defined algorithms and compare it to the Stat-of-the-Art performance.

Due to the fact that the vast majority of the algorithms perform a circuit for each bit of the input numbers, the final computational complexity is directly related to those. However, reducing the number of bits reduces the representable number space. The tradeoff between the number of bits and computational time must be studied.

As shown in Table 5.1, doubling the bits increases, approximately, the computation time by 100% for operations like addition and 300% for multiplication. The difference in computational time gains is higher in multiplication compared with other operations because, in it, the number of bit-wise operations increases exponentially with the number of bits. This relation between bit count and duration is illustrated in Figure 5.6, Figure 5.7, and Figure 5.8 for constant, linear, and exponential bit-wise operations count, respectively.

If the bit operation performance is compared, it is possible to see that the TFHE-rs library does add a significant performance advantage when compared to the original TFHE library (Table 5.2). However, the $\sim$ 45% improvement in performance is only part of the 51% speed increase in 8 bits and 59% increase in 32 bits (Figure 5.9), with the remaining performance being acquired through the optimizations applied.

5.2 Matrix Operations

In NN computations, matrix operations can be seen as a way to compile multiple individual number operations at once. That is, instead of manually summing each element individually,

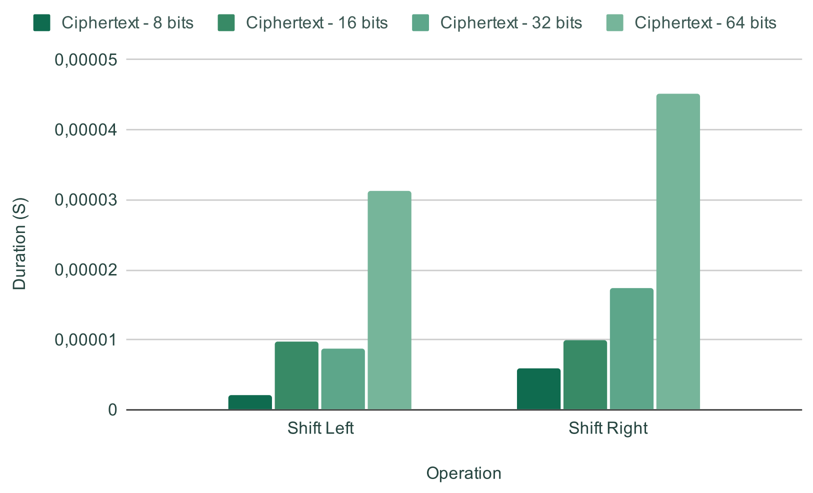

Figure 5.6: Constant bit-wise operation count operation duration (in seconds) for different bit quantities using TFHE-rs

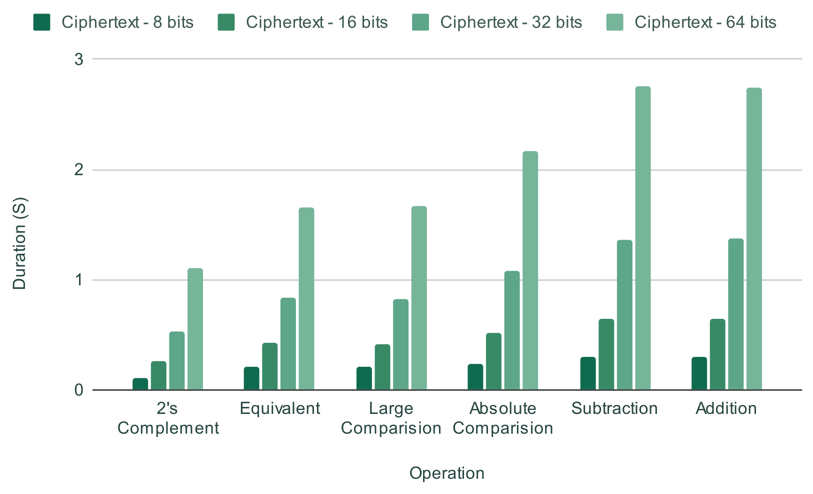

Figure 5.7: Linear bit-wise operation count operation duration (in seconds) for different bit quantities using TFHE-rs

	Plaintext	**TFHE-rs**	**TFHE**
NOT	3.30E-08	1.99E-04	1.62E-07
NOR	3.15E-08	6.43E-03	1.19E-02
XNOR	3.65E-08	6.43E-03	1.19E-02
NAND	3.32E-08	6.44E-03	1.19E-02
XOR	3.03E-08	6.46E-03	1.19E-02
OR	3.15E-08	6.48E-03	1.19E-02
AND	3.58E-08	6.59E-03	1.19E-02
MUX	3.35E-08	9.37E-03	2.24E-02

Table 5.2: Duration (in seconds) of the boolean gate operations in plaintext, using the TFHE-rs library and using the original TFHE library

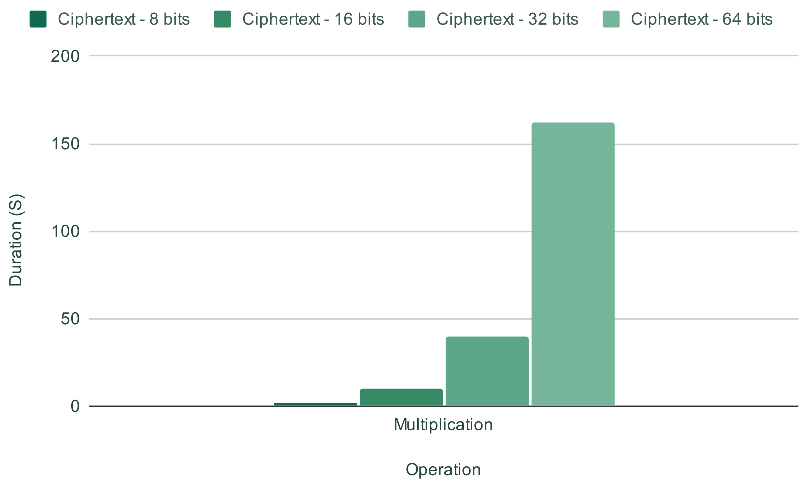

Figure 5.8: Exponential bit-wise operation count operation duration (in seconds) for different bit quantities using TFHE-rs

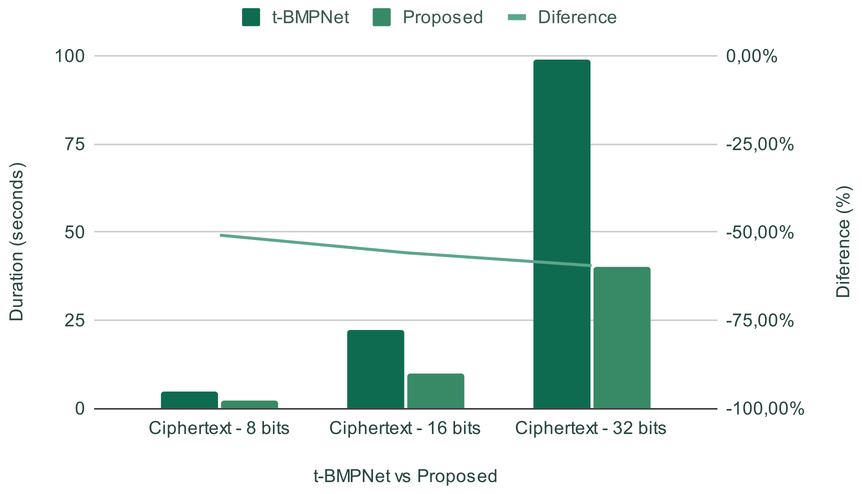

Figure 5.9: Comparison of the multiplication operation time duration (in seconds) between t-BMPNet and the proposed solution

it is possible to put them in two matrices and sum the matrices instead. This combination of operations can bring advantages, but their gains need to be studied on a use-case basis.

These computational time gains can be attributed to some specific attributes. One of them is the better memory layout and management (Section 5.2.1), while the other is the specific optimization for some of the operations, namely, matrix multiplication (Section 5.2.2). These two will be explained next.

5.2.1 Memory Allocation and Layout

Well-structured RAM usage allows applications to take advantage of various memory optimization strategies. So, representing numbers in matrix form can be one way to achieve that well-structured layout. This is because well-organized memory allows more efficient use of the cache, and matrix-arranged numbers allow specific matrix optimization algorithms to be used.

Due to this fact, the operations inside the layers in both forward and backward propagation are done using matrices.

5.2.2 Matrix Multiplication

Several element-wise matrix operations are harder to optimize due to the fact that they use each element only once. However, operations such as matrix multiplication are not element-wise and often require redundant operation and the usage of a single element more than once, which means that there is more room for improvement.

A simple improvement is to see if there is a way to share computations. For example, as explained in the previous section, the defined multiplication operation requires the absolute value of the number to be computed prior to the main multiplication step. Consequently, this means that there is an opportunity to optimize the matrix multiplication algorithm by calculating the absolute value of a number just once and reusing that calculation for the remaining uses of that specific number.

Besides that specific optimization, some algorithms leverage the numeric redundancy of matrix multiplication to replace some multiplications with cheaper sum operations. Nevertheless, these algorithms carry an immense upfront cost, making them only advantageous for very big matrices. In the next subsection, one such algorithm (the Strassen Matrix Multiplication Algorithm) will be analyzed and compared to traditional matrix multiplication.

5.2.3 Strassen Matrix Multiplication Algorithm

Traditional matrix multiplication algorithms require n^3 multiplication for the multiplication of two matrices of size n. In contrast, algorithms like the Strassen Matrix multiplication algorithm only require approximately $n^{2.8074}$ multiplication operations, with the remaining being replaced by the significantly cheaper sum operation. For example, in a multiplication of two $2x2$ matrices, the number of multiplication reduces from eight to seven.

Even though the Strassen algorithm is only defined for matrices of size $2x2$, using a divide-and-conquer approach, it is possible to adapt this algorithm to any square matrix whose size is a power of two. Using padding to make matrices square and of a size that is a power of two means that even non-square matrices can be used with this algorithm.

When analyzing the approximate size of the matrices of the SotA solutions to the trainable NN with HE, it is possible to observe a consistent pattern: the number of neurons per layer is always very small. This fact makes the usage of matrix multiplication algorithms not a guaranteed advantage. To exemplify, tests showed that multiplication of a 7x6 matrix with a 6x7 matrix utilizing ciphertext takes 1520.18 seconds using the traditional multiplication and 1996.36 seconds using the Strassen algorithms. This results in a 31% increase in computational time for the Strassen algorithm.

Given the small dimensions of the matrices used in PNN and the Strassen algorithm's inherent static computational overhead, the observed difference is not significant. Nonetheless, this underscores the critical point that while the Strassen algorithm has proven merits in certain scenarios, its application in the context does not offer a performance benefit.

5.3 Activation Functions

As seen in the SotA review, one recurring problem when using HE with NN is the computation of AFs. This occurs because most AFs require the computation of non-linear operations, such as exponential and division. Yoo and Yoon [108] claim to have implemented all the operations required to compute the sigmoid, but, as will be explained next, it was not possible to replicate their approach. Consequently, the alternative was to explore the use of an approximated version. In particular, the possibility of using the ReLU as an AF was also analyzed, and its limitations for this use case will also be presented.

5.3.1 Non-approximated Sigmoid

As previously mentioned, the sigmoid is composed of four operations: addition, subtraction, division, and exponentiation. The work by Song et al. [18] proposes algorithms for the first three, but it does not provide an implementation for the last one. That implementation is provided in the work of Yoo and Yoon [108]. However, when trying to replicate the algorithm, the results obtained do not match the expected output of an exponential function. Also, even though their code is publicly available, our attempt to use their implementation of the exponential function was unsuccessful because the results obtained when evaluating that function were incorrect.

5.3.2 Approximated Sigmoid

As seen in Section 3.2.2, there are several proposed approximations of the sigmoid in the literature, and they can be divided into two main groups: polynomial approximations and piece-wise linear approximations. Choosing the correct type depends on the use case.

For this particular use case, obtaining the values using the table lookup method was set aside primarily because of its issues with precision. So, in the approximation landscape, the piece-wise linear approximation was chosen due to its higher NN performance when compared to Taylor series approximation [112], particularly when considering input values far from the origin. Furthermore, the Taylor series also often requires high degrees of polynomials to achieve acceptable results, which exacerbates its already high computational cost.

The approximation from Myers and Hutchinson [19], provided on the left side of (5.1), is one of the best performing sigmoid approximations [112], and its reliance on power-of-2 coefficients means that it can be optimized by replacing multiplications with shifts in base 2. The transformation of the power-of-2 coefficients is illustrated on the right side of (5.1). Due to the fact that the right shift operation is significantly faster than the multiplication, this change results in a significant performance increase, as can be seen in Table 5.3. These decreases in computational time are significant, with gains reaching 60% for the base function when using 64-bit numbers and reaching 70% for the combined calculation of the base function and derivative in 16-bit numbers.

$$\begin{cases} 1 & x > 0 \\ 0.015625x + 0.875 & 4 < x \leq 8 \\ 0.03125x + 0.8125 & 2 < x \leq 4 \\ 0.125x + 0.625 & 1 < x \leq 2 \\ 0.25x + 0.5 & -1 < x \leq 1 \\ 0.125x + 0.375 & -2 < x \leq -1 \\ 0.03125x + 0.1875 & -4 < x \leq -2 \\ 0.015625x + 0.125 & -8 < x \leq -4 \\ 0 & x \leq -8 \end{cases} = \begin{cases} 1 & x > 0 \\ x \gg 6 + 0.875 & 4 < x \leq 8 \\ x \gg 5 + 0.8125 & 2 < x \leq 4 \\ x \gg 3 + 0.625 & 1 < x \leq 2 \\ x \gg 2 + 0.5 & -1 < x \leq 1 \\ x \gg 3 + 0.375 & -2 < x \leq -1 \\ x \gg 5 + 0.1875 & -4 < x \leq -2 \\ x \gg 6 + 0.125 & -8 < x \leq -4 \\ 0 & x \leq -8 \end{cases} \tag{5.1}$$

5.3.3 ReLU

The ReLU AF, in recent years, has been the subject of substantial growth, with it now being more employed than the sigmoid AF in various applications. Despite this popularity, its characteristics make it not ideal for the proposed PNN because of its unbounded nature, which means that its output is not normalized. This is a significant barrier since, when using the chosen fixed-point number representation, it leads to frequent overflow of the representable number space, which, in turn, makes the NN unusable.

Its well-known learning performance, coupled with its minimal computational overhead, would have made it an excellent candidate for these NNs. Yet, the challenges it poses in terms of representation make it a less-than-ideal choice for this specific neural network configuration.

On the other hand, functions like the sigmoid AF intrinsically prevent this problem, ensuring that the output remains confined within a representable space. Given that, the Sigmoid AF will be used instead of the ReLU AF in this PNN.

	Base Function			**Base Function & Derivative**		
	Original	**Optimized**	**%**	**Original**	**Optimized**	**%**
8 bits	4.0714	2.6885	**-33.96%**	5.8844	2.7366	**-53.49%**
16 bits	11.8379	5.0462	**-57.37%**	17.7536	5.2502	**-70.43%**
32 bits	24.3357	9.8677	**-59.45%**	46.9998	24.9574	**-46.90%**
64 bits	48.9663	19.2550	**-60.68%**	133.7095	75.7095	**-43.38%**

Table 5.3: Comparison of the number of seconds taken to calculate the Myers Sigmoid approximation and its derivative in ciphertext using numbers represented in different amounts of bits

5.4 Loss Functions

The computation of the LF is not as commonly referred to in the literature as the computation of the AF. However, there are still some cases where the use of HE does limit the ability to calculate the LF. As will be possible to see next, from the two commonly used LFs, only one is fully supported by the TFHE scheme.

5.4.1 Mean-Squared Error

As already mentioned, the MSE is one of the most commonly used LF in regression and single-class classification NNs. One of the primary advantages of using MSE is its simplicity in terms of computational complexity. The function's focus on square differences emphasizes larger errors over smaller ones, which can be advantageous when significant errors are more impactful. However, this LF has some limitations due to the fact that it squares the errors, which can make a single large outlier disproportionately impact the loss value, potentially leading to a wrongful representation of the inference performance. Furthermore, MSE assumes a constant variance across all levels of the independent variable. This assumption may not hold true for all types of data, thereby affecting the accuracy of the model.

As explained in Chapter 2, the optimizer does not care about the loss value itself but instead about the direction and intensity needed to minimize the error, which is the derivative. This means that to use this LF, it is only required to calculate the derivative of the actual LF, which is given by the function presented in (5.2). Given that calculating this derivative only requires a simple subtraction and a multiplication/shift, no adaptations are needed for its usage with HE.

$$MSE' = -2 * (exp - rec) = 2^1 * (rec - exp) \tag{5.2}$$

5.4.2 Cross-Entropy

As already mentioned, the use of Cross-Entropy loss for multi-class classification over MSE is essential to allow the model to be able to achieve better performance. This difference is due to the fact that MSE evaluates the average squared difference between actual and predicted values, which means it tends to treat errors uniformly. This approach can be a problem in a multi-class classification where each output node corresponds to a class. The MSE may not provide a sufficient penalty for incorrect confident predictions across the various classes. Cross-entropy, with its logarithmic component, addresses this by magnifying the penalties for wrong high-confidence predictions, making the model's learning more capable of identifying errors in the classifications.

Regardless, the distinction between the two becomes less pronounced when we narrow it down to binary or single-class classification. In these scenarios, there is only one output node; therefore, both LFs essentially penalize the deviation of this prediction from the ground truth. Even so, Cross-Entropy possesses a slight advantage due to its ability to amplify the penalty for highly confident, yet wrong, predictions.

Besides this, its requirement for the computation of logarithms is a serious barrier to its usage, as none of the FHE schemes used in the current literature for data privacy in NN support it. One alternative would be the use of approximations, namely, table lookups. This will not be explored in this thesis because all the NN models presented in Chapter 6 are single-class classification PNNs.

5.5 Validation and Experimentation

Guaranteeing that the implementation of the PNN is correct is paramount to ensure that its results can be trusted. This section presents the processes used to validate each part of the implemented PNN.

5.5.1 Number and Matrix Operations

Number and matrix operations are the base of all NNs, which means that an error at this level will be propagated throughout the system and eventually will end up affecting the results. Fortunately, a solution for this problem exists, which consists of leveraging the tried-and-tested numeric operations implemented by the programming environment to ensure the correct output of these computations.

To adopt this approach, a testing strategy was developed where tests were programmed to compare the output of the custom operations with the ones produced by the system's native arithmetic operations for known input values. Given that system computations might utilize a higher number of bits, introducing a tolerance threshold became necessary. This threshold ensured that the comparisons were not affected by precision issues, eliminating false errors.

5.5.2 Approximation Functions

Following the number and matrix operations, the correct implementation of approximation functions is also essential to ensure the correct behavior of the NN. A deviation in implementation can introduce errors that, when accumulated, could potentially threaten the correctness of the NN's output. To counteract these potential pitfalls, the original function is used as a ground truth to compare outputs.

In this process, the results produced by the implemented approximation functions were compared with those obtained from the original function for a set of predetermined input values. Given that these are approximation functions, a tolerance threshold was set in place to ensure that small variations, which are inevitable when approximating, were factored into the comparison.

5.5.3 Neural Network inference

The usage of NN, independent of the task being performed, always utilized the forward propagation step. To test this step, the well-known and well-tested framework TensorFlow[2] was used because, even though this NN is focused on privacy, their mathematical foundation is the same.

The validation procedure is made of several steps. Initially, a DNN model is created and trained using TensorFlow, and when the training is done, its weights and biases are extracted. Using those extracted weights, a DNN model matching, as closely as possible, the TensorFlow one must be created, and each neuron must have its respective weights and biases assigned. The validation procedure itself is done by feeding both models with the

[2]https://www.tensorflow.org/

same input data and subsequently comparing their outputs. It is important to mention that if an approximation function is used on the private NN, the comparison of the output must have a threshold proportional to the approximation's error. However, if AFs that do not require approximation are utilized, the output must be nearly identical, accounting only for small discrepancies due to precision differences.

5.5.4 Neural Network training

Contrary to the inference step, and due to the higher complexity of the backpropagation phase, the approach to test the training capability of the NN was different. Notably, instead of transferring a trained model from TensorFlow, this library was used to understand the ability of a specific NN structure to learn a predetermined problem. To validate the training performance, metrics obtained after *n* epochs were compared and analyzed in such a way that errors from approximations and precision were not counted as implementation errors.

This testing was done using different dataset sizes. The smallest one is one containing the pairs of Boolean inputs and its XOR outputs. The second one is a binary classification of banknotes using the Bank Note Authentication dataset [113], which is made up of four features. After those, more complex datasets were used, namely, the one used in the following chapter.

5.6 Discussion

Even though a practical application of this PNN will only be presented in the next Chapter (Chapter 6), it is possible to analyze the presented PNN implementation and draw some conclusions.

Starting with the numeric operations, it is possible to see that there were significant performance improvements. When compared to the work of Yoo and Yoon [108], up to 59% computational speed reduction was registered on the individual number operations, with approximately 45% being achieved by using a new optimized TFHE library and the remaining being provided by several optimizations applied at different stages of the number operation algorithms. Unfortunately, it was not possible to replicate the exponential implementation presented by Yoo and Yoon [108], but an alternative approximation function to the Sigmoid was used instead. The computational cost of this approximation was also significantly reduced by replacing the multiplication with the much cheaper shift operations.

Moving on to AFs, the sigmoid itself could not be implemented, but an optimization was used instead. This approximation was further optimized, which resulted in a 60% reduction in computational time when compared to the non-optimized one. The usage of the ReLU was also considered, but its unbounded nature made it not suitable for this fixed-point-based PNN implementation.

Similar to the AFs, only one LF was implemented. In this case, this happened because, from the two most common LFs, only one is supported. Although the non-supported one could possibly be approximated, the use case does not justify it.

Chapter 6

DDoS Attack Detection Model

With the PNN designed and implemented, this chapter will detail the process of creating an NN model for DDoS attack detection and present its results. To do this, firstly, the dataset used will be presented and detailed. Then, the pre-processing steps that were executed to transform the used data into an NN-friendly format are presented. The NN structure, including the number of layers, number of neurons, AFs, LF, and others, will then be defined. In the end, there are two final sections: one presents the results of the experiments executed, and the other discusses them.

6.1 Dataset

The CIC-DDoS2019 dataset is one of the most recent entries in a family of IDS datasets published by the University of New Brunswick, Canada[1]. It is provided in two forms, where one is the raw network traffic data, and the other is composed of the 80 features extracted from that network traffic data using a tool developed by them called CICFlowMeter. This dataset contains a total of 14 different DDoS attacks as well as benign network traffic.

Considering that the goal is to make a model capable of detecting DDoS attacks, the chosen dataset was the CIC-DDoS2019 dataset. This is because, besides the fact that it is recent and from a well-known organization, it is also the dataset of choice of the most recent publications (as seen in Section 3.1.2). This is important to ensure that the comparisons with the SotA are as fair as possible.

6.1.1 Data Collection

Correctly collecting data for a dataset is one of the most important steps to ensure that the resulting dataset is usable and clear of any bias. Besides that, given the DDoS attack dataset, there is also the need to generate benign data, which can be as hard, if not more complex, than collecting attack data. Given this, Sharafaldin et al. [92], in order to generate the utmost realistic dataset possible, used a strategy named B-Profile system to generate lifelike benign background traffic for 25 simulated users for HTTP, HTTPS, FTP, SSH, and email protocols.

The scenario where the dataset was collected was formed by one server running Ubuntu, four personal computers running Windows, and a Fortinet firewall. The dataset was divided into two days, each with a slightly different network configuration and attacks. For each day, network traffic and event logs were recorded for each machine. As already mentioned, the

[1]https://www.unb.ca/

final dataset features were extracted from this raw network traffic using the CICFlowMeter tool.

6.1.2 Attacks

In the CIC-DDoS2019 dataset, various modern DDoS attacks are represented, including PortMap, NetBIOS, LDAP, MSSQL, UDP, UDP-Lag, SYN, NTP, DNS, and SNMP [92]. These attacks were launched during one of the two days of the dataset. In total, the first day contains seven different attacks: PortScan, NetBIOS, LDAP, MSSQL, UDP, UDP-Lag, and SYN, while the second day includes twelve different DDoS attacks: NTP, DNS, LDAP, MSSQL, NetBIOS, SNMP, SSDP, UDP, UDP-Lag, WebDDoS, SYN, and TFTP.

These attacks can be divided into two main categories: reflection attacks and exploitation attacks. In the first, the attacker sends a flood of packets to various servers, impersonating the IP address of the target system. These servers, assuming the requests are legitimate, then send responses to the target system. Because these servers typically send a much larger volume of data in response to the initial request, the target is quickly overwhelmed by a high volume of traffic that it did not request. This amplification effect enables the attacker to magnify the impact of the attack by using innocent third-party servers. It is essentially a way to bounce traffic off other servers to hide the attacker's identity and amplify the amount of traffic directed at the victim.

On the other hand, exploitation-based DDoS attacks rely on vulnerabilities within the target system itself. In these types of attacks, the attacker exploits weak points or bugs in the system's software to initiate a service outage. Rather than overwhelming the system with external traffic, as is the case with reflection-based DDoS attacks, exploitation-based DDoS attacks disable the system by taking advantage of internal flaws.

Although both types of attacks present significant challenges for defenders, reflection-based attacks can be tough to trace, given that the traffic appears to come from legitimate servers. Nevertheless, the dataset contains both types, which allows their users to choose to focus on one type, one both, or even on individual types of attacks.

6.1.3 Features

As already said, this dataset provides its data in two different formats: one is the raw captured network traffic from where features must be extracted or used directly in NN, and the other is a large set of features extracted from that raw network traffic using the CICFlowMeter tool. To train this model, the features format of the dataset will be used.

In total, 80 features are provided, and they vary from basic details like IP addresses, port numbers, and protocols to more insights like session statistics and rate metrics. This comprehensive set of features allows it to be used to build refined and potentially more accurate NN models. However, a suitable pre-processing stage is essential to avoid overwhelming the model with features.

6.1.4 Known dataset flaws

Even with good data collection practices, the datasets are not immune to errors or biases, so it is essential to know these flaws and, if possible, use versions of the dataset with these flaws rectified or mitigated.

In the case of the CIC-DDoS2019 dataset, the known possible flaw lies in its use of the CICFlowMeter tool which has known errors, which may lead to some faults in the features generated. However, unlike the CICIDS2017 dataset, which has a corrected version provided by Engelen, Rimmer, and Joosen [114], no rectified version of the CIC-DDoS2019 dataset has been provided so far. As such, the CIC-DDoS2019 dataset will be used in this work as-is.

6.2 Pre-processing

To maximize the performance of any NN model, it is essential to select and pre-process the data correctly. The proposed model is no different, so it is necessary to choose the most relevant features and ensure that all of them are pre-processed and ready to be fed into the NN. Even with the most relevant features selected, they may not be immediately suitable for efficient NN training and inference. For that, they usually need to be numeric and normalized. To solve this problem, it is necessary to use specific strategies to transform them into an equivalent optimal form.

Three different NN structures will be used to be able to compare the performance of the proposed PNN in different scenarios. The first two are based on the work of Sbai and boukhari [93], so their objective will be to identify only UDP DDoS attacks. In contrast, the third will be based on the attack detection NN model proposed by Cil, Yildiz, and Buldu [96], which means that its objective will be to identify all types of attacks present in the dataset. Next, the preprocessing of the dataset for each of these two categories of NN models will be presented.

6.2.1 UDP DDoS Attack Detection

This dataset provides several features, but it is essential only to select the relevant ones. After analyzing the problem at hand, and given that the basis of the NN model will be the work of Sbai and boukhari [93], it is possible to conclude that the objective is to detect only UDP attacks. With the objective demarcated, it is now possible to choose the relevant features to this specific attack. To select these features, it is essential to analyze the relation between each input feature and the target attack type. Fortunately, the authors of the datasets provided, together with the dataset itself, this exact analysis. They concluded that, for the UDP attacks, the most important features were: "Destination Port", "Fwd Packet length Std", "Packet Length Std", "min_seg_size_forward", and "Protocol". So, similar to [93], these were the features selected to train this NN.

Categorical features can often be pictured as labels, which can sometimes not have direct numeric meaning. While size labels such as "Big", "Medium", and "Small" could be easily replaced by numeric values, others, such as city labels, cannot. There are some possible methods to transform the features without clear numeric relation, and the most common ones are One-Hot-Encoding and Binary Encoding. One-Hot-Encoding creates a feature for each label, where its value is a Boolean value representing whether the sample is of that category or not. On the other hand, binary encoding creates a single feature where each bit of the number represents a single label, where its value defines whether the sample is of that category or not. While the first creates a clear separation between labels, the second can offer performance benefits due to the fact that it does not increase the number of features.

Among the features selected, only the Protocol feature is categorical, and it is composed of three classes. Despite the computational complexity imposed by the use of HE in the proposed PNN, the One-Hot-Encoding strategy was still used for this feature. This decision was made based on two factors: firstly, given that there are only three classes in this feature, the difference in feature count is minimal - from five to seven features. Secondly, using One-Hot-Encoding facilitates comparison with the work of Sbai and boukhari [93] as this model will be similar to theirs. Furthermore, the usage of One-Hot-Encoding has already proven to be successful in that same work.

The normalization of the features is essential to ensure that the NN works within the numeric representation bounds and to avoid vanishing and exploding gradients due to large absolute numbers. To normalize the features, the min-max method was used. This method can be calculated for each entry i of all feature values F using (6.1). Given that the features generated from the application of One-Hot-Encoding on the Protocol feature are already between 0 and 1, applying this normalization procedure to them is unnecessary.

$$Output_i = \frac{F_i - min(F)}{max(F) - min(F)} \tag{6.1}$$

6.2.2 Generic DDoS Attack Detection

While the objectives are different, the preprocessing step is similar between the two categories of NNs. The pre-processing of this dataset sample was inspired by the one performed by Cil, Yildiz, and Buldu [96]. From the features provided by the dataset, the “Flow ID”, “SourceIP”, “SourcePort”, “DestinationIP”, “DestinationPort”, “Protocol”, “Timestamp”, and “SimilarHTTP” were discarded due to their lack of contribution to the final model [96]. The features “Bwd PSH Flags”, “Fwd URG Flags”, “Bwd URG Flags”, “Fwd Bytes/Bulk Avg”, “Fwd Packet/Bulk Avg”, “Fwd Bulk Rate Avg”, “Bwd Bytes/Bulk Avg”, “Bwd Packet/Bulk Avg”, and “Bwd Bulk Rate Avg” were removed because they all contained a single value. In the end, a total of 69 remaining features were selected.

The problem with categorical features is non-existent in this subsection of the dataset due to the fact that all 69 features are numeric. Nevertheless, they still need to be normalized to allow them to be at their optimum state for the NN. As was in the UDP subsection of the dataset, these features were normalized using the Min-Max normalization method.

6.3 Neural Network Model

After selecting the dataset and its features, it is necessary to define the structure of the NN. Among the various aspects that need to be defined are the number and arrangement of layers and neurons, AFs employed, optimizer, and LF used. Besides those NN hyper-parameters, the proposed PNN also required the definition of other parameters, which are the cryptographic parameters and the number of bits utilized for number representations. This section will start with an explanation of the NN-related hyper-parameters, then focus on the Optimizer choice, and only then will it go over the HE hyper-parameters and finish off with the number representation ones.

6.3.1 Structure

As already mentioned, there will be three different NN models, two based on the work of Sbai and boukhari [93], and the other based on the work of Cil, Yildiz, and Buldu [96]. They will share most configurations but will differ in the number of layers and neurons, which, in turn, also requires some adjustments on the number of bits used for number representation.

Starting with the first two, the input and output sizes of the NN are defined, respectively, by the number of input features and the number of outputs. As already seen, the number of input features is seven, so the NN will have an input of size seven, and the classification is binary, which means that there will be only a single output. The work of Sbai and boukhari [93] provides results in such a way that it is possible to evaluate the impact of the number of hidden layers on the final model performance. Seeing that, and to be able to compare different size PNNs, two different NN configurations were used: one with a single hidden layer and one with none. This way, there can be comparisons with the results of Sbai and boukhari [93] while also having a faster alternative for ciphertext computational efficiency tests. Nevertheless, the performance of the latter will also be analyzed and presented. From now on, the 1-hidden layer NN will be referred to as M_1, and the one with no hidden layers will be referred to as M_0.

The third model is significantly more complex, as its input size is 69, and it contains three hidden layers of 50 neurons each, as is the NN model presented by Cil, Yildiz, and Buldu [96]. The only difference is that the NN structure presented by them has two purposes, and the one tested in this thesis is only for identifying attacks, which means that the output size is one. Due to its complexity, this NN structure will be only trained in plaintext to be used for the analysis of the classification performance of the proposed PNN. This is possible due to the guarantees provided by HE. Nevertheless, this will be further explained in Section 6.4.1. This NN will be referred to as M'.

Even though the NN structure proposed is based on the work of Sbai and boukhari [93], they do not specify what AFs were used, neither the optimizer, neither the LF, nor the size of the hidden layers. Due to this fact, these hyper-parameters were chosen without considering their work. Cil, Yildiz, and Buldu [96] does specify the AF as being the Sigmoid and the LF as being the binary cross entropy. Considering that, the choice of AF was straightforward for both works as there is currently only a single one implemented, which is the Sigmoid approximation by Myers and Hutchinson [19]. Although Cil, Yildiz, and Buldu [96] uses binary cross-entropy, the chosen LF was the MSE because of two factors: firstly, because both problems at hand are binary classification problems, and secondly, while Binary Entropy could provide better results, its dependence on the logarithm makes it an unsuitable choice for HE scenarios. The optimizer choice is more complex, and it will be explained in more detail in the next subsection.

6.3.2 Optimizer

Choosing the right optimizer is essential to ensure that the NN is trained as efficiently and effectively as possible. Although modern optimizers incorporate the momentum concept into the learning rate value to achieve faster training times, the traditional Gradient Decent can prove more advantageous due to its simplicity. This is because this momentum concept requires additional computations to be made, namely, the expensive multiplication operations, so the increase in computational cost might not justify the increased learning performance.

Furthermore, exchanging the learning rate by a power of two can also provide a decrease in the computational time required to calculate this step. Yet, as will be possible to see in Section 6.4, because this multiplication is only a tiny part of the whole backpropagation step, its gains are insignificant when compared to the overall training time. Nevertheless, the traditional Gradient Decent was chosen to reduce computational complexity.

Now, it remains to be chosen how many samples to train on before updating the NN weights. Base Gradient Decent trains all samples and only then updates the NN model weights. This means that it comes at a significant computational cost, namely, memory requirements. Stochastic Gradient Decent, on the other hand, updates the model parameters after each training entry. Although it drastically reduces memory requirements, it is not as stable as the base one. Finally, the Mini-bach Gradient Decent is a middle point between the two by calculating the gradient of *n* entries at once and updating the model parameters at the end of each. The latter was chosen as it is the most balanced approach to this problem. Nevertheless, by changing the number of entries in each batch, it is possible to mirror one or another strategy more closely.

The chosen batch size was 240, so each of the 24 threads of the most capable system, which was used to perform the ciphertext training, was attributed to ten training samples per batch.

6.3.3 Homomorphic Encryption Parameters

As referred to in Chapter 4, the chosen HE scheme for this implementation was TFHE. Similarly to most HE schemes, TFHE can be configured with several parameters that will directly affect the security and performance of its computations. For example, since this scheme relies on ciphertext noise to ensure the security of the ciphertext, its amount can be configured through parameters, and it will directly affect the reliability and performance of each operation.

The parameters chosen were the recommended ones provided as default by the TFHE-rs library. They provide the standard 128-bit security while only presenting an error probability of 2^{-40}. The parameters recommended by the library were chosen because creating custom cryptographic parameters should only be done by specialists on that specific cryptosystem to avoid the definition of insecure parameters.

6.3.4 Number representation

The number representation used in this model is the one supported by the PNN, which is, as explained in Chapter 4, fixed-point. Nonetheless, the number of bits used to represent each number still needs to be defined on a per-use-case basis and must be the lowest possible to ensure that the computational cost of the NN is reduced to a minimum. Defining such values requires some knowledge of the NN architecture and features domains, but, very importantly, it does not require the values of the features themselves to be shared.

Besides the total number of bits, the number of them reserved for fractional representation must also be defined. The higher the number of bits used for precision, the more granular detail the NN allows. However, this also comes at the cost of reduced computational speed.

Starting with M_0 and M_1, and after analyzing the problem at hand, it is possible to infer that all seven features are normalized between zero and one, the maximum number of neurons per layer is seven, and the output of each neuron is always between zero and one. This means that the minimum number of required integer bits for each feature and the neuron output is one, and considering the seven features (and normalized weights), there is a need for at least 3 bits. However, because weights are unbounded and can grow to absolute values higher than one during training, additional space must be added to accommodate the multiplication results of the features with these non-normalized weights. Tests indicate that, in these conditions, adding two to three bits should ensure stable training. In the end, 5 bits were used to represent the integer part (+1 used for the signal representation).

The NN M', on the other hand, has 69 features all normalized, a maximum of 50 neurons per layer, and the output of each neuron is between zero and one. This means that, once again, considering normalized weights, a maximum weighted sum will be 69, which means that 7 bits is the minimum required. However, due to the fact that 7 bits can represent 128 different numbers, there are still almost 60 extra integer numbers that can be represented. Given this, only seven bits were used to represent the integer part (+1 used for the signal representation).

Choosing the number of precision bits is not as easy as selecting the number of integer bits. This is because there is no formula to calculate this quantity, but instead, a trade-off between precision and computational time. This is very important because, as mentioned in Section 5.1.9, the number of bits significantly impacts the duration of each operation. As such, this decision must be carefully analyzed.

A plaintext test executed for 25% of the CICDDoS2019 dataset using N_0 clearly shows that the number of bits directly affects the NN loss. Figure 6.1 shows the relation between the MSE of the test data after training the NN during 50 epochs and the number of precision bits. From it, it is possible to see that at least 15 bits of precision are required for this use case.

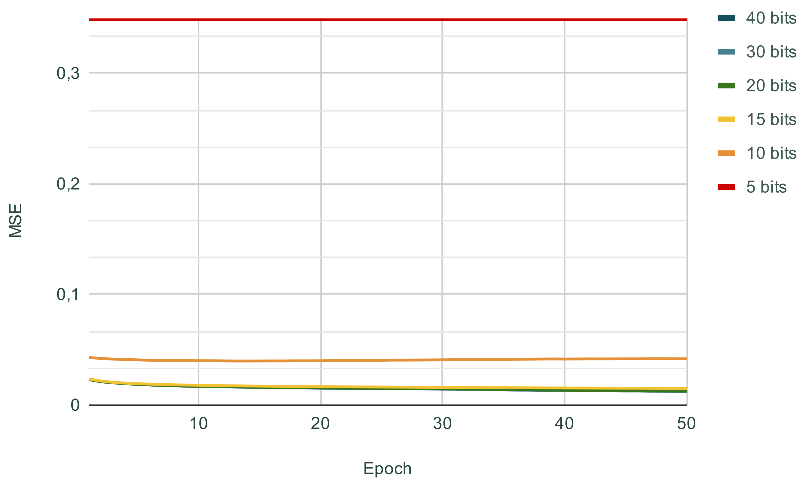

Figure 6.1: M_0 - Evolution of test MSE across training epochs for different quantities of bits used to represent numbers

6.4 Results

With the dataset, pre-processing, and NN presented, it is now time to evaluate the NN model performance. A series of carefully designed tests have been executed to assess the models under various scenarios. To provide an idea of how tests were executed, we start with a subsection detailing the experimental setup, and only after that, we present the test results. The results will be divided into two parts: one for the performance of the models, where standard metrics for DDoS attack detection will be used to compare the PNN DDoS model with SotA models, and the second part shifts the focus to computational efficiency, measured through time, throughput, and memory usage metrics.

6.4.1 Experimental Setup

A clear explanation of the experimental setup is pivotal to ensure transparent and trustworthy results. Two computers were primarily used for these experiments: one for the long-running tests, particularly ciphertext ones, and another for more quick tests. The first one is equipped with a 16-core and 24 threads Intel Core i7-12850HX CPU and 64GB of DDR5 memory. The second is equipped with an 8-core and 16 threads AMD Ryzen 7 5700X CPU and 32GB of DDR4 memory.

Every time duration presented was calculated using the "Instant" and "Duration" structures provided by the **std** module of the Rust language. The usage of these structures allows duration to be measured with precision up to nanoseconds.

Furthermore, the tests that analyzed the PNN's ability to learn from the data were executed in plaintext to allow more rapid iterations without being limited by the computational complexity of HE operations. This is possible due to the homomorphic guarantees provided by the HE schemes - operations on encrypted data generate a result equivalent to operations on the plaintext. To formalize this, let f be the HE encryption function, $\odot$ be an abstract NN training operation, and i and o the input and expected output, respectively. This property can be expressed as illustrated in (6.2). To further validate this homomorphic consistency, we also executed tests in both ciphertext and plaintext formats, which confirmed the existence of this property.

$$f(i \odot o) = f(i) \odot f(o) \tag{6.2}$$

6.4.2 DDoS Attack Classification Metrics

Being able to compare the current model with other SotA approaches is essential. To do this, the PNN DDoS attack detection model will be evaluated using the following metrics: accuracy, precision, recall, and the F1-score. These metrics were chosen because they are, according to Ali, Chong, and Manickam [12] and Mittal, Kumar, and Behal [91], the most utilized in the DDoS attack detection domain.

Starting with the comparison with the work of Sbai and boukhari [93], both M_1 and M_0 NNs were trained using 87971 training entries and 21993 test entries. The obtained test metrics and their relative error, when compared with the specified work, are presented in Table 6.1. It is important to interpret these differences with caution, as M_1 and M_0 exhibit superior performance in both recall and F1-score when compared to the original ones. Nevertheless, these results show a very small impact of the TFHE restrictions on the PNN performance.

The work of Cil, Yildiz, and Buldu [96] does provide more insights about their NN, so M' will be used for the remaining DDoS attack detection performance evaluation. The test metrics obtained by M' and their relative error with the work of Cil, Yildiz, and Buldu [96] are depicted in Table 6.2.

Table 6.1: PNN DDoS model metrics and relative error to SotA work

Metric	[93]	M_1	RE	M_0	RE
Accuracy	99.995%	99.04%	0.96%	98.98	1.01%
Precision	99%	98.74%	0.26%	98.67	0.26%
Recall	98%	99.99%	2.03%	99.99	2.03%
F1-Score	99%	99.36%	0.36%	99.33	0.33%

Table 6.2: Adapted PNN DDoS model metrics and relative error to SotA work

Metric	[96]	M'	RE
Accuracy	99.97%	99.87%	0.10%
Precision	99.99%	99.96%	0.03%
Recall	99.98%	99.89%	0.09%
F1-Score	99.98%	99.92%	0.06%

Even then, to make the training environment as close as possible, ten samples of the CI-CDDoS2019 dataset were taken randomly. The usage of ten different samples instead of a single one cancels out any advantages provided by a specific sample. The NN test metrics were recorded after ten epochs. Since there are no equivalent results in the original paper, those results were obtained using the TensorFlow library. Besides this primary objective, this experiment allows the consistency of the proposed PNN to be analyzed. The average and standard deviation of the metrics obtained in this experiment and their comparison with the TensorFlow ones are presented in Table 6.3.

Table 6.3: Adapted PNN DDoS model metrics and standard deviation for ten different samples and five epochs and their comparison with an equivalent TensorFlow model

	M'		$M'_{\text{TensorFlow}}$	
	Mean	**Std. deviation**	**Mean**	**Std. deviation**
Accuracy	0.986907449	0.000561599	0.993835419	0.00010242
Precision	0.993819436	0.000118783	0.993823498	0.000125635
Recall	0.993001315	0.00047724	1.00	0.00
F1-score	0.993410171	0.000284419	0.996899313	6.32853×10^{-5}

In plaintext, the epoch number is not critical because each one only takes a few seconds to finish. However, in ciphertext, as will be possible to see in the next part of the results, epochs can take days to complete. Because of this, it is essential to know how the PNN compares with non-private NN from a learning pace point of view. Figure 6.2 shows the evolution of the accuracy metric over time until epoch 15. Here, it is possible to see one of the advantages of using traditional Gradient Decent because, even though it cannot reach as high accuracies, it does not suffer from a cold-start problem, which allows the model created with the proposed PNN to be near peek performance right at the end of the first epoch.

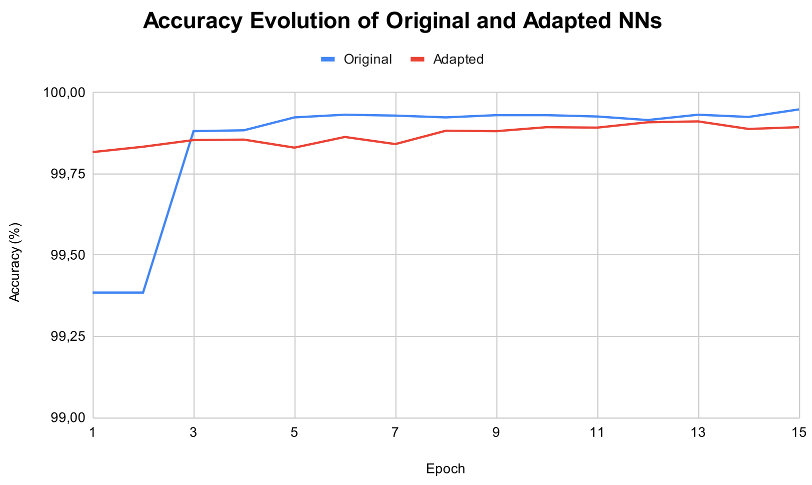

Figure 6.2: Accuracy evolution comparison along each training epoch between the adapted PNN model and the TensorFlow model

6.4.3 PNN Computational Efficiency

As seen in Chapter 3, one of the most prominent limitations of NNs that use HE for data privacy is the computational overhead introduced by the usage of HE itself. Given this, it is important to analyze how the proposed PNN performs from this perspective.

First of all, M_0 and M_1 were executed in ciphertext, and the duration of each epoch was registered to be able to estimate the computational time of different numbers of epochs. They were trained in the 24-thread computer using a 27491 entry sample of the CICDDoS2019 dataset, from which 80% were used for training and 20% for testing. During this training, the numbers were represented using 21 bits, where one bit was for the signal, five for the integer part, and fifteen bits of precision. Each 240-sized batch of M_0 took, on average, 5994.36 seconds (1h and 40m), while M_1 batches took 45908.75 seconds (12h and 45m). Considering a total of 92 batches, and due to the fact that all training homomorphic operations occur inside each batch, it is possible to estimate the duration of the epochs, which are 551481.12 (6d 9h 11m 21s) seconds and 4223605 seconds (48d 21h 13m 25s) for M_0 and M_1, respectively. These results and more estimated durations are presented in Table 6.4.

NN	Batch	Batch Std.	1 Epoch	5 Epochs	10 Epochs
M$_0$	5994.36	141.4357569	6d 9h	31d 21h	63d 20h
M$_1$	45908.75	215.7087156	48d 21h	244d 10h	488d 20h

Table 6.4: Training batch duration and respective standard deviation in seconds for M_0 and M_1 as well as different quantity of epochs duration (epoch duration rounded to the nearest hour)

By further exploring these results, it is possible to draw other conclusions. Considering that there were 24 threads, on average, it took 599.436 (9m 59s) seconds and 4590.875 seconds

(1h 16m 30s) to train ten samples in M_0 and M_1, respectively. Likewise, the throughput of each model was, respectively, 144.14 and 18.82 samples per hour.

With the overall training times presented, it is also essential to look into each training step individually. The significant steps pinpointed in the backpropagation were the forward propagation step (with the calculation of the AF derivative), the calculation of the LF, the computation of the bias gradient, the multiplication of that gradient by the learning rate, and the calculation of the weight gradients. Using the 16-thread computer, the duration of these steps for each sample was recorded for a different number of precision bits to allow the impact of those to be further studied (Table 6.5). It is important to remember that 5 bits were chosen to represent the integer part of the numbers and an additional one for the signal of the number, so, for example, with 15 precision bits, the total number of bits is 21. Also, these durations can be higher than the previously reported because these tests were executed on the 16-thread computer.

	15 Precision bits	**20 Precision bits**	**25 Precision bits**
forward propagation[1]	183.27587	226.01656	365.89293
Loss Calculation	1.93221	2.24880	2.63082
Bias Gradient	9.50089	11.18579	13.79061
Learning Rate	3.29E-05	2.97E-05	3.98E-05
Weight Gradients	236.60424	328.43702	502.31179

Table 6.5: Duration (in seconds) of each major training step of the M_0 ([1] with AF derivative calculation)

As seen in Section 2.3.2, calculating the Bias Gradient (and the subsequent Weight Gradients) requires the multiplication of the gradient by the learning rate. Due to the optimization, which replaces this multiplication with a shift operation, its computational cost is orders of magnitude lower than its original counterpart. Yet, because this operation is just a small part of the backpropagation step, this plus 99.99% velocity increase only results in a 0.67% reduction in time when using 15 bits of precision and less if using more bits (Figure 6.3).

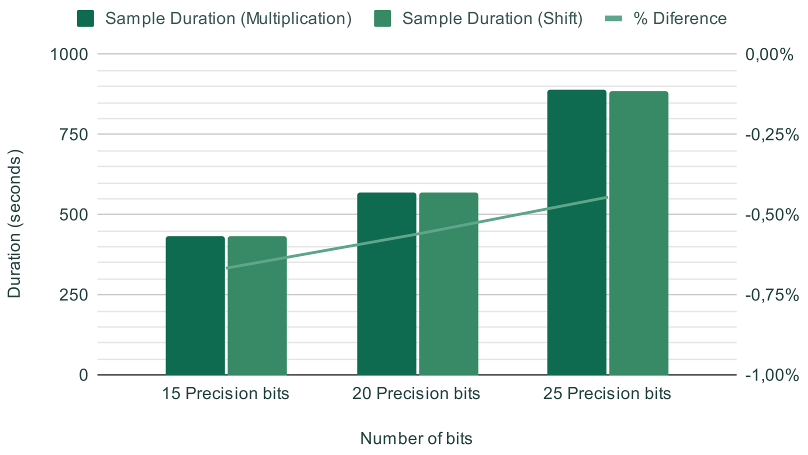

Figure 6.3: Total sample training duration comparison between the usage of shift and multiplication in the Learning Rate step

Looking into inference performance, its analysis is more straightforward due to the fact that the backpropagation step is not required. To study the throughput of this PNN, several samples were classified in the 16-bit computer, and the time of each classification was recorded. Since classification is efficiently executed in parallel, the throughput of the PNN is directly affected by the number of threads in the system. The total duration of inference for a single sample and the AF computational time inside that inference for M_0 are presented in Table 6.6.

	AF Computation Time	**Total Inference Time**
15 Precision bits	17.72	131.56
20 Precision bits	21.27	198.70
25 Precision bits	25.05	266.82

Table 6.6: M_0 PNN model AF computation time (in seconds) and total inference time (in seconds) at different precision bit count

From the durations presented in Table 6.6, it is possible to observe that, using 15 precision bits, each inference takes less than two and a half minutes. Therefore, using n threads would make it possible to classify at least $n*2$ entries every 5 minutes. For example, the 24-thread computer can classify at least (but probably more than) 48 entries every 5 minutes, which is equivalent to more than 576 entries per hour. Using computers with more RAM and better CPUs could increase this number even further.

Finally, looking into memory usage, it is clear that training using the proposed PNN uses significantly more memory than using a non-private NN. Approximate estimations show that training the N_0 PNN with 21992 training samples and 5499 test samples with 15 bits of precision requires 20GB of RAM. Interestingly, although this amount of RAM is reserved, the fact that the training is done in batches means that the part of it that is read and written during each batch is about 2GB. This suggests that, if necessary, ROM memory could be used to store the unused memory, and only the tiny part that was used in that batch would be held in RAM. Using such a technique could make it possible to train in low-memory devices.

6.5 Discussion

With all of the results presented, it is time to discuss them. This section will go into detail in the two main areas of these results: the first is model performance and computational efficiency, and the second one is data privacy. The first will dive into all the tangible results presented in this thesis, which are the basis for the complete fulfillment of most defined objectives. The second is a broader and less specific area where most discussion will not be based on presented tangible results; nevertheless, this latter area is of utmost importance to this work as it is the fundamental base of all the work presented in this document.

6.5.1 Model Performance and Computational Efficiency

Looking at the classification performance results, it is clear that adapting the NN for HE does have an impact on the final NN learning and classification performance. However, these results also show that this impact is minimal, with relative errors of less than 2.04% across all metrics. The acceptable error is dependent on the domain, and for DDoS attack detection, it is possible to say that these losses in model classification performance are non-significant. Besides that direct comparison, the one with an equivalent TensorFlow model also allows several conclusions to be taken. First of all, from a stability standpoint, although it is less than the TensorFlow equivalent, it is still good, which means that it does not need to be a consideration when creating NN models using the proposed solution. Secondly, the learning pace can also be analyzed using these results, where it can be seen that the NN model using the PNN reaches near its top classification performance faster when compared to the TensorFlow equivalent, leading to fewer epochs being required to achieve near-top model accuracy.

Switching to computational efficiency and computational performance, the tests showed that using an optimized number of bits, it is possible to significantly reduce the computational cost without affecting accuracy. This point is critical because tests showed that an increase from 8 to 16 bits doubles the computational time for every single addition operation and triples for every multiplication operation. That, together with the complexity of the NN, are the most important factors that determined the final training time. The less complex NN training completed a single epoch in six days, while the more complex one took more than a month, with ten epochs of the latter taking more than a year. This clearly shows that the application of the proposed solution with the current optimization is heavily dependent on the complexity of the NN. The DDoS attack detection domain is an example of where it could be used. However, the two-and-a-half-minute inference latency can still be a problem for real-time systems. Still, it is not that far from a sub-minute inference, with it probably already achievable with existing SotA CPUs.

Finally, the results presented also show that the memory usage under HE is higher than a cleartext alternative. This can create problems when training with a large dataset or with a large number of features, which could limit the scalability of the proposed PNN. However, tests also showed that the amount of memory used at each time is a small part of that memory, so, theoretically, it should be possible to store unused in persistent memory, and, with that, training could be done with a smaller RAM footprint. This latter strategy does add some more complexity to the final system but could make training practical in lower-resource systems.

6.5.2 Data Privacy

Unlike non-private models that only support data encryption during transmission and storage, the one presented in this thesis can guarantee data privacy from the moment data leaves the client system all the way to the moment its processing result is returned to the client. Not only is data encrypted as it enters the third-party server, but this encryption lasts throughout the entire lifecycle of the NN training or inference. This way, any window of vulnerability where the data might be exposed in plaintext format is eliminated. Furthermore, it is also essential to mention that the key necessary to decrypt the ciphertext is never sent to the server. This is very important because, as already seen, the security of the ciphertext is directly related to the ability to keep the decryption key secret.

When compared to the SotA privacy preservation using HE in NN, it is clear that this makes this model significantly more secure than the alternative of Onoufriou, Mayfield, and Leontidis [98], which calculates the backpropagation step in cleartext because this latter is susceptible to the exposure of the cleartext data, or, even worse, the leakage of the decryption key. When compared to the work by Hesamifard et al. [103], the privacy advantages are not that clear. However, any form of external communication can represent a potential vector for data exposure or attacks, regardless of the encryption measures in place. Therefore, by completely isolating the computational process from external communications, the proposed PNN model further minimizes the risk of data leakage or unauthorized access.

As explained in Chapter 2, other data privacy-preserving solutions for NN often depend on techniques such as data partitioning or noise addition. In strategies where data is distributed across multiple servers, the privacy guarantees come from its data distribution that limits the reach of any potential data exposure by ensuring that no single entity possesses all the information. However, this approach only mitigates the risk to a certain extent because it can still be possible to extract information from the portion of the data each server holds, as well as from the shared training weights that circulate during the learning process. Similarly, methods that rely on adding noise to data aim to obfuscate the actual values, making it difficult to obtain accurate information about a single individual. Yet, these techniques offer only probabilistic guarantees of privacy. With enough computational effort and access to auxiliary input, it might be possible to reverse-engineer the original data.

In summary, the proposed model ensures that the data remains encrypted throughout its lifecycle and does not require the decryption key to eliminate the risk of data compromise effectively. Therefore, while other methods provide privacy to varying degrees and under certain conditions, the proposed solution offers a more certain guarantee of data security.

Chapter 7

Conclusions and Future Work

This chapter will conclude this document. To do so, it will review the results obtained in this thesis. Afterward, the initially defined objectives will be revisited to assess their level of completion, the phase at which they were fulfilled, and their respective scientific contributions. Subsequently, the challenges and limitations of the proposed solution will be explored, and future research across the various areas of this work will then be enumerated and discussed. Finally, this chapter will conclude with remarks by the author reflecting on the journey undertaken to complete this thesis.

7.1 Results Overview

The primary objective of this thesis was to develop a PNN using HE and create a DDoS attack detection model. While the first is very important to accelerate the research on this PNN approach, the second is essential to kickstart the usage of such solutions in the NIDS domain.

The first step to solving those problems was making a SotA review. Here, it was found that there existed several challenges and limitations to the use of HE in NNs. Looking deeper into the results, it was concluded that the most promising HE scheme was TFHE. Given this, a comprehensive solution, called ENNigma, was designed taking into account the various limitations and advantages of this scheme while always maintaining it modular to allow the used scheme to be easily changed at any time. The subsequent implementation was done using the programming language Rust and the TFHE-rs library.

With ENNigma implemented, several experiments were conducted, which showed that our framework minimizes the impacts of applying data privacy techniques based on HE with significant performance gain compared to similar SotA solutions. The optimizations applied achieved an incredible boost in computational efficiency compared to equivalent PNNs, namely, the one proposed by Yoo and Yoon [108]. These optimizations were applied at various stages of the PNN architecture and implementation, where each provided different computational efficiency gains. Using a more updated TFHE library was one of the first optimizations that resulted in approximately a 45% performance increase across all HE operations. In the end, these library changes and algorithm optimizations resulted in an increase between 51% to 59% in number across-the-board numeric operations' performance.

Besides this, ENNigma's ability to work with numbers represented using any pre-defined number of bits also allows the resulting NN models to be fine-tuned for each use case. This functionality is essential because the number of bits can significantly impact operation performance, with tests showing that an increase from 8 to 16 bits doubles the time taken to perform an addition. In the same scenario, the multiplication more than quadruples the computational time. Algorithms such as the Strassen matrix multiplication were also tested to increase the computational efficiency of the final model. However, the results did not show an advantage over traditional matrix multiplication and instead increased the computational time in the test scenarios.

Nonetheless, an innovative approach to significantly decrease the computational time of some operations was used due to the high computational cost of multiplications. This approach was applied in one of the calculations required during the backpropagation step, and it changed the traditional representation of the learning rate into a value that could be represented as power-of-2 so that the scaling of the gradient by the learning rate could be exchanged by a shift operation, which is orders of magnitude faster. For comparison, a multiplication operation in 16 bits takes, on average, 9.81 seconds, while the shift operations only take about 9.79 microseconds. However, even though this multiplication was performed multiple times for each layer in each backpropagation step, the impact of this optimization proved to be insignificant because it only reduced the total sample backpropagation time by less than 1%. Yet, this switching from multiplication to shifts was also used to implement the AF approximation, and, in that case, it achieved significant results with a decrease in computation time between 33.96% and 70.43%.

Regarding the DDoS attack detection model created using ENNigma, its performance shows that the impact of data privacy preservation is non-significant from the perspective of the NN model performance. Experiments show that, when comparing the most used metrics in the DDoS attack detection field, our model only presents a slight decrease in performance compared to equivalent, non-private NN models. The relative error of this decrease is in the magnitude of 0.10% when compared to the NN model presented by Cil, Yildiz, and Buldu [96], and between 0.26% and 2.03% when compared to the work of Sbai and boukhari [93]. To isolate the impact of privacy preservation techniques, we created an equivalent NN using the TensorFlow library. This allowed us to perform controlled tests between non-private NNs and PNNs. These tests allowed the impact of the privacy preservation techniques to be isolated, and they showed a maximum loss in the Recall metric with a difference of 0.6999%. The results of these tests were also used to evaluate the consistency of ENNigma, and they showed results similar to the TensorFlow model, with the most impact being reported in the Accuracy metric with a difference in the standard deviation of just 0.000459. Yet, even with all the computational efficiency improvements, this capability to correctly identify attacks still comes with a high computational cost, with inference taking about two and a half minutes per entry and a single training epoch taking six days on a small PNN model.

Finally, from a privacy standpoint, ENNigma also increases the data privacy guarantees compared to some SotA approaches. This occurs due to some specific particularities of this proposed solution, namely, the fact that HE is used during all NN operations, the non-presence of the decryption key on the server, and the minimization of external communications, which can be potential attack vectors.

7.2 Objectives Overview

Before concluding this thesis, we revisit the defined objectives, discuss their level of achievement, and evaluate the success of this work.

Starting with O1, which implies the development of an NN that preserves data privacy using HE, it is possible to say that it was fully fulfilled. Even though the fulfillment of the main objective means that the individual ones were also fulfilled, analyzing each particular objective can give some additional insights. Objectives O1.1 to O1.3 were attained with the research made in Chapter 3, O1.4 was satisfied with the detailed architecture explanation provided in Chapter 4, and O1.5 and O1.6 were met in Chapter 5.

Switching over to O2, which refers to the usage of ENNigma to create a model for DDoS attack detection, it is also possible to affirm that it was completed. Objective O2.1 was satisfied by the research made for Chapter 3, while objectives O2.2 to O2.4 were accomplished in Chapter 6.

The last primary objective, O3, was also fulfilled, with both O3.1 and O3.2 being realized in Chapter 6 and Chapter 7.

These conclusions and the contributions resulting from each objective are summarized in Table 7.1.

Objective	Status	Chapters	Contributions
O1	Full	3,4,5	C1, C2, C3, C4, C5
O1.1	Full	3	C1
O1.2	Full	3	C1
O1.3	Full	3	C1
O1.4	Full	4	C2 (Arquitecture)
O1.5	Full	5	C2 (Implementation), C3, C4, C5
O1.6	Full	5	-
O2	Full	3,6	C7
O2.1	Full	3	-
O2.2	Full	6	-
O2.3	Full	6	C7
O2.4	Full	6	-
O3	Full	6	C6
O3.1	Full	6	-
O3.2	Full	6	C6

Table 7.1: Objectives Overview (Full - Objective Fulfilled)

7.3 Challenges and Limitations

Even though all objectives were met, the results presented in this document show that ENNigma and respective DDoS attack detection models still have some challenges and limitations. Scrutinizing this is essential to guide future work.

The main limitation of ENNigma, as was the case in the SotA works that use HE for privacy preservation in NN, is the computational overhead introduced by HE. Although several improvements have been made in this area, it is not yet sufficient to eliminate this problem, which is something that is required by real-time applications like DDoS attack detection. While in other domains this problem can be less relevant, the 2-and-a-half-minute latency for classification in DDoS attack detection can be considered a constraint for real-time traffic analysis. Regardless, substantial speed gains were already achieved in this work, so the prospect of attaining real-time classification in the future is a possibility worth exploring.

Not so noteworthy, but nevertheless, a constraint of such PNN models is that they are not fully reliable. This issue arises from the fact that HE operations do not guarantee the correct result 100% of the time. This happens because of the reliance of some HE on noise to ensure security, so there exists a slight chance that this noise corrupts the output. As reported, the use of cryptographic parameters yields an error probability of 2^{-40}, implying that an error may occur once in every 2^{40} operations or, roughly, once in a trillion operations. While this probability is quite low, it is a factor to consider in critical scenarios, such as the medical domain. However, in domains like DDoS attacks and most others, this aspect is not a significant factor when deciding whether to employ such an approach.

7.4 Future Work

Even though ENNigma provides significant value to the research community, it can still be further improved in several areas. In this section, these possible future research paths will be presented. To facilitate their compression, they were grouped into three main groups: the computational efficiency group, where the future work related to reducing the impact of the HE computational overhead is presented; the PNN performance, which will focus on possible ways to improve the performance of PNN models; and the last one will talk about the need to validate this solution in other application domains.

7.4.1 Computational Efficiency

Given the limitations presented in the previous sub-section, there is still room for improvement to make this solution the preferred method when data privacy in NN is a significant requirement. Currently, similar to most solutions that use HE, the throughput is the main limitation. Even though several improvements were made in this area, the high computational cost still needs to be addressed.

One alternative that could be analyzed in future works to increase computational efficiency is the usage of GPU acceleration, which could provide significant performance gains. However, GPU acceleration works best for easily parallelized tasks, and making TFHE operations parallelized is still a work in progress. This means that moving this task into a GPU will not yet provide the returns possible for easily parallelizable tasks. However, future research could make this possible and offer significant performance gains.

Another alternative is to use Stochastic Weight Averaging in Low Precision Training [115] to significantly reduce the number of bits used during training to decrease the computational time. This strategy was used in the work of Lou et al. [106], and the results are promising, so implementing it in this solution seems to be an excellent approach.

Finally, the last possible future research alternative is in relation to the use of multiple HE schemes as done by Lou et al. [106]. However, unlike their work, the aim of employing such a technique in ENNigma would be to rely on TFHE primarily and only utilize another scheme for weight gradient calculation and, potentially, in the computation of the activation function and its derivative. Yet, it is required to ensure that using such an aproach does not impact the NN inference latency.

Nevertheless, all these alternatives are non-exclusive and can be further researched in parallel as their combined application will probably achieve the best results.

7.4.2 Private Neural Network Performance

In terms of reducing the impact of the restrictions imposed by HE other than computational time, there is also room for improvement, namely regarding the use of approximations.

Concerning memory usage, future research could look into the mentioned strategy of storing memory that is not currently being used, which will be referred to as "cold memory" in persistent memory. This way, lower RAM usage could be achieved, but its drawbacks need to be examined.

7.4.3 Application to other domains

The chosen application domain was NIDS, more specifically, the DDoS attack detection. Due to this, the conclusions presented in this thesis are directly related to this application domain, so further research is needed to apply and validate this solution in other areas.

This is particularly important in medical and other critical fields, where implementing this solution without careful analysis could have serious consequences. Therefore, in-depth research into its application is essential if it is to be safely employed in such domains.

7.5 Final Remarks

Finishing this thesis marks the end of a journey that started almost a year ago. So, it is worth looking back at it and seeing what was learned, the challenges, and the solutions. Beginning with a foundation in other privacy-preserving techniques, I was intrigued by the prospect presented by HE. The challenge of adapting this technique to the domain of NIDS was an intriguing challenge that I would later find out to be a steep mountain to climb. This is because, as I dived deeper into the subject, the complexity and computational expense of applying HE in NN proved to be a daunting challenge.

As I explored the SotA research regarding this problem of computational complexity when joining HE and NN, it was clear that it was a significant obstacle still to be solved. Instead of being discouraged, this constraint urged me to innovate, which resulted in the implementation of the PNN solution presented and used to create a DDoS attack detection model in this thesis.

The results were promising but nuanced, as they showed that data privacy in NN could be achieved at a small cost of performance metrics. Yet, even though significant performance gains were obtained, the computational demands remain a challenge, signaling necessary future research.

On a more personal note, it is unmistakable that this path was challenging and demanding, but it was also instructive. The mystery of encryption, mainly the complex mechanics of HE, became a subject of deep attention. Each challenge faced offered a new layer of understanding and expertise, making the journey itself incredibly rewarding.

The broader implications of the proposed solution extend beyond the scope of this thesis. It serves as a stepping stone in the ever-important quest for secure and private data processing in NN using HE, offering insights that could be vital for future advancements.

As I contemplate the trials, errors, and eventual successes of this journey, a sense of achievement is impossible to ignore. The challenges faced have been stepping stones to both professional and personal development. As this chapter and thesis conclude, there is an optimistic feeling of curiosity for what the following stages of this research avenue may bring.

"The only way to discover the limits of the possible is to go beyond them into the impossible."
- Arthur C. Clarke

Bibliography

[1] Eleanor Carey and Ida Mc Donnell. *Powering an Inclusive, Digital Future for All.* Accessed: 2023-06-29. 2023. url: https://newsroom.cisco.com/c/r/newsroom/en/us/a/y2023/m01/powering-an-inclusive-digital-future-for-all.html.

[2] Adrienn Lukács. "What is privacy? The history and definition of privacy". In: (2016).

[3] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Applied Cryptography*. Ed. by Taylor and Francis Inc. 1996. isbn: 9780849385230.

[4] Abhishek Sharma et al. "An investigation of security risk & taxonomy of Cloud Computing environment". In: Oct. 2021, pp. 1056–1063. doi: 10.1109/ICOSEC51865.2021.9591954.

[5] Amine Boulemtafes, Abdelouahid Derhab, and Yacine Challal. "A review of privacy-preserving techniques for deep learning". In: *Neurocomputing* 384 (Apr. 2020), pp. 21–45. doi: 10.1016/j.neucom.2019.11.041. url: https://doi.org/10.1016/j.neucom.2019.11.041.

[6] Li Li et al. "A review of applications in federated learning". In: *Computers & Industrial Engineering* 149 (2020), p. 106854. issn: 0360-8352. doi: https://doi.org/10.1016/j.cie.2020.106854. url: https://www.sciencedirect.com/science/article/pii/S0360835220305532.

[7] Cynthia Dwork. "Differential Privacy". In: *Automata, Languages and Programming*. Ed. by Michele Bugliesi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–12. isbn: 978-3-540-35908-1.

[8] Ping Li et al. "Multi-key privacy-preserving deep learning in cloud computing". In: *Future Generation Computer Systems* 74 (Sept. 2017), pp. 76–85. doi: 10.1016/j.future.2017.02.006. url: https://doi.org/10.1016/j.future.2017.02.006.

[9] *Hacktivists step back giving way to professionals: a look at DDoS in Q3 2022*. Accessed: 2023-06-30. Nov. 2022. url: https://www.kaspersky.com/about/press-releases/2022%5C_hacktivists-step-back-giving-way-to-professionals-a-look-at-ddos-in-q3-2022.

[10] Jessica Haworth. *UK cryptocurrency exchange EXMO knocked offline by 'massive' DDoS attack*. https://portswigger.net/daily-swig/uk-cryptocurrency-exchange-exmo-knocked-offline-by-massive-ddos-attack. Accessed: 2023-06-29. 2021.

[11] Mary James. *The 8 Most Devastating DDoS Attacks of 2022 and What We Can Learn from Them*. https://allaboutcookies.org/the-worst-ddos-attacks. Accessed: 2023-06-29. Feb. 2023.

[12] Tariq Emad Ali, Yung-Wey Chong, and Selvakumar Manickam. "Machine Learning Techniques to Detect a DDoS Attack in SDN: A Systematic Review". In: *Applied Sciences* 13.5 (Mar. 2023), p. 3183. doi: 10.3390/app13053183. url: https://doi.org/10.3390/app13053183.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[14] Elena Fedorchenko, Evgenia Novikova, and Anton Shulepov. "Comparative Review of the Intrusion Detection Systems Based on Federated Learning: Advantages and Open Challenges". In: *Algorithms* 15.7 (July 2022), p. 247. doi: `10.3390/a15070247`. url: `https://doi.org/10.3390/a15070247`.

[15] Julia H. Allen, Derek Gabbard, and Christopher May. "Outsourcing Managed Security Services". In: (Jan. 2003). doi: `10.1184/R1/6575933.v1`. url: `https://kilthub.cmu.edu/articles/report/Outsourcing_Managed_Security_Services/6575933`.

[16] Bernardo Pulido-Gaytan et al. "Privacy-preserving neural networks with Homomorphic encryption: Challenges and opportunities". In: *Peer-to-Peer Networking and Applications* 14.3 (Mar. 2021), pp. 1666–1691. doi: `10.1007/s12083-021-01076-8`. url: `https://doi.org/10.1007/s12083-021-01076-8`.

[17] Robert Podschwadt et al. "A Survey of Deep Learning Architectures for Privacy-Preserving Machine Learning With Fully Homomorphic Encryption". In: *IEEE Access* 10 (2022), pp. 117477–117500. doi: `10.1109/ACCESS.2022.3219049`.

[18] Baek Kyung Song et al. "A Bitwise Design and Implementation for Privacy-Preserving Data Mining: From Atomic Operations to Advanced Algorithms". In: *Security and Communication Networks* 2019 (Oct. 2019), pp. 1–14. doi: `10.1155/2019/3648671`. url: `https://doi.org/10.1155/2019/3648671`.

[19] D.J. Myers and R.A. Hutchinson. "Efficient implementation of piecewise linear activation function for digital VLSI neural networks". In: *Electronics Letters* 25.24 (1989), p. 1662. doi: `10.1049/el:19891114`. url: `https://doi.org/10.1049/el:19891114`.

[20] Ivone Amorim et al. *Data Privacy with Homomorphic Encryption in Neural Networks Training and Inference*. Ed. by Rashid Mehmood et al. Cham, 2023.

[21] Ivone Amorim et al. "Privacy-Preserving NN for IDS: A Study on the impact of TFHE constraints". In: *Security and Trust Management*. In press. The Hague, Netherlands: Springer Cham, 2023.

[22] Daniel J. Solove. "Conceptualizing Privacy". In: *California Law Review* 90.4 (July 2002), p. 1087. doi: `10.2307/3481326`. url: `https://doi.org/10.2307/3481326`.

[23] Daniel J Solove. *The digital person: Technology and privacy in the information age*. Vol. 1. NyU Press, 2004.

[24] Daniel J Solove. *Nothing to hide: The false tradeoff between privacy and security*. Yale University Press, 2011.

[25] *General Data Protection Regulation*. 2016. url: `https://eurlex.europa.eu/eli/reg/2016/679/oj`.

[26] *California Consumer Privacy Act*. California Civil Code § 1798.100 et seq. 2018. url: `https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?division=3.&part=4.&lawCode=CIV&title=1.81.5`.

[27] Niels Ferguson and Bruce Schneier. *Practical cryptography*. Vol. 141. Wiley New York, 2003.

[28] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018.

[29] R. Lippmann. "An introduction to computing with neural nets". In: *IEEE ASSP Magazine* 4.2 (1987), pp. 4–22. doi: `10.1109/massp.1987.1165576`. url: `https://doi.org/10.1109/massp.1987.1165576`.

[30] D. Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996. isbn: 9781439103555. url: `https://books.google.pt/books?id=SEH_rHkgaogC`.

[31] Philippe Guillot. "Auguste Kerckhoffs et la cryptographie militaire". In: *BibNum* (May 2013). doi: 10.4000/bibnum.555. url: https://doi.org/10.4000/bibnum.555.

[32] *Advanced encryption standard (AES)*. Nov. 2001. doi: 10.6028/nist.fips.197. url: https://doi.org/10.6028/nist.fips.197.

[33] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (Feb. 1978), pp. 120–126. doi: 10.1145/359340.359342. url: https://doi.org/10.1145/359340.359342.

[34] Jean-Philippe Aumasson. *Serious cryptography: A Practical Introduction to Modern Encryption*. 2017. isbn: 978-1-59327-826-7.

[35] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. "ON DATA BANKS AND PRIVACY HOMOMORPHISMS". In: *Foundations of Secure Computation, Academia Press* (1978), pp. 169–179.

[36] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology — EUROCRYPT '99*. Springer Berlin Heidelberg, pp. 223–238. doi: 10.1007/3-540-48910-x_16. url: https://doi.org/10.1007/3-540-48910-x_16.

[37] T. Elgamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *IEEE Transactions on Information Theory* 31.4 (July 1985), pp. 469–472. doi: 10.1109/tit.1985.1057074. url: https://doi.org/10.1109/tit.1985.1057074.

[38] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. "Evaluating 2-DNF Formulas on Ciphertexts". In: *Theory of Cryptography*. Springer Berlin Heidelberg, 2005, pp. 325–341. doi: 10.1007/978-3-540-30576-7_18. url: https://doi.org/10.1007/978-3-540-30576-7_18.

[39] Craig Gentry. "A Fully Homomorphic Encryption Scheme". AAI3382729. PhD thesis. Stanford, CA, USA, 2009. isbn: 9781109444506.

[40] Michele Minelli. "Fully homomorphic encryption for machine learning". Université Paris sciences et lettres, 2018. url: https://tel.archives-ouvertes.fr/tel-01918263v2.

[41] Zvika Brakerski and Vinod Vaikuntanathan. "Efficient Fully Homomorphic Encryption from (Standard) LWE". In: (Oct. 2011). doi: 10.1109/focs.2011.12. url: https://doi.org/10.1109/focs.2011.12.

[42] Zvika Brakerski and Vinod Vaikuntanathan. "Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages". In: (2011), pp. 505–524. doi: 10.1007/978-3-642-22792-9_29. url: https://doi.org/10.1007/978-3-642-22792-9_29.

[43] Craig Gentry, Amit Sahai, and Brent Waters. "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based". In: (2013), pp. 75–92. doi: 10.1007/978-3-642-40041-4_5. url: https://doi.org/10.1007/978-3-642-40041-4_5.

[44] Ran Gilad-Bachrach et al. "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 201–210. url: https://proceedings.mlr.press/v48/gilad-bachrach16.html.

[45] Ahmad Al Badawi and Yuriy Polyakov. *Demystifying Bootstrapping in Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2023/149. https://eprint.iacr.org/2023/149. 2023. url: https://eprint.iacr.org/2023/149.

[46] Jung Hee Cheon et al. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: (2017), pp. 409–437. doi: 10.1007/978-3-319-70694-8_15. url: https://doi.org/10.1007/978-3-319-70694-8_15.

[47] *HElib*. 2013. url: https://github.com/homenc/HElib.

[48] N. P. Smart and F. Vercauteren. *Fully homomorphic SIMD operations*. July 2012. doi: 10.1007/s10623-012-9720-4. url: https://doi.org/10.1007/s10623-012-9720-4.

[49] Craig Gentry, Shai Halevi, and Nigel P. Smart. "Homomorphic Evaluation of the AES Circuit". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 850–867. doi: 10.1007/978-3-642-32009-5_49. url: https://doi.org/10.1007/978-3-642-32009-5_49.

[50] *Microsoft SEAL (release 4.1)*. 2023. url: https://github.com/microsoft/SEAL.

[51] Joppe W. Bos et al. "Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme". In: *Cryptography and Coding*. Springer Berlin Heidelberg, 2013, pp. 45–64. doi: 10.1007/978-3-642-45239-0_4. url: https://doi.org/10.1007/978-3-642-45239-0_4.

[52] Jung Hee Cheon et al. *HEAAN*. 2016. url: https://github.com/snucrypto/HEAAN.

[53] Léo Ducas and Daniele Micciancio. *FHEW: Bootstrapping Homomorphic Encryption in less than a second*. 2014. url: https://github.com/lducas/FHEW.

[54] Ilaria Chillotti et al. *TFHE: Fast Fully Homomorphic Encryption Library*. 2016. url: https://tfhe.github.io/tfhe/.

[55] Zama. *TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data*. https://github.com/zama-ai/tfhe-rs. 2022.

[56] Ahmad Al Badawi et al. *OpenFHE: Open-Source Fully Homomorphic Encryption Library*. Cryptology ePrint Archive, Paper 2022/915. https://eprint.iacr.org/2022/915. 2022. url: https://eprint.iacr.org/2022/915.

[57] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. issn: 1532-4435.

[58] T.F. Lunt and R. Jagannathan. "A prototype real-time intrusion-detection expert system". In: *Proceedings. 1988 IEEE Symposium on Security and Privacy*. IEEE Comput. Soc. Press. doi: 10.1109/secpri.1988.8098. url: https://doi.org/10.1109/secpri.1988.8098.

[59] D.E. Denning. "An Intrusion-Detection Model". In: *IEEE Transactions on Software Engineering* SE-13.2 (Feb. 1987), pp. 222–232. doi: 10.1109/tse.1987.232894. url: https://doi.org/10.1109/tse.1987.232894.

[60] Stefan Axelsson. *Intrusion Detection Systems: A Survey and Taxonomy*. 2000. url: https://www.researchgate.net/publication/2597023.

[61] Ansam Khraisat et al. "Survey of intrusion detection systems: techniques, datasets and challenges". In: *Cybersecurity* 2.1 (July 2019). doi: 10.1186/s42400-019-0038-7. url: https://doi.org/10.1186/s42400-019-0038-7.

[62] Zhen Yang et al. *A systematic literature review of methods and datasets for anomaly-based network intrusion detection*. May 2022. doi: 10.1016/j.cose.2022.102675. url: https://doi.org/10.1016/j.cose.2022.102675.

[63] Emad E. Abdallah, Wafa' Eleisah, and Ahmed Fawzi Otoom. "Intrusion Detection Systems using Supervised Machine Learning Techniques: A survey". In: vol. 201. Elsevier BV, 2022, pp. 205–212. doi: 10.1016/j.procs.2022.03.029. url: https://doi.org/10.1016/j.procs.2022.03.029.

[64] Eirini Anthi et al. "A Supervised Intrusion Detection System for Smart Home IoT Devices". In: *IEEE Internet of Things Journal* 6.5 (Oct. 2019), pp. 9042–9053. doi: 10.1109/jiot.2019.2926365. url: https://doi.org/10.1109/jiot.2019.2926365.

[65] Shaohua Teng et al. "SVM-DT-based adaptive and collaborative intrusion detection". In: *IEEE/CAA Journal of Automatica Sinica* 5.1 (Jan. 2018), pp. 108–118. doi: 10.1109/jas.2017.7510730. url: https://doi.org/10.1109/jas.2017.7510730.

[66] Pedro Casas, Johan Mazel, and Philippe Owezarski. "Unsupervised Network Intrusion Detection Systems: Detecting the Unknown without Knowledge". In: vol. 35. 7. Elsevier BV, Apr. 2012, pp. 772–783. doi: 10.1016/j.comcom.2012.01.016. url: https://doi.org/10.1016/j.comcom.2012.01.016.

[67] Levent Koc, Thomas A. Mazzuchi, and Shahram Sarkani. "A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier". In: *Expert Systems with Applications* 39.18 (Dec. 2012), pp. 13492–13500. doi: 10.1016/j.eswa.2012.07.009. url: https://doi.org/10.1016/j.eswa.2012.07.009.

[68] C. Edwin Singh and S. Maria Celestin Vigila. "Fuzzy based intrusion detection system in MANET". In: *Measurement: Sensors* 26 (Apr. 2023), p. 100578. doi: 10.1016/j.measen.2022.100578. url: https://doi.org/10.1016/j.measen.2022.100578.

[69] R. Vinayakumar et al. "Deep Learning Approach for Intelligent Intrusion Detection System". In: *IEEE Access* 7 (2019), pp. 41525–41550. doi: 10.1109/access.2019.2895334. url: https://doi.org/10.1109/access.2019.2895334.

[70] Sara Althubiti et al. "Applying Long Short-Term Memory Recurrent Neural Network for Intrusion Detection". In: *SoutheastCon 2018*. IEEE, Apr. 2018. doi: 10.1109/secon.2018.8478898. url: https://doi.org/10.1109/secon.2018.8478898.

[71] Lirim Ashiku and Cihan Dagli. "Network Intrusion Detection System using Deep Learning". In: *Procedia Computer Science* 185 (2021), pp. 239–247. doi: 10.1016/j.procs.2021.05.025. url: https://doi.org/10.1016/j.procs.2021.05.025.

[72] Yuansheng Dong, Rong Wang, and Juan He. "Real-Time Network Intrusion Detection System Based on Deep Learning". In: *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, Oct. 2019. doi: 10.1109/icsess47205.2019.9040718. url: https://doi.org/10.1109/icsess47205.2019.9040718.

[73] Hakan Can Altunay and Zafer Albayrak. "A hybrid CNNLSTM-based intrusion detection system for industrial IoT networks". In: *Engineering Science and Technology, an International Journal* 38 (Feb. 2023), p. 101322. doi: 10.1016/j.jestch.2022.101322. url: https://doi.org/10.1016/j.jestch.2022.101322.

[74] Phanindra Reddy Kannari, Noorullah Shariff Chowdary, and Rajkumar Laxmikanth Biradar. "An anomaly-based intrusion detection system using recursive feature elimination technique for improved attack detection". In: *Theoretical Computer Science* 931 (Sept. 2022), pp. 56–64. doi: 10.1016/j.tcs.2022.07.030. url: https://doi.org/10.1016/j.tcs.2022.07.030.

[75] Subhash Waskle, Lokesh Parashar, and Upendra Singh. "Intrusion Detection System Using PCA with Random Forest Approach". In: (July 2020). doi: 10.1109/icesc48915.2020.9155656. url: https://doi.org/10.1109/icesc48915.2020.9155656.

[76] Ruoyuan Zhang, Yang Song, and Xinghang Wang. "Network Intrusion Detection Scheme Based on IPSO-SVM Algorithm". In: (Apr. 2022). doi: 10.1109/ipec54454.2022.9777568. url: https://doi.org/10.1109/ipec54454.2022.9777568.

[77] Yesi Novaria Kunang et al. "Attack classification of an intrusion detection system using deep learning and hyperparameter optimization". In: *Journal of Information Security and Applications* 58 (May 2021), p. 102804. doi: 10.1016/j.jisa.2021.102804. url: https://doi.org/10.1016/j.jisa.2021.102804.

[78] Saeid Sheikhi and Panos Kostakos. "A Novel Anomaly-Based Intrusion Detection Model Using PSOGWO-Optimized BP Neural Network and GA-Based Feature Selection". In: *Sensors* 22.23 (Nov. 2022), p. 9318. doi: 10.3390/s22239318. url: https://doi.org/10.3390/s22239318.

[79] Aliya Tabassum et al. "FEDGAN-IDS: Privacy-preserving IDS using GAN and Federated Learning". In: *Computer Communications* 192 (Aug. 2022), pp. 299–310. doi: 10.1016/j.comcom.2022.06.015. url: https://doi.org/10.1016/j.comcom.2022.06.015.

[80] Zacharias Anastasakis et al. "Enhancing Cyber Security in IoT Systems using FL-based IDS with Differential Privacy". In: *2022 Global Information Infrastructure and Networking Symposium (GIIS)*. IEEE, Sept. 2022. doi: 10.1109/giis56506.2022.9936912. url: https://doi.org/10.1109/giis56506.2022.9936912.

[81] Mohanad Sarhan et al. "Cyber Threat Intelligence Sharing Scheme Based on Federated Learning for Network Intrusion Detection". In: *Journal of Network and Systems Management* 31.1 (Oct. 2022). doi: 10.1007/s10922-022-09691-3. url: https://doi.org/10.1007/s10922-022-09691-3.

[82] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *4th International Conference on Information Systems Security and Privacy (ICISSP)*. Portugal, Jan. 2018.

[83] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. SCITEPRESS - Science and Technology Publications, 2018. doi: 10.5220/0006639801080116. url: https://doi.org/10.5220/0006639801080116.

[84] Nour Moustafa. *The Bot-IoT dataset*. 2019. doi: 10.21227/R7V2-X988. url: https://ieee-dataport.org/documents/bot-iot-dataset.

[85] Muna Al-Hawawreh, Elena Sitnikova, and Neda Aboutorab. *X-IIoTID: A Connectivity- and Device-agnostic Intrusion Dataset for Industrial Internet of Things*. 2021. doi: 10.21227/MPB6-PY55. url: https://ieee-dataport.org/documents/x-iiotid-connectivity-and-device-agnostic-intrusion-dataset-industrial-internet-things.

[86] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed: 2023-9-26.

[87] Mahbod Tavallaee et al. "A detailed analysis of the KDD CUP 99 data set". In: (July 2009). doi: 10.1109/cisda.2009.5356528. url: https://doi.org/10.1109/cisda.2009.5356528.

[88] Liyan Yang et al. "Griffin: Real-Time Network Intrusion Detection System via Ensemble of Autoencoder in SDN". In: *IEEE Transactions on Network and Service Management* 19.3 (Sept. 2022), pp. 2269–2281. doi: 10.1109/tnsm.2022.3175710. url: https://doi.org/10.1109/tnsm.2022.3175710.

[89] Laylon Mokry et al. “Efficient and Privacy-Preserving Collaborative Intrusion Detection Using Additive Secret Sharing and Differential Privacy”. In: *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, Dec. 2021. doi: 10.1109/bigdata52589.2021.9671428. url: https://doi.org/10.1109/bigdata52589.2021.9671428.

[90] Georgios Spathoulas, Georgios Theodoridis, and Georgios-Paraskevas Damiris. “Using homomorphic encryption for privacy-preserving clustering of intrusion detection alerts”. In: *International Journal of Information Security* 20.3 (June 2020), pp. 347–370. doi: 10.1007/s10207-020-00506-7. url: https://doi.org/10.1007/s10207-020-00506-7.

[91] Meenakshi Mittal, Krishan Kumar, and Sunny Behal. “Deep learning approaches for detecting DDoS attacks: a systematic review”. In: *Soft Computing* 27.18 (Jan. 2022), pp. 13039–13075. doi: 10.1007/s00500-021-06608-1. url: https://doi.org/10.1007/s00500-021-06608-1.

[92] Iman Sharafaldin et al. “Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy”. In: *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, Oct. 2019. doi: 10.1109/ccst.2019.8888419. url: https://doi.org/10.1109/ccst.2019.8888419.

[93] Oussama Sbai and Mohamed El boukhari. “Data Flooding Intrusion Detection System for MANETs Using Deep Learning Approach”. In: *Proceedings of the 13th International Conference on Intelligent Systems: Theories and Applications*. ACM, Sept. 2020. doi: 10.1145/3419604.3419777. url: https://doi.org/10.1145/3419604.3419777.

[94] Faisal Hussain et al. “IoT DoS and DDoS Attack Detection using ResNet”. In: *2020 IEEE 23rd International Multitopic Conference (INMIC)*. IEEE, Nov. 2020. doi: 10.1109/inmic50486.2020.9318216. url: https://doi.org/10.1109/inmic50486.2020.9318216.

[95] Marcos V.O. Assis et al. “A GRU deep learning system against attacks in software defined networks”. In: *Journal of Network and Computer Applications* 177 (Mar. 2021), p. 102942. doi: 10.1016/j.jnca.2020.102942. url: https://doi.org/10.1016/j.jnca.2020.102942.

[96] Abdullah Emir Cil, Kazim Yildiz, and Ali Buldu. “Detection of DDoS attacks with feed forward based deep neural network model”. In: *Expert Systems with Applications* 169 (May 2021), p. 114520. doi: 10.1016/j.eswa.2020.114520. url: https://doi.org/10.1016/j.eswa.2020.114520.

[97] G.C. Amaizu et al. “Composite and efficient DDoS attack detection framework for B5G networks”. In: *Computer Networks* 188 (Apr. 2021), p. 107871. doi: 10.1016/j.comnet.2021.107871. url: https://doi.org/10.1016/j.comnet.2021.107871.

[98] George Onoufriou, Paul Mayfield, and Georgios Leontidis. “Fully Homomorphically Encrypted Deep Learning as a Service”. In: *Machine Learning and Knowledge Extraction* 3.4 (Oct. 2021), pp. 819–834. doi: 10.3390/make3040041. url: https://doi.org/10.3390/make3040041.

[99] Jiawei Yuan and Shucheng Yu. “Privacy Preserving Back-Propagation Neural Network Learning Made Practical with Cloud Computing”. In: *IEEE Transactions on Parallel and Distributed Systems* 25.1 (Jan. 2014), pp. 212–221. doi: 10.1109/tpds.2013.18. url: https://doi.org/10.1109/tpds.2013.18.

[100] Ana-Maria Ghimes et al. “Applying Neural Network Approach to Homomorphic Encrypted Data”. In: *2018 10th International Conference on Electronics, Computers and*

Artificial Intelligence (ECAI). IEEE, June 2018. doi: 10.1109/ecai.2018.8679085. url: https://doi.org/10.1109/ecai.2018.8679085.

[101] Tingting Chen and Sheng Zhong. "Privacy-Preserving Backpropagation Neural Network Learning". In: *IEEE Transactions on Neural Networks* 20.10 (Oct. 2009), pp. 1554–1564. doi: 10.1109/tnn.2009.2026902. url: https://doi.org/10.1109/tnn.2009.2026902.

[102] Ankur Bansal, Tingting Chen, and Sheng Zhong. "Privacy preserving Back-propagation neural network learning over arbitrarily partitioned data". In: *Neural Computing and Applications* 20.1 (Feb. 2010), pp. 143–150. doi: 10.1007/s00521-010-0346-z. url: https://doi.org/10.1007/s00521-010-0346-z.

[103] Ehsan Hesamifard et al. "Privacy-preserving Machine Learning in Cloud". In: *Proceedings of the 2017 on Cloud Computing Security Workshop*. ACM, Nov. 2017. doi: 10.1145/3140649.3140655. url: https://doi.org/10.1145/3140649.3140655.

[104] Karthik Nandakumar et al. "Towards Deep Neural Network Training on Encrypted Data". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2019. doi: 10.1109/cvprw.2019.00011. url: https://doi.org/10.1109/cvprw.2019.00011.

[105] Jack L. H. Crawford et al. "Doing Real Work with FHE". In: *Proceedings of the 6th Workshop on Encrypted Computing &amp Applied Homomorphic Cryptography - WAHC '18*. ACM Press, 2018. doi: 10.1145/3267973.3267974. url: https://doi.org/10.1145/3267973.3267974.

[106] Qian Lou et al. "Glyph: Fast and Accurately Training Deep Neural Networks on Encrypted Data". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020. isbn: 9781713829546.

[107] Hao Chen et al. *Logistic regression over encrypted data from fully homomorphic encryption*. Oct. 2018. doi: 10.1186/s12920-018-0397-z. url: https://doi.org/10.1186/s12920-018-0397-z.

[108] Joon Soo Yoo and Ji Won Yoon. "t-BMPNet: Trainable Bitwise Multilayer Perceptron Neural Network over Fully Homomorphic Encryption Scheme". In: *Security and Communication Networks* 2021 (Dec. 2021). Ed. by Junggab Son, pp. 1–19. doi: 10.1155/2021/7621260. url: https://doi.org/10.1155/2021/7621260.

[109] Philippe Kruchten. "Architectural Blueprints-The "41" View Model of Software Architecture". In: *IEEE Software* 12 (6 1995), pp. 42–50.

[110] Simon Brown. *The C4 model for visualising software architecture*. Feb. 2010. url: https://c4model.com/.

[111] *About the Unified Modeling Language Specification Version 2.5.1*. Dec. 2017. url: https://www.omg.org/spec/UML/.

[112] Qiao Zhang, Chunsheng Xin, and Hongyi Wu. "SecureTrain: An Approximation-Free and Computationally Efficient Framework for Privacy-Preserved Neural Network Training". In: *IEEE Transactions on Network Science and Engineering* 9.1 (Jan. 2022), pp. 187–202. doi: 10.1109/tnse.2020.3040704. url: https://doi.org/10.1109/tnse.2020.3040704.

[113] Volker Lohweg. *banknote authentication*. 2012. doi: 10.24432/C55P57. url: https://archive.ics.uci.edu/dataset/267.

[114] Gints Engelen, Vera Rimmer, and Wouter Joosen. "Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study". In: *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, May 2021. doi: 10.1109/spw53761.2021.00009. url: https://doi.org/10.1109/spw53761.2021.00009.

[115] Guandao Yang et al. “SWALP: Stochastic Weight Averaging in Low-Precision Training”. In: (2019). url: `https://api.semanticscholar.org/CorpusID:135465686`.