Universidade Estadual de Campinas
Instituto de Computação

Leonardo Henrique Neumann

Weightless Neural Network training and inferencing over encrypted data

Treinamento e inferência de Redes Neurais Sem Pesos sobre dados criptografados

CAMPINAS
2025

Leonardo Henrique Neumann

Weightless Neural Network training and inferencing over encrypted data

Treinamento e inferência de Redes Neurais Sem Pesos sobre dados criptografados

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Prof. Dr. Edson Borin
Co-supervisor/Coorientador: Prof. Dr. Diego de Freitas Aranha
Dr. Antônio Carlos Guimarães Júnior

Este exemplar corresponde à versão final da Dissertação defendida por Leonardo Henrique Neumann e orientada pelo Prof. Dr. Edson Borin.

CAMPINAS
2025

Ficha catalográfica
Universidade Estadual de Campinas (UNICAMP)
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Neumann, Leonardo Henrique, 1994-
N397w Weightless neural network training and inferencing over encrypted data / Leonardo Henrique Neumann. – Campinas, SP : [s.n.], 2025.

Orientador: Edson Borin.
Coorientadores: Diego de Freitas Aranha, Antônio Carlos Guimarães Júnior.
Dissertação (mestrado) – Universidade Estadual de Campinas (UNICAMP), Instituto de Computação.

1. Criptografia homomórfica. 2. Aprendizado de máquina. 3. Redes neurais sem peso. I. Borin, Edson, 1979-. II. Aranha, Diego de Freitas, 1982-. III. Guimarães Júnior, Antônio Carlos, 1982-. IV. Universidade Estadual de Campinas (UNICAMP). Instituto de Computação. V. Título.

Informações complementares

Título em outro idioma: Treinamento e inferência de redes neurais sem pesos sobre dados criptografados
Palavras-chave em inglês:
Homomorphic encryption
Machine learning
Weightless neural networks
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Edson Borin [Orientador]
Priscila Machado Vieira Lima
Hilder Vitor Lima Pereira
Data de defesa: 21-02-2025
Programa de Pós-Graduação: Ciência da Computação

Objetivos de Desenvolvimento Sustentável (ODS)
Não se aplica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-4409-6118
- Currículo Lattes do autor: http://lattes.cnpq.br/1853169528146353

- Prof. Dr. Edson Borin
  Universidade Estadual de Campinas
- Profa. Dra. Priscila Machado Vieira Lima
  Universidade Federal do Rio de Janeiro
- Prof. Dr. Hilder Vitor Lima Pereira
  Universidade Estadual de Campinas

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Young man, in mathematics
you don't understand things.
You just get used to them.

– John von Neumann

Agradecimentos

Primeiramente, agradeço a minha família, que me forneceu o suporte emocional e financeiro para que eu conseguisse chegar até aqui. São pessoas que sempre me incentivaram a atingir os meus objetivos, além de estarem ao meu lado nos momentos mais difíceis.

Agradeço também aos meus colegas de trabalho, em especial ao meu orientador Edson Borin, co-orientadores Diego Aranha e Antônio Guimarães, por todo o apoio e inspiração. Além de exímios profissionais, forneceram todo o suporte para concluir esta dissertação.

Por último mas não menos importante, agradeço a todos os demais amigos e colegas que contribuíram de alguma forma para este trabalho.

Este trabalho foi financiado por:

- Becas Santander / Santander Universidades
- Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

As opiniões, hipóteses, conclusões e recomendações expressas neste material são de responsabilidade do autor e não necessariamente refletem a visão das entidades acima.

Resumo

A adoção em massa de algoritmos de aprendizado de máquina trouxe preocupações dentro da comunidade de pesquisa de privacidade de dados, exigindo esforços para desenvolver técnicas de preservação de privacidade. Entre essas abordagens, a avaliação homomórfica de algoritmos de aprendizado de máquina se destaca por ser capaz de computar diretamente sobre dados encriptados, oferecendo garantias de confidencialidade robustas. Enquanto houve progresso significativo em algoritmos eficientes de criptografia homomórfica (HE) para inferência em Redes Neurais Convolucionais (CNNs), ainda não existem soluções eficientes para treinamento encriptado. As soluções atuais geralmente dependem de protocolos interativos, que, embora preservem a privacidade, impõem um enorme custo de comunicação. Essa limitação destaca a demanda por soluções de aprendizado de máquina mais rápidas que preservem a privacidade e possam manter a confidencialidade dos dados e o desempenho do modelo em uma ampla gama de aplicações.

Este trabalho apresenta uma nova abordagem para o aprendizado de máquina com preservação de privacidade por meio da avaliação homomórfica do Algoritmo de Reconhecimento de Wilkie, Stonham e Aleksander (WiSARD) (Aleksander *et al.*, 1984) e subsequentes Redes Neurais Sem Pesos (WNNs) de última geração, usando o esquema TFHE de criptografia totalmente homomórfica (FHE). Apresentamos várias contribuições, incluindo extensões para o TFHE, otimizações de parâmetros e modificações nas WiSARDs para melhorar a acurácia. Nossa abordagem permite o treinamento e a inferência baseada em FHE, juntamente com técnicas complementares, como balanceamento homomórfico.

Avaliamos nossos modelos homomórficos WiSARD em relação a abordagens de última geração em três conjuntos de dados de referência: MNIST, HAM10000 e Wisconsin Breast Cancer. Nossos resultados demonstram melhorias significativas de desempenho, alcançando níveis de latência competitivos em minutos de treinamento criptografado em comparação aos dias exigidos por trabalhos anteriores. Para o MNIST, alcançamos 91,71% de acurácia após apenas 3,5 minutos de treinamento encriptado, aumentando para 93,76% após 3,5 horas. No HAM10000, alcançamos 67,85% de precisão em apenas 1,5 minutos, aumentando para 69,85% após 1 hora. Comparado ao Glyph (Lou *et al.*, 2020), o estado da arte em treinamento homomórfico, esses resultados representam ganhos de desempenho de até 1200 vezes com uma perda máxima de acurácia de 5,4%. Para HAM10000, alcançamos até mesmo uma melhoria de acurácia de 0,65% sendo 60 vezes mais rápidos.

Nossos modelos oferecem um bom balanço entre velocidade, acurácia e preservação de privacidade. Também demonstramos a praticidade da nossa abordagem em *hardware* nível consumidor, treinando mais de 1000 imagens do MNIST em 12 minutos ou o conjunto inteiro do Wisconsin Breast Cancer em apenas 11 segundos usando um único núcleo e menos de 200 MB de memória. A nossa técnica se destaca pela flexibilidade em cenários como aprendizado distribuído, federado e contínuo. Embora ainda não alcance a acurácia das CNNs, as WiSARDs homomórficas representam um passo significativo para tornar o aprendizado de máquina baseado em FHE mais acessível para aplicações de dados sensíveis.

Abstract

The generalized adoption of machine learning algorithms has brought concerns within the data privacy research community, demanding efforts to develop privacy-preserving techniques. Among these approaches, the homomorphic evaluation of machine learning algorithms stands out for its ability to perform computations directly on encrypted data, offering robust and inherent confidentiality guarantees. While homomorphic encryption (HE) has made significant progress in enabling practical inference for Convolutional Neural Networks (CNNs), the challenge of efficient encrypted training remains unsolved. Current solutions often rely on interactive protocols, which, while preserving privacy, imposes a huge communication overhead. This limitation highlights the demand for faster privacy-preserving machine learning solutions that can maintain both data confidentiality and model performance across a wide range of applications.

This work presents a novel approach to privacy-preserving machine learning through the homomorphic evaluation of Wilkie, Stonham, and Aleksander's Recognition Device (WiSARD) (Aleksander *et al.*, 1979) and subsequent state-of-the-art Weightless Neural Networks (WNNs), using the TFHE Fully Homomorphic Encryption (FHE) scheme. We introduce several contributions, including extensions to TFHE, optimizations of cryptographic parameters, and modifications to the WiSARD algorithm to improve accuracy. Our approach enables FHE training and inference, alongside complementary techniques such as homomorphic dataset balancing.

We evaluate our Homomorphic WiSARDs against state-of-the-art approaches on three benchmark datasets: MNIST, HAM10000, and Wisconsin Breast Cancer. Our results demonstrate significant performance improvements, achieving competitive accuracy levels in minutes of encrypted training compared to days required by previous works. For MNIST, we achieve 91.71% accuracy after only 3.5 minutes of encrypted training, rising to 93.76% after 3.5 hours. On HAM10000, we reach 67.85% accuracy in just 1.5 minutes, increasing to 69.85% after 1 hour. Compared to Glyph (Lou *et al.*, 2020), the state-of-the-art in homomorphic training, these results represent speed-ups of up to 1200 times with a maximum accuracy loss of 5.4%. For HAM10000, we even achieved a 0.65% accuracy improvement while being 60 times faster.

Our models offer a compelling trade-off between speed, accuracy, and privacy preservation. We also demonstrate the practicality of our approach on consumer-grade hardware, training over 1000 MNIST images in 12 minutes or the entire Wisconsin Breast Cancer dataset in just 11 seconds using a single thread and less than 200MB of memory. The flexibility of our technique in scenarios like distributed, federated, and continuous learning is highlighted. While not yet matching the peak accuracy of CNN approaches, Homomorphic WiSARDs represent a significant step towards making FHE-based machine learning more accessible and practical for sensitive data applications.

List of Figures

List of Tables

List of Algorithms

List of Symbols

$\mathbb{B}$	The set of binary digits $\{0, 1\}$
$\mathbb{Z}$	The set of integer numbers
$\mathbb{Q}$	The set of rational numbers
$\mathbb{R}$	The set of real numbers
$\mathbb{T}$	The real torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ (real numbers modulo 1)
$\mathbb{S}_q$	The set of elements in some set $\mathbb{S}$ modulo q
$\mathbb{S}[X]$	The set of polynomials in the variable X with coefficients in $\mathbb{S}$
$\Phi_{2^d}(X)$	The 2^d-th cyclotomic polynomial in the variable X
$\mathcal{S}$	The polynomial ring $\mathbb{S}[X]/\Phi_{2^d}(X)$ in the variable X for a set $\mathbb{S}$
$\chi_{\mathbb{S}}$	A fixed probability distribution over samples in a set $\mathbb{S}$
p	Modulus for plaintext space $\mathbb{Z}_p$
q	Modulus for ciphertext space $\mathbb{Z}_q$
σ	Standard deviation of the error distribution in samples
n	Dimension of the secret key vector in LWE samples
k	Number of polynomials in RLWE samples
N	Power-of-two degree of polynomials in RLWE samples
ℓ	Levels in the gadget decomposition for RGSW samples
ℓ_{KS}	Levels in the gadget decomposition for packing key switching
β	Power-of-two gadget decomposition base for RGSW samples
β_{KS}	Power-of-two gadget decomposition base for packing key switching

Contents

Chapter 1

Introduction

The demand for Machine Learning (ML) has increased substantially in the past decade, driven by advances in machine learning techniques and the ability to leverage hardware acceleration. These developments enable the implementation of deeper architectures that can address increasingly complex problems. Notable examples include computer vision [64], natural language processing [53], and recommendation systems [70].

In response to this demand, cloud services provide solutions for training and inferencing over ML models, simplifying and often reducing the cost of resource allocation compared to hosting and maintaining hardware on-premises. While being a practical solution for many applications, cloud services also raise privacy concerns regarding both the models and the data, particularly when dealing with sensitive information.

Privacy-preserving Machine Learning (PPML) may seem contradictory, as ML involves extracting insights from data while privacy hides data. However, Fully Homomorphic Encryption (FHE) provides a solution to this dilemma. FHE enables computations on encrypted data without the need for decryption or knowledge of secret keys, allowing data to be processed in an untrusted environment without exposition.

Recent years have seen significant progress in FHE-based ML inference. Since then, the most successful approaches have been based on Convolutional Neural Networks (CNNs) [23, 35, 36], achieving near state-of-the-art precision compared to plaintext models. However, efficient and accurate encrypted training remains an open problem, as existing solutions either fall short of state-of-the-art accuracy or require prohibitively long execution times [45]. Alternative approaches like client-assisted FHE [14] offer promising results in specific contexts but come with the inherent limitations of multiparty protocols.

In this dissertation, we explore the intersection of PPML and Weightless Neural Network (WNN) architectures by proposing a novel approach: a homomorphically encrypted version of WiSARDs [3] (Wilkie, Stonham, and Aleksander's Recognition Device) using the TFHE scheme [20]. WiSARD nets offer unique advantages in terms of computational efficiency and adaptability, making them an intriguing candidate for PPML applications.

Using the TFHE scheme, we demonstrate that it is possible to efficiently train and evaluate WiSARD nets on encrypted data, preserving the privacy of both the input data and the trained model. This approach combines the efficiency of WiSARD nets with the security guarantees of fully homomorphic encryption, potentially opening new avenues for privacy-preserving machine learning applications in cloud environments.

This work explores the foundational principles of both TFHE and WiSARDs. We then present our methodology, the first to investigate this intersection and subsequently analyze its performance and security characteristics. Our goal is to contribute to the field of PPML by offering a practical and computationally efficient solution that maintains data confidentiality without significantly compromising efficiency or model accuracy.

1.1 Contributions

In this work, we focus on advancing the field of privacy-preserving machine learning through the development and evaluation of homomorphically encrypted WiSARD [3] models based on the TFHE scheme [20]. Our contributions are summarized as follows:

- We extend the TFHE scheme's functionality for encrypted data structures by developing the *Controlled Demultiplexer* (CDEMUX) gate, as well as defining the *CDEMUX Tree* and the *Inverse Vertical Packing* (IVP) techniques.
- We conduct a comprehensive search over the TFHE parameter space, focusing on improving the efficiency of the RGSW-RLWE external product operation while maintaining the required precision and a minimum security level of 128 bits.
- We generalize the bleaching technique in WiSARD models as a non-linear activation function, also introducing the logarithmic and bounded logarithmic activations, which presented interesting properties and accuracy improvements in our tests.
- We design and implement fully homomorphic WiSARD models, with homomorphically encrypted training, inference, and model activation phases.
- We develop complementary techniques that enhance the core methodology, including a privacy-preserving dataset balancing technique and strategies for distributed, federated, and continuous learning implementations.
- We perform a comprehensive evaluation of Homomorphic WiSARDs against state-of-the-art, assessing accuracy, training and evaluation times, and resource utilization.
- We demonstrate the practical applicability and efficacy of Homomorphic WiSARDs by evaluating against three datasets in the literature, including MNIST [41], HAM10000 [67], and Wisconsin Breast Cancer [68], comparing our results with existing privacy-preserving machine learning techniques.
- We analyze and quantify the trade-offs between different model activation approaches in terms of privacy preservation capabilities, computational cost, and efficiency.

The results of this work are presented in our paper titled *"Homomorphic WiSARDs: Efficient Weightless Neural Network Training over Encrypted Data"* [52], which has been accepted for publication in the *proceedings of the 23rd International Conference on Applied Cryptography and Network Security* (ACNS 2025).

1.2 Structure

The rest of this document is organized into six chapters. Chapter 4 provides an overview of related developments in FHE-based machine learning. Chapter 2 introduces notation, concepts about LWE-based cryptography, Fully Homomorphic Encryption, and the TFHE scheme. Chapter 3 provides the theoretical foundations for Weightless Neural Networks (WNNs), WiSARD nets, and presents relevant state-of-the-art techniques for WNNs. Chapter 5 contains our methodology, including contributions to the TFHE scheme, the results of our parameter search, our proposed modifications to the WiSARD algorithm, our implementation of Homomorphic WiSARDs, and discussions of additional techniques. Chapter 6 contains our experimental configurations, a comparative analysis against the state-of-the-art, a resource usage analysis, and the model activation cost analysis. Chapter 7 presents our conclusions and introduces future work.

Chapter 2

Homomorphic Encryption

Homomorphic Encryption (HE) is a cryptographic solution that allows computations to be performed on encrypted data. Its theoretical foundation dates back to a 1978 paper by Rivest, Adleman, and Dertouzos [60]. Since then, many HE schemes have been developed to provide solutions for privacy-preserving computation.

In Section 2.1, we introduce the fundamental concepts related to homomorphic encryption. In Section 2.2, we present the notation used throughout the chapter. In Section 2.3, we explain the LWE problem and the RLWE variant. In Section 2.4, we detail the TFHE scheme, the foundation for the development of our techniques.

2.1 Fundamental Concepts

A homomorphism is a structure-preserving map between two algebraic structures. In the context of homomorphic encryption, this mathematical concept is applied to create a correspondence between operations in the plaintext space and operations in the ciphertext space. This correspondence allows computations to be performed on encrypted data without decrypting it, while still yielding meaningful results when decrypted. An encryption scheme is homomorphic with respect to an operation $\diamond$ in the plaintext space if there exists a corresponding operation $\circ$ in the ciphertext space such that:

$$dec_{sk}(enc_{pk}(m_1) \circ enc_{pk}(m_2)) = m_1 \diamond m_2, \forall m_1, m_2 \in M$$

Where M is the set of possible messages, enc_{pk} is the encryption function, given a public key pk, and dec_{sk} is the decryption function, given a secret key sk. This ensures that applying the operation $\circ$ on ciphertexts and decrypting is equivalent to applying $\diamond$ on plaintext messages. Figure 2.1 represents homomorphic encryption as a category, where M and M' are messages before and after an operation and E is a functor that maps plaintext operations to the ciphertext domain, preserving their algebraic structure.

Homomorphic encryption schemes are designed to support various types of computations on encrypted data. They typically include algorithms for key generation, encryption, decryption, and evaluation of functions over ciphertexts. It is important to note that the operations on plaintexts and ciphertexts may differ significantly. For example, the addition of plaintexts might correspond to a more complex operation when performed on

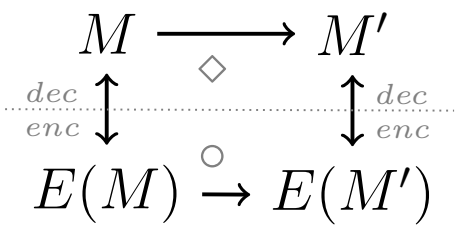

Figure 2.1: Homomorphism in encryption schemes.

ciphertexts.

Several traditional encryption schemes exhibit homomorphic properties. Examples include RSA [61], ElGamal [24], and Paillier [54]. While the Paillier cryptosystem is homomorphic over addition, RSA and ElGamal are homomorphic over multiplication.

Although they have homomorphic capabilities, these capabilities alone are not sufficient to enable arbitrary computations. To achieve that, an encryption scheme must implement a functionally complete set of operations while also allowing arbitrarily long operation chains. For instance, homomorphism over addition and multiplication enables polynomial evaluation, which in turn enables function approximation to arbitrary precision. In boolean logic, the *AND* and *XOR* gates together constitute a functionally complete set of operations, not only because they form a universal set of logic gates, but also because they correspond to multiplication and addition over the Galois field $GF(2)$, respectively.

Homomorphic encryption schemes are classified by their computational limitations, with each successive category offering broader capabilities:

1. *Partially Homomorphic Encryption (PHE)*: Implements homomorphism over at least one operation, typically either addition or multiplication.

2. *Somewhat Homomorphic Encryption (SHE)*: Implements a functionally complete set of operations, allowing one to evaluate operation chains with limited depth.

3. *Fully Homomorphic Encryption (FHE)*: Can evaluate any computable function on encrypted data, with operation chains unbounded in depth.

Beyond computational capabilities, HE schemes must provide strong security guarantees against cryptographic attacks. Consider the deterministic version of RSA: while homomorphic over multiplication, encrypting any plaintext message always produces the same ciphertext, making it vulnerable to Chosen-Plaintext Attacks (CPA) [38]. This means that if an attacker is able to encrypt carefully chosen plaintexts and observe the resulting ciphertexts, they can potentially obtain all or part of the secret key.

To be indistinguishable against CPA, most SHE and FHE schemes introduce randomness into the ciphertext, ensuring that the same message encrypted twice will, with *overwhelming probability*, result in distinct ciphertexts [27]. The major drawback is that the result of each operation slightly diverges in value with the introduced randomness, causing intermediate results in an operation chain to become increasingly imprecise. This divergence, known as noise, can grow until the message is corrupted. Consequently, the maximum length of an operation chain is limited by the noise propagation, as illustrated in Figure 2.2.

$$\begin{array}{ccccccc} \text{Plaintext} & & & & & & \text{Plaintext} \\ \text{Input} & \longrightarrow f(x) & \longrightarrow g(x) & \longrightarrow h(x) & \longrightarrow & & \text{Output} \\ = & & \overset{\text{low}}{\text{noise}} & \overset{\text{medium}}{\text{noise}} & \overset{\text{high}}{\text{noise}} & & \neq \\ \text{Ciphertext} & & & & & & \text{Ciphertext} \\ \text{Input} & \longrightarrow f'(x) & \rightarrow g'(x) & \rightarrow h'(x) & \longrightarrow & & \text{Output} \end{array}$$

Figure 2.2: Noise propagation over the operation chain.

The breakthrough leading to the first FHE scheme was achieved by Gentry in 2009 [27] with the introduction of bootstrapping. It involves creating a SHE scheme with a simple enough decryption function and then homomorphically evaluating this function to produce a clean ciphertext. This process is then used to reduce the noise in the operation chain.

Even though bootstrapping aims for simple decryption functions, it is still a computationally expensive operation in most modern FHE schemes. For this reason, they must be carefully inserted between operation chains, otherwise performance can be hindered. However, simpler functions can be evaluated without the need for bootstraps in some FHE schemes by improving the robustness of cryptographic parameters. This approach is known as leveled function evaluation and can be a practical solution in many situations.

In the next subsections, we will explore computational problems that lay the foundation for some SHE and FHE schemes, as well as going in detail over an FHE scheme that can evaluate arbitrary functions both during bootstrap and in a leveled setting.

2.2 Notation

We denote $\mathbb{S}_q^n$ as the set of vectors with n elements, where each element belongs to a set $\mathbb{S}$ modulo q. Subscripts are also used to index elements of a vector, where $s_i \in \mathbb{S}$ represents the i-th element of $s \in \mathbb{S}^n$. $\mathbb{B}$ denotes the set $\{0, 1\}$. $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ denote the sets of natural, integer, rational, and real numbers, respectively. $\mathbb{T}$ denotes the real torus $\mathbb{R}/\mathbb{Z}$, which is also $\mathbb{R}$ mod 1. $\mathbb{S}[X]$ denotes the set of polynomials in the variable X with coefficients belonging to $\mathbb{S}$. $\Phi_{2^d}(X)$ denotes the 2^d-th cyclotomic polynomial in the variable X. $\mathcal{S}$ denotes the polynomial ring $\mathbb{S}[X]/\Phi_{2^d}(X)$ in the variable X for a set $\mathbb{S}$. $\chi_{\mathbb{S}}$ denotes a fixed probability distribution over samples in a set $\mathbb{S}$.

2.3 Learning With Errors

Learning With Errors (LWE) is a computational problem introduced by Regev in 2005, which has become foundational in modern cryptography, particularly in the development of post-quantum cryptosystems. LWE involves distinguishing between random linear equations perturbed by a small amount of noise and uniformly random equations [58]. Regev demonstrated that the LWE problem is at least as hard as several worst-case lattice problems [59]. The LWE problem has served as the basis for many public-key cryptosystems, including homomorphic encryption schemes and key-exchange algorithms.

The LWE problem is defined as follows. There exists an unknown linear function $f : \mathbb{Z}_q^n \to \mathbb{Z}_q$ in the form $f(x_i) = (s \cdot x_i) + e_i$, where $s \in \mathbb{Z}_q^n$ is a secret and $e_i \sim \chi$ is a small amount of noise drawn from a discrete Gaussian distribution χ. Given a limited number of sample pairs in the form $(x_i, f(x_i)) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$:

- **Search LWE**: Find the value of s.
- **Decision LWE**: Differentiate between pairs from f and uniformly random pairs.

The security of LWE-based cryptosystems depends on the choice of parameters n, q, and the distribution χ. Larger values of n and appropriate choices of q and χ lead to stronger security but at the cost of increased computational overhead.

Lyubashevsky *et al.* [47] introduced Ring Learning With Errors (RLWE), an algebraic variant of LWE. RLWE offers improved efficiency by operating in polynomial rings, allowing for more compact representations and faster operations.

The RLWE problem is defined as follows. There exists an unknown linear function $f : \mathcal{R}_q \to \mathcal{R}_q$ in the form $f(x_i) = (s \cdot x_i) + e_i$, where $s \in \mathcal{R}_q$ is a secret, $x_i \in \mathcal{R}_q$ is chosen uniformly at random, and $e_i \sim \chi$ is a small amount of noise. Given a limited number of sample pairs in the form $(x_i, f(x_i)) \in \mathcal{R}_q \times \mathcal{R}_q$:

- **Search RLWE**: Find the value of s.
- **Decision RLWE**: Differentiate between pairs from f and uniformly random pairs.

Ongoing research in LWE and RLWE focuses on optimizing parameter selection, improving implementation efficiency, and exploring new applications in cryptography. These problems remain central to the development of quantum-resistant cryptographic protocols and homomorphic encryption schemes.

2.4 TFHE Scheme

TFHE is a FHE scheme introduced by Chillotti *et al.* [19] in 2016, built on the hardness assumptions of the LWE problem and its ring variant. It significantly improves performance over previous FHE schemes by introducing a fast bootstrap procedure, along with generalized and scale-invariant representations over the Torus.

This section introduces the algebraic foundations and fundamental algorithms underlying TFHE, which enable homomorphic evaluation of both arithmetic and arbitrary functions. Subsection 2.4.1 discusses the ciphertext types in TFHE. Subsection 2.4.2 presents the basic operations, including arithmetic, encryption, and decryption. Subsection 2.4.3 defines the CMUX gate. Subsection 2.4.4 unveils the CMUX tree method. Subsection 2.4.5 outlines Blind Rotation procedure. Subsection 2.4.6 explains Sample Extraction procedure. Subsection 2.4.7 details Vertical Packing technique. Subsection 2.4.8 describes Key Switching. Subsection 2.4.9 discloses Programmable Bootstrapping.

2.4.1 Ciphertexts

TFHE operates through three primary types of ciphertext, each serving distinct purposes in the scheme. Although the original TFHE definition employs the Torus notation, we opt for the representation over $\mathbb{Z}_q$, which is prevalent in the literature. Table 2.1 introduces the parameters in TFHE, which will be detailed in this section.

Parameter	Description
p	Modulus for plaintext space $\mathbb{Z}_p$.
q	Modulus for ciphertext space $\mathbb{Z}_q$.
σ	Standard deviation of the error distribution in samples.
n	Dimension of the secret key vector in LWE samples.
k	Number of polynomials in RLWE samples.
N	Power-of-two degree of polynomials in RLWE samples.
ℓ	Levels in the gadget decomposition for RGSW samples.
ℓ_{KS}	Levels in the gadget decomposition for packing key switching.
β	Power-of-two gadget decomposition base for RGSW samples.
β_{KS}	Power-of-two gadget decomposition base for packing key switching.

Table 2.1: Cryptographic parameters used in the TFHE scheme.

LWE is the most basic ciphertext, encrypting a scalar in $\mathbb{Z}_p$ and defined by a pair $(a, b) \in \mathbb{Z}_q^{n+1}$, where $b = \langle a, s \rangle + e$. Here, a (the basis vector) is uniformly sampled from $\mathbb{Z}_q^n$, s (the secret key) is uniformly sampled from $\mathbb{B}^n$, and e (the random noise) is sampled from a Gaussian distribution over $\mathbb{Z}$ with mean 0 and standard deviation σ.

On the other hand, *RLWE* is a composite ciphertext, which encrypts a polynomial in $\mathcal{R}_p$ and is defined by a pair $(a, b) \in \mathcal{R}_q^{k+1}$, where $b = a \cdot s + e$. In this case, a (the basis vector) is uniformly sampled from $\mathcal{R}_q^k$, s (the secret key) is uniformly sampled from $\mathcal{R}_2^k$, and e (the random noise) is sampled from a Gaussian distribution over $\mathcal{R}$ with mean 0 and standard deviation σ. A polynomial packs multiple scalar values as coefficients, allowing for batched, SIMD-like evaluation (Single Instruction, Multiple Data).

Finally, a *RGSW* ciphertext, based on the *Gentry-Sahai-Waters* scheme [28], is a special structure composed of $\ell(k+1)$ RLWE ciphertexts. It encrypts a gadget decomposition matrix $G := [[0, \frac{q}{p\beta^1}], \ldots, [0, \frac{q}{p\beta^\ell}], [\frac{q}{p\beta^1}, 0], \ldots, [\frac{q}{p\beta^\ell}, 0]]$. G represents the decomposition of a scalar $s \in \mathbb{Z}_p$ into ℓ terms, formed by powers of a given basis β. This decomposition enables a fundamental arithmetic operation that will be explained in the next subsection.

A ciphertext is said to be "fresh" if it encrypts an element within a noise variance of σ^2, which means that no operations were performed on it, except its own initialization.

2.4.2 Basic Operations

The original TFHE scheme defines a functionally complete set of operations to achieve FHE. Basic arithmetic operations involving LWE and RLWE ciphertexts are described in Table 2.2. All LWE operations are performed modulo q, whereas all RLWE operations described are performed modulo $X^N + 1$.

Operation	Operands		Output
Addition	LWE (a, b)	Scalar c	$(a, b + c)$
Subtraction	LWE (a, b)	Scalar c	$(a, b - c)$
Multiplication	LWE (a, b)	Scalar c	$(a, b \cdot c)$
Addition	LWE (a_1, b_1)	LWE (a_2, b_2)	$(a_1 + a_2, b_1 + b_2)$
Subtraction	LWE (a_1, b_1)	LWE (a_2, b_2)	$(a_1 - a_2, b_1 - b_2)$
Addition	RLWE (a, b)	Polynomial c	$(a, b + c)$
Subtraction	RLWE (a, b)	Polynomial c	$(a, b - c)$
Multiplication	RLWE (a, b)	Polynomial c	$(a, b \cdot c)$
Addition	RLWE (a_1, b_1)	RLWE (a_2, b_2)	$(a_1 + a_2, b_1 + b_2)$
Subtraction	RLWE (a_1, b_1)	RLWE (a_2, b_2)	$(a_1 - a_2, b_1 - b_2)$

Table 2.2: Basic arithmetic operations in the TFHE scheme.

Notice that no multiplication between LWE or RLWE ciphertexts is defined. Instead, an operation named *External Product* ($\boxdot$) is defined between RGSW and RLWE ciphertexts. This operation enables the element-wise multiplication between a scalar, encrypted as a RGSW ciphertext, and each coefficient in a polynomial, encrypted as a RLWE ciphertext, resulting in an RLWE ciphertext. Algorithm 1 details this operation.

Algorithm 1 RGSW-RLWE External Product ($\boxdot$).

Input: A RGSW sample C encrypting a message m_1 with noise σ_C;
Input: A RLWE sample $d = (a, b)$ encrypting a message m_2 with noise σ_d;
Output: A RLWE sample encrypting a message $m_1 \cdot m_2$ with noise σ;
Noise: $\sigma^2 \leq \sigma_d^2 + (k+1)\ell N(0.5\beta)^2\sigma_C^2 + (1 + kN)(2\beta^\ell)^{-2}$;
1: **let** $\hat{a}$ be the decomposition of a such that $a = \sum_{i=1}^{\ell} \hat{a}_i \cdot \beta^i$
2: **let** $\hat{b}$ be the decomposition of b such that $b = \sum_{i=1}^{\ell} \hat{b}_i \cdot \beta^i$
3: **return** $\sum_{i=1}^{\ell} C_i \cdot \hat{a}_i + \sum_{i=1}^{\ell} C_{\ell+i} \cdot \hat{b}_i$

Given a fresh LWE or RLWE ciphertext encrypting zero, a message can be encrypted by simple cleartext addition. An RGSW ciphertext is basically a row-wise encryption of the gadget decomposition matrix described in the last subsection. The decryption of a ciphertext, known as phase [19], is calculated as follows.

- **LWE**: Given an LWE ciphertext (a, b) and a LWE secret s, return $\left\lceil \frac{p}{q}(b - \langle a, s \rangle) \right\rfloor$.
- **RLWE**: Given an RLWE ciphertext (a, b) and a RLWE secret s, return $\left\lceil \frac{p}{q}(b - a \cdot s) \right\rfloor$.
- **RGSW**: Given an RGSW ciphertext $C = \{(a_1, b_1), \ldots, (a_{2\ell}, b_{2\ell})\}$ and a RLWE secret s, return $\left\lceil \frac{p}{q} \sum_{i=1}^{\ell}(b - a \cdot s) \right\rfloor$.

In the remainder of this section, we describe high-level constructs that can be realized by means of the above operations, enabling the TFHE scheme to achieve leveled discrete function evaluation, key switching, and bootstrapping.

2.4.3 CMUX Gate

The *Controlled Multiplexer* gate (*CMUX*) is a fundamental primitive for multiple ciphertext operations in the TFHE scheme, being characterized by three input channels and one output channel. The control input channel accepts an RGSW ciphertext that encrypts a binary message $C \in \mathbb{B}$, while the two data input channels receive RLWE ciphertexts. The CMUX gate, through homomorphic evaluation, replicates one of the two data input messages to the output channel based on the value of C. Specifically, if $C = 0$, the first input data message is routed to the output channel. If $C = 1$, the second input data message is directed to the output channel. In Figure 2.3, we depict the CMUX gate.

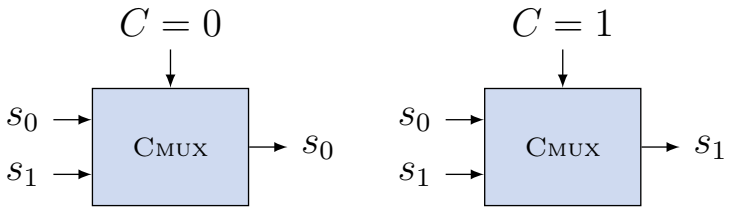

Figure 2.3: Representations of the CMUX gate with distinct control messages.

The CMUX gate enables one to secretly select between two encrypted input messages based on an encrypted control message. This functionality is achieved by using the external product operation to conditionally select between the zero element and the difference between the two encrypted input messages and then adding this value to the first encrypted input message. Specifically, given two encrypted input messages m_0 and m_1 and an encrypted control message C, the output is described by Algorithm 2.

Algorithm 2 Controlled Multiplexer Gate (CMUX).

Input: A binary RGSW sample C with noise σ_C;
Input: A RLWE sample s_0 encrypting a message m_0 with noise σ_{s_0};
Input: A RLWE sample s_1 encrypting a message m_1 with noise σ_{s_1};
Output: A RLWE sample with noise σ encrypting either m_0 or m_1;
Noise: $\sigma^2 \leq max(\sigma_{s_0}^2, \sigma_{s_1}^2) + (k+1)\ell N(0.5\beta)^2\sigma_C^2 + (1+kN)(2\beta^\ell)^{-2}$;
1: **return** $s_0 + C \boxdot (s_1 - s_0)$

2.4.4 CMUX Tree

The functionality of the CMUX gate can be extended to select between an arbitrary number of encrypted input messages. This is accomplished through a hierarchical tree-like arrangement of CMUX gates. The process begins by applying the CMUX gate between all pairs of input messages. Subsequently, CMUX operations are recursively applied to the outputs of the preceding stage, with each level of the tree utilizing a distinct control message. In Figure 2.4, we represent the CMUX tree arrangement for two control messages.

This recursive construction yields a single ciphertext resulting from the selection between all input ciphertexts. The selection of the input message is determined by the sequence of control messages used at each level of the tree. This method effectively enables

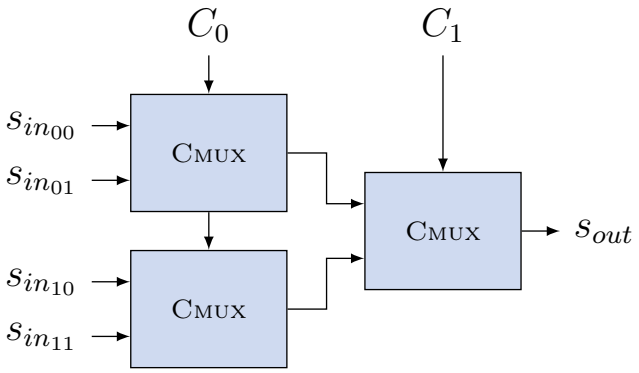

Figure 2.4: A representation of a two-level CMUX tree.

secretly indexing an array of ciphertexts based on an encrypted control message array, where each control message encrypts an individual bit from the desired index. Given an array of encrypted input messages m and an array of encrypted control messages C, the output array of the CMUX tree is described by Algorithm 3.

Algorithm 3 Controlled Multiplexer Tree (CMUXTREE).

Input: Array of d binary RGSW samples C with noise σ_C;
Input: Array of 2^d RLWE samples s encrypting an array m with noise σ_s;
Output: A RLWE sample with noise σ encrypting N messages from m;
Noise: $\sigma^2 \leq \sigma_s^2 + d((k+1)\ell N(0.5\beta)^2\sigma_C^2 + (1+kN)(2\beta^\ell)^{-2})$;

$s' \leftarrow s$
for $i \leftarrow 0$ **to** $d-1$ **do**
    $t \leftarrow 2^{d-i-1}$
    for $j \leftarrow 0$ **to** t **do**
        $s'_j \leftarrow \text{CMUX}(C_i, s'_j, s'_{j+t})$
return s'_0

2.4.5 Blind Rotation

Blind Rotation is an operation used to perform a controlled rotation on the coefficients of an encrypted RLWE ciphertext. This rotation is based on a sequence of RGSW ciphertexts encrypting binary values, referred to as the control sequence. This operation is "blind" as it does not reveal information about the ciphertext or the control sequence.

This operation exploits that multiplying a polynomial in X of degree N by X modulo $X^N + 1$ results in a counterclockwise shift of the polynomial's coefficients. Using this principle, the blind rotation algorithm strategically combines multiplications by fixed powers of X with CMUX gate operations. This results in a RLWE ciphertext that has its coefficients rotated by a total amount determined by the sequence of rotations conditionally applied according to the control sequence. The procedure is described in Algorithm 4.

We use the sequence of powers of two to achieve counterclockwise rotations, *i.e.*, $r_i = 2^i$. As multiplication occurs modulo the $2N$-th cyclotomic polynomial, rotation can also be performed in the clockwise direction employing the sequence $r_i := 2N - 2^i$, effectively reversing a rotation over a RLWE ciphertext provided the same control sequence.

Algorithm 4 Blind Rotation (BLINDROTATE).

Input: An array of d binary RGSW samples C with noise σ_C;
Input: An array of d rotation offsets r;
Input: A RLWE sample s with noise σ_s;
Output: A RLWE sample with noise σ and rotation determined by C;
Noise: $\sigma \leq \sigma_s + d((k+1)\ell N(0.5\beta)^2\sigma_C + (1+kN)(2\beta^\ell)^{-2})$;

$t \leftarrow s$
for $i \leftarrow 0$ **to** $d-1$ **do**
    $t' \leftarrow t \cdot (X^{r_i} - 1) \bmod X^N + 1$
    $t \leftarrow \text{CMUX}(C_i, t, t')$
return t

2.4.6 Sample Extraction

The *Sample Extract* operation in TFHE extracts individual LWE ciphertexts from specific coefficients of an RLWE ciphertext. Given an RLWE ciphertext $s = (a, b)$ and a coefficient position i within the range $[0, N)$, this operation produces an LWE ciphertext $s' = (a', b')$ of dimension N. The method is described in Algorithm 5.

Algorithm 5 Sample Extraction (SAMPLEEXTRACT).

Input: A RLWE sample $s = (a, b)$ encrypting a message m with noise σ_s;
Input: An index $c \in [0, N)$;
Output: A LWE sample encrypting the c-th coefficient of m with noise σ;
Noise: $\sigma^2 = \sigma_s^2$;

for $i \leftarrow 0$ **to** $k-1$ **do**
    for $j \leftarrow 0$ **to** c **do**
        $a'_{N\cdot i+j} \leftarrow a_{c-j}$
    for $j \leftarrow c+1$ **to** $N-1$ **do**
        $a'_{N\cdot i+j} \leftarrow q - a_{N+c-j}$
$b' \leftarrow b_c$
return (a', b')

2.4.7 Vertical Packing

The *Vertical Packing* (VP) technique [19] homomorphically evaluates arbitrary functions employing a combination of CMUX trees and blind rotation over RLWE ciphertexts. It uses RLWE ciphertext arrays as encrypted lookup tables, allowing the selection and decryption of individual LWE ciphertexts representing the outputs of the functions.

In VP, a sequence of RGSW ciphertexts containing elements in $\mathbb{B}$, known as the control sequence, encodes the input. Let N be the polynomial degree of the RLWE ciphertexts encrypting the lookup table entries. The control sequence is then split into two subsets: the former comprises the initial $log_2(N)$ RGSW ciphertexts, used for blind rotation, while the latter contains any remaining RGSW ciphertexts for the CMUX tree.

The CMUX tree, controlled by the second subset, selects the appropriate RLWE ciphertext from the array. This ciphertext contains the output corresponding to the

encoded input. If the second subset is empty, no RGSW ciphertexts are available after partitioning, and the look-up table is encrypted by a single RLWE. The first subset guides the blind rotation of the selected RLWE. This rotation shifts the polynomial coefficients to move the output to the first coefficient. Finally, the output is extracted as an LWE. We detail this procedure in Algorithm 6.

Algorithm 6 Vertical Packing (VERTICALPACKING).

Input: Array of d binary RGSW samples C with noise σ_C;
Input: Array of $2^d/N$ RLWE samples s encrypting array m with noise σ_s;
Output: LWE encrypting message extracted from m, with noise σ;
Noise: $\sigma^2 \leq \sigma_s^2 + d((k+1)\ell N(0.5\beta)^2\sigma_C^2 + (1+kN)(2\beta^\ell)^{-2})$;

1: $n \leftarrow log_2(N)$
2: $r \leftarrow \{2N-1, \ldots, 2N-2^{d-n}\}$
3: $s' \leftarrow \text{CMUXTREE}(\{C_n, \ldots, C_{d-1}\}, s)$
4: $s' \leftarrow \text{BLINDROTATE}(\{C_0, \ldots, C_{n-1}\}, r, s')$
5: **return** $\text{SAMPLEEXTRACT}(s', 0)$

2.4.8 Key Switching

Key Switching is a family of procedures in TFHE that performs the homomorphic evaluation of the ciphertext phase. These techniques allow for the transformation of ciphertexts encrypted under one secret key to be decryptable under a different key, while also facilitating transitions between different parameter sets. Key-switching procedures are fundamental to various ciphertext manipulations, including bootstrapping, transitioning between ciphertext types, and enabling multi-key homomorphic encryption [17].

The original TFHE [20] introduces two variants: public and private functional key switching. Both procedures enable the transition from an array of LWE ciphertexts of dimension n, encrypted under a secret key $s \in \mathbb{B}^n$, to a RLWE ciphertext encrypted under a new secret key $s' \in \mathcal{R}_q$. This transition is enabled by a specialized key switching key $ks \in \mathcal{R}_q^{2nt}$, where $t \in \mathbb{N}$ is a decomposition precision parameter. Additionally, a linear morphism $f : \mathbb{Z}_q^n \to \mathcal{R}_q$ maps the input LWE ciphertexts to the output RLWE ciphertexts. The difference between the public and private variants is that f is publicly known in the public version, whereas f is encoded as part of the key ks in the private version.

Although detailed algorithms for both functional key-switching procedures are well documented in the literature [20] and beyond the scope of this work, it is important to highlight a special case: the packing key-switching procedure. This procedure is effectively an application of public functional key switching, employing a specific linear morphism f that maps each input LWE ciphertext to sequential coefficients in the output RLWE ciphertext. In particular, both functional key switching procedures use the same gadget decomposition principles that drive the external product operation. This implies that the parameters for packing key switching, namely ℓ_{KS} and β_{KS}, share similar properties and distributions with the parameters ℓ and β, used in the RGSW ciphertexts.

2.4.9 Programmable Bootstrapping

As discussed previously, the ciphertexts in the FHE schemes incorporate randomness to ensure security. As noise accumulates during arithmetic operations, noise management is required for long operation chains, eventually requiring a reset to smaller noise levels to enable further computations. This process is known as *Bootstrapping*.

The TFHE literature contains various bootstrap techniques. In particular, one of them enables arbitrary evaluation of functions during bootstrap, which is known as *Programmable Bootstrapping* [10, 21] (PBS). It exploits the fact that TFHE's gate bootstrap is a special LUT evaluation of the identity function that also reduces noise. By composing arbitrary functions in the bootstrap, they can be evaluated for no additional cost.

Given two bootstrap precision parameters $t, t' \in \mathbb{N}$ and a bootstrap key $kb \in \mathcal{R}_q^{\ell n(k+1)^2}$, the programmable bootstrap procedure applies a morphism $f : \mathbb{Z}_t \mapsto \mathbb{Z}_{t'}$ over an LWE encrypting a message m, resulting in an LWE encrypting $f(m)$ with reduced noise.

Although other details of PBS are beyond the scope of this work, it is important to note its role in enabling function evaluation with LWE ciphertexts as input, whereas VP requires RGSW ciphertexts. This is required when performing function evaluation using the result from previous homomorphic computations. However, it is significantly more expensive than VP and is only capable of evaluating small LUTs with good performance.

Chapter 3

Weightless Neural Networks

Machine Learning (ML) is a research field focused on enabling computational systems to learn and improve based on experience rather than explicit programming. This learning process is based on observations or data, from which underlying patterns can be learned. *Brain models* stand out as a prominent family of ML algorithms, which resembles the behavior of the nervous system found in animal brains [62]. The development of these algorithms is primarily motivated by the pursuit of understanding the mechanisms of the biological brain and their ability to address complex and loosely defined problems.

The *Perceptron*, introduced by Rosenblatt (1958) [62], based on the artificial neuron model by McCulloch and Pitts (1943) [48], represents one of the earliest brain models. This model later evolved into what is now known as an *Artificial Neural Network* (ANN). On a high level, an ANN is an arrangement of artificial neurons that communicate by sending signals over numerous weighted connections. These weights are dynamically adjusted during training as the network is exposed to new data, allowing the model to approximate the behavior of arbitrary functions [63].

Around the same time, alternative brain models, such as the *N-Tuple Classifier* from Bledsoe and Browning (1959) [8], were also proposed, further contributing to the machine learning field. N-tuple classifiers, later known as *Weightless Neural Networks* (WNNs), explore a radically different approach than ANNs, and compatible with the decoding that occurs in the neurons' dendritic trees [39]. The basic principle behind WNNs involves segmenting the binary representation of the input data into small, deterministic, usually assigned groups of N bits, known as N-tuples. This segmentation breaks the input data into smaller feature combinations, recognizing the information by means of statistical correlation, which vaguely resembles how visual information is processed by animal brains.

WNNs demonstrated potential in performing classification tasks without relying on weight multiplications. *WiSARDs* [3] were among the first practical implementations of WNNs and the first to be realized as hardware devices. It also presented a modular architecture that conveyed more flexibility in training and composing different classes.

In Section 3.1, we introduce the notation used throughout the chapter. In Section 3.2, we detail Random Access Memory (RAM) devices. In Section 3.3, we present Discriminators, which are trainable recognition structures designed to address some of the limitations of RAMs. In Section 3.4, we provide an overview of the WiSARD models. In Section 3.5, we discuss some of the techniques that lay the foundation for the state-of-the-art.

3.1 Notation

We denote $\mathbb{S}_q^n$ as the set of vectors with n elements, where each element belongs to a set $\mathbb{S}$ modulo q. Subscripts are also used to index elements in a vector, which means that $s_i \in \mathbb{S}$ represents the i-th element of $s \in \mathbb{S}^n$. $\mathbb{B}$ denotes the set $\{0, 1\}$. $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ denote the sets of natural, integer, rational, and real numbers, respectively.

3.2 Random Access Memories

Random Access Memory (RAM) devices are programmable memory units that are analogous to lookup tables, capable of dynamically storing discrete functions. A RAM with a N-bit input and P-bit output space can represent any function $f : \mathbb{B}^N \mapsto \mathbb{B}^P$. RAMs can store a unique output for every possible input, making them sufficiently expressive to encode any sort of function that may fit within their representation domain.

However, this expressiveness comes with a drawback: the size of a RAM unit grows exponentially with the number of input bits and linearly with the number of output bits, which may potentially limit their practicality for functions with a large input space. To demonstrate their representation capabilities, we present the *NOR* and *XOR* logic gates encoded as binary functions, by using 2-bit RAMs with 1-bit output each, in Figure 3.1.

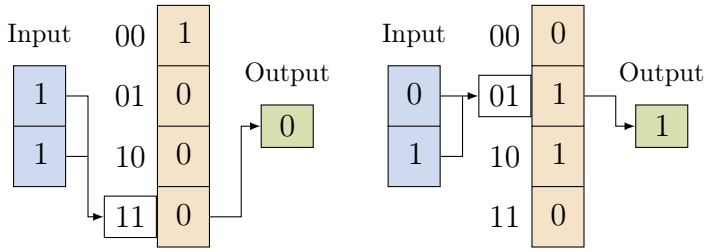

Figure 3.1: Evaluation of the *NOR* and *XOR* logic gates using RAMs.

RAMs can dynamically learn functions by updating their stored output based on input-output pairs provided during the training process. This allows them to directly store the output corresponding to each possible input combination, effectively memorizing any given function. Learning is achieved by using input values as addresses to access specific memory locations, where the desired output values are then written or updated.

This learning capability is particularly effective for functions where all possible input combinations can be explicitly provided and stored, making RAMs ideal for implementing static and deterministic functions like logic gates. However, this approach also implies that RAMs can only reproduce behaviors that have been explicitly taught during training, without the ability to interpolate outputs for any inputs not seen during training.

3.3 Discriminators

One way to overcome the interpolation limitations of RAMs and reduce the size of the model is to organize them into structures known as discriminators. In this setup, all the

s input bits are segmented into k equal-sized tuples of size a. Each a-tuple is provided as input into k distinct RAMs. Each RAM is then trained to be a binary function that determines whether a a-tuple is relevant for general pattern recognition. The contents of their positions are summed to form a score that quantifies the number of relevant a-tuples present in the input. We formalize the evaluation procedure in Algorithm 7.

Algorithm 7 Discriminator Evaluation (DiscEval).

Input: Input size s;
Input: Address size a;
Input: Evaluation sample $S \in \mathbb{B}^s$;
Input: Discriminator state $\mathcal{D} \in \mathbb{B}^{2^a \lceil s/a \rceil}$;
Output: Recognition score Φ;

$\Phi \leftarrow 0$ ▷ *reset the recognition score accumulator.*
for $r \leftarrow 0$ **to** $\lceil s/a \rceil$ **do** ▷ *loop over all RAMs in the discriminator.*
  $k \leftarrow 0$ ▷ *reset the address for the current RAM.*
  for $i \leftarrow 0$ **to** a **do** ▷ *loop over sample bits associated to RAM r.*
    $k \leftarrow k + 2^i \cdot S_{ar+i}$ ▷ *append the sample bit to the RAM address.*
  $\Phi \leftarrow \Phi + \mathcal{D}_{2^a r+k}$ ▷ *retrieve the value for the given RAM address.*
return Φ ▷ *return the final recognition score.*

As an example, we represent two evaluations of a discriminator, composed of two RAMs, using two distinct inputs in Figure 3.2.

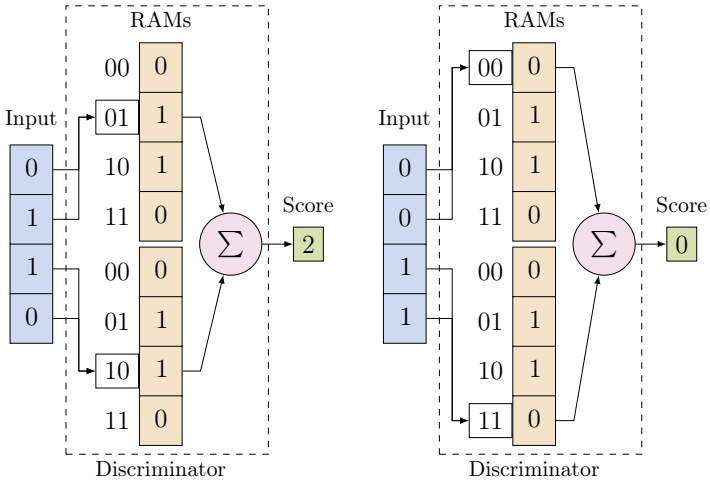

Figure 3.2: Discriminator evaluations with input size 4 and address size 2.

The training of a discriminator begins with all k RAMs' outputs set to zero. As training data are presented to the discriminator, it considers each of the k a-tuples in the sample as important combinations of features to recognize that sample. For that reason, the outputs of the k respective RAMs for these a-tuples are set to one. Repeating this adjustment over all samples, each RAM gradually learns to recognize feature combinations in the training set, becoming specialized in recognizing inputs that share similar features. As the outputs of all a-tuples are set to one, all training samples will consistently receive the maximum score. We formalize the discriminator training procedure for a single input in Algorithm 8.

Algorithm 8 Discriminator Training (DiscTrain).

Input: Input size s;
Input: Address size a;
Input: Sample $S \in \mathbb{B}^s$;
Input: Discriminator state $\mathcal{D} \in \mathbb{B}^{2^a \lceil s/a \rceil}$;
Output: Updated discriminator state $\mathcal{D}$;

for $r \leftarrow 0$ **to** $\lceil s/a \rceil$ **do** ▷ *loop over all RAMs in the discriminator.*
  $k \leftarrow 0$ ▷ *reset the address for the current RAM.*
  for $i \leftarrow 0$ **to** a **do** ▷ *loop over sample bits associated to RAM r.*
    $k \leftarrow k + 2^i \cdot S_{ar+i}$ ▷ *append the sample bit to the RAM address.*
  $\mathcal{D}_{2^a r+k} \leftarrow 1$ ▷ *update the value for the given RAM address.*
return $\mathcal{D}$ ▷ *return the updated discriminator state.*

3.4 WiSARD Nets

A standalone discriminator provides a similarity score between a sample and the set of features it represents. However, for classification problems, each class requires its own discriminator to decide how closely a sample matches the learned features of that specific class. Using a discriminator per class, the WiSARD net can generate individual scores, allowing it to determine the most likely class. The selected class is typically the one with the highest score, except in case of a tie, for which an untie criterion must be defined.

A WiSARD net consists of three main components: a permutation procedure that provides a fixed, deterministic, pseudo-random mapping between the input data and the corresponding N-tuples, shared among all discriminators; a set of discriminators, each responsible for generating a similarity score for a class in the dataset; and a scoreboard criterion that determines the classification of the input sample based on the similarity scores. We show an overview of the WiSARD net architecture in Figure 3.3.

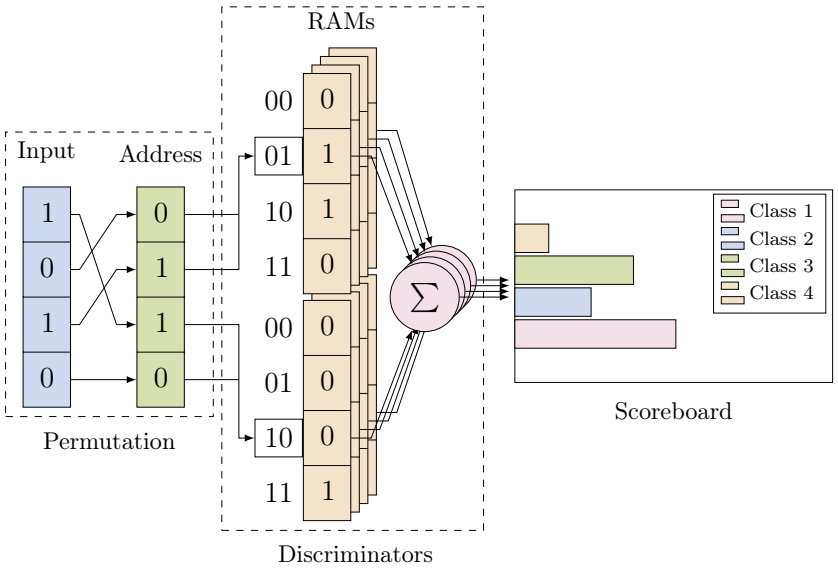

Figure 3.3: Overview of the WiSARD net architecture.

Training a WiSARD model consists in training the discriminator associated with each label from a training set. However, evaluating a WiSARD model involves evaluating all discriminators to determine the highest score for a sample. The permutation is fixed prior to training and reused throughout the lifetime of the model. Additionally, due to the incremental nature of discriminators, a WiSARD model can be trained simultaneously by multiple parties, making it suitable for distributed and federated learning.

3.5 State-of-the-Art WNNs

In this section, we explore techniques that can improve the performance of WNN models. These techniques may not only improve the overall accuracy of these models for general and particular kinds of data, but also reduce their computational requirements, making them more efficient and practical for a wider range of applications. In Subsection 3.5.1, we introduce quantization. In Subsection 3.5.2, we present the thermometer encoding. In Subsection 3.5.3, we overview the bleaching technique.

3.5.1 Quantization

Quantization [29] refers to mapping a set of input values from a continuous or otherwise large domain, such as $\mathbb{Q}$ or $\mathbb{R}$, to a discrete finite domain, such as $\mathbb{Z}_p$ for some $p \in \mathbb{N}$. This procedure enables the representation of real-world values in digital systems and can reduce the memory required to store scalar sequences. However, this mapping can be lossy and non-linear, as input values may not have exact representations in the output domain, leading to round-off errors and blurring the distinction between close values.

The most basic form of quantization allocates uniform and equal-sized intervals to represent ranges of input values in the output domain. An example of uniform quantization is the rounding function, which partitions the continuous domain of $\mathbb{R}$ into units of intervals, represented by the values in $\mathbb{Z}$. The uniform quantization is represented in Figure 3.4.

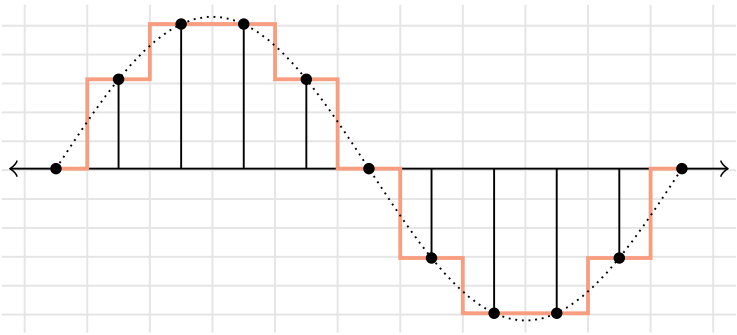

Figure 3.4: Uniform quantization applied over a continuous function (sine).

For non-uniform distributions, such as Gaussian or exponential distributions, round-off errors can be reduced by using non-uniform quantization techniques. They assign smaller ranges to more frequently occurring values in the input distribution, providing higher precision where the data density is greatest, and therefore minimizing round-off errors.

In the context of WNNs, applying quantization over the input data significantly reduces the model's memory requirements and may potentially improve the model performance, as the data are projected into a smaller, non-linear latent space that may be better recognized with smaller RAM sizes. The challenge, however, lies in the trade-off between the quantization resolution and the model's accuracy. Excessive quantization can degrade the information in the input data, negatively affecting the performance of the model.

3.5.2 Thermometer Code

Thermometer code [15, 16], also known as unary code, is a method to represent natural numbers as increasing binary sequences of ones. A thermometer code $\mathcal{T}_q : \mathbb{N} \mapsto \mathbb{B}^q$ encodes a number $x \in \mathbb{N}$ in a binary vector t containing q bits. This encoding is defined so that for all $i \in [0, x), t_i := 1$, and for all $j \in [x, q), t_j := 0$. In other words, for any natural number x less than q, its representation using a thermometer of size q starts with x ones, followed by zeros, padding the sequence to a total length of q bits.

As the bit sequence length increases linearly with the maximum value that may be encoded, thermometer encoding requires significantly more bits per value than conventional binary encoding. Therefore, this code is often used in conjunction with quantization and rescaling values to meet specific resolution requirements. The name "thermometer code" is inspired by its similarity to an analog thermometer, where the filled portion increases with increasing temperature. Table 3.1 presents numbers in decimal, binary, and horizontal thermometer representations, highlighting the number of digits required to represent them.

	Binary Code			**Thermometer Code**						
Decimal	B_0	B_1	B_2	T_0	T_1	T_2	T_3	T_4	T_5	T_6
0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
2	0	1	0	1	1	0	0	0	0	0
3	1	1	0	1	1	1	0	0	0	0
4	0	0	1	1	1	1	1	0	0	0
5	1	0	1	1	1	1	1	1	0	0
6	0	1	1	1	1	1	1	1	1	0
7	1	1	1	1	1	1	1	1	1	1

Table 3.1: Comparison between binary and thermometer codes.

In the context of WNNs, applying the thermometer code to scalar values has shown significant improvements in recognition performance, particularly when combined with a quantization technique that better preserves the characteristics of the underlying data [6]. Although recent WNN literature often discusses thermometer code as incorporating quantization techniques, we choose to address these techniques separately here for clarity.

3.5.3 Bleaching Technique

Bleaching [30] is an architectural modification of WiSARDs to adjust their sensitivity to repeated subpatterns in the training set. It consists in replacing the binary RAM cells in discriminators by integers, which now counts co-occurrences of sub-patterns over all inputs in the training set instead of only acknowledging their presence.

To preserve the original score properties in WiSARDs, a threshold operation $\phi_t(x)$ is applied to every RAM output during the inference phase, where t is a threshold value, and then the results are summed to form the score. More formally, we define $\phi_t(x) := 0$ if $x < t$, and $\phi_t(x) := 1$ if $x \geq t$. The optimal threshold value t for the best classification is heavily influenced by the statistical subpattern distributions and can be static (*i.e.*, fixed for all RAMs) or dynamically defined for each RAM in a discriminator.

In Figure 3.5, we show the architecture of a bleaching discriminator using a static threshold $t = 2$ side-by-side with the traditional binary discriminator. Note that the discriminator states do not have semantic relation besides being fed with the same input.

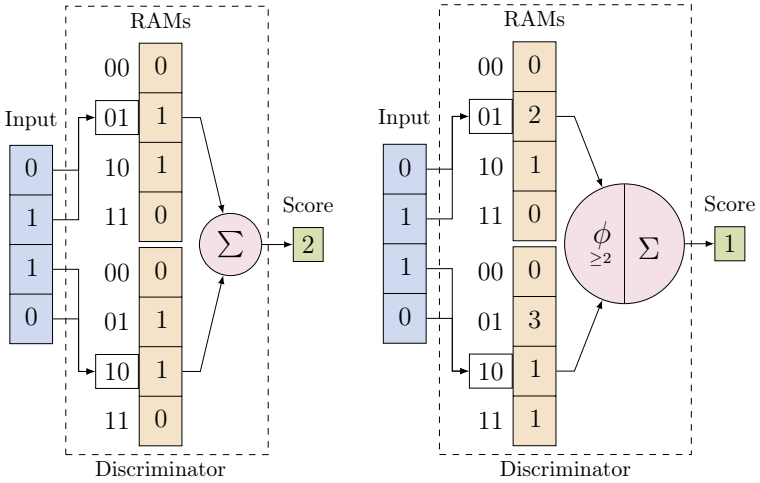

Figure 3.5: Comparison between the binary and the bleaching discriminators.

By increasing the threshold value, the bleaching technique effectively filters out noise and weak correlations, focusing on subpatterns that appear at least a minimum number of times during training. One of the key advantages of bleaching is its ability to significantly enhance the accuracy of models with smaller address sizes. This is particularly beneficial in scenarios where memory constraints limit the RAM sizes that can be used.

Chapter 4

Related Work

The field of privacy-preserving machine learning using FHE has seen significant advances in recent years. This chapter provides an overview of related developments in FHE-based machine learning models in chronological order. We outline various approaches that have pushed the boundaries of homomorphically encrypted training and inference, focusing on their methodologies and optimizations.

Bourse *et al.* (2018) [11] proposed FHE-DiNN, a framework to evaluate plain text neural networks over encrypted data using the TFHE scheme. FHE-DiNN uses bootstrapping after every neuron evaluation, making the complexity of processing each neuron independent of the network size. To ensure scale invariance, FHE-DiNN employs quantized integer weights and the sign activation function, classifying MNIST [41] with an accuracy up to 96.3% in under 1.5 seconds. Although accuracy is decreased due to quantization and sign function approximations, the authors prioritize efficient neuron-level inference.

Lou and Jiang (2019) [46] presented SHE, a shift-accumulation-based neural network for fast inference, using the TFHE scheme in a leveled setting. SHE leverages binary-friendly TFHE to implement ReLU activations and max pooling, avoiding the accuracy degradation and performance overhead of polynomial approximations. SHE adopts logarithmic quantization to replace multiplications with shifts and introduces a mixed-bit-width accumulator to optimize accumulations. SHE achieves state-of-the-art accuracy while reducing inference latency from 76.21% to 94.23% on data sets such as MNIST [41].

Park *et al.* (2020) [57] proposed the first HE-friendly algorithm for privately training a Support Vector Machine (SVM) model based on the CKKS scheme. They employed a least-squares SVM formulation and solved the linear system using a gradient descent approach, eliminating the need for expensive operations like matrix inversions or comparisons. Their implementation has shown improved performance compared to an existing secure logistic regression model in various data sets, including the Wisconsin Breast Cancer dataset [68], suggesting that SVM training with FHE is viable and could be integrated with existing homomorphic SVM inference algorithms.

Lou *et al.* (2020) [45] presented Glyph, a technique for fast and accurate training of deep neural networks on encrypted data using FHE. Glyph uses the TFHE scheme for efficient nonlinear activations and BGV for vector arithmetic in convolutional and fully connected layers. In addition, Glyph incorporates transfer learning to reduce computational overhead and improve test accuracy. Their experiments using data sets such as MNIST [41] and

HAM10000 [67] demonstrate significant speed-ups compared to previous homomorphic training approaches while achieving state-of-the-art accuracy.

Folkerts *et al.* (2021) [26] proposed REDsec, a framework that uses the TFHE scheme to perform homomorphic inference on binary and ternary neural networks. The framework contains tools for plaintext training and modules for encrypted inference. REDsec introduces optimizations for faster inference, such as novel bidirectional bridging techniques that allow it to use binary and integer arithmetic. Their evaluation shows an inference time of 3.6 seconds in MNIST [41] with 98% accuracy using a multi-GPU server.

Stoian *et al.* (2022) [65] presented a quantization-aware training methodology for neural networks compatible with the TFHE scheme, allowing for exact computations without depth limitations or specialized activation functions, using the programmable bootstrap technique. They achieve competitive accuracy in data sets such as MNIST [41]. In particular, their method simplifies FHE-compatible networks by making it a ML problem rather than requiring cryptographic expertise. While showcasing up to 9 layer networks, they acknowledge the potential for implementing networks with unbounded depth.

Montero *et al.* (2024) [50] proposed a quantized neural network training approach that uses the TFHE scheme to ensure data confidentiality throughout the training process. Their method supports both logistic regression and MLP models. Utilizing TFHE's programmable bootstrapping for activation functions and a novel rounding operator for quantization, they demonstrated competitive training accuracy with lower bit representations, while improving upon prior work in terms of speed for MLP training.

The related work presented in this chapter demonstrates the rapid progress and diverse approaches to applying FHE to machine learning. Although challenges still remain, particularly in computational efficiency, these works lay a strong foundation for advances in privacy-preserving machine learning. As this field continues to evolve, we can expect further innovations that bring FHE-based machine learning closer to practical real-world applications across various domains.

Chapter 5

Homomorphic WiSARDs

This chapter presents the methodological foundations and novel techniques developed to enable privacy-preserving machine learning using Homomorphic WiSARDs.

In Section 5.1, we introduce several key contributions to the TFHE cryptosystem, allowing more flexible and efficient operations on encrypted data structures. In Section 5.2, we detail our approach to optimize TFHE parameters, focusing on balancing performance and security in the context of the RGSW-RLWE external products. In Section 5.3, we discuss modifications to the WiSARD algorithm, introducing novel activation functions designed to enhance its compatibility with TFHE while preserving recognition capabilities. In Section 5.4, we provide a comprehensive breakdown of the training, inference, and model activation phases, each designed to operate fully within the homomorphic encryption domain. In Section 5.5, we address other techniques that complement our methodology.

5.1 TFHE Contributions

This section presents several novel techniques that extend and enhance the functionality of TFHE, enabling more flexible and efficient operations on encrypted data structures. We introduce three key innovations: the Controlled Demultiplexer Gate (CDEMUX), the CDEMUX tree, and the Inverse Vertical Packing (IVP) technique.

These contributions address important challenges in TFHE, particularly in the manipulation of encrypted arrays and lookup tables (LUTs). The CDEMUX gate provides a mechanism for conditionally positioning encrypted ciphertexts based on a binary control ciphertext, while the CDEMUX tree extends this concept to enable positioning using arbitrarily sized indices. The IVP technique builds on these foundations to enable the modification of encrypted LUTs, which complements the existing VP method used for homomorphic LUT evaluation. We note that algorithms similar in functionality to CDEMUX and CDEMUX tree have already been proposed in the literature. In Cong *et al.* [22], they introduce *HomSel* and *HomTrav* to evaluate decision trees. Our proposal, being independently developed, aims specifically at building the IVP algorithm.

Together, our proposed techniques offer new tools for constructing and manipulating encrypted data structures. In the following subsections, we detail each of these contributions, their implementations, and some potential applications.

5.1.1 CDEMUX Gate

We define the Controlled Demultiplexer Gate (CDEMUX) as the functional inverse of TFHE's Controlled Multiplexer Gate (CMUX), characterized by two input channels and two output channels. The control input channel accepts an RGSW ciphertext that encrypts a binary message $C \in \mathbb{B}$, while the data input channel receives an RLWE ciphertext. The CDEMUX gate, through homomorphic evaluation, directs the input message to the first or second output channels based on the value of C. Specifically, if $C = 0$, the input data is routed to the first output channel, while the second output channel produces a ciphertext that encrypts the zero element. In contrast, if $C = 1$, the input data is directed to the second output channel, and the first output channel produces a ciphertext encrypting the zero element. In Figure 5.1, we illustrate the CDEMUX gate.

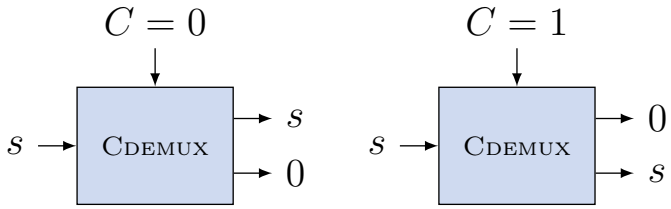

Figure 5.1: Representations of the CDEMUX gate with distinct control messages.

The CDEMUX gate effectively enables conditional routing of an encrypted input message according to the internal state of an encrypted control message. This functionality is achieved by using the external product operation to conditionally select between the zero element and the encrypted input message. Specifically, given an encrypted input message m and an encrypted control message C, the tuple containing the output of the CDEMUX gate is described by the Algorithm 9.

Algorithm 9 Controlled Demultiplexer Gate (CDEMUX).

Input: A binary RGSW sample C with noise σ_C;
Input: A RLWE sample s encrypting a message m with noise σ_s;
Output: RLWE samples with noise σ encrypting $(0, m)$ or $(m, 0)$;
Noise: $\sigma^2 \leq \sigma_s^2 + (k+1)\ell N(0.5\beta)^2\sigma_C^2 + (1+kN)(2\beta^\ell)^{-2}$;

1: $m' \leftarrow C \boxdot m$
2: **return** $(m - m', m')$

5.1.2 CDEMUX Tree

The functionality of the CDEMUX gate can be extended to accommodate an arbitrary number of output gates. This is accomplished through a hierarchical tree-like arrangement of CDEMUX gates, analogous to the CMUX tree structure employed by the Vertical Packing technique. The process begins with a single CDEMUX operation. Subsequently, CDEMUX operations are recursively applied to both outputs of the preceding stage, and each level of the tree utilizes a distinct control message. In Figure 5.2, we represent the CDEMUX tree arrangement for two controlling RGSW ciphertexts.

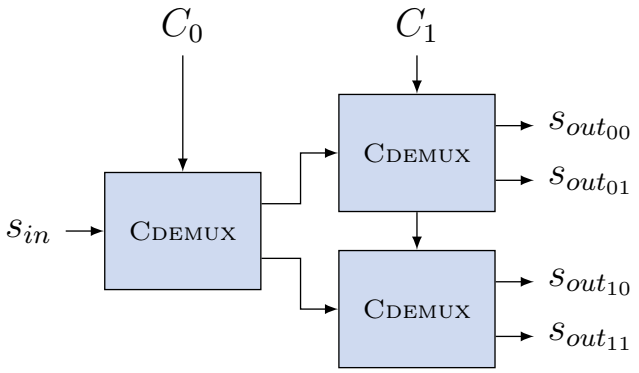

Figure 5.2: A representation of a two-level CDEMUX tree.

This recursive construction yields an array of ciphertexts that is predominantly encrypting zero elements, except for one specific position that contains the message routed from the input data channel. The position of the input message within the array is determined by the sequence of control messages used at each level of the tree. This method enables the creation of an array of any desired size, providing precise control over the initialization of a single secret position within the array. Control messages can be seen as a sequence of routing instructions that guide the input data to its designated location. Specifically, given an encrypted input message m and an array of encrypted control messages C, the output array of the CDEMUX tree is described by the Algorithm 10.

Algorithm 10 Controlled Demultiplexer Tree (CDEMUXTREE).

Input: An array of d binary RGSW samples C with noise σ_C;
Input: A RLWE sample s encrypting a message m with noise σ_s;
Output: An array of 2^d RLWE samples with noise σ, one encrypting m;
Noise: $\sigma^2 \leq \sigma_s^2 + d((k+1)\ell N(0.5\beta)^2\sigma_C^2 + (1+kN)(2\beta^\ell)^{-2})$;

$s'_0 \leftarrow s$
for $i \leftarrow d-1$ **downto** 0 **do**
    $t \leftarrow 2^{d-i-1}$
    for $j \leftarrow 0$ **to** t **do**
        $(s'_j, s'_{j+t}) \leftarrow \text{CDEMUX}(C_i, s'_j)$
return s'

5.1.3 Inverse Vertical Packing

The *Inverse Vertical Packing* (IVP) technique addresses an important challenge in the TFHE cryptosystem: secretly modifying specific coefficients within arrays of RLWE ciphertexts. While the standard Vertical Packing (VP) allows for element retrieval in LUTs, IVP complements this by allowing the modification of elements in LUTs, enhancing the flexibility of homomorphic operations on encrypted data structures.

Specifically, IVP combines the blind rotation operation over RLWE ciphertexts with the CDEMUX tree procedure to create an encrypted single-valued LUT that follows TFHE's vertical packing encoding. Single-valued LUTs are arrays of RLWE ciphertexts composed of zeroed coefficients except for one secretly designated coefficient.

In IVP, a sequence of RGSW ciphertexts containing binary elements, *i.e.*, elements in $\mathbb{B}$, known as the control sequence, encodes the target address. Let N be the polynomial degree of the initial RLWE ciphertext encoding the single value in its first coefficient. The control sequence is then split into two subsets: the former comprises the initial N RGSW ciphertexts, used for blind rotation, while the latter contains any remaining RGSW ciphertexts used to control the CDEMUX tree.

The first subset guides the blind rotation of the selected RLWE ciphertext. This rotates the polynomial coefficients to move the input coefficient to a position defined by the sequence of RGSW ciphertexts encoding the index. Then, the CDEMUX tree, controlled by the second subset, routes the RLWE message into the target range inside the array. If the second subset is empty, no RGSW ciphertexts are available after the split, implying that the resulting look-up table should contain a single RLWE ciphertext, which is already positioned from the previous step. We detail this procedure in Algorithm 11.

Algorithm 11 Inverse Vertical Packing (InverseVerticalPacking).

Input: An array of d binary RGSW samples C with noise σ_C;
Input: A RLWE sample s encrypting a single message m_0 with noise σ_s;
Output: Array of $2^d/N$ RLWE samples with noise σ, one encrypting m_0;
Noise: $\sigma \leq \sigma_s + d((k+1)\ell N(0.5\beta)^2\sigma_C + (1+kN)(2\beta^\ell)^{-2})$;

1: $n \leftarrow log_2(N)$
2: $r \leftarrow \{1, \ldots, 2^{d-n}\}$
3: $s' \leftarrow \text{BlindRotate}(\{C_{n-1}, \ldots, C_0\}, r, s)$
4: $s' \leftarrow \text{CdemuxTree}(\{C_{d-1}, \ldots, C_n\}, s')$
5: **return** s'

The IVP technique's ability to create single-valued LUTs opens up powerful possibilities when combined with other TFHE operations, particularly for updating existing LUTs. Updating an existing LUT can be achieved in a few steps: First, we apply the VP technique to select a target element, then we extract it to an LWE sample for arbitrary manipulation; Then, we compute the difference between the original and modified values, use IVP to reposition this difference and finally update the original LUT by using RLWE addition. Moreover, we may generate LUTs of arbitrary complexity simply by adding multiple single-valued LUTs together, enabling the secret positioning of its elements.

5.2 TFHE Parameter Search

In TFHE, the choice of cryptographic parameters dictates the execution time and noise propagation of the operations, allowing for trade-offs between output precision and performance. To optimize the performance of the RGSW-RLWE external product operation while preserving the precision required for accurate evaluation, we conducted a comprehensive exploration of the TFHE parameter space, specifically focusing on N, σ, ℓ and β. This emphasis on the external product stems from its significant contribution to the overall execution time of both the VP and IVP methods.

To take advantage of the optimized RLWE ciphertext multiplication offered by the MOSFHET library [32], which utilizes Fast Fourier Transforms (FFT), we restrict our search

to RLWE instances with a single high-degree polynomial by fixing $k = 1$. Furthermore, given the characteristics of the FFT algorithm, we also restrict our search for values of N that are powers of two. The MOSFHET library supports values of N between 256 and 4096. Taking into account our restrictions for N and k, we estimated the normalized noise standard deviation (σ) for each value of N by using the Lattice Estimator library [2], to ensure a minimum 128-bit security. These values of σ are described in Table 5.1.

N	std. dev. (σ)
256	1.00×2^{-5}
512	1.15×2^{-11}
1024	1.85×2^{-25}
2048	1.10×2^{-51}
4096	1.00×2^{-104}

Table 5.1: Estimated noise standard deviation σ for each value of N.

Taking into account all combinations of ℓ and $log_2(\beta)$ satisfying the constraint $\ell \cdot log_2(\beta) \leq 54$, where 54 is the number of usable bits in each ciphertext component defined by the MOSFHET implementation, we then estimate the upper bound for the noise propagation of the average case external product, based on Corollary 3.14 from TFHE [20]. This analysis determines values of $log_2(\beta)$ that minimize the noise variance for each choice of ℓ and N, which, in turn, maximize the precision of these parameter choices, enabling the selection of computationally cheaper sets. Our findings are visualized in Figure 5.3.

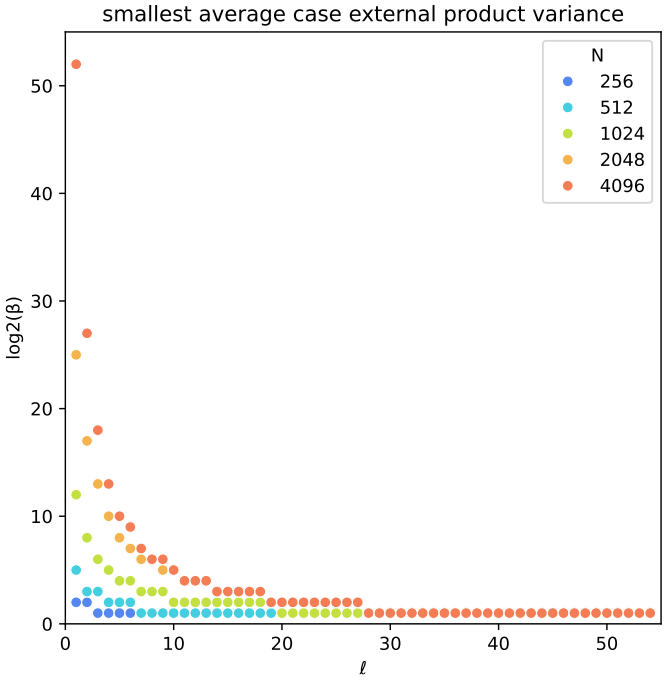

Figure 5.3: Optimal values of $log_2(\beta)$ for each choice of ℓ and N.

We conducted exhaustive testing of the parameter sets using the MOSFHET library, observing a minor discrepancy between theoretical predictions and experimental results. In particular, this deviation is marginal, being no more than two units for smaller values of ℓ, and can be explained by implementation details, such as numerical imprecision in FFTs.

5.3 WiSARD Activation Functions

The literature on WNNs frequently emphasizes on minimizing resource usage, as these models are often employed as a cost-effective alternative to traditional ANNs. However, in our use case, many of these techniques are not applicable due to the constraints of HE. Consequently, we investigated several modifications to WiSARDs to enhance their compatibility with FHE while preserving their classification performance.

Our approach maintains a balance between preserving the efficiency for which WNNs are known and adapting them to work within the cryptographic environment. In Table 5.2, we summarize the functions we investigated to activate the RAM outputs. In the remainder of this section, we detail our investigation and the improvements we achieved.

approach	**function**	**observation**
Binary	$x \geq 1$	From original WiSARD paper
Bleaching	$x \geq t$	From bleaching technique
Linear	x	Biased towards frequent sub-patterns
Logarithmic	$log_2(x+1)$	Consistent accuracy across permutations
Bounded Log	$min(log_2(x+1), c)$	Consistent accuracy with improvements

Table 5.2: Activation functions we investigated using our WiSARD models.

5.3.1 Linear Integer Models

Building upon WiSARD models that employ the bleaching technique, we aimed to mitigate the computational costs associated with nonlinear functions, particularly the threshold operation, as they are expensive to evaluate in TFHE and require bootstraps.

Our initial approach involved completely removing these operations and instead calculating scores by directly summing the counters from each RAM output. However, this modification alone has proven to be ineffective, as the models became severely biased towards the most frequently occurring patterns observed during the training phase, ultimately leading to poor overall classification performance. This outcome suggests that non-linearity plays a critical role in maintaining accurate input classification.

5.3.2 Non-linear Integer Models

Drawing some inspiration from ANNs, we reconceptualized the threshold operation as an instance within a broader class of non-linear activation functions. This generalization

led us to explore alternative forms of score composition that could potentially improve recognition performance while remaining compatible with existing bleaching models.

As discussed earlier, adding RAM counter values for the score composition did not yield improved accuracy. Our hypothesis is that this method overemphasizes frequently occurring feature combinations during inference, obscuring the capacity to recognize less frequent, yet relevant, patterns. However, it is important to note that frequently seen features can indicate strong correlation, which could improve recognition performance.

Considering that, we explored the use of the magnitude of RAM counters instead of their raw values. To quantify this magnitude for positive integer values, we used the function $f(x) = log_2(x + 1)$, since log_2 is not defined for $x = 0$. This method, designated as logarithmic (*log*) activation, produced models with an accuracy almost as good as our bleaching models for our experimental data sets without requiring threshold adjustments. In addition, they maintained consistent accuracy even with varying input permutations.

Further experimentation revealed that defining an upper bound on the *log* activation, specifically employing $f(x) = min(log_2(x + 1), c)$ with a small constant $c > 0$, resulted in even better recognition performance. This approach, which we name bounded logarithmic (*bounded-log*) activation, demonstrated superior accuracy compared to bleaching models when utilizing smaller WiSARD parameters for our experimental datasets. Furthermore, it maintained comparable accuracy levels with larger WiSARD parameters. In particular, the consistent accuracy observed with *log* activation was retained in this variant.

5.3.3 Activations as Ensembles

Interestingly, non-linear activation functions can be interpreted as particular cases of ensembling of bleaching models, where multiple WiSARDs trained with the same WiSARD parameters and inputs, using distinct threshold values, have their scores summed to form a stronger classification score. For example, five models using thresholds of 1, 2, 4, 8, and 16 would produce a score equivalent to a single WiSARD model using the *b-log* activation function, as a score point is added for each magnitude level a RAM counter can reach.

This ensemble equivalence explains some of the properties observed in our empirical experimentation and suggests that other activation functions may simulate different ensemble configurations, potentially improving classification for specific problems.

5.4 Homomorphic WiSARDs

This section details the core components of our proposed Homomorphic WiSARD model, which enables pattern recognition on encrypted data without compromising privacy. We break down the process into three phases: the training phase, where the model learns from encrypted samples; the inference phase, where it classifies new encrypted inputs; and the model activation phase, which determines the final classification output. Each phase is designed to operate fully within the homomorphic encryption domain, ensuring data confidentiality throughout the machine learning pipeline. By explaining these phases, we illustrate how traditional WiSARD concepts are adapted to work with encrypted data, highlighting the unique challenges and solutions in this privacy-preserving paradigm.

5.4.1 Training Phase

The training phase begins on the client side. This step involves applying quantization and thermometer encoding to the input data. Fundamentally, for each sample, we employ only preprocessing techniques that are independent of the knowledge of the entire dataset. Although techniques such as average or min-max normalization could improve our recognition performance, they would require data knowledge that is not always available in scenarios where the data are already encrypted or in cases such as distributed, federated, or continuous learning [56, 69]. The client then encrypts each sample with its respective label bit-by-bit as RGSW ciphertexts and sends this data to the server.

At the start of the training phase, the server initializes an array of $u(2^n)$ RLWE samples, referred to as the model state, where n is the sum of the address and the label sizes and u is the number of RAMs in the model. For each pair of address and respective label, the server then concatenates the RGSW ciphertexts as a single array. Then, it executes the IVP procedure over a special RLWE ciphertext, whose first coefficient contains an encryption of the number one, followed by the rest of the coefficients encrypting zeros, using this concatenated array as the control sequence. This yields a RLWE array of size 2^n, which is then summed to the slice of the model state that corresponds to the RAM that the specific address is targeting. For each address in the sample, each RAM will be trained exactly once, which means that a total of u IVP evaluations per sample are performed. We formalize the training procedure for an individual sample in Algorithm 12. Repeating this step for each sample, the training phase is then completed.

Algorithm 12 Homomorphic WiSARD Training (HomWisardTrain).

Input: TFHE parameter N;
Input: Input size s; Address size a; Label size l;
Input: Number of RAMs $k := \lceil s/a \rceil$;
Input: Label set $\mathcal{L}$, discretely mapped to $[0, 2^l)$;
Input: An array of $k(a+l)$ binary RGSW ciphertexts C (the sample);
Input: An array of $k2^{a+l-log_2(N)}$ RLWE ciphertexts $\mathcal{W}$ (the model state);
Output: Updated model state $\mathcal{W}$;

$m \leftarrow \{1, 0, 0, \ldots, 0\}$
for $r \leftarrow 0$ **to** k **do**
    $m' \leftarrow$ InverseVerticalPacking$(\{C_{r(a+l)}, \ldots, C_{(r+1)(a+l)-1}\}, m)$
    $e \leftarrow r2^{a+l-log_2(N)}$
    for $i \leftarrow 0$ **to** $2^{a+l-log_2(N)}$ **do**
        $\mathcal{W}_{e+i} \leftarrow \mathcal{W}_{e+i} + m'_i$
return $\mathcal{W}$

5.4.2 Inference Phase

The inference phase begins on the client side. Similarly to the training phase, this step involves applying quantization and thermometer encoding to the input data. The client then encrypts each sample bit by bit as RGSW ciphertexts and sends them to the server.

The server, possessing the model state, evaluates the VP procedure using the RGSW

ciphertexts provided by the client, which is now evaluated $u \cdot l$ times, being u the number of RAMs and l the number of labels in the model. Note that despite executing the $u \cdot l$ VP evaluations, as opposed to the u IVP evaluations in the training phase, each of these VP evaluations is performed at a smaller depth, with precisely $\lfloor log_2(l) \rfloor$ fewer bits. This procedure is formalized for a single sample in Algorithm 13. This results in $u \cdot l$ RAM scores that are then used in the model activation phase, described in the next subsection.

Algorithm 13 Homomorphic WiSARD Evaluation (HomWisardEval).

Input: TFHE parameter N;
Input: Input size s; Address size a; Label size l;
Input: Number of RAMs $k := \lceil s/a \rceil$;
Input: Label set $\mathcal{L}$, discretely mapped to $[0, 2^l)$;
Input: An array of ka binary RGSW ciphertexts C (the sample);
Input: An array of $k2^{a+l-log_2(N)}$ RLWE ciphertexts $\mathcal{W}$ (the model state);
Output: An array of lk LWE ciphertexts Φ (associated RAM counts);

1: **for** $b \in \mathcal{L}$ **do**
2: **for** $r \leftarrow 0$ **to** k **do**
3: $c \leftarrow r(a+l)$
4: $c' \leftarrow (r+1)(a+l) - 1$
5: $e \leftarrow r2^{a+l-log_2(N)}$
6: $e' \leftarrow (r+1)2^{a+l-log_2(N)} - 1$
7: $\Phi_{bk+r} \leftarrow$ VerticalPacking$(\{C_c, \ldots, C_{c'}\}, \{\mathcal{W}_e, \ldots, \mathcal{W}_{e'}\})$
8: **return** Φ

5.4.3 Model Activation Phase

The activation phase has the goal of applying the activation function f_{act} over the individual RAM scores of a sample and combining them to form the discriminator scores, which are then compared to determine the most likely class for a sample. In this subsection, we define three approaches for the score composition, summarizing them in Table 5.3.

approach	evaluation cost	model privacy	continuous learning
PD-act	very low	!	✔
OTF-act	high (inference)	✔	✔
FM-act	amortized low	✔	✘

Table 5.3: Comparison between model activation approaches.

Post-Decryption Activation – PD-act

Nonlinear operations, such as `argmax`, are common at the end of neural network inference processes. When evaluating these processes homomorphically, a typical strategy is to send all inputs for nonlinear operations to the client, who decrypts them and performs

the operation in cleartext. This approach avoids the complexity of evaluating nonlinear operations homomorphically but exposes intermediate computation results to the client, which can potentially reveal information about the training data.

This principle can be applied to the model activation in Homomorphic WiSARDs. The server can perform inference, retrieve encrypted RAM output, and send them to the client. The client then decrypts the scores and completes the model activation by applying f_{act}, summing the scores, and determining the class with the highest score. Although the simplest and most cost-effective method for activations, it presents two challenges:

- Output size: Models may comprise numerous RAMs (up to 6860 in our MNIST experiments), each producing an encrypted score. Initially, this would require many LWE ciphertexts, potentially using 26.8 MiB of memory. To reduce the resulting array size, we can employ the packing key switch procedure [20] to pack all scores into a single RLWE ciphertext, decreasing usage to 131 KiB.
- Model privacy: Exposing individual RAM results during evaluation allows the client to gather information about the model, potentially enabling its reconstruction.

Although the PD-ACT approach offers a cost-effective solution for model activation, it requires careful consideration of the output size and privacy requirements.

On-the-Fly Activation – OTF-ACT

Building upon the previous activation method, we now explore approaches that perform the model activation entirely on the server side. Our second approach for performing the activation utilizes TFHE's programmable bootstrapping (PBS) to evaluate f_{act} over the RAM scores during inference. The process unfolds as follows.

1. Perform the inference process, obtaining the encrypted scores of each RAM cell.
2. Use PBS to apply f_{act} to encrypted scores.
3. Add the activated scores to calculate each discriminator's score.
4. *(Optional)* Evaluate the homomorphic `argmax` function.
5. Send the encrypted class scores or the inferred class to the client.

The main drawback of this approach is that performing multiple PBS operations for each inference can be computationally expensive, potentially increasing latency and implementation costs. However, it offers several advantages over the PD-ACT method:

- More privacy: The model's internal states remain fully encrypted, revealing only the final prediction to the client.
- No training cost: The method does not require changes to the training process, which is desirable for distributed, federated, and continuous learning scenarios.

- Reduced communication overhead: Only scores or the final prediction are transmitted, minimizing data transfer requirements.

The suitability of this method depends on the specific application. For use cases where privacy is critical, OTF-ACT could be the ideal choice. For example, in medical diagnosis systems where patient data confidentiality is crucial, this method could provide the necessary privacy guarantees. Additionally, the computational overhead for OTF-ACT can be reduced with future research, optimizing PBS operations and potentially making this approach more viable for general applications.

Full Model Activation – FM-ACT

The Full Model Activation approach presents a distinctive strategy for model activation. This method involves using PBS to evaluate f_{act} over each element of each RAM throughout the model. Unlike previous methods that activate during or after inference, FM-ACT performs activation after training. The activation process is performed as follows.

1. Train the Homomorphic WiSARD model.
2. Use PBS to apply f_{act} to every element in the model state.
3. Store the pre-activated model for future use.

After precomputation of the activated scores, inference proceeds as follows.

1. Perform the inference process, obtaining the encrypted preactivated scores of each corresponding RAM cell.
2. Add the activated scores to calculate each discriminator's score.
3. *(Optional)* Evaluate the homomorphic `argmax` function.
4. Send the encrypted class scores or the inferred class to the client.

Compared to PD-ACT and OTF-ACT, FM-ACT is significantly more expensive and hinders implementation in distributed, federated or continuous learning scenarios. However, it offers several advantages over the previous approaches:

- Even more privacy: The model is fully protected, and the RAM scores cannot be exposed even after a leak, as pre-activation occurs before any inference.
- Reduced training cost: Noise removal after training with PBS may allow the use of cheaper cryptographic parameters, accelerating the training phase.
- Reduced inference cost: Activation is performed only once per model, which does not affect subsequent inference times. Additionally, pre-activated scores with removed noise may allow the use of cheaper parameters, accelerating the inference phase.

The FM-ACT approach could be particularly beneficial in scenarios where a model is trained once and then deployed for long-term use with strict privacy requirements. For example, it might be ideal for a medical diagnosis system that uses a fixed and thoroughly validated model. As in OTF-ACT, the computational overhead for FM-ACT can be reduced with future research, in particular with novel batched PBS techniques.

5.5 Additional Techniques

The design of the Homomorphic WiSARD algorithm facilitates the straightforward implementation of various techniques commonly required for PPML training and inference. This section will discuss some of these techniques.

5.5.1 Federated Learning

Federated Learning is a collaborative machine learning approach in which multiple entities train a single model together without sharing their data [69]. This scenario presents unique privacy challenges that Homomorphic WiSARDs are well suited to address.

In federated learning, each participating entity has its own privacy concerns and refuses to share input data with others. This requires techniques such as threshold homomorphic encryption [4] or multi-key homomorphic encryption [17]. These methods enable computation over data encrypted with different or jointly generated keys, with decryption requiring participation from all involved parties. Similar privacy requirements arise in other scenarios, such as encrypted inference in public clouds, where the input data owner and model owner may be distinct entities seeking to protect their assets.

Homomorphic WiSARDs offer several advantages for federated and distributed learning:

- Trivial merging: Homomorphic WiSARDs allow for merging models trained by distributed or federated entities through simple model state addition, if they share the same parameters. This enables performance equivalent to centralized training.
- Scalability: Both distributed and federated approaches for Homomorphic WiSARDs scale well with an increasing number of entities, as the merging process remains computationally efficient regardless of the number of participants.
- Continuous learning: The model state in Homomorphic WiSARDs can be continuously updated as new training samples are presented, with a similar effect to having these samples trained all at once. This facilitates asynchronous collaborative updates from distributed or federated entities, allowing parties to send differential updates.
- Threshold HE compatibility: Only the programmable bootstrapping performance is affected by threshold HE, while other predominant procedures such as VP and IVP remain largely unaffected. This can be implemented using methods such as proposed by Lee *et al.* [42], which supports multiple parties with minimal overhead.

5.5.2 Dataset Balancing

The class imbalance represents a significant challenge for machine learning, particularly in privacy-preserving contexts where data access is restricted. This section explores the existing solutions for imbalanced datasets and proposes a novel balancing technique.

A data set with n samples associated with c classes is considered balanced if each class is represented by approximately $\lceil n/c \rfloor$ samples. More formally, we can define a balance ratio $r_i = c \cdot n_i/n$ for each class i, with perfect balance achieved when $r_i = 1$. Although

minor imbalances ($0.8 \leq r_i \leq 1.2$) are generally not problematic, significantly unbalanced data sets pose a substantial challenge for many ML algorithms. Training a classifier on an unbalanced data set often results in biased models that may exhibit artificially high accuracy on the majority class while performing poorly on minority classes. This bias comes from the model's tendency to overfit the dominant class, failing to learn the patterns necessary for accurate classification across all classes [1, 40].

Several techniques have been developed in the literature to address class imbalance [34], most of them falling into more general concepts, described below.

- Undersampling: Remove samples from the majority classes until all classes are equally represented. Although simple, it may lead to the loss of valuable information.
- Oversampling: Duplicate samples from minority classes. Although it preserves information, it may introduce bias from repeated samples.
- Data augmentation: Create samples using transformations. Although it increases variability, it requires domain-specific knowledge and can introduce artificial patterns.
- Algorithm adjustment: Modify learning algorithms to be more sensitive to minority classes. Does not modify the dataset, but can have similar issues to oversampling.

In conventional neural networks, algorithm adjustment often involves scaling the learning rate linearly according to the class represented by the sample, thus increasing its impact during training. For WNNs, the sensitivity to specific samples can be adjusted by applying a scaling factor to the increments applied to the associated RAM cells.

However, these traditional approaches present challenges in privacy-preserving machine learning scenarios. Many balancing techniques require knowledge of the dataset's class distribution, which may not be available when working with encrypted or distributed data. In federated learning contexts, where data are dispersed across multiple parties, applying consistent balancing techniques becomes even more complex.

To address these limitations, we propose an alternative that is FHE-friendly, inspired by the scaling technique described for WNN algorithm adjustment. It works as follows.

- During the training phase, we maintain a separate array of RLWE ciphertexts and use the IVP technique to count occurrences of each label in an encrypted manner, reusing the same encrypted label provided by the client for each sample.
- For PD-ACT, during the inference phase, we share the encrypted label counts with the client, who will use this information to dynamically adjust the scores before evaluating the rest of the circuit.
- For OTF-ACT, during the inference phase, we dynamically adjust the output scores using encrypted label counts.
- For FM-ACT, the same process proposed for OTF-ACT is used, except that the balancing occurs after the training phase.

Our homomorphically encrypted balancing technique, unlike other approaches in the literature, does not involve previously modifying the dataset, being also applicable in distributed, federated, and continuous learning scenarios, for the model activation approaches that are compatible. The solution for PD-ACT has obvious privacy concerns that must be considered, but the labels are never exposed to the server. However, the solutions for OTF-ACT and FM-ACT preserve privacy for both the client and the server.

Furthermore, solutions for both OTF-ACT and FM-ACT can be implemented without additional PBS evaluations by composing the f_{act} activation function with the balancing technique. As introduced by Guimarães *et al.* [31], and pointed out by Chillotti *et al.* [21], functions to be evaluated using PBS can be dynamically created using encrypted data, which allows both function composition and multivariate evaluation.

5.5.3 Input Compression

A notable challenge in our method is the significant ciphertext expansion of encrypted input samples compared to other neural network training and evaluation approaches. In homomorphic encryption, ciphertext expansion refers to the ratio of the ciphertext size to the size of the data it encrypts. This subsection explores the challenges and potential solutions related to ciphertext expansion in our implementation.

Our approach uses RGSW ciphertexts to encrypt the sample and the label bits. Although this enables fast VP and IVP evaluations, it also results in considerable ciphertext expansion, as each RGSW is composed of $\ell(k+1)$ RLWE ciphertexts, which is amplified by bit-by-bit decomposition of both the addresses and labels. Quantitatively, each RGSW expands each input bit in a rate of $\ell(k+1)log_2(q)N$.

However, computing Homomorphic WiSARD circuits using RGSW ciphertexts generates significantly less noise, thus requiring a much smaller ciphertext modulus q. Our approach also does not require batched input samples, which is beneficial for scenarios such as distributed or federated learning where data availability is restricted. In contrast, RLWE batched approaches, while generally yielding smaller expansions, are dependent on circuit depth. For reference, Legiest *et al.* [43] reported an expansion factor of $2 \cdot 389/4 \approx 194$ for a CNN in MNIST using 4-bit linear quantization.

There are some potential solutions to mitigate ciphertext expansion for the TFHE scheme. In particular, transciphering offers complete mitigation of ciphertext expansion [7, 51]. Alternative approaches are to pack multiple bits as coefficients in RLWE ciphertexts, reducing the expansion by a factor of $\ell(k+1)N$ at the cost of requiring circuit bootstraps, or to pack them inside RGSW ciphertexts, reducing the expansion by a factor of N at the cost of a key switch per input bit [33].

Chapter 6

Experimental Results

In this chapter, we evaluate the performance of Homomorphic WiSARDs in terms of latency and classification performance across three popular machine learning datasets, presenting detailed analyses of these results against state-of-the-art in PPML training.

In Section 6.1, we present the hardware configuration used for our experiments. In Section 6.2, we introduce characteristics of the data sets used in our experiments. In Section 6.3, we assess the parameter sets for WiSARD and TFHE used in our experiments. In Section 6.4, we perform a comparative analysis of our results against the state-of-the-art. In Section 6.5, we argue about the practicality of our models by evaluating execution times and memory usage on consumer-grade hardware. In Section 6.6, we estimate the execution time for different model activation approaches. In Section 6.7, we analyze and discuss the communication costs for each model.

6.1 Experimental Setup

To evaluate the performance and scalability of our homomorphic WiSARD models, we conducted experiments in two distinct environments, representing consumer-grade and enterprise-grade hardware. These environments were chosen to provide information on computational requirements across a spectrum of potential use cases. The experiments were conducted in February 2024, using the following setups.

1. **Consumer Desktop Computer**:
    - Processor: Intel Core i7-12700k w/ AVX-512 (5.0 GHz, 8 cores, 16 threads)
    - Memory: 64GB (DDR5, 4800MHz)
    - Operating System: NixOS 23.11
2. **AWS Cloud Instance**:
    - Instance Type: *i4i.metal*
    - Processor: Dual Intel Xeon Platinum 8375C (3.5 GHz, 64 cores, 128 threads)
    - Memory: 1TB (DDR4, 3200MHz)
    - Operating System: Ubuntu Linux 22.04

For both environments, we used the Clang compiler version 17.0.6 and the Rust compiler version 1.76.0. Furthermore, the experiment binaries were compiled with *-O3* optimization level, and we applied the *cross-language LTO* technique to further improve optimization. Using these diverse environments, we aim to provide a comprehensive understanding of how our homomorphic WiSARD models perform under different hardware constraints.

6.2 Evaluation Datasets

This section details the datasets selected for our experimental evaluation, outlining the inherent challenges for each dataset, and describing the preprocessing techniques implemented to optimize their data representation. In Subsection 6.2.1, we overview the MNIST [41] dataset. In Subsection 6.2.2, we introduce the HAM10000 [67] dataset. In Subsection 6.2.3, we present the Wisconsin Breast Cancer [68] dataset.

6.2.1 MNIST

The MNIST [41] dataset consists of 70,000 grayscale images of handwritten digits, each with a resolution of 28×28 pixels and represented using 8 bits per pixel. We follow the standard train-test split for this dataset, with 60,000 images for training and 10,000 images for testing. We also applied linear quantization to the entire dataset, reducing the bit depth from 8 to 4 bits per pixel, effectively halving the model size.

6.2.2 HAM10000

The HAM10000 [67] dataset, also known as Skin Cancer MNIST or DermaMNIST, consists of 10,015 dermatoscopic images representing seven types of pigmented skin lesions: *Melanocytic Nevi* (nv), *Melanoma* (mel), *Benign Keratosis-like Lesions* (bkl), *Basal Cell Carcinoma* (bcc), *Actinic Keratoses* or *Intraepithelial Carcinoma* (akiec), *Vascular Lesions* (vasc), and *Dermatofibroma* (df). The images have a resolution of 450×600 pixels and are represented in RGB format, with 8 bits per color channel. This data set is known in the literature to be heavily unbalanced [1]. We used the official 28×28 downscale version of this dataset, following a 80% train and a 20% test split. We also applied linear quantization to the entire dataset, decreasing the bit depth from 8 to 4 bits per channel.

6.2.3 Wisconsin Breast Cancer

The Wisconsin Breast Cancer [68] dataset comprises 569 samples, each featuring 30 attributes of breast cell nuclei, calculated from digitized Fine Needle Aspirate (FNA) images, and labeled benign or malignant. For this dataset, we employ a 80% train and a 20% test set split. We apply Min-Max scaling followed by linear quantization, converting the features from their original floating-point representation to 8-bit unsigned integers.

6.3 Parameter Sets

Based on the results of our TFHE parameter search, outlined in Section 5.2, we evaluated the error rate for each set using Equation 6 of Guimarães *et al.* [31]. To minimize execution times, the parameter sets were then selected through a greedy choice of ℓ, which is associated with higher evaluation costs, prioritizing values of β that could support the plaintext space $\mathbb{Z}_p$ for each model while ensuring the error rate remained below 2^{-16}. Both ℓ_{KS} and β_{KS} were selected using the same criteria, with the additional constraint of maintaining an output noise variance comparable to that resulting from VP. These restrictions led us to two candidate parameter sets, presented in Table 6.1.

set	q	σ	N	ℓ	β	ℓ_{KS}	β_{KS}
HE_0	2^{64}	1.1×2^{-51}	2048	1	2^{23}	2	2^{15}
HE_1				2	2^{15}	3	2^{11}

Table 6.1: Selected parameter sets for the TFHE scheme.

For WiSARD parameters, we conducted empirical experiments using our cleartext implementation to optimize the balance between memory requirements and accuracy, provided that WiSARDs are memory bound. Our experiment assessed the impact of each parameter on memory usage. While increasing the address size leads to exponential growth in model state, the thermometer size causes only linear growth. Considering that and the model accuracy across various combinations, we identified optimal candidates for address sizes, thermometer sizes and types, thresholds, and number of training samples. For a fair comparison, we used the entire corresponding test set in all experiments. Table 6.2 presents the selected sets with associated TFHE parameter sets.

set	addr. size	therm. size	therm. type	act.	thr.	training samples	testing samples	encrypt set	encrypt p
MNIST_T	9	4	log	b-log	0	1000	10000	HE_0	2^8
MNIST_S	12	4	log	b-log	0	7500	10000	HE_0	2^{10}
MNIST_M	14	4	log	b-log	0	30000	10000	HE_1	2^{12}
MNIST_L	16	4	log	b-log	0	60000	10000	HE_1	2^{13}
HAM10000_S	12	5	lin	bin	0	1002	2003	HE_0	2^{10}
HAM10000_M	14	5	lin	bin	1	4006	2003	HE_1	2^{13}
HAM10000_L	16	5	lin	bin	1	8012	2003	HE_1	2^{13}
Wisconsin	10	5	lin	log	0	455	114	HE_0	2^9

Table 6.2: Selected parameter sets for the WiSARD models.

For the MNIST dataset, we prepared four parameter sets, named MNIST_T (tiny), MNIST_S (small), MNIST_M (medium), and MNIST_L (large). For the HAM10000 dataset, we prepared three parameter sets, named HAM10000_S (small), HAM10000_M (medium), and HAM10000_L (large). For the Wisconsin Breast Cancer dataset, we prepared only one parameter set, as increasing the parameters did not result in improved accuracy.

6.4 Comparative Analysis

This section presents a comparative analysis of our proposed models against the current state-of-the-art in both homomorphically encrypted training and inference. We provide a comparison based on the accuracy and execution time for each of our models, obtained from our evaluations in the cloud-based environment described in Section 6.1.

The accuracy is calculated as the average of 100 evaluations over the entire unbalanced test set, each using a unique random seed to ensure statistical relevance. For all experiments, we consider the PD-ACT activation approach, described in Subsection 5.4.3, and the use of our balancing technique, described in Subsection 5.5.2. Other classification metrics were not considered due to the lack of comparative metrics in other works.

The following subsections present the comparison results, highlighting both quantitative improvements and qualitative insights gained from our methodology. Subsection 6.4.1 includes the techniques that perform both homomorphically encrypted training and inference for the three datasets. Subsection 6.4.2 considers inference-only techniques for the MNIST dataset, which is more prevalent in the literature.

6.4.1 Homomorphic Training

In this subsection, we present the comparison against techniques that perform both homomorphically encrypted training and inference, being a direct correspondence to what our methodology proposes. Training times were measured by employing the amount of training samples reported in Table 6.2, and are expected to scale linearly for increasing amounts of samples if the same TFHE parameters are used. The choice for a reduced amount of samples reflects one of the strengths of the WiSARD models, that can effectively recognize patterns using lower amounts of data.

MNIST

For the MNIST dataset, we benchmark our models against Glyph [45], including their fastest result (one training epoch) and their result optimized for better accuracy, both results representing the current state-of-the-art in homomorphically encrypted training. In both cases, we focus on the baseline model performance without applying transfer learning, which we consider as a context-specific optimization outside the scope of this work. Note that transfer learning can potentially be applied to WNNs [49], as we will discuss later.

Table 6.3 presents a comprehensive comparison between our models and Glyph's. In particular, our implementations achieve significantly faster training times while maintaining competitive accuracy. The accuracy trade-off ranges from a minimal 0.4% to a maximum of 5.4% reduction compared to Glyph, depending on the specific configuration.

When interpreting the speed-up results, we provide both nominal values and values adjusted for the difference in computational resources. Glyph's experiments were conducted on an Intel Xeon E7-8890 v4 at 3.4GHz with 48 threads and 256GB of memory. Our speedup calculations account for the difference in thread count only, though other factors such as architecture specifics and compiler optimizations may also influence performance. It is

model	epochs	accuracy	train time	nominal speedup	adjusted speedup
MNIST_S	1	91.71%	3m28s	3291.4	1234.3
MNIST_M	1	93.06%	38m18s	300.6	112.7
MNIST_L	1	93.76%	3h30m	54.9	20.6
Glyph	1	94.10%	1.5 days	5.4	5.4
Glyph	5	97.10%	8 days	1.0	1.0

Table 6.3: Cloud results comparison for MNIST dataset.

difficult to consider these factors in our comparison without access to their implementation. Note that these differences alone do not explain the gap in model performance.

The results demonstrate that our approach offers a compelling trade-off between training speed and accuracy. For applications where rapid training is crucial and a slight decrease in accuracy is acceptable, our fastest model provides a remarkable 1234x adjusted speedup compared to Glyph's. This dramatic reduction in training times could be particularly valuable in scenarios requiring frequent model updates.

HAM10000

For the HAM10000 dataset, we benchmark our models against Glyph [45], which also represents the current state-of-the-art. As in the MNIST comparison, we adjusted the speedup for differences in execution environments between our benchmark and Glyph's, while also considering the model trained without the transfer learning technique.

Table 6.4 presents a comparison of our models with Glyph's. In particular, our implementations also achieved faster training times, but for this dataset our largest model achieved an improvement of 0.35% in accuracy. Compared to Glyph, our small and medium models maintained an accuracy gap of 1.35% and 0.6%, respectively.

model	epochs	accuracy	train time	nominal speedup	adjusted speedup
HAM10000_S	1	67.85%	1m35s	6720.0	2520.0
HAM10000_M	1	68.60%	13m35s	746.7	280.0
HAM10000_L	1	69.85%	1h03m	160.0	60.0
Glyph	15	69.20%	7 days	1.0	1.0

Table 6.4: Cloud results comparison for HAM10000 dataset.

Our results show a substantial improvement in training time while maintaining competitive accuracy. Specifically, our fastest model achieves a training time of less than 2 minutes, representing a 2520x adjusted speedup over Glyph's approach. These results are significant as they demonstrate our method's applicability to complex, real-world datasets.

Furthermore, these results also demonstrate the effectiveness of our balancing approach. As discussed in Subsection 6.2.2, HAM10000 is a heavily unbalanced dataset, requiring

additional measures to prevent overfitting. For example, without any balancing, our models can achieve accuracies up to 83.37%. Despite the higher values, the model completely overfits to a super-represented class, failing to classify samples from any of the other classes. As we balance the model, the accuracy drops significantly, but the model is able to consistently classify samples from all classes. Glyph does not discuss data set balancing in their work, so it is unclear whether this problem was addressed in their results.

Wisconsin Breast Cancer

For the Wisconsin Breast Cancer dataset, we compare our models with the approaches presented by Montero *et al.* [50] and Park *et al.* [57]. Taking into account that this is a much smaller data set than the others, simpler alternative techniques for classification are also applicable. In this case, we are comparing the results against the *Multi-Layer Perceptron* (MLP) and *Support Vector Machines* (SVM) algorithms, respectively.

Table 6.5 presents a comparison of our models with both MLP and SVM for the Wisconsin Breast Cancer dataset. In particular, our implementation also achieved faster training time while maintaining competitive accuracy, with a reduction of less than 1%.

model	accuracy	train time	nominal speedup	adjusted speedup
Wisconsin	97.30%	316ms	9303.8	1163.0
PBL$^+$20 SVM	98.00%	5m34s	8.8	7.0
MFK$^+$24 MLP	98.25%	49m	1.0	1.0

Table 6.5: Cloud results comparison for Wisconsin Breast Cancer dataset.

When interpreting the speed-up results, we provide both nominal values and values adjusted for the difference in computational resources. Montero *et al.* [50] run their experiments in an Intel i7-11800H CPU at 4.6Ghz with 16 threads, whereas Park *et al.* [57] uses an Intel Xeon CPU E5-2660 v3 at 3.3 GHz with 20 threads. As in previous comparisons, our speedup calculations account for the difference in thread count only, though other factors such as architecture specifics and compiler optimizations may also influence performance.

Adjusting for differences in the thread counts, our model provides a speed-up ranging from 166x to 1163x compared to alternatives. The considerable speed-up we achieve is particularly significant given the importance of heterogeneous tabular data for real-life use cases, such as financial analysis and medical diagnosis. Heterogeneous tabular data is the most ubiquitous form of data in machine learning applications, and is often seen as a challenge even for deep learning models [9].

6.4.2 Homomorphic Inference

In this subsection, we present the comparison against techniques that perform only homomorphically encrypted inference using TFHE. We compare our models with the approaches presented by Bourse *et al.* (FHE-DiNN) [11], Lou and Jiang (SHE) [46], Folkerts *et al.* (REDsec) [26], and Stoian *et al.* [65]. Considering the availability of results

in the literature, here we focus on the MNIST dataset, which is widely regarded as a benchmark standard in the HE evaluation of machine learning algorithms. Table 6.6 presents the comparison between TFHE-based approaches for encrypted MNIST inference. For our results, we present single-threaded and multithreaded evaluations.

model	accuracy	time (s)	security bits (λ)	threads	nominal speedup	adjusted speedup
MNIST_S	91.40%	0.774	128	1	23.8	2282.2
MNIST_M	92.81%	1.470	128	1	12.5	1201.6
MNIST_L	93.43%	2.184	128	1	8.4	808.8
MNIST_S	91.40%	0.048	128	128	383.3	287.5
MNIST_M	92.81%	0.067	128	128	274.6	206.0
MNIST_L	93.43%	0.092	128	128	200.0	150.0
FHE-DiNN-30N	93.71%	0.490	80	1	37.6	3604.9
FHE-DiNN-100N	96.30%	1.500	80	1	12.3	1177.6
SHE	99.54%	9.300	128	20	2.0	9.5
SFB^+23-150N	92.20%	31.000	128	16	0.6	3.6
SFB^+23-90N	96.50%	77.000	128	16	0.2	1.2
REDsec-Sign	98.00%	12.300	128	96	1.5	1.5
REDsec-ReLU	99.00%	18.400	128	96	1.0	1.0

Table 6.6: Cloud inference results comparison for MNIST dataset.

When interpreting the speed-up results, we provide both nominal values and values adjusted for the difference in computational resources. Bourse *et al.* [11] ran their experiments in an Intel i7-4720HQ CPU at 3.6GHz using one thread; Lou and Jiang [46] ran their experiments in an Intel Xeon E7-4850 CPU at 2.4GHz with 20 threads; Folkerts *et al.* [26] ran their experiments in a *r5.24xlarge* AWS instance, with 96 vCPUs running at 3.1 GHz; Stoian *et al.* [65] ran their experiments in an Intel i7-11800H CPU at 4.6GHz with 16 threads. As in our previous comparisons, our speedup calculations account for the difference in thread count only, even though other factors may also influence performance.

Our results demonstrate significant performance improvements over existing methods at the same security level, even for single-thread execution. Overall, the best performance goes for the FHE-DiNN-30N technique, with an adjusted speed-up of 3605x over the slowest method. However, it is important to note that FHE-DiNN models only offers a security level of 80 bits for the inputs, leaving the model weights unencrypted. Additionally, the adjusted speedup also benefits single-threaded approaches as they do not account for the parallelism overhead, as demonstrated by our single-thread and multithread executions.

In particular, we observe that our approach maintains a maximum of 6.1% in accuracy degradation compared to the best accuracy between the inference-only methods. However, the raw execution time is orders of magnitude faster than the other approaches, and can be scaled with an increasing number of threads. Comparing the adjusted speedup between the single-thread and multi-thread results, we notice that the parallelism imposes an overhead of 5.4x, which is very small considering that 128 threads are being used.

6.5 Resource Usage Analysis

To demonstrate the practicality of our approach, we conducted comprehensive benchmarks on a consumer-grade desktop computer. This analysis aims to show that small-scale encrypted training can be feasible with limited hardware resources. We measured the execution times over the whole test set for each experiment, providing the amortized time per sample for single-thread and multi-thread configurations. For the multi-thread configuration, we opt for 8 threads, each assigned to a separate physical core.

For medium and large parameter sets, where full execution would be prohibitively demanding, we estimated full execution times across a smaller set of samples. Memory usage results, however, reflects the true measurements as allocations are constant throughout execution. Table 6.7 and Table 6.8 present our benchmark results.

model	single-thread		multi-thread	
	training	inference	training	inference
MNIST_T	12m16s	2h11m	1m33s	19m40s
MNIST_S	2h34m	2h09m	19m19s	18m28s
MNIST_M	28h55m	4h05m	3h41m	33m42s
MNIST_L	156h15m	6h04m	19h57m	48m45s
HAM10000_S	54m13s	1h24m	6m50s	11m27s
HAM10000_M	9h19m	2h51m	1h11m	21m11s
HAM10000_L	43h30m	3h45m	5h27m	29m31s
Wisconsin	11s	3s	1436ms	361ms

Table 6.7: Consumer desktop benchmark times for each model.

model	single-thread		multi-thread	
	training	inference	training	inference
MNIST_T	177MiB	1.0GiB	501MiB	1.1GiB
MNIST_S	427MiB	1.1GiB	2.2GiB	1.1GiB
MNIST_M	1.2GiB	1.8GiB	7.4GiB	1.8GiB
MNIST_L	3.4GiB	3.9GiB	25.1GiB	4.0GiB
HAM10000_S	638MiB	1.0GiB	3.9GiB	1.1GiB
HAM10000_M	1.8GiB	2.3GiB	13.1GiB	2.3GiB
HAM10000_L	6.0GiB	6.2GiB	45.4GiB	6.4GiB
Wisconsin	132MiB	188MiB	146MiB	192MiB

Table 6.8: Consumer desktop memory usage for each model.

Our results highlight that while larger models demand significant memory and computational resources during training, the inference times, especially in multi-threaded configurations, are highly efficient for consumer-grade hardware. Note that the memory usage during training could be further reduced by sharing the model state between threads, potentially achieving similar memory usage to the single-threaded configuration.

6.6 Activation Cost Analysis

The choice of activation function in homomorphic encryption contexts significantly impacts performance and privacy preservation. This section analyzes computational costs associated with different activation approaches in our homomorphic WiSARD implementation, providing information about their trade-offs.

In Sections 6.4 and 6.5, the results represent the PD-ACT approach, with fast execution times. Here, we estimate costs associated with FM-ACT and OTF-ACT, which provide enhanced privacy with slower computation times, as discussed in Subsection 5.4.3.

For FM-ACT and OTF-ACT, we consider the results of the programmable bootstrapping technique implemented by Liu *et al.* [44]. Table 6.9 presents estimated execution times for each of our models using this programmable bootstrapping technique.

model	FM-ACT (hours)		OTF-ACT (min.)	
	9-bit	12-bit	9-bit	12-bit
MNIST_T	0.1	0.4	3.7	21.3
MNIST_S	0.4	2.1		
MNIST_M	1.1	6.4		
MNIST_L	3.8	22.0		
HAM10000_S	0.9	5.0		
HAM10000_M	2.8	16.4		
HAM10000_L	9.8	57.2		
Wisconsin	0.1	0.4		

Table 6.9: Estimated model activation execution times for 9 and 12 bits.

A key observation is that despite working over large plaintext spaces, the activation functions *b-log* and *bin* could be implemented using 9-bit precision for the MNIST and HAM10000 models, with a maximum probability of error of 2^{-9}. For the Wisconsin model, the *log* activation already operates with $p = 2^9$, aligning well with this precision level.

For the MNIST data set, assuming cloud training times, we observe that the FM-ACT activation times for 9-bit precision almost match the full training time, with activation times ranging from 0.4 hours (MNIST_S) to 3.8 hours (MNIST_L), while the training times range from 0.06 to 3.5 hours. For the HAM10000 dataset, the FM-ACT activation costs for 9-bit precision are 9 to 30 times more expensive than full training, ranging from 0.9 hours (HAM10000_S) to 9.8 hours (HAM10000_L), compared to training times of 0.03 to 1.05 hours. The Wisconsin data set shows the most striking contrast, with an FM-ACT activation time of 6 minutes for 9-bit precision, while the training time is 316 ms.

Due to the batching nature of the technique presented by Liu *et al.* [44], activating any of our models with 9-bit precision using OTF-ACT approach requires a constant time of 3.7 minutes, as none of their samples can fill a complete batch. In contrast, the FM-ACT approach has activation times that scale with the LUT sizes, depending on the sample, address, and thermometer sizes, as detailed in Section 6.3.

6.7 Communication Cost Analysis

Communication costs are a critical consideration in end-to-end PPML, particularly when deploying models in distributed or federated learning scenarios. A primary challenge in our approach is the significant ciphertext expansion, where encrypted data uses substantially more space than its plaintext counterpart. This expansion directly impacts bandwidth and storage requirements, especially when transmitting encrypted samples or model updates.

To quantify these costs, we calculate the encrypted sample size for each Homomorphic WiSARD model, as shown in Table 6.10. The expansion factor is determined by the TFHE parameter set (HE_0 or HE_1 of Table 6.1) and the number of bits in the input sample and label. For example, a single MNIST sample (3,136 bits after preprocessing) encrypted with HE_0 results in a 98 MiB ciphertext. Larger models, such as $HAM10000_L$ encrypted with HE_1, produce ciphertexts of 736 MiB per sample.

dataset	number of bits		enc.	exp.	sample
	sample	label	set	factor	size (MiB)
$MNIST_T$	3136	4	HE_0	262144	98
$MNIST_S$					
$MNIST_M$			HE_1	524288	196
$MNIST_L$					
$HAM10000_S$	11760	3	HE_0	262144	368
$HAM10000_M$			HE_1	524288	736
$HAM10000_L$					
Wisconsin	150	1	HE_0	262144	4.7

Table 6.10: Encrypted sample size for each Homomorphic WiSARD model.

Mitigation strategies discussed in Section 5.5.3, such as transciphering or bitpacking, could reduce these communication costs. However, they can also introduce computational overhead, which may not align with all scenarios. Despite the expansion, our approach remains practical for applications that prioritize performance over bandwidth usage.

Chapter 7

Conclusion

In this work, we introduced a novel approach to PPML through the homomorphic tranining and evaluation of WiSARD nets, as well as supporting procedures, such as the HE-friendly dataset balancing technique. We demonstrated that our approach offers significant performance improvements over existing methods, achieving good accuracy levels in minutes of encrypted training compared to days required by previous works.

In addition, we have shown that our technique advantages can go beyond performance improvements, providing unprecedented flexibility in multiple scenarios, such as distributed, federated, and continuous learning. These characteristics are not readily available in other machine learning algorithms, often requiring major modifications and involving several compromises that can make their implementation very complex or even not viable.

Although Homomorphic WiSARDs may not yet match the accuracy levels of ANN approaches, they offer a compelling trade-off between performance, privacy, and flexibility. This combination represents a significant step towards making PPML practical, opening new possibilities for privacy applications. For example, they could be used in fields such as healthcare, finance, and cybersecurity, accelerating the adoption of ML in these scenarios.

7.1 Future Work

There are many opportunities to further improve the performance and accuracy of WNN models. In this section, we uncover several promising avenues for future exploration.

Multi-shot Learning Multi-shot training procedures, particularly those using backpropagation [37], like DWN [5] and ULEEN [66], significantly improve WNN performance. By adjusting RAM scores through backpropagation, these frameworks achieved high MNIST accuracies (98.5% and 98.8%, respectively). Integrating such techniques into homomorphically encrypted WiSARD models could substantially reduce the accuracy gap compared to state-of-the-art encrypted CNNs. Notwithstanding this potential, our investigation revealed that elements borrowed from these frameworks in isolation did not yield significant improvements when implemented outside of their original methodology.

Transfer Learning Applying Transfer Learning to WNNs have shown significant improvements. By replacing a CNN's final layers by a WiSARD, Milhomem *et al.* [49] achieved a major accuracy improvement (11.2% over prior literature, 42.9% over baseline) in asphalt distress detection, using a pre-trained CNN as a feature extractor. In PPML, Transfer Learning improves Glyph [45] CNN results up to 4% for HAM10000 and up to 2% for MNIST. Panzade *et al.* [55] have shown further improvements over Glyph's results by employing TL to fine-tune encrypted image recognition models.

Circular Thermometer Code A promising direction for improving data representation lies in the use of the circular thermometer code [37] for periodic data domains, such as angular measurements or temporal features (*e.g.*, hours of the day). Instead of increasing the number of active bits, the circular thermometer maintains a fixed number of active bits and encodes wraparound values by rotating these bits within a binary vector. This ensures that proximity at the boundaries (*e.g.*, 0° and 360°) is preserved in the final representation.

Dynamic Activation Functions Our approach uses a common (static) function for all activations in a model. However, per-RAM (dynamic) activation functions are an optimistic direction for future research. As discussed, ULEEN [66] models achieve an impressive accuracy of 98.5% by combining multiple techniques, one of them being the dynamic selection of bleaching thresholds. In general, dynamic thresholds and activation functions can employ distinct behaviors for each RAM, improving noise tolerance.

Horizontal Packing The VP technique provides a solid foundation for homomorphic LUT evaluation in TFHE, but it suffers from overhead when applied to LUTs smaller than the polynomial degree N. As a complementary approach, Horizontal Packing [20] facilitates the simultaneous evaluation of several small LUTs, provided that they use the same input. It does not have performance improvements compared to VP for larger models, such as those used for MNIST or HAM10000. Nevertheless, when applied to models trained on datasets like Wisconsin Breast Cancer that demands fewer parameters, HP accelerates by evaluating in parallel multiple discriminators.

Alternative Encryption Schemes Although this work uses TFHE for its efficient function evaluation, alternative HE schemes such as CKKS [18], BFV [12, 25], and BGV [13] represent promising avenues for future homomorphic WiSARD research. BFV and BGV offer faster arithmetic than TFHE but slower nonlinear operations (potentially impacting activation functions). CKKS cheaply approximates polynomials, but requires managing precision and scale. Generally, these alternatives have slower functional bootstrapping than TFHE but higher operational throughput, being especially attractive for batch processing.

Bibliography

[1] Talha M. Alam, Kamran Shaukat, Waseem A. Khan, et al. An Efficient Deep Learning-Based Skin Cancer Classifier for an Imbalanced Dataset. *Diagnostics*, 12(9), 2022.

[2] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3), 2015.

[3] Igor Aleksander, W. V. Thomas, and P. A. Bowden. WISARD: a radical step forward in image recognition. *Sensor Review*, 4(3), 1984.

[4] Gilad Asharov, Abhishek Jain, Adriana López-Alt, et al. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *Advances in Cryptology - EUROCRYPT 2012*, pages 483–501, Berlin, Heidelberg, 2012. Springer.

[5] Alan Bacellar, Zachary Susskind, Mauricio Breternitz, et al. Differentiable Weightless Neural Networks. In *Proceedings of the 41st International Conference on Machine Learning*, pages 2277–2295, Vienna, Austria, July 2024. PMLR.

[6] Alan Bacellar, Zachary Susskind, Luis Villon, et al. Distributive Thermometer: A New Unary Encoding for Weightless Neural Networks. In *ESANN 2022*, pages 31–36, Bruges, Belgium, January 2022. i6doc.

[7] Thibault Balenbois, Jean-Baptiste Orfila, and Nigel Smart. Trivial Transciphering With Trivium and TFHE. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 69–78, New York, NY, USA, 2023. Association for Computing Machinery.

[8] Woodrow W. Bledsoe and Iben Browning. Pattern Recognition and Reading by Machine. In *1959 Proceedings of the Eastern Joint Computer Science*, pages 225–232, New York, NY, USA, 1959. Association for Computing Machinery.

[9] Vadim Borisov, Tobias Leemann, Kathrin Seßler, et al. Deep Neural Networks and Tabular Data: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(6), 2021.

[10] Christina Boura, Nicolas Gama, Mariya Georgieva, et al. Simulating Homomorphic Evaluation of Deep Learning Predictions. In *Proceedings of the Cyber Security Cryptography and Machine Learning Conference*, pages 212–230, Cham, 2019. Springer.

[11] Florian Bourse, Michele Minelli, Matthias Minihold, et al. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In *Advances in Cryptology - CRYPTO 2018*, pages 483–512, Cham, 2018. Springer.

[12] Zvika Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Advances in Cryptology - CRYPTO 2012*, pages 868–886, Berlin, Heidelberg, 2012. Springer.

[13] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[14] Michael Brand and Gaëtan Pradel. Practical Privacy-Preserving Machine Learning using Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2023/1320, 2023.

[15] Jacob Buckman, Aurko Roy, Colin Raffel, et al. Thermometer Encoding: One Hot Way To Resist Adversarial Examples. In *6th International Conference on Learning Representations, ICLR 2018*, pages 1–22, Vancouver, Canada, 2018. OpenReview.net.

[16] Hugo C. C. Carneiro, Felipe M. G. França, and Priscila M. V. Lima. Multilingual part-of-speech tagging with weightless neural networks. *Neural Networks*, 66(1), 2015.

[17] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-Key Homomorphic Encryption from TFHE. In *Advances in Cryptology - ASIACRYPT 2019*, pages 446–472, Cham, 2019. Springer.

[18] Jung Hee Cheon, Andrey Kim, Miran Kim, et al. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology - ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer.

[19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, et al. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *Advances in Cryptology - ASIACRYPT 2016*, pages 3–33, Berlin, Heidelberg, 2016. Springer.

[20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, et al. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology*, 33(1), 2020.

[21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks. In *Cyber Security Cryptography and Machine Learning*, pages 1–19, Cham, 2021. Springer.

[22] Kelong Cong, Debajyoti Das, Jeongeun Park, et al. SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. Cryptology ePrint Archive, Paper 2022/757, 2022.

[23] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, et al. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 201–210, New York, NY, USA, 2016. Association for Computing Machinery.

[24] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18, Berlin, Heidelberg, 1985. Springer-Verlag.

[25] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012(1), 2012.

[26] Lars W. Folkerts, Charles Gouert, and Nektarios G. Tsoutsos. REDsec: Running Encrypted Discretized Neural Networks in Seconds. In *Proceedings 2023 Network and Distributed System Security Symposium*, pages 1–17, San Diego, CA, USA, 2023. Internet Society.

[27] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, pages 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[28] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Advances in Cryptology - CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer.

[29] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6), 1998.

[30] Bruno P. A. Grieco, Priscila M. V. Lima, Massimo De Gregorio, et al. Producing pattern examples from "mental" images. *Neurocomputing*, 73(7), 2010.

[31] Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2), 2021.

[32] Antonio Guimarães, Edson Borin, and Diego F. Aranha. MOSFHET: Optimized Software for FHE over the Torus. *Journal of Cryptographic Engineering*, 14(1), 2024.

[33] Antonio Guimarães, Leonardo Neumann, Fernanda A. Andaló, et al. Homomorphic evaluation of large look-up tables for inference on human genome data in the cloud. In *2022 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, pages 33–38, Washington, DC, USA, 2022. Institute of Electrical and Electronics Engineers.

[34] Haibo He and Edwardo A. Garcia. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), 2009.

[35] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Deep Neural Networks Classification over Encrypted Data. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pages 97–108, New York, NY, USA, 2019. Association for Computing Machinery.

[36] Malika Izabachène, Renaud Sirdey, and Martin Zuber. Practical Fully Homomorphic Encryption for Fully Masked Neural Networks. In *Cryptology and Network Security*, pages 24–36, Cham, 2019. Springer.

[37] Rafael F. Katopodis, Priscila M.V. Lima, and Felipe M.G. França. Functional gradient descent for n-tuple regression. *Neurocomputing*, 500(1), 2022.

[38] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography: Principles and Protocols*. Chapman & Hall/CRC Cryptography and Network Security Series. Taylor & Francis, 2007.

[39] Christof Koch, Tomaso A. Poggio, and Vincent Torre. Retinal ganglion cells: a functional interpretation of dendritic morphology. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 298(1090), 1982.

[40] Miroslav Kubat and Stan Matwin. Addressing the Curse of Imbalanced Training Sets: One-Sided Selection. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 179–186, Nashville, TN, USA, 1997. Morgan Kaufmann.

[41] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. Yann LeCun's Website, 2010.

[42] Yongwoo Lee, Daniele Micciancio, Andrey Kim, et al. Efficient FHEW Bootstrapping with Small Evaluation Keys, and Applications to Threshold Homomorphic Encryption. In *Advances in Cryptology - EUROCRYPT 2023*, pages 227—256, Berlin, Heidelberg, 2023. Springer-Verlag.

[43] Wouter Legiest, Furkan Turan, Michiel Van Beirendonck, et al. Neural Network Quantisation for Faster Homomorphic Encryption. In *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design*, pages 1–5, Washington, DC, USA, 2023. Institute of Electrical and Electronics Engineers.

[44] Zeyu Liu and Yunhao Wang. Amortized Functional Bootstrapping in Less than 7 ms, with Polynomial Multiplications. In *Advances in Cryptology - ASIACRYPT 2023*, pages 101–132, Singapore, 2023. Springer Nature.

[45] Qian Lou, Bo Feng, Geoffrey C. Fox, 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*, pages 9193–9202, Red Hook, NY, USA, 2020. Curran Associates Inc.

[46] Qian Lou and Lei Jiang. *SHE: A Fast and Accurate Deep Neural Network for Encrypted Data*, pages 10035–10043. Curran Associates, Inc., Red Hook, NY, USA, 2019.

[47] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. *Journal of the ACM*, 60(6), 2013.

[48] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), 1943.

[49] Suayder Milhomem, Tiago Almeida, Warley Gramacho, et al. Weightless Neural Network with Transfer Learning to Detect Distress in Asphalt. *Journal of Advanced Engineering Research and Science*, 5(12), 2018.

[50] Luis Montero, Jordan Frery, Celia Kherfallah, et al. Machine Learning Training on Encrypted Data with TFHE. In *Proceedings of the 10th ACM International Workshop on Security and Privacy Analytics*, pages 71–76, New York, NY, USA, 2024. Association for Computing Machinery.

[51] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can Homomorphic Encryption Be Practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, pages 113–124, New York, NY, USA, 2011. Association for Computing Machinery.

[52] Leonardo Neumann, Antonio Guimarães, Diego F. Aranha, et al. Homomorphic WiSARDs: Efficient Weightless Neural Network training over encrypted data, 2025. To appear at the 23rd International Conference on Applied Cryptography and Network Security (ACNS 2025).

[53] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2), 2020.

[54] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology - EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer.

[55] Prajwal Panzade, Daniel Takabi, and Zhipeng Cai. I can't see it but I can Fine-tune it: On Encrypted Fine-tuning of Transformers using Fully Homomorphic Encryption. arXiv, 2024.

[56] German I. Parisi, Ronald Kemker, Jose L. Part, et al. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113(1), 2019.

[57] Saerom Park, Junyoung Byun, Joohee Lee, et al. HE-Friendly Algorithm for Privacy-Preserving SVM Training. *IEEE Access*, 8(1), 2020.

[58] Oded Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, pages 84—93, New York, NY, USA, 2005. Association for Computing Machinery.

[59] Oded Regev. The Learning with Errors Problem. In *2010 IEEE 25th Annual Conference on Computational Complexity*, pages 191–204, Washington, DC, USA, 2010. Institute of Electrical and Electronics Engineers.

[60] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. *On Data Banks and Privacy Homomorphisms*, pages 169–179. Academia Press, New York, NY, USA, 1978.

[61] Ronald L. Rivest, Adi Shamir, and Len Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2), 1978.

[62] Frank Rosenblatt. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 1958.

[63] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61(1), 2015.

[64] Himanshu Shekhar, Sujoy Seal, Saket Kedia, et al. *Survey on Applications of Machine Learning in the Field of Computer Vision*, pages 667–678. Springer, Singapore, 2020.

[65] Andrei Stoian, Jordan Frery, Roman Bredehoft, et al. Deep Neural Networks for Encrypted Inference with TFHE. In *Cyber Security, Cryptology, and Machine Learning*, pages 493–500, Switzerland, 2023. Springer.

[66] Zachary Susskind, Aman Arora, Igor D. S. Miranda, et al. ULEEN: A Novel Architecture for Ultra-low-energy Edge Neural Networks. *ACM Trans. Archit. Code Optim.*, 20(4), 2023.

[67] Philipp Tschandl. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Harvard Dataverse, 2018.

[68] William Wolberg, Olvi Mangasarian, Nick Street, et al. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1995.

[69] Chen Zhang, Yu Xie, Hang Bai, et al. A survey on federated learning. *Knowledge-Based Systems*, 216(1), 2021.

[70] Shuai Zhang, Lina Yao, Aixin Sun, et al. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Computing Surveys*, 52(1), 2019.