Universiteit Leiden
The Netherlands

Data Science & AI

Secure Machine Learning: A Comparative Study of Special Preprocessing Techniques in Multiparty Computation

Alexander Bazba

Supervisors:
Eleftheria Makri and Nusa Zidaric

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 25/08/2025

Abstract

Machine learning (ML) is progressively utilised in sensitive domains such as healthcare and finance, where privacy issues limit its implementation. One promising cryptographic approach to this problem is Secure Multi-Party Computation (MPC), which enables several parties to jointly compute on private data without disclosing their inputs. Many recent MPC protocols adopt the preprocessing model, in which part of the work is shifted to an offline phase so that the online execution can proceed more efficiently. Over the past decade, a number of state-of-the-art protocols have been developed under this model, each designed with different adversarial settings and security assumptions in mind. This thesis analyses six protocols—CDNN15, SecureML, SecureNN, CKRRSW20, RRHK23, and LowGear 2.0—that utilise specialised preprocessing approaches to facilitate privacy-preserving machine learning (PPML). The experimental investigation utilised the MP-SPDZ framework, documenting both computation time and communication volume during the offline and online periods. The benchmarks highlight clear contrasts. Semi-honest protocols such as CDNN15 and SecureML keep online costs extremely low but require heavy preprocessing. Maliciously secure schemes like CKRRSW20, RRHK23, and LowGear 2.0 demand far more offline resources yet provide stronger protection. SecureNN occupies an intermediate position, combining moderate offline effort with significant online interaction in a three-party setting. The findings illustrate how design decisions impact various efficiency-security trade-offs, providing practical recommendations for adopting protocols for PPML in real-world applications.

Contents

1 Introduction

1.1 Motivation

Fast digital data spread is driving up demand for effective privacy-preserving technologies. Many different sectors, including healthcare, finance and autonomous systems, are increasingly depending on machine learning (ML), which has generated major controversy over data security and user privacy. [19] High-profile data leaks and rigorous legal frameworks like the General Data Protection Regulation (GDPR) [23] have underlined even more the need for secure computing techniques allowing data analysis without confidentiality compromise.

Multiparty computation (MPC) provides a solution to the problem of computing methods exposing sensitive data. MPC works by facilitating cooperative computation over private data without revealing the underlying inputs. In a typical MPC environment, all parties (user devices, servers or both) agree on what function needs to be calculated ahead of time. The protocol ensures that as long as the number of parties working together remains below a certain limit, they cannot reconstruct one another's input. This preserves input privacy under the protocol's security guarantees.

Machine learning is shaping the future of technology even if privacy concerns are restricting its use. Further research on MPC can help machine learning models to attain a new degree of practical application. By employing specialised preprocessing techniques, modern MPC protocols can significantly reduce the computational burden typically associated with secure computation, so enabling privacy-preserving machine learning (PPML) to be feasible in practical applications.

Recent advances in MPC-based PPML systems, such as SecureML [20], SecureNN [25], and CKRRSW20 [6], indicate that secure computation is a practical tool and not only a theoretical idea. As ML models get more complex, the efficiency of PPML systems remains a key concern. This thesis seeks to comprehensively and methodically evaluate the performance of current preprocessing techniques, with particular attention to the level of security they provide. The goal is to produce a comparative study that will help researchers and practitioners choose the best strategy for their individual use case and system needs.

This thesis evaluates how MPC can be applied to privacy-preserving machine learning by focusing on specialised preprocessing. The aim is to quantify practical costs and security trade-offs and to provide evidence-based guidance for protocol selection in PPML

1.2 Preliminaries

This section introduces essential terminology and primitives used throughout the thesis. Understanding these foundational concepts is crucial for interpreting the design, implementation, and evaluation of MPC protocols, especially in the context of PPML.

1.2.1 Multiparty Computation

Secure Multiparty Computation is a subfield of cryptography that studies how multiple parties, each holding a private input, can jointly compute a publicly defined function over their inputs in such a way that no party learns anything beyond what can be inferred from their own input and the final output. [18] In computer science and cryptography, a protocol is a well-defined set of rules that multiple participants follow to achieve a specific goal. It tells the participants how to communicate to each other, in what order, and what to do with the data they get. Each participant follows a set of steps in a certain order, and if all follow the guidelines, the desired result is reached. MPC is a cryptographic protocol that enables n parties, each with a private input x_i to jointly compute a function

$$F(x_1, x_2, ..., x_n) \tag{1}$$

such that each party learns the correct output of $F(x_1, x_2, ..., x_n)$ and no party learns anything more about other parties' inputs beyond what is inherently revealed by the output [18].

The real–ideal world paradigm is often used to analyse and create MPC protocols. This paradigm defines security by comparing how the protocol works in the real world to how it would work in an ideal situation with a trusted third party. If an adversary's view in the real world can be simulated in the ideal world, then the protocol is secure. This means that no more information is released beyond what was meant to be output.

There are a few major factors that determine how an MPC system is designed in practice and in theory. First, it must be obvious how many parties there are and what their roles are, including whether everyone gets the output or just some parties. Second, protocols need to define their cryptographic security assumptions, that is, the underlying guarantees (such as hardness of certain mathematical problems or the honesty of a majority of parties) that the protocol relies on.

A lot of modern MPC protocols use a preprocessing model that splits the computation into two parts: an offline phase, where parties generate correlated randomness that does not depend on their inputs, and an online phase, where inputs are given and the actual function is evaluated quickly using the correlated randomness and auxiliary values generated during preprocessing. This separation is particularly useful in PPML, where fast inference or training is required.

1.2.2 Parties

Different MPC protocols are designed to work with a specific number of participating parties. In the context of MPC, a party is an independent computer that is taking part in the protocol. This could be a user device, a server, or any other system that holds confidential input values and participates in the joint computation. Protocols might include two parties as in client-server interactions or more than two parties as in distributed collaborative learning among several organisations. The protocol's complexity communication costs, and security assumptions are affected by the number of parties. As this number increases, communication overhead and computational requirements typically grow, which can impact the protocol's overall scalability and efficiency. Moreover, the security model may also depend on how many parties are assumed to be honest and how many are assumed to be dishonest, an issue further discussed below.

1.2.3 Adversary Models

In MPC, security is defined according to the adversarial behaviour the protocol is designed to withstand. The two most common categories of adversaries are semi-honest and malicious [13].

A semi-honest adversary observes the rules but strives to get more information from the messages it gets. In this approach, secure protocols make sure that each party only learns what it can figure out from its own input and output.

On the other hand, a malicious adversary may randomly break the rules in order to violate correctness or privacy. To make these models more secure, additional forms of protection are needed such as verifiable secret sharing, which ensures that all parties give equally valid inputs during the protocol's execution.

Protocols that achieve malicious security, such as CKRRSW20 [6], RRHK23 [22], and LowGear 2.0 [21], which are designed to tolerate actively misbehaving parties, provide stronger guarantees of correctness and privacy. However, these protocols are more complex and typically incur significantly higher communication and computational costs compared to those secure only against semi-honest adversaries

1.2.4 Corruption threshold

The security model of a multiparty computation protocol describes the conditions under which the protocol is regarded as secure. It combines two aspects: the behaviour expected from adversaries and the number of corrupted parties that can be tolerated. The adversary model focuses on behaviour alone, distinguishing for example between semi-honest adversaries, who obey the protocol but attempt to glean extra information, and malicious adversaries, who may act arbitrarily to compromise privacy or correctness. The corruption threshold complements this by setting a concrete limit on how many parties can be compromised before security fails.

In semi-honest settings, certain protocols remain sound even if all but one participant is corrupted. By contrast, protocols with information-theoretic guarantees in the multiparty case typically withstand fewer than half of the parties being dishonest. Maliciously secure protocols often require an even stronger honest majority, in some cases tolerating fewer than a third of the parties being corrupted. A higher threshold naturally improves robustness, but it usually comes with additional computation and communication overhead.

It is also common to distinguish between static corruption, where the set of corrupted parties is fixed in advance, and adaptive corruption, where the adversary can decide whom to corrupt during execution. Supporting the latter generally requires more sophisticated design choices and extra safeguards, but it provides stronger resilience in adversarial environments.

1.2.5 Preprocessing Models

MPC protocols, designed under the so-called preprocessing model, break the calculation down into two parts: an online phase and an offline phase. The offline phase takes place before the private inputs are provided. This phase prepares various types of auxiliary data needed for efficient secure computation. These data elements are generated without access to the parties' private inputs and are designed to minimize the computational and communication cost of the online phase. After the preprocessing phase, the parties input their private data into the protocol using a secret-sharing or encryption mechanism (explained further in this thesis), depending on the setting. This starts the online phase. At this point, the secure computation is performed using both the private inputs and the auxiliary data that were prepared during the offline phase. This separation of phases is especially useful in PPML, where structured preprocessing can help speed up operations like matrix multiplications and non-linear activations that require significant computational effort.

1.3 Current Limitations of MPC

MPC for PPML has seen notable development, yet some important issues remain unresolved, for instance real-world application in large-scale machine learning is still held back by core trade-offs between security, efficiency, and scalability.

Traditional MPC protocols, even those employing preprocessing, suffer from substantial computational and communication costs, particularly for complex machine learning operations such as matrix multiplications, convolutions, and activation functions [20, 25]. These protocols are generally designed for generic secure computation, where preprocessing produces generic resources such as multiplication triples, suitable for any computation but not tailored to machine learning. In contrast, the six protocols evaluated in this thesis make use of specialised preprocessing techniques tailored to machine learning workloads. Instead of relying only on generic triples, they precompute resources optimised for operations such as convolutions, matrix multiplications, or fixed-point arithmetic. This targeted approach is what differentiates them from traditional MPC and allows them to achieve better efficiency in PPML tasks. When training or making predictions with deep neural networks that have millions of parameters, the resources needed for preprocessing can become too expensive, making it impossible to use them on a broad scale [16].

Existing MPC protocols like TopGear [5] are mostly meant for general-purpose secure computation and are not customised for machine learning tasks [17]. Certain studies for MPC protocols like CKRRSW20 [6] and RRHK23 [22] offer particular preprocessing for matrix computations. However, there is currently no comprehensive evaluation comparing these techniques within the context of PPML. The lack of such a comparison restricts academics from making educated decisions regarding which MPC protocol they should choose to work with.

Existing MPC-based PPML solutions struggle to remain efficient as machine learning models increase in size (e.g., large-scale neural networks, Recurrent Neural Networks and Convolutional Neural Networks). Most of the present methods concentrate on small to medium-sized networks, therefore lacking research on securing large models such as BERT or ResNet-152 [14]. Particularly in distributed environments with several parties, the preprocessing demands both computational and communication related for such architectures remain a major bottleneck.

1.4 Research Aim

The goal of this thesis is to understand how special preprocessing techniques in MPC can enable efficient PPML. The central research question guiding the work is: *"How can special preprocessing techniques in MPC enable efficient privacy-preserving machine learning?"*. This main research question is broken down into two smaller, more concrete subquestions. *"Which special preprocessing techniques specifically target machine learning computations?"*. This subquestion has already been answered in detail by prior work, particularly in [24]. In this thesis, these results are taken as a starting point for further analysis. The focus is on the following MPC protocols: CDNN15 [9], SecureML [20], SecureNN [25], CKRRSW20 [6], RRHK23 [22], and LowGear 2.0 [21], which were identified as special preprocessing protocols for ML applications in [24]. This thesis aims to provide a thorough comparison of these protocols, by implementing them, and concretely comparing its computation and communication costs.

Second, *"How do special preprocessing models compare in terms of computation and communication costs?"* Each model will be implemented in a controlled environment, and their computation and communication costs will be benchmarked. Unlike in [24], which only provides comparative cost estimates, this research will benchmark the protocols by executing them and reporting concrete time measurements for both the online and offline phases of each protocol.

The results from this research will help us figure out which preprocessing methods are best for particular PPML use cases by carefully looking at the pros and cons of each one. In the end, the goal of this research is to provide practical guidelines for choosing and improving MPC protocols in real-world machine learning applications, where privacy and computational feasibility are both very important.

1.5 Structure of the Thesis

This thesis set up is to take the reader from basic ideas to real-world evaluation and ultimate conclusions. Chapter 2 gives an overview of the background and related work, as well as the main ideas behind MPC, how it has changed over time, and how it might be used in PPML. Chapter 3 describes the six unique preprocessing techniques that are the main focus of this study. It explains how each one works and what makes it different. Chapter 4 discusses the methods utilised for evaluation, such as the criteria, instruments, and experimental design. Chapter 5 compares the performance and security of each protocol and summarises the pros and cons of each one for different use cases. Chapter 6 sums up the main points, highlighting the limitations of this thesis, and suggesting areas for future research.

2 Background and Related work

2.1 Theoretical foundations of MPC

Theoretical research conducted in the 1980s and 1990s demonstrated the feasibility of secure computation based on general cryptographic assumptions. Oded Goldreich [17], provided rigorous definitions for different types of adversaries and made a distinction between semi-honest adversaries and malicious adversaries. Goldreich made the real–ideal world paradigm the mainstream way to define security in MPC. In this paradigm, a protocol is secure if any attack that happens in the real world can also be simulated in an ideal world with a trusted third party, which means that no extra information is revealed. They also proved completion theorems, which showed that some basic cryptographic building blocks, like oblivious transfer (explained further in this paper), are sufficient to make general-purpose MPC protocols.

Meanwhile, Du and Atallah [12] redirected focus from general feasibility to specific problem identification, introducing a classification of domain-specific multiparty computation tasks—such as privacy-preserving database queries, statistical analysis, and intrusion detection—and contending that specialised protocols can significantly outperform black-box reductions in efficiency.

2.2 Real world implementation of MPC

Recent developments have been driven by the need for greater efficiency and deployability. An essential discovery is to divide computation into an offline preprocessing phase, during which correlated randomness is generated, and a rapid online phase, in which inputs are utilised. Sun and Makri's systematization of knowledge [24] differentiates between traditional preprocessing, which is used to prepare general purpose resources such as multiplication triples, and specialised preprocessing for several non-generic tasks such as convolutions and matrix multiplication, which are commonly required in PPML. Preprocessing has facilitated high-throughput protocols tolerating a dishonest majority of parties, such as SPDZ [8], a protocol family that supports secure arithmetic over secret data where only one party is honest.

Simultaneously, academics have developed open-source frameworks that obscure cryptographic intricacies behind elevated programming interfaces. Keller's MP-SPDZ [16] provides over 30 protocol variants within a single virtual machine. The FLASH framework by Byali et al. [4] demonstrates that the inclusion of a fourth server allows for computing dot-products and efficient truncation. A dot-product is an operation in linear algebra that computes the sum of two vectors, and appears frequently in neural network layers and linear classifiers. Truncation, on the other hand, is necessary when working with fixed-point arithmetic to simulate real-number operations. The impact of these frameworks reduces the transition from theoretical protocol design to practical secure analytics.

Zhou et al. [27] provide a comprehensive survey of MPC solutions for linear models, deep neural networks, and federated learning, highlighting the superiority of secret-sharing protocols in extensive arithmetic operations and the effectiveness of homomorphic encryption (explained further in the thesis) settings where parties possess highly imbalanced data volumes or features.

Protocols written using FLASH and MP-SPDZ exhibit image classification on MNIST in seconds instead of minutes. These empirical results illustrate the practical viability of specialised MPC for privacy-preserving machine learning, while supporting Du and Atallah's [12] earlier theoretical argument that domain-specific MPC can achieve significantly greater efficiency than generic MPC protocols.

2.3 Privacy-Preserving Machine Learning (PPML)

The term Privacy-Preserving Machine Learning refers to a group of statistical and cryptographic methods that allow machine-learning models to be trained or used for inference without disclosing private information. While allowing standard machine learning algorithms, such as classification, regression, clustering, or deep learning, to operate at realistic speeds, a PPML pipeline seeks to guarantee that no party learns more about another party's input than can be deduced from the final output [26]. With secret sharing (SS) and homomorphic encryption (HE) serving as the two main cryptographic paradigms, the instruments that enable this can be broadly divided into statistical and cryptographic approaches. This section will only focus on cryptographic approaches.

2.3.1 Homomorphic encryption

Homomorphic encryption (HE) enables arithmetic operations to be performed on encrypted data without decrypting it first. The result is the same as if the operations had been done on the original data after decryption. This allows secure computation without ever having to look at the raw data. Gentry [3] introduced fully homomorphic encryption (FHE), which enables arbitrary computations to be performed directly on encrypted data. However, FHE remains computationally impractical for most real-world applications due to its significant performance overhead. Consequently, many machine learning tasks instead rely on more efficient variants, such as partially homomorphic encryption (PHE) or somewhat homomorphic encryption (SHE), which support only restricted sets of operations.

When a user sets up PPML, they encrypt their data and transfer it to a server that performs the computations without seeing the original unencrypted values. Most of the time, these tasks include operations such as addition and multiplication. It is well-established that addition and multiplication over finite fields are computationally complete, meaning that any function over a finite field can be expressed using only these two operations [1]. A detailed mathematical treatment of finite fields is beyond the scope of this thesis. Upon computation completion, it sends back the encrypted result, which the user decrypts to get the final output. This increases computational overhead and latency but improves security by ensuring that the server cannot access the plaintext data. Libraries like SEAL [7] help enhance performance by letting numerous inputs be handled at the same time.

Homomorphic encryption is helpful because it means that there is no need to trust the server with private information. But it is still hard and slow to use HE for complicated tasks like training deep neural networks with a lot of layers. To solve this problem, current research is looking towards hybrid systems that combine HE with other technologies, including trusted hardware or garbled circuits, to find a balance between performance and security. The technical details of hardware implementations and garbled circuits are beyond the scope of this thesis and are only referenced here at a conceptual level.

2.3.2 Secret Sharing

Secret Sharing is a cryptographic technique used to split up a private value into several shares. No one share can give away any information about the original value, but when enough shares are combined, the original value can be rebuilt [10].

The most common type of secret sharing is additive secret sharing. In this method, a secret value x is split among n parties by choosing random shares $x_1, x_2, ..., x_{n-1}$ and setting $x_n = x - \sum_{n-1}^{i=1} x_i$. Each party gets one share, x_i, and the total of all the shares makes the original value, x. Since each share is randomly chosen, no one party can learn anything about the secret until all the shares are put together.

One of the advantages of secret sharing in the MPC setting is that addition and multiplication with a constant can be done without communicating with any other party. If parties own shares of two secrets, x and y, they can figure out shares of $x + y$ by just adding up their shares. But multiplication is more complicated and usually needs extra tools like Beaver Triples (on which see 2.1.4, below).

Secret sharing schemes can be used in finite fields or rings, and they can work in both two-party and multiparty settings. SecureML, SecureNN, and LowGear 2.0 are examples of protocols that use secret sharing to make training and inferring machine learning models more efficient and secure.

2.3.3 Message Authentication Codes (MACs)

Message Authentication Codes (MACs) are cryptographic primitives used to ensure the authenticity and integrity of secret-shared values in MPC [11]. They are particularly important in protocols that allow dishonest majorities or operate under the malicious adversary model, since they enable parties to detect misbehaviour without revealing their private inputs.

In the variant used by protocols such as SPDZ and its successors, every secret-shared value x is accompanied by a MAC tag $m = \alpha \cdot x$, where α is a global MAC key known to the parties in shared form. Each party holds a share of both x and its tag m. When values are later opened, the parties can check consistency by verifying that the reconstructed tag matches $\alpha \cdot x$. If the check fails, cheating is detected and the protocol can abort.

This SPDZ-style construction is not the only way to build a MAC, but it is the method employed in the MPC protocols considered in this thesis.

2.3.4 Beaver Triples

Beaver triples are made up of a precomputed random triplet (a, b, c), where a and b are randomly chosen values and $c = a \cdot b$ [2]. These values are made in the offline phase, when the private inputs are not yet known, and they are shared among the parties taking part using a secret-sharing scheme.

The goal of Beaver triples is to securely multiply two private inputs, x and y, without letting anyone know what their values are while the calculation taking place. By hiding the inputs with the precomputed values a and b, only the masked differences $\triangle_x = x - a$ and $\triangle_y = y - b$ are shown. The final result is then found using the formula:

$$xy = c + \triangle_x \cdot b + \triangle_y \cdot a + \triangle_x \cdot \triangle_y \tag{2}$$

Each term in this formula can be computed securely using only local operations and minimal interaction, since the randomness and structure of the Beaver triple ensure that no party learns anything about the actual inputs x or y. This makes it highly suitable for use in PPML, where large numbers of multiplications (e.g, in matrix operations or neural network layers) must be performed securely and efficiently.

2.3.5 Masks

Masks are random numbers that are used to hide private inputs during computation to ensure security and privacy. [10] They are important in secret sharing schemes, where each party only has a masked or partial view of the real data. Before the computation starts, each input is usually combined with a random mask that makes it impossible for anyone to understand. During the protocol, these masked values are used in calculations, and the results are then unmasked to show the correct output without ever showing the raw inputs. This method prevents information from leaking, which is very important for keeping sensitive data private when working together on calculations.

2.3.6 Fields vs. Rings

The choice between fields and rings as the underlying algebraic setting has a direct impact on the design of MPC protocols. It determines how basic operations are carried out and, in turn, affects efficiency, communication, and scalability. A full algebraic treatment is beyond the scope of this thesis, but it is important to highlight the differences most relevant in practice.

Many MPC protocols are described in terms of arithmetic circuits, where private inputs are represented so that the computation can be broken down into additions and multiplications. Whether these circuit operations are performed over a field or a ring influences what types of functions can be supported efficiently. Fields are typically required when protocols involve division or inversion. For example, CDNN15 [9] uses a finite field to carry out the Newton–Raphson iterations for secure matrix inversion in linear regression. Rings, by contrast, are well suited for binary-style computations and for fixed-point arithmetic. Their modular structure allows fast integer operations and makes it straightforward to approximate real numbers in a way that aligns with standard practice in machine learning.

2.3.7 Zero Knowledge Proofs of Plaintext Knowledge

Zero-Knowledge Proof of Plaintext Knowledge (ZKPoPK) is a cryptographic method that enables an entity to demonstrate knowledge of the plaintext contained within a ciphertext without disclosing the plaintext itself [15].

This evidence is crucial in maliciously secure systems like as CKRRSW20 [6] and RRHK23 [22], where participants collaboratively produce Beaver triples or convolution triples utilising homomorphic encryption. In the absence of ZKPoPK, a deceitful entity could skew the computation or induce overflows, hence jeopardising correctness or compromising confidentiality.

Utilising ZKPoPK, each participant verifies that their encrypted values are properly structured without disclosing them. This verification strengthens malicious security independently of a trustworthy initialiser and prevents the leakage of critical information during preprocessing. Despite ZKPoPK introducing additional computational and communication overhead, it is significantly more cheaper than re-executing the complete preprocessing step or using a more complex zero-knowledge systems.

3 Special Preprocessing Models Overview

3.1 Summary of Special Preprocessing Techniques

To provide a clearer overview of the protocols examined in this thesis, Table 1 summarises their key characteristics in a structured format. The table highlights the main workloads each protocol supports, the computational domain in which it operates, its security assumptions, the number of parties and tolerated corruption threshold, and the cryptographic primitives on which it relies. It also distinguishes between the core operations performed in the offline and online phases, allowing for a direct comparison of where computational and communication costs are concentrated. This summary serves as a reference point for the detailed protocol descriptions that follow.

Table 1: Summary of special preprocessing MPC protocols

Protocol	Functionality / Workload	Domain	Adversary Model	Parties & Threshold	Cryptographic Primitives	Offline Phase Operations	Online Phase Operations
CDNN15	Linear regression	Finite field	Semi-honest	2 parties, 1 corrupt	Beaver triples, fixed-point encoding, secure truncation	Precompute Beaver triples (no inputs used)	Matrix multiplications, Newton–Raphson inversion
SecureML	Linear, logistic regression, neural networks	Finite field	Semi-honest	2 parties, 1 corrupt	LHE, COT, Beaver triples	Generate triples via LHE + COT	Gradient descent updates, activation evaluation
SecureNN	Neural network training (linear, conv, ReLU, MaxPool)	Ring Field	Semi-honest or malicious (1 corrupt)	3 parties	PRFs, Beaver triples	PRF-based random triple generation	Matrix multiplications, secure ReLU/MaxPool
CKRRSW20	Matrix multiplication, convolution	Finite field	Malicious (dishonest majority)	2 parties, 1 honest	BFV HE, ZKPoPK, bilinear triples	Homomorphic generation of matrix/convolution triples	Secure multiplication using triple masking
RRHK23	Convolution layers	Finite field	Malicious (dishonest majority)	2 parties, 1 honest	BGV HE, ZKPoPK, convolution triples	Homomorphic triple generation + MAC authentication	Secure convolution via triple masking
LowGear 2.0	Matrix multiplication, convolution	Finite field	Malicious (dishonest majority)	2 parties, 1 honest	BGV HE, integrated MAC-authenticated triples	Authenticated triple generation (no sacrifice step)	Multiplication using masked differences

3.2 CDNN15

The goal of the CDNN15 [9] paper is to compute a linear regression model across multiple parties securely without the parties revealing their private datasets. The core idea is to compute the linear regression coefficients, shown in this formula: $\beta = (X^T X)^{-1} X^T y$ where X is the design matrix (feature values), y is the response vector (labels), and β is the regression coefficient vector that needs to be learned. This is done without revealing X or y to the other party.

During the offline phase, a Trusted Initializer (TI) generates correlated randomness, such as Beaver Triples, which are later used to enable secure computation. This phase does not involve any input data, meaning that no sensitive information is exposed. The same functionality could be achieved using homomorphic encryption techniques, such as the Paillier cryptosystem, which allows parties to jointly generate correlated randomness without a trusted party using additive homomorphism.

In the online phase, each party uses their private data, secret-shared using an additive secret sharing scheme over a finite field, and performs the operations using the randomness provided during the offline phase. The operations include the generation and use of Beaver Triples for secure multiplication, fixed-point encoding, secure truncation, and the Newton-Raphson method. Beaver Triples, which are preprocessed during the offline phase, enable secure two-party multiplication by allowing parties to compute products of secret-shared values without revealing intermediate results. Since linear regression involves real-valued inputs, fixed-point encoding is applied by scaling real numbers to integers using a fixed precision parameter before they are secret shared. After multiplications, secure truncation protocols are used during the online phase to rescale fixed-point results, preserving numeric precision while avoiding overflows. The Newton-Raphson method is used to compute the inverse of the covariance matrix, which is required to solve the linear regression equation securely. All these steps are carefully orchestrated to ensure that no party learns the private inputs of the others.

In this thesis, CDNN15 was evaluated on a secure linear regression task using a dataset of 20 samples with 5 features and a corresponding label vector. Each party contributed secret shares of the input, encoded using fixed-point representation with 16-bit fractional precision. The computation followed the standard linear regression formula $\beta = (X^\top X)^{-1} X^\top y$, with all matrix operations executed over secret shares. Matrix inversion was performed using the Newton-Raphson method, and the final result β was securely reconstructed and revealed.

3.3 SecureML

Unlike CDNN15, which is used to compute linear regression models, SecureML [20] computes not only linear models, but also neural network using a two-server MPC model. SecureML operates under a semi-honest adversarial model, meaning that both servers are expected to follow the protocol correctly, but one of them may try to learn additional information from the messages it receives. The security guarantee holds as long as at most one server is corrupted. In the offline phase, the goal is to generate multiplication triplets, that perform secure multiplication on secret-shared data during the training phase. To compute the triples, SecureML uses Linearly Homomorphic Encryption (LHE), a cryptographic technique that allows parties to compute linear operations directly on encrypted data without decryption [20], and Correlated Oblivious Transfer (COT). Using LHE, one of the parties encrypts their share, while the other performs matrix multiplication on that share. The result is masked with randomly generated numbers and sent back for decryption. COT is an optimized variant of Oblivious Transfer designed to reduce communication and computational overhead. In SecureML, it is used to securely precompute multiplication triples. Specifically, one party holds a set of correlated random values, while the other uses a selection bit to choose one of them without revealing its choice.

During the online phase, SecureML performs each batch of stochastic gradient descent only over additive secret-shares. The forward pass multiplies the shared batch matrix X_B with the current weight vector w, thanks to the Beaver triples, which allow secure and efficient multiplication. This results in predictions $\widehat{y}$, which are used to compute the error term by subtracting from y. The gradient is computed as $X_B^T(\widehat{y} - y)$ using Beaver triples. In linear regression, this gradient directly drives the model update, since the loss function is quadratic and has a simple closed-form derivative. In logistic regression, the only difference is that a non-linear activation (sigmoid) is typically required, SecureML approximates it using a piecewise-linear function called double-ReLU, evaluated in a small garbled circuit and re-shared via a Yao-to-arithmetic conversion. The underlying design of garbled circuits is not explored in detail here, as it falls outside the scope of this thesis. After the gradient is calculated, the servers update the weight vector. Just like in CDNN15, secure truncation is applied to maintain the correct precision after multiplication.

This thesis implements three versions of SecureML (linear, logistic and neural network) In this implementation of SecureML linear regression, the input consists of a feature matrix with 100 rows and 10 columns and a label vector with 100 values. In the offline phase, Beaver triples are generated using homomorphic encryption and correlated oblivious transfer. In the online phase, the parties perform mini-batch gradient descent with a batch size of 20 and a learning rate of 0.01 over 5 epochs. For each batch, the protocol computes predictions, calculates the error, computes the gradient, and updates the weight vector.

For logistic regression, SecureML uses a dataset with 100 samples and 10 features. Training is done over 5 epochs with mini-batches of size 20 and a learning rate of 0.01. In the offline phase, Beaver triples are generated using the same homomorphic encryption and COT mechanisms as in the linear case. In the online phase, the protocol applies a piecewise-linear approximation of the sigmoid activation function, known as "double ReLU", on the linear logits. Predictions are compared with the ground truth labels to compute the error, which is then used to compute gradients and update the weight vector. All arithmetic operations rely on the triples prepared in the offline phase.

For the neural network, SecureML trains a simple two-layer neural network with one hidden layer consisting of 8 neurons. The input consists of 40 examples with 10 features each and a label vector of 40 values. Full-batch training is performed over 2 epochs with a learning rate of 0.01. In the offline phase, the protocol generates Beaver triples to support the matrix multiplications required for both forward and backward passes. In the online phase, the protocol executes the forward pass through two layers, applying a ReLU-like activation function using secure comparisons. It then performs backpropagation by computing gradients for both layers and updating the corresponding weight matrices.

3.4 SecureNN

SecureNN [25] is an MPC protocol designed for training convolutional and deep neural networks. Unlike CDNN15 and SecureML, SecureNN is designed for three parties (P_0, P_1, P_2). It assumes that only one of the three servers may act under adversarial control, either in a semi-honest or malicious way, while the other two are honest and follow the protocol. This setup lets the protocol work even if one party is corrupt, while still making sure that the data is correct and private. The system works by allowing multiple data owners to secret-share their input data among three servers, then collaboratively train the model without learning the underlying data. SecureNN supports NN components like matrix multiplication, convolutions, ReLU, Maxpool, and normalization. Importantly, SecureNN avoids the heavy use of garbled circuits which created bottlenecks in earlier protocols such as SecureML.

In the offline phase, the protocol precomputes randomness such as Beaver triples using pseudo-random functions (PRFs), which are cryptographic functions that take a short, secret key and generate output that appears random. PRFs enable parties to generate shared randomness efficiently without relying on expensive public-key cryptography. This eliminates the need for public-key cryptographic tools like homomorphic encryption or oblivious transfer. These precomputed values include random matrices and correlated randomness necessary for secure multiplication. All parties use these shared random values during the online phase to mask intermediate results and maintain security. The system uses additive secret sharing over rings such as $\mathbb{Z}_{2^{64}}$ and, when needed, performs secure conversion to smaller or odd-sized rings $\mathbb{Z}_{2^{64}-1}$ to support certain operations like computing the most significant bit.

During the online phase, SecureNN executes training over secret-shared data using secure building blocks. For linear layers, it uses Beaver triples to compute matrix multiplications efficiently. Convolutions are treated as large matrix multiplications and implemented similarly. For non-linear functions, SecureNN innovatively computes functions like ReLU and Maxpool without using garbled circuits. For example, ReLU is computed by securely extracting the MSB of a number and using it to conditionally select between the input and zero. These comparisons and bit-extraction operations are carried out using efficient three-party protocols based on secure masking, comparisons, and minimal communication. For backpropagation, derivatives such as ReLU' are computed using the same secure comparison mechanisms.

In this thesis, SecureNN was implemented for training a simple neural network on synthetic image-classification data. The workload consisted of 10 mini-batches, each containing 128 samples of flattened grayscale images (784 pixels) with one-hot encoded labels of length 10 (digits 0–9). Party 0 provided the input data, while parties 1 and 2 held no input shares. Training involved secure matrix multiplications, ReLU activations, a softmax-like normalization, and backpropagation with a learning rate of 0.01.

3.5 CKRRSW20

CKRRSW20 [6] tackles maliciously-secure linear-algebra for deep learning in the dishonest-majority setting, where all but one party may behave maliciously. Building on SPDZ, the authors replace Beaver triples with bilinear triples, which are large blocks of correlated randomness that enable the secure evaluation of an entire matrix multiplication or 2D convolution in a single step. By homomorphically generating these "matrix triples" and "convolution triples" with an efficient BFV (Brakerski–Fan–Vercauteren) [6] homomorphic encryption scheme, they cut the communication for an $n \times n$ product from $O(n^3)$ to $O(n^2)$. This works because the input and output matrices, which each being the size of $n \times n$ need to be exchanged, rather than all intermediate values as in element-wise multiplication.

Each party locally samples random matrices A and B and encrypts them under a common BFV public key. A lightweight ZKPoPK guarantees that those ciphertexts really hide small, well-formed values. The parties homomorphically multiply the ciphertexts to obtain an encryption of $C = AB$; inside the same ciphertext they also compute information-theoretic MACs, so $(\langle A\rangle, \langle B\rangle, \langle C\rangle)$ leave preprocessing already authenticated. Because the computation depth is only two, CKRRSW20 eliminates the "triple-sacrifice" step that older SPDZ versions needed, halving both time and bandwidth in preprocessing.

Once a matrix or convolution triple has been made in the offline phase, it can be used to securely multiply two secret-shared inputs together. Each side has its own additive shares of the matrices X and Y, as well as shares of a precomputed triple (A, B, C), where $C = AB$. To accomplish multiplication, the parties show masked discrepancies between their real inputs and the random matrices from the triple. Then, local calculations are used to combine the revealed differences with the common parts of the triple to get the final result.

This thesis implements the CKRRSW20 protocol to securely compute a matrix multiplication using special preprocessing. The input consists of two secret-shared square matrices of size 32×32, both provided entirely by party 0. Party 1 receives no input. The protocol evaluates $Z = X \cdot Y$ without revealing intermediate values or the input matrices. In the offline phase, the protocol simulates the generation of matrix triples (A, B, C), where A and B are random matrices and $C = A \cdot B$ is their product. These triples are computed using local multiplications and are generated in practice using homomorphic encryption and zero-knowledge proofs. In the online phase, the protocol subtracts A and B from the private inputs X and Y, opens the masked differences, and uses them to compute the final result Z by combining the opened values with the preprocessed triple (A, B, C).

3.6 RRHK23

The RRHK23 [22] protocol aims to speed up PPML training and inference when the neural-network layer is a convolution. Operating in the malicious adversary model with a dishonest majority, it introduces convolution triples—random, correlated tensors shaped to match 2D convolutions. Once these triples are in place, each secure convolution during the online phase requires only local additions and a single Beaver-style "reveal-and-recombine" step, ensuring that no party learns the other's inputs. Two implementation variants are provided: a LowGear-style version optimized for two parties with low communication and small proofs, and a HighGear-style version that scales better for more parties.

Like SPDZ and Overdrive, RRHK23 uses a preprocessing phase separate from the online phase, but does not require a trusted initializer. Instead, the necessary randomness is jointly generated using homomorphic encryption and zero-knowledge proofs. Each party encrypts random plaintext images A and filters B using a shared BGV public key and provides zero-knowledge proofs to confirm the encrypted values are properly structured. With efficient polynomial-packing, the convolution result $C = A * B$ is computed using a single ciphertext-ciphertext multiplication, even across batches. The protocol also includes MAC-based authentication with a global key α,which ensures that values remain verifiable and untampered with across the computation.

In the online phase, each data owner secret-shares their input image X, and the model owner shares their filter Y with the servers. The servers use a precomputed triple (A, B, C) to open the masked differences $\triangle X = X - A$ and $\triangle Y = Y - B$. They then use $Z = C + \triangle X \cdot B + A \cdot \triangle Y + \triangle X \cdot \triangle Y$ to find the answer locally, getting their part of the convolution, which reconstructs $Z = X \cdot Y$ without further interaction. The output is then either revealed using authenticated openings or passed as input to the next layer in the neural network pipeline. For non-linear layers such as ReLU and MaxPool, RRHK23 reuses SecureNN's efficient three-party comparison protocols to compute them securely without revealing intermediate values.

For RRHK23, this thesis benchmarks the original implementation provided by the authors using their Docker repository. No modifications were made to the protocol. The input consists of secret-shared tensors configured to match a simple 3×3 convolutional layer, one of the supported operations in the RRHK23 framework. In the offline phase, the protocol generates convolution triples using homomorphic encryption and zero-knowledge proofs to ensure correctness under a malicious adversary model. These triples contain random input and filter tensors, as well as their corresponding convolution outputs, all authenticated using MACs. In the online phase, the parties provide their private inputs and perform a single convolution step using the precomputed values. The output is securely reconstructed using authenticated triples.

3.7 LowGear 2.0

LowGear 2.0 [21] is a newer member of the SPDZ family of MPC protocols. Its goal is simple: make the "offline" setup cheaper while keeping full malicious-security, even when most parties could cheat. LowGear 2.0 does not use traditional preprocessing methods that separate triple generation and authentication. Instead, it tightly links the two steps. This is especially useful for secure machine learning, where many authenticated multiplications, like matrix products and convolutions, are needed.

The key innovation is to create the random Beaver triples and their authenticity checks in one shot, so a separate “sacrifice” step (used in older SPDZ versions) is no longer needed. The sacrifice procedure in SPDZ involves producing additional Beaver triples and thereby exposing a subset of these at random to confirm the accuracy of the MACs, with any invalid triples eliminated. This introduces additional rounds of communication and computation, however guarantees the reliability of the remaining triples. By including authentication during the production of triples, LowGear 2.0 eliminates the need for this additional verification step. This alone cuts the data sent per triple by roughly a third and shortens the setup from three network rounds to two.

In the offline phase, each party randomly picks inputs a and b (which can be scalars, vectors, or matrices depending on the computation) and then encrypts them using lightweight BGV homomorphic encryption [21]. We use these ciphertexts to homomorphically compute $c = a \cdot b$ and at the same time add a MAC tag to the result using a global secret key α. There is no need for a ”triple sacrifice” step because the MAC is calculated during encryption. Also, no extra triples are thrown away. This makes communication much less necessary and increases efficiency.

In the online phase, when the data arrives, secure multiplication still works the classic Beaver way: reveal masked differences, do a few local additions. If a square, inner product, or a big matrix multiplication shows up, LowGear 2.0 can burn one of its special tuples to save about half the traffic for that step. For neural networks, treating a whole convolution as one large matrix multiply means often only a handful of triples per layer are required.

This thesis implements LowGear 2.0 with the input consisting of a 64×128 matrix provided by party 0 and a 128×32 matrix provided by party 1. The goal is to compute the matrix product $C = A \cdot B$ without revealing either input. In the offline phase, the protocol generates Beaver triples that are already authenticated with MAC tags, using a homomorphic encryption-based mechanism. In the online phase, the parties input their secret shares of matrices A and B, and use the preprocessed authenticated triples to securely compute the matrix product. The result matrix C is then revealed to both parties.

4 Methodology

This section discusses how the six different MPC protocols are going to be evaluated. This thesis focuses on two main criteria: computation and communication costs and security characteristics. The subsections below explain the criteria used and the experimental setup on MP-SPDZ.

4.1 Evaluation Criteria

This thesis considers the runtime and communication demands during the offline and online phases to evaluate how well each protocol performs. Performance is measured using two metrics: computation time (seconds) and communication volume (MB). Both metrics are recorded separately for the offline and online phases to capture the costs of preprocessing as well as the costs of the actual secure computation.

The analysis examines a protocol’s adversarial model and corruption threshold to assess its security. The adversarial model specifies whether the protocol presumes semi-honest adversaries, who adhere to the protocol while attempting to acquire further information, or malicious adversaries, who may deviate arbitrarily. Protocols designed to withstand malicious adversaries typically require more robust cryptographic mechanisms and increased interaction, although they offer stronger guarantees of correctness and privacy. The corruption threshold indicates the number of parties that can be compromised before the guarantees fail: a higher threshold signifies greater robustness but generally increases costs. Collectively, these two elements reflect the level of security attained by the protocol.

Cryptographic primitives also indicate the strength of security. Linearly Homomorphic Encryption (LHE) guarantees that operations performed on encrypted data do not disclose plaintext values, enhancing privacy during preprocessing. Oblivious Transfer (OT) enables parties to choose inputs without disclosing their selections, preventing information leakage during the selection process. Zero-Knowledge Proofs (ZKPs) enable one side to demonstrate correctness without revealing its inputs, which is essential in malicious settings to prevent adversaries from supplying erroneous data. Protocols that employ these primitives generally provide stronger assurances, but at the expense of increased computation time and communication volume.

This review aims to identify protocols that optimise the balance among privacy, performance, and scalability. The balance is assessed by evaluating: the privacy level indicated by the adversary model, corruption threshold, and primitives used; the computation time and communication volume in both offline and online phases; and the scalability of these costs with respect to larger datasets and additional parties. A protocol is considered well-balanced if it ensures adequate security for its designated adversarial model while controlling costs and preserving practicality for real machine learning applications. Scalability refers to how a protocol's costs grow as the dataset, model size, or number of parties increases. In this thesis, protocols are considered more scalable if their computation time and communication volume grow moderately with larger workloads, rather than becoming prohibitive.

4.2 Experimental Setup

Table 2: MacBook Pro (13-inch, 2020) System Specifications

Component	**Specification**
Model	MacBook Pro (13-inch, 2020, Four Thunderbolt 3 ports)
Processor	2 GHz Quad-Core Intel Core i5
Graphics	Intel Iris Plus Graphics, 1536 MB
Memory	16 GB 3733 MHz LPDDR4X
Operating System	macOS Sonoma 14.6.1

The MP-SPDZ [16] framework was used for all experiments (hardware specifications are provided in Table 2). Each protocol was compiled with the appropriate arithmetic domain flags (e.g., `-F 64` for finite fields and `-R 64` for integer rings) using the `compile.py` script. All runs were executed on a single machine (localhost); no LAN or WAN measurements were taken, and therefore, the reported timings and communication costs exclude real network effects. Protocols were executed with their respective party executables (e.g., `mascot-party.x`, `semi-party.x`, `lowgear-party.x`) together with the `-v` flag to capture detailed logs. For local benchmarking, the `--unencrypted` option was occasionally used to disable secure sockets.

The MP-SPDZ framework automatically records both computation time and communication volume for each run. These values are printed directly to the terminal after execution. Computation time is reported in seconds for both the offline and online phases, while communication is reported in bytes. The measurements were therefore taken directly from the MP-SPDZ output, without requiring any external tools.

Table 3: MP-SPDZ Party Executables Used in This Thesis

Executable	**Security Model**	**Arithmetic Domain**
`mascot-party.x`	Malicious (Dishonest Majority)	Finite Field
`replicated-ring-party.x`	Malicious (Honest Majority)	Integer Ring
`semi-party.x`	Semi-Honest (Flexible)	Field or Ring
`lowgear-party.x`	Malicious (Dishonest Majority)	Finite Field

Custom Python scripts were created for each protocol so that each protocol received the expected input structure. Each script allowed setting a custom random seed, ensuring that all protocols were tested on equivalent input distributions, which made the comparisons fair and consistent. The verbose flag `-v` was used to get detailed logs of both the offline and online phases, including metrics for runtime and communication. In some cases, extra shell tools were used to get the process logs so they could be compared across protocols.

Each experiment was repeated five times to reduce run-to-run variability ("noise") from OS scheduling, caching, and randomized seeds and to report stable averages. To avoid interference from other processes, all experiments were run on a single machine by themselves. This controlled and consistent experimental setup makes sure that the comparison of preprocessing protocols is fair and can be repeated, which makes it possible to get reliable performance benchmarks in a wide range of secure machine learning settings. This controlled and consistent experimental design makes guarantee that the comparison of preprocessing techniques is fair and can be repeated. All the code needed to run the experiments (except RRHK23), could be found at this github repository `https://github.com/LaPetiteBird/six-mpc-preprocessing.git`. The code used in this thesis for RRHK23 could be accessed from their repository [22].

Each protocol was tested on workloads matching those described in its original paper and summarised in Chapter 3. For CDNN15, the workload consisted of a secure linear regression on a dataset with 20 samples and 5 features. SecureML was evaluated on three workloads: linear regression (100 samples, 10 features), logistic regression (100 samples, 10 features), and a two-layer neural network (40 samples, 10 features, hidden layer of 8 neurons). SecureNN was tested on a simple image-classification task using batches of 128 synthetic MNIST-like vectors of dimension 784 with one-hot encoded labels of size 10. CKRRSW20 and LowGear 2.0 were both benchmarked on secure matrix multiplication, with input matrices of size 32×32 and 64×128 by 128×32 respectively. RRHK23 was tested on a secure 2D convolution using 3×3 filters.

In addition to recording computation time and communication volume, this thesis also considers the scalability of each protocol. Scalability is not measured directly through increasing dataset sizes or party counts in this work, but rather inferred from the design of the workloads and the structure of the protocols. In practice, scalability is reflected in how well each protocol's offline and online costs would grow if the input matrices, number of samples, or number of parties were scaled up. The workloads chosen in this study (see above) therefore serve as representative baselines from which scalability trends can be discussed in Chapter 5.

All protocols were evaluated in the two-party setting, except for SecureNN which inherently requires three parties. This choice aligns with the design of CDNN15, SecureML, CKRRSW20, RRHK23, and LowGear 2.0, all of which target two-party computation (see Table 1). Using two parties also reflects common PPML deployment models such as client–server setups and ensures comparability across protocols.

Scalability was not measured systematically across all protocols. Only SecureML was evaluated on multiple workloads of increasing complexity (linear, logistic, and neural), which allows some limited observations on how performance changes as the task grows. For the remaining protocols, scalability is inferred from their design rather than from direct experimental evidence.

5 Comparative Evaluation

This section evaluates the six preprocessing-based MPC protocols in terms of computational time, communication volume, and security trade-offs. The chosen workloads are both indicative of the procedures for which each protocol was initially developed and correspond directly to practical machine learning jobs. Secure linear regression and logistic regression (CDNN15 and SecureML) exemplify predictive modelling problems in sectors such as healthcare and finance. Neural network training tasks (SecureML and SecureNN) pertain to classification challenges, including image and speech recognition. Benchmarks for matrix multiplication and convolution (CKRRSW20, RRHK23, LowGear 2.0) simulate the secure execution of fundamental layers in deep learning architectures, such as CNNs, which support applications including medical image analysis, facial recognition, and natural language processing.

5.1 Performance Metrics

The main performance parameters tracked are offline time, online time, and communication overhead. Offline time is the amount of time it takes to create the preprocessed resources, like Beaver triples, matrix triples, or convolution triples, before any inputs are known. Online time tells how fast each protocol can do secure computation once it has access to private inputs. Communication overhead is the entire amount of data that is sent and received between the two parties during both phases.

It is crucial to establish a clear distinction between conducting these investigations on localhost, over a local area network (LAN), and across a wide area network (WAN). On localhost, all parties operate on the same physical machine, resulting in zero network latency, no packet loss, and an unlimited bandwidth in relation to the process requirements. In general, LAN configurations are fast and stable, despite the introduction of some latency and bandwidth limitations, which are a result of the multiple machines connected within the same local network. In contrast, WAN environments are characterised by multiple machines across remote locations, such as the public internet. In these environments, the runtimes of both online and offline operations can be substantially impacted by higher latency, limited bandwidth, and unpredictable network conditions. The benchmarks in this thesis were conducted on localhost, rather than in real-world deployment scenarios. These metrics already indicate that the protocols realise different trade-offs. For example, CDNN15 achieves extremely low online latency (0.0586 seconds, 0.382 MB) but requires heavier offline computation (84.7 seconds, 846 MB). SecureNN shows the opposite tendency, with modest offline demands (48.3 seconds, 1.19 GB) but very high online interaction (115 seconds, 200 MB). Such contrasts highlight the importance of analysing both phases separately when comparing protocols.

5.2 Experimental Results

The results in Table 4 and Table 5 show distinct patterns across the different protocol families. For CDNN15 and SecureML, most of the cost is in the offline phase, which allows their online stage to run with very low latency. This property makes them particularly well suited to inference tasks where quick responses are important. SecureNN presents the opposite trade-off: because of its interactive three-party design, a larger share of the workload shifts into the online phase, resulting in heavier communication. Within the maliciously secure protocols, CKRRSW20 achieves relatively modest costs in both phases due to its homomorphic triple generation, while RRHK23 requires the most expensive preprocessing but benefits from efficient convolutions once that setup is complete. LowGear 2.0 falls between these extremes, with a somewhat demanding offline phase but a very lightweight online execution. Taken together, the results underline a broader trend: protocols targeting semi-honest security tend to minimise online latency, whereas those designed for malicious adversaries trade higher preprocessing overheads for stronger security guarantees.

The data also supports the trade-off patterns expected from each protocol design. Semi-honest protocols such as SecureML invest heavily in the offline phase (up to 940 seconds and 9.1 GB for the neural workload) to achieve fast online execution (2.88 seconds, 3.6 MB). Maliciously secure protocols such as RRHK23 accept much higher preprocessing costs (625 seconds, 7.0 GB) in order to support secure convolutions with relatively modest online requirements (14.4 seconds, 22 MB). LowGear 2.0 demonstrates an intermediate strategy, with moderate offline cost (64 seconds, 1.5 GB) and the lowest online latency among malicious protocols (0.23 seconds, 8.6 MB).

Table 4: Computation Time for Offline and Online Phases

Protocol	**Parties**	**Offline Computation Time (Average in Seconds)**	**Online Computation Time (Average in Seconds)**
CDNN15	2	84.73	0.0586
SecureML	2	Linear: 277.86 Logistic: 776.30 Neural: 940.44	Linear: 0.4590 Logistic: 1.2004 Neural: 2.8778
SecureNN	3	48.30	114.72
CKRRSW20	2	18.98	1.544
RRHK23	2	624.66	14.38
LowGear 2.0	2	64.4	0.229

Table 5: Communication Cost for Offline and Online Phases

Protocol	**Parties**	**Offline Communication Volume (Average in MB)**	**Online Communication Volume (Average in MB)**
CDNN15	2	845.57	0.3821
SecureML	2	Linear: 2829.56 Logistic: 7332.24 Neural: 9126.61	Linear: 1.4654 Logistic: 2.5062 Neural: 3.6315
SecureNN	3	1190.25	200.31
CKRRSW20	2	101.56	1.081
RRHK23	2	7014.05	22.48
LowGear 2.0	2	1524.89	8.554

The workloads therefore function as approximations for realistic PPML applications, including regression models for tabular prediction tasks, neural networks for classification and decision making, and matrix/convolution operations for vision-based deep learning. The results emphasise the protocol's suitability for practical deployment scenarios in addition to their raw performance, as a consequence of the evaluation's foundation in these standard ML use cases.

The data in Tables 4 and 5 are taken from the tables found in the appendix A. They indicate notable differences in the computational and communication efficiency of the protocols, with CDNN15 and CKRRSW20 identified as relatively lightweight options, while SecureML, SecureNN, RRHK23 and LowGear 2.0 exhibit greater resource demands. The average offline computing time for CDNN15 is 84.73 seconds, whereas the average online computation time is 0.0586 seconds. The cost of offline communication is 845.57 MB, whereas the cost of online communication is 0.3821 MB. Three workloads were evaluated for SecureML. For linear regression, the offline computation averages 277.86 seconds, whereas online computation averages 0.4590 seconds. Offline communication averages 2829.56 MB, whereas online communication averages 1.4654 MB. For logistic regression, the average duration for offline computing is 776.30 seconds, whereas the average duration for online computation is 1.2004 seconds. Offline communication averages 7332.24 MB, whereas online communication averages 2.5062 MB. For neural network training, the offline computation averages 940.44 seconds, whereas online computation averages 2.8778 seconds. Offline communication averages 9126.61 MB, whereas online communication averages 3.6315 MB. For SecureNN, including three parties, the average offline computation duration is 48.30 seconds, whereas the average online computation duration is 114.72 seconds. The average cost of offline communication is 1190.25 MB, whereas the average cost of online communication is 200.31 MB. For CKRRSW20, the average offline computing time is 18.98 seconds, but the average online computation time is 1.544 seconds. Offline communication averages 101.56 MB, whereas online communication averages 1.081 MB. For RRHK23, the average offline computing time is 624.66 seconds, while the average online computation time is 14.38 seconds. Offline communication amounts to 7014.05 MB, and online communication totals 22.48 MB. For LowGear 2.0, the average offline computing duration is 64.4 seconds, whereas the average online computation duration is 0.229 seconds. The cost of offline communication is 1524.89 MB, whereas the cost of online communication is 8.554 MB.

5.3 Discussion of Trade-offs (Efficiency vs. Security)

Table 6: Protocol characteristics relevant to performance discussion

Protocol	**Adversary Model**	**Key Offline Primitive(s)**	**Offline Cost**	**Online Cost**
CDNN15	Semi-honest	Beaver triples, auxiliary randomness	Low	High
SecureML	Semi-honest	LHE-based triple generation, COT	Medium	Medium
SecureNN	Semi-honest / Malicious	PRF-based triple generation	Medium	Medium
CKRRSW20	Malicious	HE-based matrix/convolution triples, ZK proofs	Very High	Low
RRHK23	Malicious	HE-based convolution triples, MAC authentication	Very High	Low
LowGear 2.0	Malicious	Authenticated HE triples (no sacrifice)	High	Low

All protocol trade-offs discussed in this section are summarised in Table 6, which provides a comparative overview of their adversary models, offline primitives, and offline/online costs. The six protocols vary in terms of computing efficiency, communication overhead, and security assurances, with their performance showing the inherent trade-offs of their designs.

CDNN15 exhibits one of the most minimal results regarding online execution. CDNN15 shows an average computation time of merely 0.0586 seconds and an online communication cost of 0.3821 MB. This is however counterbalanced by significant offline requirements: 84.73 seconds of processing and 845.57 MB of transmission. As shown in Table 6, CDNN15 relies on Beaver triples and auxiliary randomness during the offline phase, which keeps preprocessing cost low but leaves more computation for the online phase, resulting in higher online runtimes. According to the 'Adversary Model' column in Table 6, CDNN15 targets semi-honest environments, making it suitable for deployments where the offline phase can be prepared in advance without concern for active adversaries.

SecureML is a design tailored for semi-honest security and two-party computation. The findings indicate that efficiency declines as workload complexity increases. In linear regression, the online computation and communication are moderate at 0.4590 seconds and 1.4654 MB respectively. For logistic regression and neural networks, the online phase extends to 1.2004 seconds and 2.8778 seconds, with communication expenses increasing to 2.5062 MB and 3.6315 MB, respectively. Offline requirements are higher, varying from 277.86 to 940.44 seconds of processing time and 2829.56 to 9126.61 MB of data transmission. The costs associated with these figures arise from SecureML's implementation of LHE and COT in the preprocessing phase. Although SecureML provides a straightforward setup and satisfactory speed for minor tasks, its semi-honest framework and associated scaling expenses render it less feasible for adversarial or more computationally intense implementations.

SecureNN optimises speed and security with more memory usage. The three party architecture shows a brief offline computing time of 48.30 seconds, although entails an online computation duration of 114.72 seconds and a online communication requirement of 200.31 MB. These numbers stem from SecureNN's preference for repeated secret sharing and PRF-based randomisation over public-key cryptography, which diminishes offline workload but necessitates continuous communication during training, particularly for non-linear layers like as ReLU and MaxPool. The three party requirement constrains its deployment efficiency, even if it offers superior resilience against individual corruptions compared to semi-honest two-party systems.

CKRRSW20 is the lightest maliciously secure protocol evaluated, requiring merely 18.98 seconds of offline computation and 1.544 seconds of online execution. The communication cost is modest, measuring 101.56 MB offline and 1.081 MB online. These results arise because the protocol uses BFV-based homomorphic generation of matrix and convolution triples, effectively packing large multiplications into small, ready-to-use blocks that already carry MACs for correctness checks. This efficiency necessitates a complex homomorphic encryption framework and depends on zero-knowledge proofs for validation, which, although enhancing security, significantly increases implementation complexity relative to semi-honest architectures.

RRHK23 prioritises security and optimisations tailored for convolution at the expense of speed. Although its offline computation (624.66 seconds) is worse than SecureML's maximum values, its online duration is comparatively elevated at 14.38 seconds, and the communication overhead is considerable: 7014.05 MB offline and 22.48 MB online. The statistics arise from the protocol's creation of convolution triples utilising homomorphic encryption and zero-knowledge proofs, facilitating rapid convolutions in the online phase, however with significant data transfer during preprocessing. RRHK23's security features and focus on convolutional layers render it advantageous in adversarial contexts, although it could be challenging to implement it in limited networks or with extensive datasets.

LowGear 2.0 reduces runtime by compromising setup, as seen by its 0.229 seconds of online computation and 8.554 MB of online communication. The lowest online metrics among maliciously secure protocols. Nonetheless, these savings are offset by substantial offline demands: 64.4 seconds of processing and 1524.89 MB of transmission. The trade-offs occur because LowGear 2.0 combines Beaver triple creation and MAC authentication into a single step, hence reducing the need for "sacrifice" verification rounds but increasing the preprocessing workload. This design works well for large deployments where offline costs can be spread out, but it is less suitable when fast setup or low-bandwidth preprocessing is needed

The findings indicate that protocols optimised for semi-honest security (CDNN15 and SecureML) exhibit reduced offline complexity but lack robustness, whereas those tailored for malicious adversary models (CKRRSW20, RRHK23, and LowGear 2.0) sacrifice greater offline computation and communication in exchange for enhanced security during the online phase. SecureNN sits in the middle, avoiding public-key cryptography and keeping offline costs moderate, but it requires heavy online interaction and three parties to run.

5.4 Suitability for Real-World Use Cases

CDNN15 or SecureML may be enough for applications with harsh time limits but mild security needs, such as classifying medical images within trusted hospital environments. SecureNN is a good choice for speed and security when deployment happens in semi-trusted contexts, like federated banks.

CKRRSW20 and RRHK23 are better for scenarios with strong adversarial models, including sharing data across governments or countries, because they have malicious security models. CK-RRSW20 is great for ML tasks that need a lot of structure, like safe matrix multiplication. For convolutional networks with small batch sizes, RRHK23 is the best choice.

LowGear 2.0 is appropriate for big ML deployments where communication costs need to be kept to a minimum while the program is running. It can be used with MP-SPDZ and supports a wide variety of operations, which makes it easy to use in many environments.

6 Conclusion and Future Work

This part of the thesis talks about the main results and what they mean for safe machine learning utilising particular preprocessing methods. It also talks about possible future study.

6.1 Limitations

Although the implementations in this thesis were developed using MP-SPDZ [16] and were designed to follow each protocol as described in the original publications, several limitations should be noted.

First, while the experiments maintained consistent matrix dimensions and inputs across protocols, they did not investigate scalability beyond the tested batch sizes. As a result, the conclusions may not directly extend to substantially larger models such as ResNet-152 or BERT.

Second, all experiments were carried out on a localhost setup. This ensured consistency and removed the variability introduced by network conditions, but it also prevented an accurate assessment of protocol performance in real network environments. The results therefore do not reflect the potential effects of network latency, bandwidth constraints, or packet loss—factors that are especially relevant for preprocessing-heavy protocols like SecureML and LowGear 2.0. While the reported computation times and baseline communication volumes are correct for a controlled setting, actual deployments over LAN or WAN connections may produce noticeably different runtimes.

A further limitation concerns the difference in functional scope across the evaluated protocols. SecureML was tested on three complete machine learning workloads—linear regression, logistic regression, and neural network training—whereas CDNN15, CKRRSW20, LowGear 2.0, and RRHK23 were each evaluated on only a single computational primitive. This creates an imbalance in functional complexity, meaning that the comparisons in the findings chapter mainly reflect the performance of these specific operations rather than equivalent end-to-end workloads. This distinction should be kept in mind when interpreting the results and assessing the scalability of each preprocessing method.

A further limitation is that scalability was not evaluated uniformly across all six protocols. Only

SecureML was tested on multiple workloads (linear, logistic, and neural), providing some indication of how performance changes with task complexity. The other protocols were each benchmarked on a single representative workload, which restricts the ability to draw general conclusions about their scalability. Future work should therefore include a broader range of workloads for each protocol to allow a more balanced comparison.

Lastly, all experiments were run exclusively on macOS using Intel-based hardware. Performance results may therefore differ when reproduced on other platforms, such as AMD or ARM architectures, or in cloud-based virtual machines.

6.2 Future Directions

This thesis considers methods that could lead to many different research paths. First, experiments should be expanded to include neural networks that are bigger and deeper. Finding out how well these preprocessing methods work when they are used on huge CNNs or transformers could help a lot with deciding whether or not to use them. Second, it might be possible to look at hybrid protocol integration. For example, integrating CKRRSW20's secure matrix triple creation with SecureNN's rapid arithmetic might make trade-offs more worthwhile.

These benchmarks should be expanded to encompass distributed environments, specifically over LAN and WAN networks, in future research. This would enable the measurement of the impact of network latency, packet loss, and bandwidth limitations on both the offline and online phases. These tests would offer a more realistic measurement on the computation and communication cost of these six MPC protocols.

In general, this thesis shows how important specialised preprocessing is for secure machine learning and gives real-world examples of how to compare MPC protocols. The ideas and technologies evaluated here will help with future research and real-world use of AI systems that protect people's privacy.

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

[2] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology—CRYPTO'91*, pages 420–432. Springer, 1992.

[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[4] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. FLASH: Fast and robust framework for privacy-preserving machine learning. Cryptology ePrint Archive, Paper 2019/1365, 2019.

[5] Hyunho Cha, Intak Hwang, Seonhong Min, Jinyeong Seo, and Yongsoo Song. MatriGear: Accelerating authenticated matrix triple generation with scalable prime fields via optimized HE packing. Cryptology ePrint Archive, Paper 2024/1502, 2024.

[6] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously secure matrix multiplication with applications to private deep learning. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, 2020, Proceedings*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 31–59, Germany, 2020. Springer Science and Business Media Deutschland GmbH. Publisher Copyright: © 2020, International Association for Cryptologic Research.; 26th International Conference on the Theory and Application of Cryptology and Information Security, ASIACRYPT 2020 ; Conference date: 07-12-2020 Through 11-12-2020.

[7] Hao Chen, Kim Laine, and Rachel Player. Simple encrypted arithmetic library - SEAL v2.1. Cryptology ePrint Archive, Paper 2017/224, 2017.

[8] Valerie Chen, Valerio Pastro, and Mariana Raykova. Secure computation for machine learning with spdz, 2019.

[9] Martine de Cock, Rafael Dowsley, Anderson C.A. Nascimento, and Stacey C. Newman. Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, AISec '15, page 3–14, New York, NY, USA, 2015. Association for Computing Machinery.

[10] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing.* Cambridge University Press, 2015.

[11] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure mpc for dishonest majority – or: Breaking the spdz limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security – ESORICS 2013*, pages 1–18, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[12] Wenliang Du and Mikhail J. Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *Proceedings of the 2001 Workshop on New Security Paradigms*, NSPW '01, page 13–22, New York, NY, USA, 2001. Association for Computing Machinery.

[13] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, USA, 2004.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition.* Chapman & Hall/CRC, 2nd edition, 2014.

[16] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. Cryptology ePrint Archive, Paper 2020/521, 2020.

[17] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making spdz great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 158–189, Cham, 2018. Springer International Publishing.

[18] Yehuda Lindell. Secure multiparty computation (MPC). Cryptology ePrint Archive, Paper 2020/300, 2020.

[19] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. When machine learning meets privacy: A survey and outlook. *CoRR*, abs/2011.11819, 2020.

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

[21] Pascal Reisert, Marc Rivinius, Toomas Krips, and Ralf Kuesters. Overdrive lowgear 2.0: Reduced-bandwidth mpc without sacrifice. pages 372–386, 07 2023.

[22] Marc Rivinius, Pascal Reisert, Sebastian Hasler, and Ralf Küsters. Convolutions in overdrive: Maliciously secure convolutions for mpc. *Proceedings on Privacy Enhancing Technologies*, 2023:321–353, 07 2023.

[23] Giovanni Sartor and Francesca Lagioia. The impact of the general data protection regulation (gdpr) on artificial intelligence. Study EPRS_STU(2020)641530, European Parliamentary Research Service (EPRS), 2020. Scientific Foresight Unit (STOA).

[24] Shuang Sun and Eleftheria Makri. SoK: Multiparty computation in the preprocessing model. Cryptology ePrint Archive, Paper 2025/060, 2025.

[25] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019:26–49, 07 2019.

[26] Sergey Zapechnikov. Secure multi-party computations for privacy-preserving machine learning. *Procedia Computer Science*, 213:523–527, 2022. 2022 Annual International Conference on Brain-Inspired Cognitive Architectures for Artificial Intelligence: The 13th Annual Meeting of the BICA Society.

[27] Ian Zhou, Farzad Tofigh, Massimo Piccardi, Mehran Abolhasan, Daniel Franklin, and Justin Lipman. Secure multi-party computation for machine learning: A survey. *IEEE Access*, 12:1–1, 01 2024.

A Appendix

Table 7: CDNN15 Performance Metrics

	Computation Time (s)		Communication (MB)	
Run	Online	Offline	Online	Offline
1	0.0422	84.0121	0.3821	845.57
2	0.0573	86.3688	0.3821	845.57
3	0.0782	83.1649	0.3821	845.57
4	0.0515	85.2350	0.3821	845.57
5	0.0637	84.8921	0.3821	845.57
Average	0.0586	84.7346	0.3821	845.57

Table 8: SecureML (Linear) Performance Metrics

	Computation Time (s)		Communication (MB)	
Run	Online	Offline	Online	Offline
1	0.471	283.7	1.475	2829.56
2	0.456	266.6	1.451	2829.56
3	0.449	283.7	1.468	2829.56
4	0.462	275.2	1.460	2829.56
5	0.457	280.1	1.473	2829.56
Average	0.4590	277.86	1.4654	2829.56

Table 9: SecureML (Logistic) Performance Metrics

	Computation Time (s)		Communication (MB)	
Run	Online	Offline	Online	Offline
1	1.160	775.7	2.496	7332.24
2	1.231	779.4	2.514	7332.24
3	1.205	772.1	2.501	7332.24
4	1.189	776.3	2.508	7332.24
5	1.217	778.0	2.512	7332.24
Average	1.2004	776.30	2.5062	7332.24

Table 10: SecureML (Neural) Performance Metrics

	Computation Time (s)		Communication (MB)	
Run	Online	Offline	Online	Offline
1	2.928	954.1	3.6267	9126.61
2	2.883	934.9	3.633	9126.61
3	2.800	927.7	3.6374	9126.61
4	2.910	945.3	3.6295	9126.61
5	2.868	940.2	3.6311	9126.61
Average	2.8778	940.44	3.6315	9126.61

Table 11: SecureNN Performance Metrics

	Computation Time (s)		Communication (MB)	
Run	Online	Offline	Online	Offline
1	109.8	53.4	200.35	1190.25
2	114.9	52.7	200.13	1190.25
3	119.5	41.0	200.37	1190.25
4	112.3	48.6	200.28	1190.25
5	117.1	45.8	200.40	1190.25
Average	114.72	48.30	200.31	1190.25

Table 12: CKRRSW20 Performance Metrics

	Computation Time (s)		Communication (MB)	
Run	Online	Offline	Online	Offline
1	1.575	19.1	1.081	101.56
2	1.515	18.9	1.081	101.56
3	1.542	19.0	1.081	101.56
4	1.560	18.9	1.081	101.56
5	1.528	19.0	1.081	101.56
Average	1.544	18.98	1.081	101.56

Table 13: RRHK23 Performance Metrics

	Computation Time (s)		Communication (MB)	
Run	Online	Offline	Online	Offline
1	14.64	641.2	22.48	7014.05
2	14.02	605.6	22.48	7014.05
3	14.41	623.5	22.48	7014.05
4	14.25	634.1	22.48	7014.05
5	14.58	618.9	22.48	7014.05
Average	14.38	624.66	22.48	7014.05

Table 14: LowGear 2.0 Performance Metrics

	Computation Time (s)		Communication (MB)	
Run	Online	Offline	Online	Offline
1	0.266	65.7	8.554	1524.89
2	0.194	63.2	8.554	1524.89
3	0.231	64.5	8.554	1524.89
4	0.242	64.9	8.554	1524.89
5	0.212	63.8	8.554	1524.89
Average	0.229	64.4	8.554	1524.89