Universiteit Leiden

Master Computer Science

Privacy-Preserving Machine Learning in the Training Phase: A Comparative Study of Privacy Enhancing Technologies

Name:	Qiuxuan Ma
Student ID:	3798976
Date:	28/07/2025
Specialisation:	Data Science
1st supervisor:	Eleftheria Makri
2nd supervisor:	Nusa Zidaric

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Contents

Abstract

Privacy-Preserving Machine Learning (PPML) is a methodology designed to maintain data privacy and security throughout the machine learning pipeline. Research in this area mainly focuses on two stages of machine learning: training and inference. Training refers to the process by which a model learns from large datasets, while inference involves using the trained model to produce results based on new input data. Studies on PPML during the training stage are relatively limited due to the high computational costs and the additional burden introduced by Privacy-Enhancing Technologies (PETs). This study applies two commonly used PETs: Homomorphic Encryption (HE) and Secure Multiparty Computation (MPC), to the training phase of classification tasks. By using open-source libraries such as TenSEAL and CrypTen, the model can be trained on encrypted input data without compromising data privacy. We also compare and analyze the performance, accuracy, and communication costs of using HE and MPC. The results show that both methods can achieve accuracy levels close to those of plaintext models, though they introduce significant performance overhead, particularly in the case of HE. Regarding communication costs, HE outperforms MPC in terms of communication time but results in a larger transmission data size. Despite the increased overhead, HE and MPC offer clear benefits, including regulatory compliance, privacy protection, and comparable accuracy.

Keywords: Privacy-Preserving Machine Learning, Homomorphic Encryption, Secure Multiparty Computation, GDPR

1 Introduction

This project was initiated by Bynder, a company that provides Digital Asset Management software to business customers. The objective of the project is to explore Privacy-Preserving Machine Learning (PPML) during the training phase, leveraging transfer learning with a transformer architecture and using images as the primary input data. Transformer is a deep learning architecture that uses the attention mechanism to capture relationships between inputs and outputs, rather than relying on recurrence or convolutions [1].

Nowadays, many corporations provide Artificial Intelligence (AI) features to enhance the user experience. From intelligent assistants (chatbots) to autonomous agents, these products have become increasingly efficient and appealing. However, privacy concerns cannot be overlooked. The European General Data Protection Regulation (GDPR) requires organizations handling customer data to state how the data will be used clearly, and prohibits its use for other purposes without obtaining additional consent [2]. Similarly, the Health Insurance Portability and Accountability Act (HIPAA) requires that hospitals and insurance companies in the United States keep patients' medical records secure and private. The California Consumer Privacy Act (CCPA), a state-level data privacy law, grants individuals the right to know what personal data is collected, request deletion of their data, and opt out of the sale of their data [3]. International companies that serve customers worldwide must comply with various local regulations. Corporations providing

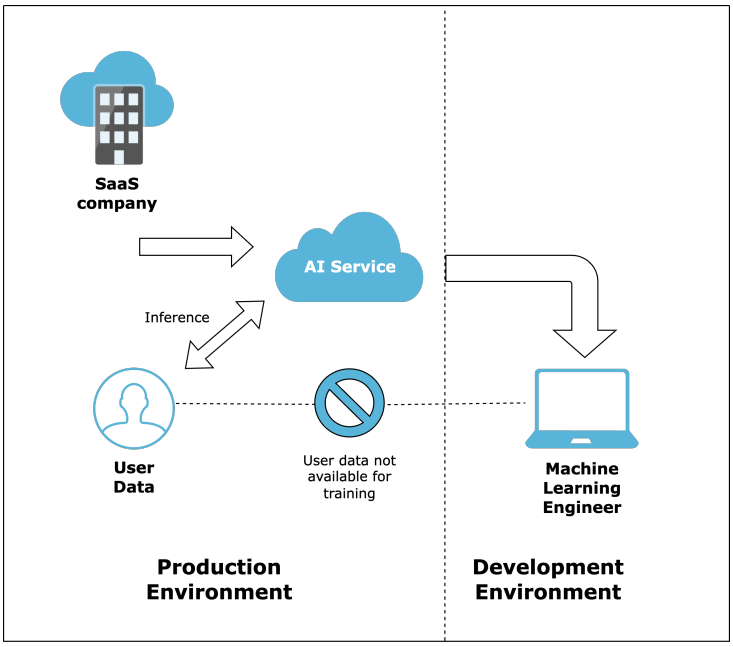


Figure 1: Machine Learning development pipeline

AI features typically assert that they will not use customer data for training purposes without explicit consent. In such cases, machine learning engineers are not allowed to access and use customer data for model training. Customer data is only used during the inference stage in the production environment, where it serves as input to generate output. A typical machine learning development pipeline used within a company is illustrated in Figure 1.

The main objective of this research is to enable machine learning engineers to utilize customer data earlier in the ML pipeline without compromising customer privacy or violating regulations. Involving actual data in the training stage would be beneficial, as fine-tuning a pre-trained model on target datasets (i.e., datasets used to adapt a pre-trained model to a new task or domain) has been shown to enhance model accuracy [4]. This approach allows the model to adapt to the unique characteristics of the data, ultimately improving its ability to make accurate predictions within the target domain.

In this research, we explore two Privacy-Enhancing Technologies (PETs): Homomorphic Encryption (HE) and Secure Multiparty Computation (MPC). The goal is to investigate how these approaches can be utilized during the training phase and to compare their performance and impact on model accuracy. HE and MPC were chosen over other PETs, such as Differential Privacy [5] and Federal Learning [6], because they are suitable for our problem setting, as many of other PETs are infeasible for the situation when a data processor (machine learning engineer in the context) cannot access the data.

1.1 Problem Statement

Recent research has made significant progress in PPML. However, most advancements focus on the inference stage of the machine learning pipeline, while the training stage re-

mains underexplored. Training models requires significantly more computational resources, and incorporating PETs makes it even more challenging [7]. Some studies [7, 8, 9, 10, 11] have demonstrated that applying PPML during training is possible, but they primarily use support vector machines, logistic regression models, or simple, shallow neural networks. These models are insufficient for many industry applications, where more complex architectures like transformers [1] are commonly used.

Instead of focusing on PPML for training an entirely new model from scratch, we aim to investigate its application in transfer learning [12] for pre-trained models (PTMs). Typically, untrained models are trained on large public datasets, a process that is both time-consuming and computationally expensive, especially when incorporating PTMs on a specific dataset is a widely adopted approach in the industry and aligns well with our research goals.

In this work, we will explore and implement existing research findings while investigating the feasibility of fine-tuning PTMs using PPML in real-world industry scenarios.

1.2 Research Questions

How can we efficiently integrate PETs, specifically HE and MPC, into the machine learning training process so that customer data remains private while still letting the model benefit from it?

Homomorphic Encryption How does HE impact the model's accuracy? What is the associated overhead in terms of additional training time and communication costs?

Multiparty Computation How does Multi-Party Computation (MPC) affect the model's accuracy? What is the overhead involved, including additional training time and communication costs?

Comparison How can HE and MPC be compared in terms of training time overhead, additional operational costs, implementation complexity, and communication costs?

2 Background and Related Work

2.1 Homomorphic Encryption

2.1.1 Terminology and Background

HE is a cryptographic technique that allows computations to be performed directly on encrypted data without decrypting it [13].

LWE problem The Learning With Errors (LWE) problem was first introduced by Regev in 2009 [14]. The core challenge is to distinguish whether a sample originates from a noisy linear system or a uniform random distribution. LWE provides quantum resistance by adding a small random error to linear relations during encryption. Ring Learning With

Errors (RLWE) is a variant of LWE that extends operations from vectors to polynomial rings. The LWE problem is the foundational assumption of HE [13].

Encryption and decryption During encryption, the system combines the message with a secret key and a random matrix sampled from a uniform distribution, along with a small random error. The ciphertext is constructed as a linear function of the secret key and the message. Decryption involves inverting this linear relation. Although errors are added to enhance security, they are kept small enough to allow accurate decryption of the original message [13].

Relinearization In HE, the sum of two ciphertexts results in another ciphertext that remains linear and can be decrypted directly. However, multiplication of two ciphertexts produces a degree-2 polynomial, increasing the ciphertext size from $n+1$ to approximately $n^2/2$, which complicates decryption. Relinearization is a technique used to reduce the ciphertext size back to $n+1$ [15].

Noise Noise (also called error) is introduced initially for security purposes. However, noise accumulates through addition and multiplication operations. When the noise level exceeds a certain threshold, it can cause decryption errors [16].

Bootstrapping Bootstrapping, proposed by Gentry in 2009 [16], is used to refresh ciphertexts and reduce the noise accumulated through homomorphic operations. The key idea is to homomorphically evaluate the decryption function on a noisy ciphertext using an encrypted secret key:

$$c' = Eval(Decrypt(c,s)) \tag{1}$$

Here, $Eval$ denotes the homomorphic evaluation of the decryption function $Decrypt$ on the ciphertext c, using the encrypted secret key s. The output of $Decrypt(c,s)$ is a plaintext, so the result of this homomorphic operation is a new ciphertext c' with lower noise.

2.1.2 Types of Homomorphic Encryption

It is classified based on its ability to support different types and numbers of operations:

1. Partially Homomorphic Encryption (PHE) supports only one type of operation, either addition or multiplication. The first well-known PHE scheme is RSA [17], which enables multiplication on encrypted data. However, this is impractical for our research since PETs applied to neural networks require both addition and multiplication, as all computations can be reduced to these two operations [18].

2. Somewhat Homomorphic Encryption (SHE) allows both addition and multiplication on encrypted data, but with a limited number of operations. This limitation arises

from the increasing noise in the ciphertext after each operation. Once the noise exceeds a certain threshold, the ciphertext can no longer be decrypted [19]. SHE is commonly used in limited-depth PPML and basic statistical computations on encrypted data [20].

3. Fully Homomorphic Encryption (FHE) supports both addition and multiplication on encrypted data without any restrictions on the number of operations. The first practical FHE scheme was introduced in 2009 by Gentry [16], enabling encrypted data to be processed without decryption. A variant of Gentry's scheme was later implemented in paper [21], building upon prior work, implemented by Smart and Vercauteren [22], which successfully implemented the underlying SHE scheme but failed to implement bootstrapping, a key process necessary for achieving full homomorphism. FHE requires significantly more computational resources than PHE and SHE due to the overhead of noise management. Bootstrapping is a noise management technique that refreshes intermediate ciphertexts [19].

4. Leveled Homomorphic Encryption was introduced in 2011 by Brakerski, Gentry, and Vaikuntanathan [23]. They proposed a new approach to FHE without bootstrapping, which had previously caused significant performance issues in FHE. In Leveled HE, the traditional bootstrapping process is not required; instead, it features a more efficient method for managing noise during homomorphic operations. Leveled HE requires a pre-defined depth L, and the homomorphic circuit can evaluate any circuit with depth at most L. Leveled HE is built upon SHE. The key difference between Leveled HE and SHE is that, unlike SHE, which can only evaluate low-degree polynomials. Leveled HE can evaluate an arbitrary degree of polynomials, as long as the total depth does not exceed L.

Since PHE supports only a single operation, it is insufficient for general computation and will not be considered in our research. While SHE supports both addition and multiplication, its limited number of operations restricts its applicability to complex computations. On the other hand, FHE, despite being computationally intensive, enables complex encrypted computations. Leveled HE also has a computational limitation defined by a pre-set depth parameter L, similar to SHE. However, Leveled HE advances further by providing a mechanism to manage noise up to the defined depth. In contrast to the arbitrary noise threshold in SHE, leveled HE makes this threshold a configurable parameter. In SHE, the noise threshold is not explicitly defined but is instead implicitly determined by other core encryption parameters, such as the plaintext and ciphertext modulus. Selecting appropriate parameter combinations is, therefore, central to the design of Leveled HE [23]. In our research, we integrate HE into our problem setting using the open-source library TenSEAL [24], which is based on SEAL [25], developed by Microsoft. SEAL employs a leveled HE mechanism that does not require bootstrapping and uses a pre-defined parameter to control the maximum multiplication level, where a higher level typically results in lower performance.

2.1.3 Homomorphic Encryption Schemes

FHE schemes can be categorized based on their methods for encrypting inputs and performing operations [26].

FHEW and TFHE Fastest Homomorphic Encryption in the West (FHEW), proposed by Ducas and Micciancio [27], addresses a fundamental challenge in FHE: the inefficiency of the bootstrapping process. Earlier implementations, such as HElib [28], exhibited substantial latency during bootstrapping. FHEW introduces a novel bootstrapping technique that operates at the bit level, specifically evaluating the logical NAND of two encrypted bits. This approach reduces the amount of noise introduced during gate evaluation, thereby simplifying the subsequent ciphertext refreshing steps. An optimized bootstrapping procedure is then employed to refresh the ciphertext [27]. Fast Fully Homomorphic Encryption over the Torus (TFHE), introduced by Chillotti et al. [29], builds upon FHEW to further accelerate the bootstrapping process. Like FHEW, TFHE performs encryption at the bit level. It improves performance by replacing the computationally intensive internal product of TGSW (i.e., Gentry-Sahai-Waters, a combined analogue from [30, 31]) used in FHEW's accumulator with a more efficient external product between a TGSW ciphertext and an LWE ciphertext. The authors explain this by comparing the speed of matrix-vector multiplication with matrix-matrix multiplication. Since a TGSW sample is a matrix, where each row is an LWE sample, the external product (between TGSW and LWE) is simpler than the internal product (TGSW multiplied by TGSW) [29]. Since in FHEW the message space is in $\mathbb{Z}_t$, where t defines the message space in encryption schemes (e.g., $t = 2$ means the scheme supports binary messages 0 or 1), FHEW is designed for homomorphic computation over integers [27]. On the other hand, TFHE operations use values in the real torus, which represents a set of real numbers. In practice, TFHE rescales these elements to 32-bit integers, which appears to be more stable and efficient than using floating-point numbers [29].

BGV and BFV Brakerski-Gentry-Vaikuntanathan (BGV) [23] and levelled Brakerski/Fan-Vercauteren(BFV) [32] encrypt data at the word level, meaning they process data in larger chunks rather than individual bits. Although these schemes typically exhibit lower computational efficiency than a bit-wise encryption scheme, they allow multiple data elements to be combined into a single ciphertext. This enables computations over bundled data through a Single Instruction Multiple Data (SIMD) approach [33], potentially improving per-slot execution times compared to bit-wise methods. BGV and BFV support only integer arithmetic because they rely on polynomial rings with integer coefficients modulo an integer polynomial. Consequently, they are less suitable for our study, which requires fractional and continuous computations such as gradient descent, weights, and biases. The process of converting features and parameters from high-precision floating-point representations to a discrete set, such as integers, is quantization. This technique is primarily used to reduce computational overhead in machine learning models [34].

However, as this study does not focus on quantization, all data within the deep learning pipeline will be maintained in floating-point format.

CKKS The Cheon-Kim-Kim-Song (CKKS) is specifically designed to handle computations involving approximate, real-valued numbers. Like BGV and BFV, CKKS supports SIMD computations but provides approximate results that are valid up to a certain precision, meaning that computations lose accuracy incrementally [35]. Due to its native support for real-valued plaintexts, CKKS is widely applied in PPML scenarios [36].

The effectiveness and practicality of CKKS largely depend on selecting appropriate parameters. A key parameter is the polynomial modulus degree, denoted by N, which typically is a power of two (e.g., 1024, 2048, 4096, 8192, 16384, or 32768). This polynomial defines a ring:

$$R = \mathbb{Z}_q[X]/(X^N + 1) \tag{2}$$

Where q is the coefficient modulus.

All plaintexts and ciphertexts are represented as polynomials within this ring, where computations also occur. A larger N enhances security by increasing the difficulty of solving the Ring Learning With Errors (RLWE) problem [37]. However, increasing N also raises computational complexity by increasing the number of coefficients in plaintexts and ciphertexts [35]. Homomorphic operations, particularly multiplication, cause rapid noise growth; excessive noise can lead to decryption errors. To manage the growth of noise, Cheon et al. [35] introduced a rescaling technique involving modulus reduction after multiplication operations.

In the leveled implementation of CKKS, the modulus q is typically a product of several small prime moduli, forming a moduli chain:

$$q = q_0 \cdot q_1 \cdot \ldots \cdot q_L \tag{3}$$

Where each q_k is one of a small prime (e.g., $2^{30} + 7$), and L is the supported level of multiplication.

Rescaling will reduce this composite modulus by dividing out one of these primes. HE inherently operates over integers, so real numbers must first be scaled by a global scale factor, denoted as Δ, before encoding into plaintext polynomials. A larger Δ allows higher precision. When two scaled encrypted numbers are multiplied, the resulting scale becomes Δ^2; thus, the rescaling operation is essential to reduce the scale back to Δ. The rescaling operation involves dividing ciphertexts by one of the modulus primes q_k. Choosing q_k close to Δ ensures that after rescaling, the scale approximately returns to its original value.

The specific parameter selections used in our experiments will be detailed in Section 3.3.

2.2 Secure Multiparty Computation

Secure Multiparty Computation (MPC) protocols allow multiple participants to jointly compute a function such that only the final result is revealed, and each participant's inputs stay private. No party can learn anything about the inputs of other parties, except what can be inferred by the output [38]. Yao originally introduced the idea for secure two-party computation [39], and later, Goldreich et al. [40] proposed a version for multiple parties with an honest majority, wherein the number of potentially corrupted parties is less than half of the total.

2.2.1 Security Properties

Based on Lindell's work [38], several key security properties are used to determine whether a protocol can be considered secure. Among them, the following five are the most fundamental:

Privacy. This property ensures that no party learns anything about the inputs of other parties, beyond what can be inferred from the output.

Correctness. This guarantees that all parties receive the correct output of the computation.

Independence of Inputs. The inputs of corrupted parties must be independent of those of honest parties. A corrupted party, under the control of an adversary, acts according to the adversary's instructions. Importantly, privacy alone does not guarantee input independence. For example, in a private auction, each party's bid remains confidential, and the goal is to determine the highest bid without revealing the losing bids. It is possible that a corrupted party sees a ciphertext representing $100 and manipulates it to generate a valid ciphertext that represents a value greater than $100. Although this does not violate privacy, it does compromise the independence of inputs [38].

Guaranteed Output Delivery. All parties must receive the output, without being prevented from doing so, for instance, through denial-of-service attacks.

Fairness. Corrupted parties should only receive the output if honest parties do as well. It should never happen that only corrupted parties learn the result while honest parties do not.

In practice, not all of these properties are always required. Some protocols exclude guaranteed output delivery and fairness, a relaxation referred to as security with abort. This approach is typically justified by two main reasons: first, achieving fairness may be impossible in certain settings; second, more efficient protocols can be designed by omitting fairness. In such cases, the security requirements are relaxed to reflect the specific needs of the application [38].

2.2.2 Adversarial Models

There are three main types of adversaries in secure computation: semi-honest, malicious, and covert adversaries.

In the semi-honest (or "passive" or "honest-but-curious") adversarial model, all parties follow the protocol correctly, but an adversary may attempt to infer private information from corrupted parties. In contrast, the malicious (or "active") adversarial model assumes that corrupted parties may deviate arbitrarily from the protocol to break security guarantees. For example, a malicious adversary may send incorrect values to disrupt communication or alter the final output [38].

Covert adversaries lie between the semi-honest and malicious models and better reflect real-world settings. In this model, adversaries are willing to cheat but prefer to avoid detection. Unlike malicious adversaries, who act without concern for exposure, covert adversaries weigh the risk of being caught [41].

2.2.3 Corruption Threshold

Lindell [38] also discusses the feasibility of MPC in relation to the number of total parties and the number of corrupted parties. Let n denote the total number of parties and t the number of corrupted parties:

- For $t < n/2$ (i.e., an honest majority), it is possible to construct secure multiparty protocols that satisfy all five properties.
- For $t \geq n/2$ (dishonest majority), secure multiparty protocol is still feasible, but only without fairness and guaranteed output delivery.

When more than half of the parties are honest, secure computation can be achieved for any function with fairness and guaranteed output delivery, provided that communication occurs through a secure channel [40]. However, if the number of corrupted parties exceeds half, MPC protocols can still be executed with privacy, correctness, and independence of input. This means that inputs remain confidential, the computed result is correct, and the inputs of corrupted parties do not depend on the inputs of honest parties. However, fairness and guaranteed output delivery are no longer ensured, allowing adversarial parties to withhold or manipulate when and how outputs are revealed [40, 39].

2.2.4 Techniques

Several cryptographic techniques support MPC. One foundational approach is Secret Sharing, which is particularly effective in honest-majority settings, where the number of honest parties exceeds that of dishonest ones. The dishonest parties may be semi-honest, covert, or malicious adversaries. Secret Sharing enables a dealer to distribute a secret among multiple parties, ensuring that only a sufficient subset of parties can reconstruct it, while smaller subsets learn nothing about the secret [18].

A well-known Secret Sharing implementation is Shamir's scheme [42], which splits a secret S into n shares using polynomial interpolation over a finite field. Any subset of k shares can reconstruct the secret, while fewer than k shares reveal no information. The scheme generates a random polynomial:

$$f(x) = S + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1} \bmod p \tag{4}$$

where S is the secret, a_i are random coefficients, p is a large prime to keep values in a finite field, k is the threshold that can reconstruct the secret, and x represents unique numbers assigned to each share. Let n denote the total number of shares generated. Shares are created by evaluating $f(i)$ for $i = 1, ..., n$, and the secret can be reconstructed using Lagrange interpolation [42].

Another widely used technique is Yao's Garbled Circuits, designed for semi-honest adversaries. It enables two parties to securely compute a function over their private inputs without revealing them to each other. This is done by transforming the function into a garbled circuit, ensuring that intermediate values remain hidden during computation [39].

In our research, we assume the server hosting the model is semi-honest, meaning it follows the protocol, but may attempt to infer information from the data.

2.3 Differential Privacy

Differential Privacy (DP) ensures that adding or removing a single record from a database does not significantly alter query results, preventing adversaries from inferring whether specific data points are included. DP achieves this by introducing noise to query responses for numerical values. For categorical data (e.g., strings or hierarchical data), noise addition is impractical, so DP employs an exponential mechanism, selecting an output y with probability proportional to:

$$exp(-\varepsilon u(X, y)/2) \tag{5}$$

where $u(X, y)$ is a utility function measuring the quality of an output y given a database X[5].

In our setting, developers should not access the plain customer's data, while DP assumes that the data processor has access to the data but adds noise to protect its privacy from other parties to retrieve information for a specific user; thus, it is not applicable.

2.4 Federated Learning

Federated Learning (FL) enables multiple devices to collaboratively train a model without centralizing data [6]. However, in an enterprise production setting, this approach is impractical. If customer data is stored on separate instances and training occurs on those instances, engineers performing the training would still have access to the data, which conflicts with our privacy requirements. Furthermore, expecting customers to train models themselves is unrealistic in a Software-as-a-Service (SaaS) environment. Thus, FL does not align with our goals.

2.5 Deep Learning

This section introduces foundational concepts and methodologies related to deep learning, specifically those relevant to our work.

2.5.1 Loss Function

During the training of a deep learning model, a loss value quantifies the discrepancy between true labels and predicted labels (i.e., the output of the forward process). Loss functions play a crucial role in deep learning pipelines. They guide backpropagation by providing gradients for updating model weights, and can help balance bias and prevent overfitting when appropriately designed.

Loss functions are categorized based on the type of task they are suitable for. Common loss functions for regression tasks include Mean Squared Error (MSE) and Mean Absolute Error (MAE). For classification tasks, Binary Cross-Entropy (BCE) and Categorical Cross-Entropy (CE) are widely used, which correspond to binary classification and multi-class classification, respectively [43].

In this study, the task is image-based multi-class classification. While cross-entropy is the standard choice, we also use MSE for comparison and due to its simplicity in terms of gradient linearity.

Mean Squared Error (MSE) Mean Squared Error (MSE) is a common loss function for regression tasks. It computes the average squared difference between the predicted and true labels. This makes it sensitive to outliers due to the squaring, which can lead to reduced generalization performance. For tasks such as tabular data prediction and time series forecasting, MSE effectively measures the deviation between predictions and true labels and directly reflects model performance [43].

$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{6}$$

Where y denotes the true labels, $\hat{y}$ represents the predicted labels, and N is the number of samples.

Categorical Cross Entropy Categorical Cross-Entropy (CE) is commonly used for multi-class classification tasks. The model output is a probability distribution over classes, assigning a higher probability to the correct class. CE measures the divergence between the predicted probability distribution and the true labels.

$$L(y,\hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{7}$$

Where y represents the true labels and $\hat{y}$ is the predicted probability distribution, typically produced by a softmax function. N is the number of samples.

2.5.2 Optimization Algorithms

After computing the loss, the model updates its parameters (e.g., weights and biases) to minimize this loss. This is done using the gradient of the loss for the parameters. Since neural networks can have millions of parameters, it is not practical to search the parameter space exhaustively. Moreover, there is no guarantee that the minimum found is a global minimum [44]. Optimization algorithms aim to make gradient descent faster, more stable, and more efficient.

Stochastic Gradient Descent Stochastic Gradient Descent (SGD) reduces the loss by iteratively updating model parameters in the opposite direction of the gradient. Unlike traditional gradient descent, which uses the entire dataset, SGD randomly selects a subset of training data known as a minibatch. This randomness introduces noise into the gradient estimation, which can help the optimizer escape local minima and improve generalization. SGD is often combined with additional techniques, such as momentum. In SGD with momentum, the update is a weighted combination of the current gradient and the previous update step [44].

Adaptive Moment Estimation Adaptive Moment Estimation (Adam) is another widely used optimization algorithm. It integrates ideas from both RMSprop [45] and momentum. Adam assigns individual learning rates to each parameter by computing adaptive estimates of first and second moments of the gradients. The first moment is the exponentially decaying average of past gradients, and the second moment is the exponentially decaying average of past squared gradients.

Bias correction is applied to both moments, as they are initialized as zero vectors, which introduces bias in early steps. Parameters are then updated using these bias-corrected estimates [46].

2.5.3 Transfer Learning

Transfer learning is a machine learning approach that leverages knowledge from a task with abundant data to improve performance on a target task with limited data. A model can be trained on a large, high-quality dataset and then adapted to the target task, which may involve fixing the existing layers, adding new layers, or fine-tuning the entire model [44].

One common strategy is to use a pre-trained model as a feature extractor. Although not trained on the target dataset, it can still extract useful features. EPIC [47], for example, applies transfer learning by first extracting features from plaintext data using a CNN-based model. This reduces computational overhead in the privacy-preserving stage. Then, a separate SVM-based model is used for private classification.

Our study adopts a similar approach, using transfer learning to reduce the computational cost of private training.

2.6 Reconstruction Attacks

Transformer-based models have demonstrated their effectiveness in latent feature representation for both text and vision tasks [48]. This section discusses research related to reconstructing output features and reveals how sensitive information from the original data can be exposed. These findings suggest that features do not inherently preserve privacy, highlighting the importance of using PETs to protect input data, even if it is not in its original form.

In the image domain, Mahendran et al. [49] proposed a method in 2015 to reconstruct images based on features. Their method is applicable to deep Convolutional Neural Networks (CNNs), such as AlexNet, and achieves reconstruction errors that rarely exceed 20%. The process begins with random noise and iteratively searches for images that best match the target using gradient descent and a natural image prior.

Later, in 2016, Dosovitskiy et al. [50] introduced a different approach by training separate CNNs to perform the reconstruction task. Their method achieved lower reconstruction errors compared to the previous work [49]. Although the method proposed by Mahendran et al. produces clearer images, the color and the exact placement of details in the reconstructed images do not perfectly align with the original ones, resulting in higher reconstruction errors.

While these two studies focus on CNNs, Shahreza et al. [51] investigated reconstruction attacks on transformers. They proposed a method to reconstruct facial images based on a pre-trained face foundation model. Their system can attack any black-box face recognition system by using leaked embeddings to reconstruct the corresponding face images.

2.7 Privacy-Preserving Machine Learning

The research on PPML is increasing, and there are existing applications for different model settings. Xu et al. introduce PPML in general in [52]; they also introduce a machine learning pipeline and different PPML approaches for different stages, including data preparation, model training and evaluation, model development, and model inference. To align with our research goal, we will primarily review the PPML studies for the model training and the inference stage.

2.7.1 Privacy-Preserving Machine Learning for inference

Inference is the stage in which a fully trained model processes user inputs and generates outputs, such as classification labels, regression values, or text, without performing loss calculations or parameter updates. In this sense, it is simpler than the training phase, as it primarily involves a forward pass through the network, without loss computation, regularization, and weight update [52].

Numerous protocols and frameworks for model inference have been introduced. CryptoNets [53] in 2016, allows data owners to send their encrypted data to a cloud service, the cloud service is able to apply the learned neural network to the encrypted

data without decryption, and generate encrypted predictions, send back to data owners who have private keys that can decrypt the result. CryptoNets uses SHE to achieve privacy-preserving computation, and it achieves high accuracy (99%) on MNIST optical character recognition tasks. SEALion [54] is built upon CryptoNets; it provides a modular and extensible software architecture. Chou et al. [55] introduced a faster CryptoNets framework, which uses network pruning and quantization to reduce the computational complexity.

MiniONN [56] is a framework that transforms a trained model into an oblivious form, enabling it to generate outputs for user inputs while preserving privacy. It leverages secure two-party computation (2PC) to implement oblivious protocols for operations such as linear transformations, activation functions, and pooling. Unlike previous approaches, MiniONN does not require any modifications to the model during the training phase.

2.7.2 Privacy-Preserving Machine Learning for training

While PPML at the training stage is theoretically feasible, it remains computationally intensive in practice. Under HE, addition and multiplication operations are supported. Training deep learning models typically involves non-polynomial activation functions (e.g., ReLU, Softmax) and loss functions (e.g., cross-entropy). To accommodate HE constraints, these functions are often approximated using polynomial expressions, which may lead to reduced accuracy and increased computational overhead. Backpropagation, although consisting primarily of sums and products, becomes significantly slower when applied to encrypted data. In addition, the limited computation depth supported by SHE often proves infeasible for deep models, while FHE requires even more computational resources. Another issue is that encrypted training data prevents data scientists from inspecting it directly, making it difficult to correct errors or gain insights into the data [53]. Another PET, MPC, is shown to be more time-efficient; however, it typically requires multiple rounds of communication among parties, leading to substantial overhead [38].

One of the earliest notable works to train a neural network on encrypted data using HE is by Nandakumar et al. [7], published in 2019. Their approach leverages the open-source HElib toolkit to implement an SGD-based method under FHE, training a three-layer fully connected neural network on the encrypted MNIST dataset for handwritten digit recognition. The study shows that while HE reduces accuracy compared to plaintext training, the impact is minimal. For instance, in the case of the NN2 model (a neural network configuration in the paper with 32 and 16 neurons in the two hidden layers), accuracy drops slightly from 96.4% to 96%. Despite this reduction, the trade-off is negligible compared to the security benefits of training on encrypted data. This effort was the first to successfully demonstrate neural network training on encrypted data.

An earlier study by Mohassel and Zhang [9] uses secure two-party computation to achieve PPML for linear regression, logistic regression, and neural network training. Other works have primarily focused on simpler models, such as the HE-based privacy-preserving training of Support Vector Machines (SVM) [8] and logistic regression [10, 11]

Although PETs, such as FHE, improve data privacy, they also impose computational overhead, which can restrict the complexity of machine learning models. In this context, model complexity refers to factors such as the number of parameters, depth, and architecture of the neural network, all of which influence both accuracy and training efficiency. Since PETs introduce additional computational costs, especially during training, more complex models become impractical and increased resource demands.

To mitigate these challenges, some research focuses on reducing training complexity through approaches like transfer learning, which leverages knowledge from well-labeled domains to support under-resourced ones [12]. In the context of PPML, transfer learning reduces computational requirements [4], thereby alleviating the overhead of encrypting data. PrivGD, introduced by Jin et al. [57], is the first FHE-based solution for private fine-tuning in a transfer learning setting. In this scheme, data owners employ a shared feature extractor to obtain features, encrypt them using FHE, and then send them to a pre-trained model for further fine-tuning.

HETAL [58], proposed by Lee et al. in 2024, extends the same use case as PrivGD but provides a more practical approach by supporting higher input feature dimensions and improving the efficiency of encrypted matrix multiplication. This reduces the computational bottlenecks commonly associated with privacy-preserving training. Our study is primarily based on the idea of using a transformer as a feature extractor and training a neural network layer privately on encrypted features. However, the core components of HETAL, such as the classifier and the training process, are not fully open-sourced, which makes it difficult to customize, for example, by modifying the loss function for specific applications. In contrast, our implementation and experiments are fully open-sourced, and we additionally provide an implementation using MPC, which is not available in HETAL.

This study focuses on enabling encrypted training of deep learning models, addressing a common enterprise need: integrating AI to enhance product functionality while maintaining data confidentiality. To achieve improved accuracy, it is often necessary to train models on proprietary datasets. However, simple neural networks such as fully connected networks are typically inadequate for complex business applications, particularly those involving image data, such as image classification. To meet these demands, we utilize the representational power of transformer models for feature extraction through plaintext inference. The extracted features are then used as input to a lightweight neural network for the final classification task. This approach allows us to combine the strengths of transformers with the efficiency and compatibility of encrypted training.

3 Implementation

In this section, we present the implementation of our PPML pipeline. Specifically, we integrate HE and MPC into the neural network training process. The model architecture employs a single-layer perceptron, which is trained using the output from CLIP [59], a multimodal transformer developed by OpenAI. This perceptron functions as a classification head on top of the transformer. Notably, only the training phase of the single-layer

perceptron is conducted under encryption. The prerequisite for this implementation is that the company (or the model provider) is permitted to use plaintext input data during the inference stage. For example, we assume that the plaintext Transformer inference takes place in a production environment with explicit user consent, similar to standard model serving, where the customer's input is used solely to obtain the model's output. Once the model generates the output (i.e., extracted features), that output is treated as private input. This design allows us to leverage the representational power of the transformer while avoiding the impractical computational overhead associated with fully encrypted transformer training. Upon completion of encrypted training, machine learning engineers can extract the weights and biases of the classification head for use during the inference stage (i.e., model serving).

There are various PPML frameworks designed for both HE and MPC. SEAL [25] is a widely used HE library written in C++. An extension of SEAL, TenSEAL [24], enables operations on tensors, making it more suitable for deep learning applications. Another HE-based PPML library is Concrete-ML [60], developed by Zama. This framework provides out-of-the-box capabilities for performing HE operations. However, while some are well-suited for machine learning inference, they may not be optimized for training. This thesis adopts TenSEAL for implementing HE, due to its native support for tensor computations and its flexibility for modifying and building a neural network training pipeline. For MPC, we explore MP-SPDZ [61] and Crypten [62]. MP-SPDZ supports various MPC protocols and offers flexibility in choosing the best one, but we opted for CrypTen because it naturally supports PyTorch tensors, making it easier to build the encrypted training process similar to regular PyTorch.

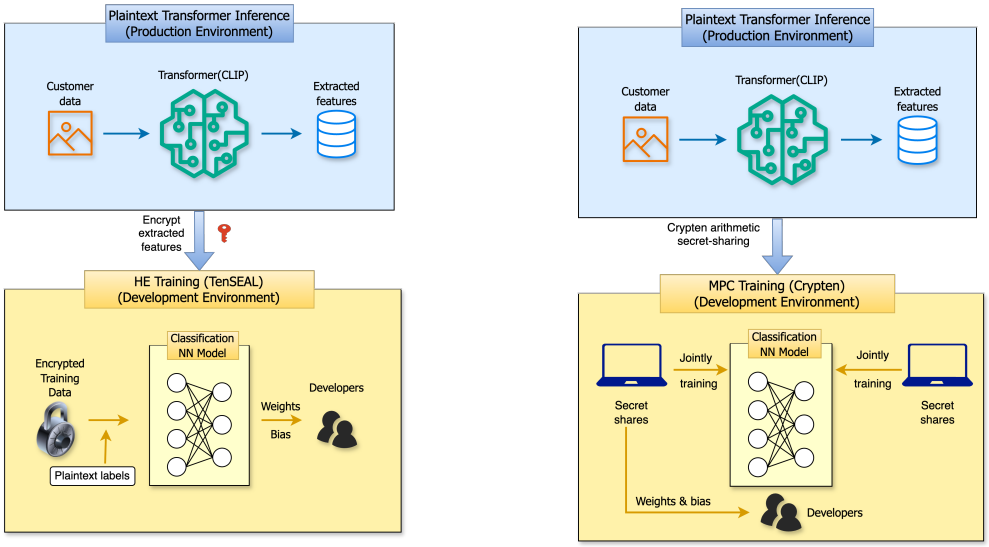
For private training, we use DeepFashion [63]. The key requirement for dataset selection is that it must be new to the pre-trained model; the model must not have been previously trained on it. This ensures that fine-tuning leads to adaptation rather than mere memorization of existing data. CLIP has not been trained on this dataset and has poor accuracy on it at the beginning of the experiment. Although fine-tuning the classification head on a domain-specific dataset improves accuracy, that's not the main goal of this study.

The implementation architecture is shown in Figure 2. The source code for the experiments is available at https://github.com/h0rs3fa11/dl_pets.

3.1 Deep Learning Model

The deep learning models used in this study include a transformer (CLIP) and a single-layer perceptron as the classification head. Specifically, the perceptron has 512 input features and predicts across 46 output classes without any hidden layers. This setup leverages the benefits of transfer learning.

For the experimental baselines, the plaintext model under the MPC setting was implemented using PyTorch, while the HE baseline was developed without the use of specialized frameworks. This design choice ensures that both baselines are closely aligned



(a) Implementation architecture of HE

(b) Implementation architecture of MPC

Figure 2: Our implementation architecture

with their respective encrypted counterparts. Training was conducted using both cross-entropy and MSE loss functions, with weight updates performed via the Adam optimizer. While cross-entropy is generally more suitable for multi-label classification tasks, MSE was included due to its linear gradient computation, which is more computationally efficient under encrypted settings.

This study defines multiple privacy levels. To maintain practicality, data remains unencrypted during loss computation and weight updates. Evaluating both MSE and CE for loss computation allows for an assessment of their effectiveness under current conditions and informs future work involving stricter privacy constraints, where encrypted training may be required throughout the entire pipeline.

For MPC experiments, the CrypTen library was employed to handle both loss calculation and model updates. Although CrypTen supports both cross-entropy and MSE losses, it currently restricts weight updates to SGD. Accordingly, the plaintext MPC baseline also utilizes SGD. While model accuracy is not the primary focus of this work, basic hyperparameter tuning was performed, such as adjusting learning rates and momentum, to ensure the MPC results remain valid and interpretable.

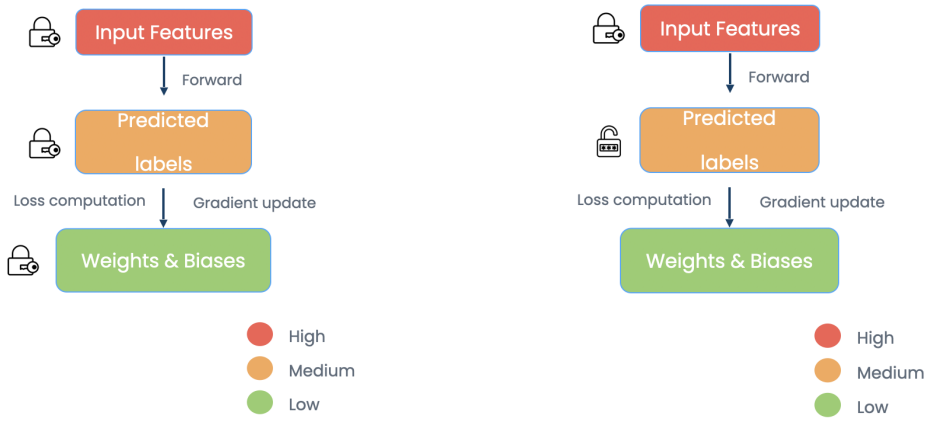
3.2 Privacy Level Definition

There are several types of data involved in this implementation: input data, true labels, weights and biases, and output data (i.e., predicted labels). In real-world scenarios, these data types naturally have different levels of privacy. The input data has the highest privacy requirement, and since protecting it is the main goal of this study, it should be encrypted during the training phase.

True labels and predicted labels are more flexible. While they can potentially reveal information about the input data, encrypting them comes with a performance cost due to the additional computation required. It becomes a tradeoff between stronger privacy and better training efficiency.

For instance, if all labels remain encrypted, then after the forward pass, the predicted labels will also be encrypted. These would then need to be used in the loss calculation and backpropagation, which would result in the weights and biases becoming encrypted as well. However, encrypting weights and biases is unnecessary in this case, as they will ultimately be decrypted for use when deploying the model. The fully private training process is shown in Figure 3a.

In our HE experiment, we chose to decrypt both the true labels and predicted labels during training to maintain a balance between privacy and performance. The semi-private training process is shown in Figure 3b. In a practical deployment, the setup typically involves two parties: one party (referred to as *prod*) holds and encrypts the data, while the other (referred to as *dev*) performs model training on the encrypted data. During training, *dev* transmits the predicted labels (i.e., the model outputs) to *prod*, which decrypts them and computes the loss, since it possesses both the decrypted predictions and the corresponding true labels. The computed loss and plaintext label are then sent back to *dev* to update the model. Although this approach introduces additional communication overhead, it preserves the privacy of both input data and true labels throughout training, while also reducing the computational burden associated with fully encrypted training. We refer to this setup as "semi-private" in the following sections. Fully encrypted training is more practical with MPC, but we also ran an experiment using encrypted forward passes with plain labels in MPC to better compare with the HE setup.



(a) Fully private training.

(b) Semi-private training.

Figure 3: Privacy Level Definition. A lock icon indicates that the data is encrypted, and an open lock indicates that the data will be decrypted and remain in plaintext during the following computations.

3.3 Homomorphic Encryption

TenSEAL supports operations using both CKKS vectors and tensors. In our implementation, we utilize `ckks_vector` as the primary operational unit. Although tensors typically offer higher computational efficiency, they require significantly more memory. Due to current environmental constraints, we cannot fully exploit the advantages of the CKKS tensor.

Encrypted computations in our setup are exclusively applied during the forward pass, specifically in the dot product operation between input data and model parameters (weights and biases). Once we obtain the predicted output, the encrypted result is decrypted, as detailed in the section 3.2. Subsequent processes, such as loss computation and parameter updates, occur entirely in plaintext.

Selecting suitable parameters for the CKKS scheme represents a critical challenge in our implementation. TenSEAL provides three main CKKS parameters: `poly_mod_degree`, `coeff_mod_bit_sizes`, and `global_scale`, which are discussed extensively in Section 2.1.3. The parameter `poly_mod_degree` denotes the degree of the polynomial modulus; higher degrees allow increased precision but cause computation times to rise exponentially. The parameter `coeff_mod_bit_sizes` specifies a list of integers, each indicating the bit-length of individual prime factors (q_k) within the modulus chain. The number of elements in `coeff_mod_bit_sizes` correlates directly to the multiplicative depth, determining the total number of permissible homomorphic multiplications before rescaling is necessary.

According to the security requirements outlined in the Homomorphic Encryption Standard [64], the sum of the coefficient modulus specified in `coeff_mod_bit_sizes` should not exceed a certain threshold determined by the value of `poly_mod_degree`. In our experiments, we set `poly_mod_degree` to 8192. To meet the 128-bit security level (i.e., requiring 2^{128} operations to break the system), the total of `coeff_mod_bit_sizes` should not exceed 218, we choose configuration `[50, 40, 40, 50]`, with a `global_scale` set to 40.

3.4 Secure Multiparty Computation

For our exploration of MPC, we initially investigated MP-SPDZ [61], a framework that supports multiple security models, including passive, active, and covert adversarial settings. However, we encountered difficulties during the experiment, particularly when executing compiled programs to read input data. Troubleshooting was challenging due to the system's reliance on a custom virtual machine, which compiles Python code into bytecode before execution, thereby obscuring error tracing and debugging.

We adopted CrypTen [62], developed by Facebook (now Meta), which supports only the passive adversarial model. It implements arithmetic and binary secret sharing [65, 66, 40] and provides a PyTorch-like interface, facilitating seamless integration with existing deep learning workflows. In this respect, CrypTen is conceptually similar to TenSEAL in how it handles encrypted training. Its `crypten.nn` module closely mirrors PyTorch's `torch.nn`, offering a variety of tools for defining and training neural networks, including components such as `crypten.nn.module.Conv2d` and `crypten.nn.loss.CrossEntropyLoss`.

Given our application context, where all computations occur within a single enterprise and all parties are assumed to behave honestly, there is no need to guard against malicious or covert behavior. The primary security requirement is to preserve data privacy, not to enforce protocol compliance. Therefore, CrypTen's support for passive security is sufficient for our purposes.

4 Experiment

This section first introduces the dataset used in the experiments in section 4.1, then presents the experimental settings and results for HE in section 4.2, followed by the experimental settings and results for MPC in section 4.3. Finally, the comparative study is discussed in section 4.4.

4.1 Datasets

We chose DeepFashion [63] as the training dataset for our encrypted model. DeepFashion offers labeled fashion images with a direct mapping between each image and its corresponding category, making it well-suited for our use case, a supervised classification task using a classification head. Additionally, zero-shot CLIP, meaning it was applied to the target dataset directly without any fine-tuning, performs poorly on DeepFashion; the accuracy on the test dataset is only 0.0530. This result highlights that, despite being trained on large-scale datasets, sophisticated transformers like CLIP require fine-tuning for domain-specific applications such as fashion or medical tasks. Our experiments demonstrate that even without retraining the transformer itself, simply leveraging transfer learning by adding and training a lightweight classification head significantly improves accuracy. This has also been demonstrated in previous work, such as EPIC [47], although it focused on less complex operations.

We used the Category and Attribute Prediction Benchmark from DeepFashion, which includes 289,222 clothing images across 50 categories and 1,000 attributes. Specifically, the dataset consists of 209,222 training samples, 40,000 test samples, and 40,000 validation samples. In our experiment, we randomly selected 10% of the dataset: 20,895 training samples, 3,982 validation samples, and 3,983 test samples, covering 46 classes. Attributes were not used in this experiment.

Extracted features. The extracted features from CLIP have 512 dimensions. The statistical properties of these training features are summarized in Table 1. Figure 4 presents the t-SNE visualization of CLIP extracted features from the training samples. t-SNE (t-distributed Stochastic Neighbor Embedding) is an unsupervised machine learning technique used for visualizing and interpreting high-dimensional data by reducing its dimensionality [67]. It is capable of separating data that a linear boundary cannot separate. The visualization reveals partial clustering patterns with significant overlap between classes, indicating that while CLIP embeddings capture meaningful semantic distinctions between categories, they do not provide perfect class separation.

	Count	Mean	Min	Max
Training	20,895	0.0005750656127929688	-0.80078125	0.387451171875
Validation	3,982	0.0006003379821777344	-0.78125	0.383544921875
Test	3,983	0.0005831718444824219	-0.7958984375	0.3740234375

Table 1: Statistics of extracted features

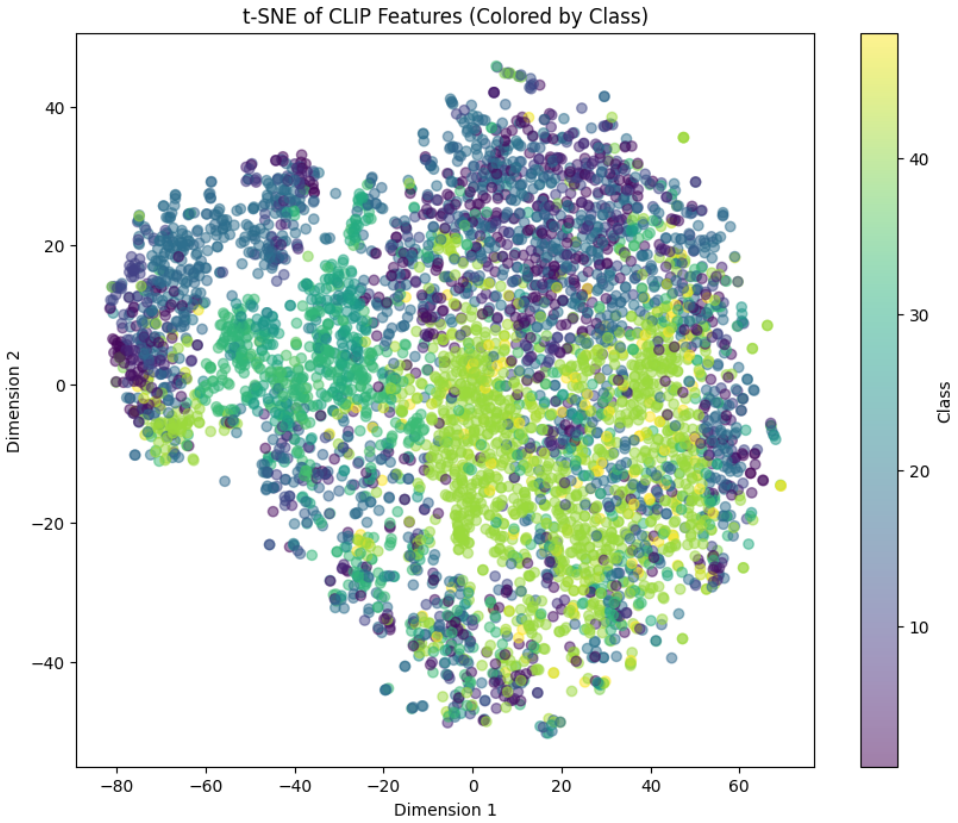


Figure 4: t-SNE of CLIP features

4.2 Homomorphic Encryption

In this section, we present the settings for the HE experiment in section 4.2.1. We compare the performance and accuracy of plaintext training against encrypted model training in sections 4.2.2 and 4.2.3. We also analyze the estimated communication costs of implementing HE in section 4.2.4. Since encrypting labels introduces additional encryption computations, as discussed in section 3.2, we use the semi-private setting in this experiment.

Additionally, we compared two loss computation methods in terms of performance and accuracy in sections 4.2.2 and 4.2.3: MSE and cross-entropy. While cross-entropy is commonly used for classification tasks, encrypting labels requires an approximated version of cross-entropy. In contrast, MSE has a linear gradient descent, eliminating the need for approximation. While this study does not include fully encrypted training with HE, the

MSE results still provide insights into the trade-offs between accuracy and performance, which could help guide future research in that direction.

4.2.1 Experimental Setup

System. Our experiment utilizes CPU processing as TenSEAL, and most homomorphic encryption libraries do not currently support GPU acceleration. All HE experiments were conducted on macOS with an Apple M1 processor, 16 GB of memory, and 8 CPU cores. The implementation operates in single-thread mode, meaning it utilizes only one CPU core.

Baselines. We implemented a plaintext model as the baseline to evaluate the impact of HE on accuracy and training time. The model does not use deep learning frameworks (e.g., PyTorch, TensorFlow) to ensure alignment with the encrypted model, as these frameworks introduce additional optimizations.

Training settings. The model training parameters are listed in Table 2. The training time per epoch does not include the duration of validation loss and accuracy calculations, as these computations are encrypted and would affect how training time is measured. Additionally, validation loss and accuracy do not influence model updates.

	Value
Epochs	20
Weight decay(Adam)	1e-5
Batch size	64
Learning rate	1e-3

Table 2: Training parameters for the classification head in the HE experiment on the DeepFashion dataset

4.2.2 Performance Evaluation

The comparison of training times is presented in Table 3. Although partial encryption is utilized in the setup, training with HE still requires significantly more time. The training durations for cross-entropy and MSE are similar, each exceeding 65 hours. This similarity arises because both methods involve identical encrypted computations, specifically during the forward pass, which includes dot product operations between input data and weights. Furthermore, while this duration may be acceptable in the current setup, the simplicity of the model structure and experimental configuration should be noted. In more complex scenarios (e.g., deeper neural network models with additional layers), the training time using HE will increase exponentially. Although the training duration is not ideal, such a requirement is typically less critical during training than inference, where rapid response is crucial.

	Avg Time/Epoch(s)	Total(s)	Total(h)
Plaintext (MSE)	1.37	27.32	0.0076
Plaintext (CE)	1.46	29.20	0.0081
Semi-private training (MSE)	12286.45	245728.91	68.26
Semi-private training (CE)	11947.35	238946.96	66.37

Table 3: Training time (plaintext vs. HE encrypted) for the classification head on 20,895 training samples. "Semi-private training" setups encrypt only the forward pass, while labels and model updates remain plaintext. Loss functions used: Cross-entropy (CE) and Mean Squared Error (MSE). The time spent does not include the validation process.

4.2.3 Accuracy Evaluation

The comparison of accuracies during the training is shown in Figure 5, and the final validation accuracy after 20 epochs of training is shown in Table 4.

	Accuracy
Semi-private training (CE)	0.6047
Semi-private training (MSE)	0.5853
Plaintext (CE)	0.6027
Plaintext (MSE)	0.5891

Table 4: Validation Accuracy of Homomorphic Encryption after 20 epochs of training. "Semi-private training" setups encrypt only the forward pass, while labels and model updates remain plaintext. Loss functions used: Cross-entropy (CE) and Mean Squared Error (MSE).

According to Figure 5, the models using MSE initially have higher accuracy but quickly plateau, exhibiting slower improvement throughout the training process. In contrast, the cross-entropy models start with lower accuracy but demonstrate rapid improvement and convergence, eventually surpassing MSE. This behavior arises from the fundamental differences between these loss functions: MSE treats outputs as numerical values and aims to minimize squared differences, whereas cross-entropy measures discrepancies between probability distributions, making it particularly suitable for multiclass classification tasks. Although MSE is less efficient and less ideal for multiclass scenarios, it still shows some incremental improvement. Thus, while cross-entropy clearly outperforms MSE in our multiclassification experiment, MSE could be advantageous in other contexts.

Moreover, comparing the plaintext model (baseline) and the HE model, the final accuracies after 20 epochs are very close, with cross-entropy reaching 0.6047 for the encrypted model and 0.6027 for the plaintext model. This near-equivalence highlights that HE models can effectively produce accurate results, provided the encryption parameters (e.g., CKKS parameters) are appropriately selected.

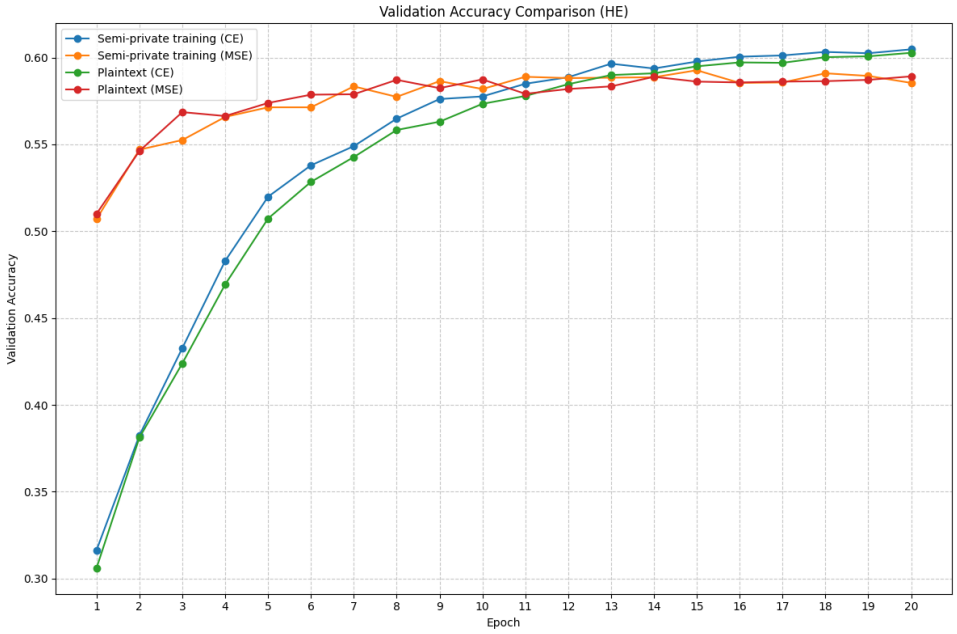


Figure 5: Accuracy of Homomorphic Encryption. "Enc. Forward, Plain Labels" setups encrypt only the forward pass, while labels and model updates remain plaintext. Loss functions used: Cross-entropy (CE) and Mean Squared Error (MSE).

4.2.4 Communication Cost

Since TenSEAL does not provide a direct method for calculating communication costs, we estimate it based on the HE computation process and the size of keys and input data.

We assume a setting with two parties: a client (denoted as *prod*), which holds and encrypts the data, which could represent a server in a production environment, and a developer (denoted as *dev*), who performs the training on encrypted data. In our setup, the output of the model (i.e., the predicted labels) is decrypted, which introduces additional communication steps. Specifically, we assume that *prod* is responsible for decrypting the predicted labels, computing the loss value, and sending them back to *dev*. The backpropagation is carried out on the *dev* side.

The communication process for each batch consists of the following steps. Figure 6 also shows the process.

- Step 1: *prod* sends the encrypted data along with the evaluation key to *dev*.The main purpose of the evaluation key is to enable the homomorphic multiplication of ciphertexts.
- Step 2: *dev* sends the encrypted predicted labels to *prod*
- Step 3: *prod* decrypts the predicted labels, computes the loss, and sends the loss value and the plaintext predicted labels back to *dev*

After the final step, the *dev* proceeds with updating the model parameters.

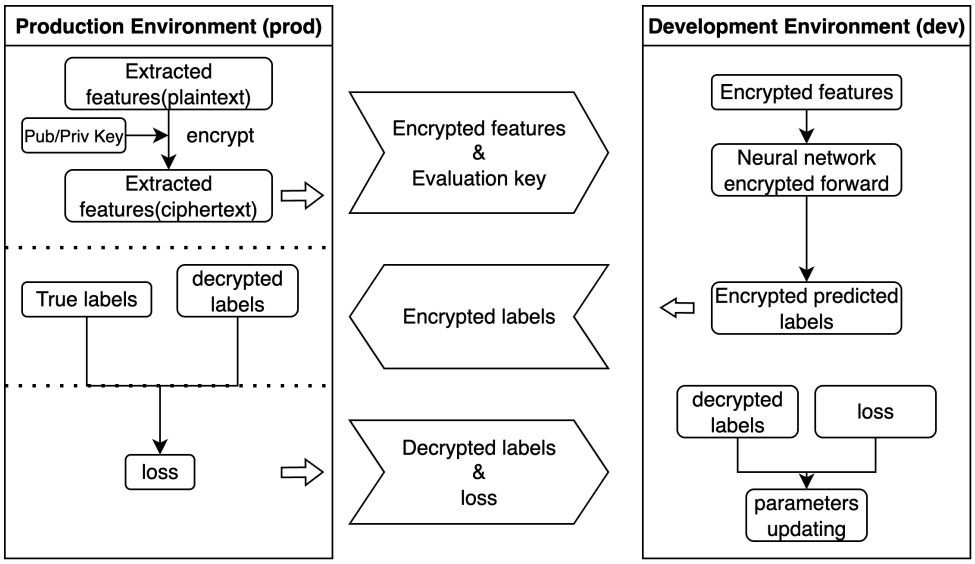


Figure 6: Communication process of the Homomorphic Encryption implementation. The diagram illustrates the process for a single batch.

The communication cost in step 1 is influenced by the size and dimensionality of the input data as well as the CKKS parameters. Step 2 is affected by the size of the output (predicted label) and the same CKKS parameters. In step 3, the cost is negligible since the transmitted loss value and labels are plaintext and small. The batch size also impacts communication cost per epoch: larger batch sizes reduce the number of communication rounds but increase the volume of data transmitted in each round.

First, we compute the size of the ciphertext. The total bit-length of the coefficient moduli is the sum of `[50, 40, 40, 50]`, which equals 180 bits. The bit size of a single polynomial's coefficient representation is the product of the polynomial modulus degree (8192) and the bit-length (180), resulting in 1,474,560 bits. Since a CKKS ciphertext consists of two polynomials [35], the total size becomes 2,949,120 bits.

In CKKS, a polynomial of degree N can encode $N/2$ approximate complex numbers. In our case, this means 4096 values. The input data for each batch is a 64 $\times$ 46 matrix, containing 2944 values, which can fit within a single ciphertext. However, since the ciphertext is computed in a vector-wise manner in our implementation, the entire input data cannot be packed into one ciphertext. Instead, to support vector-wise computation, the input data is split into individual vectors, resulting in 64 1×46 vectors per batch. Therefore, the final ciphertext size is 188,743,680 bits, or approximately 23.59 megabytes. The evaluation key, which is 33.2 megabytes in size, only needs to be transmitted once. This key size is obtained using TenSEAL's serialization method.

Step 2 uses the same CKKS parameters, but the transmitted data is only the encrypted predicted label, which can be fit into one single ciphertext, which is 2,949,120 bits, equal to 0.37 megabytes.

In Step 3, we assume the loss value is represented as a 32-bit float and the plaintext label as a 32-bit integer, resulting in a total of 64 bits, which is negligible. Thus, the total

transmission size is approximately 23.96 megabytes per batch. With 326 batches per epoch and 20 epochs in total, the overall transmission size amounts to 156,219.2 megabytes, plus a one-time transmission of evaluation keys, resulting in 156,252.4 megabytes. This estimate assumes no encoding methods (e.g., Protobuf, JSON) are used, and the overhead from transport protocols (e.g., HTTP, gRPC) is not considered.

Regarding network latency, we assume the same environment as used for MPC (see Section 4.3.1). Specifically, two AWS EC2 `t2.micro` instances are deployed within the same region and availability zone. We measure the latency between them using the `ping` command, obtaining an average round-trip time of 0.8483 milliseconds over 10 samples. Since `ping` measures round-trip time, and each communication round involves three steps, we approximate this as one and a half round-trips. Additionally, because the data is packed in a vector-wise manner into ciphertexts, there are 64 rounds per batch. Based on this, the total communication time per batch is estimated to be 81.43 milliseconds. Over 20 epochs, the total communication time amounts to 530.92 seconds.

4.3 Secure Multiparty Computation

This section presents the experimental setup in section 4.3.1, then compares the performance and accuracy of plaintext training against private training in sections 4.3.2 and 4.3.3. We also analyze the communication costs for MPC in section 4.3.4.

4.3.1 Experimental Setup

System. Our experiment utilized two Amazon EC2 t2.micro instances, each with 1 CPU and 1 GB of memory. The input data was split into two parts, and each instance only had access to its own part. Although CrypTen supports GPU acceleration, we conducted this experiment entirely using CPUs due to computational limitations and to maintain consistency with the HE experiment.

Baseline. The baseline in this experiment is designed to evaluate the impact of MPC on accuracy and training time. Unlike the baseline implementation used in the HE experiment, the MPC baseline employs PyTorch directly. This choice is because CrypTen is built upon PyTorch and provides PyTorch-like functions for privacy-preserving computations. Thus, using PyTorch for the baseline model better aligns with the MPC training conditions.

Training settings. The model training parameters are summarized in Table 5. CrypTen does not support the Adam optimizer used in the HE experiments; hence, we utilized SGD instead. Using the original parameters with SGD did not yield satisfactory results. Although achieving optimal accuracy is not the main purpose of this study, we aim to ensure the results are meaningful. Therefore, we performed simple fine-tuning to identify an optimal combination of learning rate and momentum. The momentum parameter assists SGD in stabilizing gradient directions and accelerating convergence. After fine-tuning, we selected a learning rate of `0.1` and a momentum of `0.99`. Similar to the HE experiments, we

compared accuracy and performance between private and plaintext training. Additionally, although fully private training (where data remains private throughout the entire training process) is feasible in MPC, we adopted semi-private training, matching the setup of the HE experiments, to facilitate a clearer comparison between HE and MPC performance. In the semi-private setting, the label data is held by one of the parties and is not represented as a secret-shared tensor using `crypten.cryptensor`.

Other training settings, such as the loss functions, remain consistent with those used in the HE experiments.

	Value
Epochs	20
Momentum (SGD)	0.99
Batch size	64
Learning rate	1e-1

Table 5: Training parameters for the classification head in the MPC experiment on the DeepFashion dataset

4.3.2 Performance Evaluation

The performance results are summarized in Table 6. As expected, MPC-based training is significantly slower than plaintext training. Among the loss functions, cross-entropy introduces more computational overhead than MSE, due to its reliance on more complex operations such as logarithms and exponentiation. The semi-private setup results in a shorter training time compared to the fully private configuration, indicating that reducing the number of privacy-preserving computations can improve efficiency. Although fully private training with cross-entropy is over 10,000 times slower than its plaintext counterpart, the total runtime (approximately three hours) remains manageable in practice. Training with MSE demonstrates better efficiency, offering a practical trade-off between computational overhead and model accuracy.

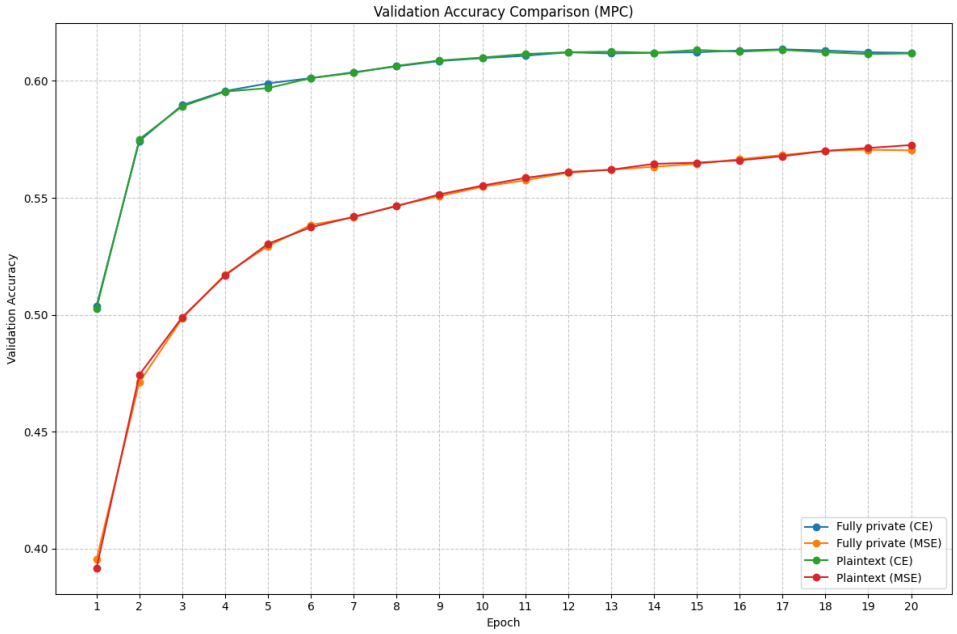


Figure 7: Accuracy of Secure Multiparty Computation.

	Avg Time/Epoch(s)	Total(s)	Total(h)
Plaintext (MSE)	0.04	0.90	0.0002
Plaintext (CE)	0.06	1.23	0.0003
Semi-private (MSE)	28.8	576.61	0.1602
Semi-private (CE)	209.93	4198.66	1.1663
Fully private MPC (MSE)	246.58	4931.55	1.3699
Fully private. MPC (CE)	616.24	12324.88	3.4236

Table 6: Training time (plaintext vs. MPC) for the classification head on 20,895 training samples. "Fully private" means both model and data (inputs, labels, loss, gradients) are treated as secret sharing. In "Semi-private" setups, only the features are secret sharing. Loss functions used: Cross-entropy (CE) and Mean Squared Error (MSE)

4.3.3 Accuracy Evaluation

Figure 7 compares the validation accuracy over 20 epochs between plaintext models and fully private models using MPC, and Table 7 shows the final accuracy after 20 epochs of training. The accuracy of MPC models is notably close to that of the plaintext model, demonstrating that encryption has minimal impact on predictive performance. Additionally, models trained with cross-entropy consistently outperform those using MSE across all epochs.

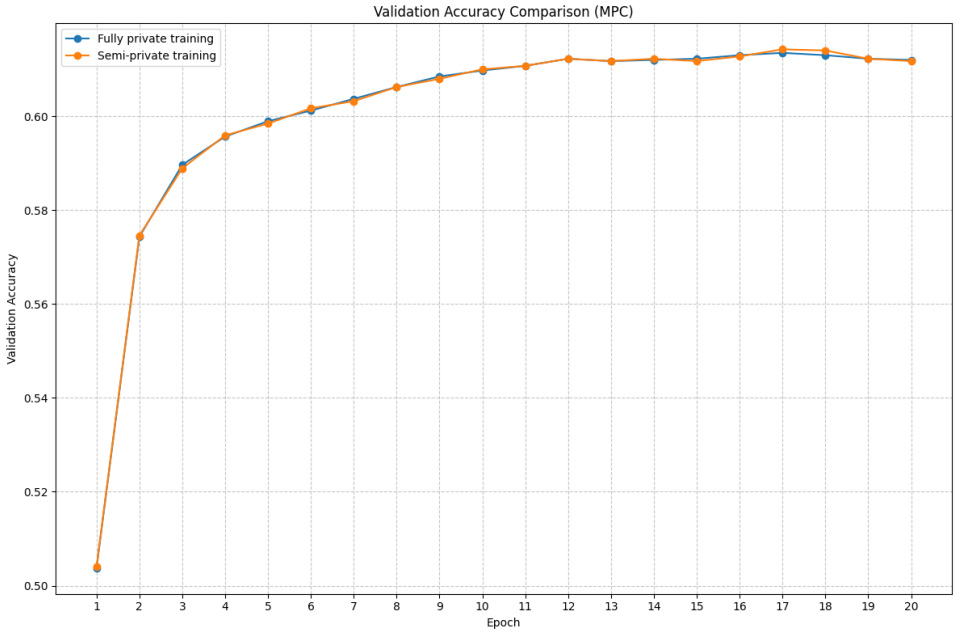


Figure 8: Accuracy Comparison: Fully private MPC vs. Semi-private MPC. Loss functions used: Cross-entropy

	Accuracy
Fully private training (CE)	0.6120
Fully private training (MSE)	0.5703
Semi-private training (CE)	0.6118
Semi-private training (MSE)	0.5726
Plaintext (CE)	0.6118
Plaintext (MSE)	0.5726

Table 7: Validation Accuracy of Multiparty computation after 20 epochs of training. Loss functions used: Cross-entropy (CE) and Mean Squared Error (MSE).

Figure 8 presents a comparison between the plaintext and fully private model. The two accuracy curves are nearly indistinguishable, indicating that fully encrypting all components has a negligible effect on model accuracy while incurring significantly higher computational overhead.

4.3.4 Communication Cost

We leverage Crypten's built-in functionality to measure the communication cost over 20 epochs of training. When using cross-entropy loss in both fully private and semi-private training settings, the total transmission volume is approximately 63.9 Gigabytes. In contrast, training with MSE results in significantly lower communication, around 14.1 Gigabytes, representing a reduction by a factor of about 4.5. This substantial gap highlights that cross-entropy requires considerably more computation under secure settings, which in turn increases the number of communication rounds needed. Specifically, training with

cross-entropy incurs roughly 1.13 million communication rounds, whereas MSE requires only about 58,000.

Although semi-private training generally incurs lower transmission and fewer communication rounds compared to fully private training, the difference is minor in comparison to the impact of the loss function. This is because the secret-shared input features (x) are still involved in the backpropagation process, which remains within the secure computation framework. As such, the fundamental communication overhead of secure backpropagation cannot be avoided, and the difference between semi-private and fully private settings is relatively limited.

In terms of communication time, semi-private training exhibits noticeable savings over fully private training. This reduction stems from the elimination of communication associated with private labels. While the total number of communication rounds in semi-private training is only slightly lower, by 1,260 rounds for both loss functions, each skipped round avoids a full network latency and kernel invocation, which together significantly decrease overall communication time. Furthermore, the communication savings, 1,260 rounds and 58,615,040 bytes, are identical across both loss functions, indicating that making labels public removes a common set of operations regardless of whether cross-entropy or MSE is used.

	Round	Gigabytes	Time(s)
Fully private (CE)	1132111	63.96	9501
Fully private (MSE)	58471	14.17	1660.37
Semi-private (CE)	1130851	63.91	2972.86
Semi-private (MSE)	57211	14.11	183.26

Table 8: Communication cost of MPC. Rounds represent the total number of global synchronisation barriers executed by the protocol. Bytes indicate the total volume of data exchanged throughout the process. Time refers to he actual elapsed time during communication.

4.4 Comparative Analysis: HE and MPC

Due to differences in underlying implementation, PyTorch is used in the MPC experiment but not in the HE setup, as it offers additional optimizations that reduce training time. Moreover, different optimizers are applied during backpropagation: the HE experiment uses the Adam optimizer, while the MPC experiment employs SGD with momentum. These choices contribute to the discrepancy in baseline accuracy. The baseline model accuracies are compared in Figure 9. Therefore, only the communication cost is directly comparable between the two setups.

4.4.1 Performance

Although a direct performance comparison is not technically appropriate, we can analyse the trend by comparing them. As shown in Table 9c, the training time in both HE and MPC

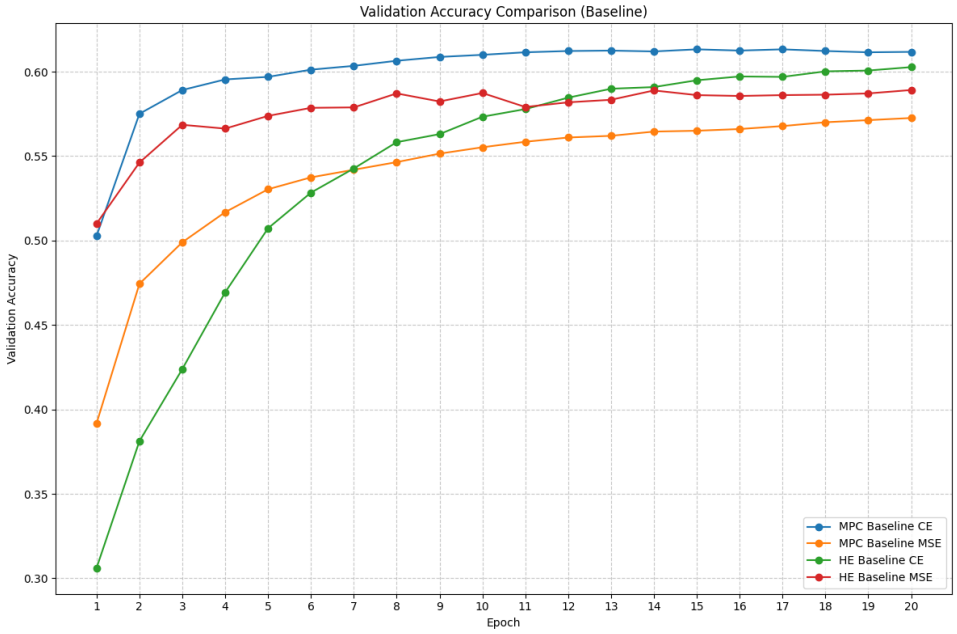


Figure 9: Comparison of accuracy between plaintext models in HE and MPC settings. Loss functions used: cross-entropy (CE) and mean squared error (MSE).

settings is significantly higher than that of its plaintext baseline. In semi-private training of HE, it increases to 66.37 hours, while in the case of MPC, under the semi-private setting, the total training time is only 1.1663 hours. Furthermore, a fully private setting is feasible in MPC but was not implemented for HE due to its excessive computational overhead.

Table 9b compares the performance of different loss functions, cross-entropy and MSE, under HE and MPC settings. For HE, the training times for MSE and cross-entropy are similar because, in the semi-private setting, only the forward pass is computed in ciphertext. The more complex operations required by cross-entropy are executed in plaintext, which results in comparable training times. In contrast, under MPC, even with semi-private training, the performance of cross-entropy is still lower than that of MSE. This is because cross-entropy involves non-linear operations that must be performed under MPC, making them computationally expensive. Finally, Table 9b also presents the comparison of the two loss functions under MPC in the fully private setting. It shows that MSE, which involves fewer purely linear operations, results in faster training than cross-entropy.

These results indicate that HE incurs a substantially higher computational cost, making it impractical in time-sensitive scenarios. Additionally, under stricter security requirements, where no data may be decrypted during computation, HE is not suitable, since the training time will be extensive. While MPC also introduces computational overhead, it remains considerably more efficient than HE and is practical even under fully private conditions.

	MSE (h)	CE (h)
HE	68.26	66.37
MPC	0.16	1.16

(a) Semi-private training time (HE vs. MPC). Loss functions used: Cross-entropy (CE) and Mean Squared Error (MSE). Fully private training with HE is not applicable.

	MSE (h)	CE (h)
MPC	1.37	3.42

(b) Fully private training time with MPC. Loss functions used: Cross-entropy (CE) and Mean Squared Error (MSE).

	Plaintext (h)	Semi-private (h)	Fully private (h)
HE	0.0081	66.37	-
MPC	0.0003	1.1663	3.4326

(c) Training time comparison (HE vs. MPC). Loss functions used: Cross-entropy (CE).

Table 9: Training time (HE vs. MPC) for the classification head on 20,895 training samples. "Fully private" means both model and data (inputs, labels, loss, gradients) are treated as secret sharing. In "Semi-private" setups, only the features are secret sharing.

4.4.2 Accuracy

Both HE and MPC perform well in terms of accuracy, with only minor differences observed between the baselines and the privacy-preserving models. As shown in Figures 5 and 7, the training curves for MPC are more distinct compared to those of HE, particularly between epochs 3 and 14. This discrepancy may stem from the limitations of HE, which cannot perform computations on floating-point numbers directly. Instead, such operations are approximated, and the encryption process introduces additional noise. These factors contribute to some inaccuracy during training; however, the model eventually converges as training progresses. In contrast, MPC exhibits nearly identical training curves to its baseline, due to CrypTen's implementation of fixed-point arithmetic and truncation protocols, which enable efficient and accurate approximations of non-linear functions [62].

Unexpectedly, the HE model shows a slightly higher result than the plaintext model, even though the difference is minor (e.g., 0.6047 in HE vs. 0.6027 in plaintext). In contrast, the accuracy between MPC and plaintext training is almost identical, with the same result in semi-private and plaintext training, and only a slight difference between fully private and plaintext training. Compared to semi-private training, fully private training includes non-linear operations (e.g., loss computation), which are approximated and may introduce discrepancies in the result. However, HE performs semi-private training, where only the dot product is encrypted. Even when the random seed is fixed, the results still differ. This may be because CKKS is ultimately an approximate method; deviations can accumulate during multiplication. In addition, there are some slight differences between the implementations of HE and plaintext training. Certain operations are not available in TenSEAL (e.g., `np.outer`); therefore, we implemented a similar approach to achieve

the same functionality. All these factors may contribute to the final accuracy differences observed in HE training.

4.4.3 Communication Cost

As discussed in Sections 4.2.4 and 4.3.4, HE results in a larger amount of transmitted data, whereas MPC leads to significantly higher communication time. A comparison of the communication overhead is presented in Table 10.

	Gigabytes	Time(s)
HE	156.25	530.92
MPC	64.61	2972.86

Table 10: Comparison of the communication cost of HE and MPC. Both results are semi-private training, and cross-entropy is used. Time refers to he actual elapsed time during communication.

HE computations are performed on a single machine, except that the client sends the ciphertext to the server for each batch and receives the corresponding encrypted label. In our setup, communication between two machines is required due to the semi-private setting, where accessing plaintext data during computation is necessary.

The results show that although the amount of transmitted data in HE is larger than in MPC, the overall communication time is lower. This is primarily due to the nature of MPC, particularly secret sharing. In MPC, private data is split between parties, and for non-linear operations such as multiplication or activation functions, both parties must collaborate to preserve correctness and privacy. As a result, MPC requires more communication rounds. Even though the size of transmitted data is smaller compared to HE, the total communication time is higher.

Regarding the actual cost of communication, for instance, in AWS, there is no charge for data transmitted between two instances within the same region when using private IP addresses. Given the requirements and security assumptions of our study, such an environment is sufficient for our purposes.

5 Conclusion

This thesis focuses on the training stage of PPML, particularly in the context of deep learning models, to evaluate the impact of PETs, specifically HE and MPC. We designed and implemented a system that utilizes features extracted from a transformer model as input to a deep neural network, incorporating HE and MPC to enable privacy-preserving training. Two sets of experiments were conducted to examine the accuracy, computational overhead, and communication cost associated with HE and MPC, followed by a comparative analysis aimed at informing both industry practices, especially under stringent data privacy regulations, and future research addressing more complex scenarios.

Experimental results show that both HE and MPC can achieve accuracy close to that of plaintext models. While HE exhibits some discrepancies during training due to its approximate arithmetic and encryption-induced noise, it gradually converges and reaches near-plaintext accuracy after 20 epochs. In contrast, MPC maintains training curves and final accuracy that are almost indistinguishable from the plaintext baseline.

In terms of performance, HE introduces a significantly higher computational overhead. Training a shallow deep neural network for 20 epochs using HE takes nearly three days, whereas the same task under MPC completes in about one hour, highlighting MPC's greater practicality in performance-critical applications.

Although MPC incurs higher communication time compared to HE, the volume of transmitted data is lower, and the actual monetary cost remains minimal in a setup involving two AWS EC2 instances within the same region using a private network. By contrast, accelerating HE through more powerful CPU and memory resources requires upgrading the instance, which may incur a higher cost than the communication overhead of MPC. However, in scenarios involving cross-region communication or inter-organizational collaboration, the increased communication demands of MPC must be carefully considered.

In summary, MPC proves to be the more feasible option. It achieves comparable accuracy to plaintext models, requires significantly less training time, and, in our experimental setting, incurs lower effective costs when considering cloud infrastructure. While HE offers strong privacy guarantees, its computational inefficiency makes it less practical in environments where performance and scalability are critical.

6 Limitations and Future Work

Despite achieving the primary objective of this study, several limitations should be considered for future work.

First, the deep learning model employed, which is a fully connected neural network consisting only of input and output layers, is relatively simple, and the resulting accuracy is suboptimal. Although improving accuracy was not the primary goal of this research, the study successfully demonstrates the potential of transfer learning. For more complex and practical applications, it is recommended that more sophisticated models be explored.

Second, the HE library used, TenSEAL, does not support PyTorch, which offers more efficient tensor operations, GPU acceleration, and multi-threading capabilities. This limitation contributes to slower training in the HE implementation. Additionally, the current experiment does not separate the roles of client and server; the data holder and the model trainer are assumed to be the same party. Future work should adopt a setup that better reflects real-world scenarios by more clearly defining and separating these roles.

The implementation of HE in this study uses vector-wise computation, which results in inefficient use of ciphertexts, as more data can be packed into a single ciphertext. Tensor-wise computation could be considered for more efficient communication, although this approach may require more memory.

Finally, there is an unexpected result in which HE shows slightly higher accuracy than plaintext. To address this discrepancy, it is important to ensure that the implementations of HE and plaintext are identical. Additionally, more carefully tuned CKKS parameters should be considered to reduce errors in the encrypted dot product as much as possible.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[2] Regulation - 2016/679 - EN - gdpr - EUR-lex. Doc ID: 32016R0679 Doc Sector: 3 Doc Title: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance) Doc Type: R Usr_lan: en. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng

[3] California consumer privacy act (CCPA) | state of california - department of justice - office of the attorney general. [Online]. Available: https://oag.ca.gov/privacy/ccpa

[4] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 3320–3328.

[5] C. Dwork, "Differential privacy: A survey of results," in Theory and Applications of Models of Computation, M. Agrawal, D. Du, Z. Duan, and A. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–19.

[6] A. Abadi, V. A. Dasu, and S. Sarkar, "Privacy-preserving data deduplication for enhancing federated learning of language models," Cryptology ePrint Archive, Paper 2024/1151, 2024. [Online]. Available: https://eprint.iacr.org/2024/1151

[7] K. Nandakumar, N. Ratha, S. Pankanti, and S. Halevi, "Towards deep neural network training on encrypted data," in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2019, pp. 40–48.

[8] M. Brand and G. Pradel, "Practical privacy-preserving machine learning using fully homomorphic encryption," Cryptology ePrint Archive, Paper 2023/1320, 2023. [Online]. Available: https://eprint.iacr.org/2023/1320

[9] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," Cryptology ePrint Archive, Paper 2017/396, 2017. [Online]. Available: https://eprint.iacr.org/2017/396

[10] J. Chiang, "Privacy-preserving logistic regression training on large datasets." [Online]. Available: http://arxiv.org/abs/2406.13221

[11] C. Bonte and F. Vercauteren, "Privacy-preserving logistic regression training," Cryptology ePrint Archive, Paper 2018/233, 2018. [Online]. Available: https://eprint.iacr.org/2018/233

[12] S. J. Pan and Q. Yang, "A survey on transfer learning," IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345–1359, 2010.

[13] J. Kim, H. Shim, and K. Han, Comprehensive Introduction to Fully Homomorphic Encryption for Dynamic Feedback Controller via LWE-Based Cryptosystem. Singapore: Springer Singapore, 2020, pp. 209–230. [Online]. Available: https://doi.org/10.1007/978-981-15-0493-8_10

[14] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," J. ACM, vol. 56, no. 6, Sep. 2009. [Online]. Available: https://doi.org/10.1145/1568318.1568324

[15] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," Cryptology ePrint Archive, Paper 2011/344, 2011. [Online]. Available: https://eprint.iacr.org/2011/344

[16] C. Gentry, "Fully homomorphic encryption using ideal lattices," in Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, ser. STOC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 169–178. [Online]. Available: https://doi.org/10.1145/1536414.1536440

[17] R. A. DeMillo, R. J. Lipton, D. P. Dobkin, and A. K. Jones, Foundations of Secure Computation. USA: Academic Press, Inc., 1978.

[18] E. Makri, "Secure and efficient computing on private data," Ph.D. dissertation, KU Leuven, 2017.

[19] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," ACM Comput. Surv., vol. 51, no. 4, Jul. 2018. [Online]. Available: https://doi.org/10.1145/3214303

[20] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, ser. CCSW '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 113–124. [Online]. Available: https://doi.org/10.1145/2046660.2046682

[21] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," Cryptology ePrint Archive, Paper 2010/520, 2010. [Online]. Available: https://eprint.iacr.org/2010/520

[22] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," Cryptology ePrint Archive, Paper 2009/571, 2009. [Online]. Available: https://eprint.iacr.org/2009/571

[23] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping," Cryptology ePrint Archive, Paper 2011/277, 2011. [Online]. Available: https://eprint.iacr.org/2011/277

[24] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal, "Tenseal: A library for encrypted tensor operations using homomorphic encryption," 2021.

[25] "Microsoft SEAL (release 4.1)," https://github.com/Microsoft/SEAL, Jan. 2023, microsoft Research, Redmond, WA.

[26] I. Iliashenko and V. Zucca, "Faster homomorphic comparison operations for BGV and BFV," Cryptology ePrint Archive, Paper 2021/315, 2021. [Online]. Available: https://eprint.iacr.org/2021/315

[27] L. Ducas and D. Micciancio, "FHEW: Bootstrapping homomorphic encryption in less than a second," Cryptology ePrint Archive, Paper 2014/816, 2014. [Online]. Available: https://eprint.iacr.org/2014/816

[28] S. Halevi and V. Shoup, "Design and implementation of HElib: a homomorphic encryption library," Cryptology ePrint Archive, Paper 2020/1481, 2020. [Online]. Available: https://eprint.iacr.org/2020/1481

[29] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," Cryptology ePrint Archive, Paper 2016/870, 2016. [Online]. Available: https://eprint.iacr.org/2016/870

[30] J. Alperin-Sheriff and C. Peikert, "Faster bootstrapping with polynomial error," Cryptology ePrint Archive, Paper 2014/094, 2014. [Online]. Available: https://eprint.iacr.org/2014/094

[31] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," Cryptology ePrint Archive, Paper 2013/340, 2013. [Online]. Available: https://eprint.iacr.org/2013/340

[32] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Paper 2012/144, 2012. [Online]. Available: https://eprint.iacr.org/2012/144

[33] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," Cryptology ePrint Archive, Paper 2011/133, 2011. [Online]. Available: https://eprint.iacr.org/2011/133

[34] T. Lu, B. Zhang, X. Zhang, and K. Ren, "A new PPML paradigm for quantized models," Cryptology ePrint Archive, Paper 2024/1132, 2024. [Online]. Available: https://eprint.iacr.org/2024/1132

[35] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," Cryptology ePrint Archive, Paper 2016/421, 2016. [Online]. Available: https://eprint.iacr.org/2016/421

[36] A. Costache, B. R. Curtis, E. Hales, S. Murphy, T. Ogilvie, and R. Player, "On the precision loss in approximate homomorphic encryption," Cryptology ePrint Archive, Paper 2022/162, 2022. [Online]. Available: https://eprint.iacr.org/2022/162

[37] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," Cryptology ePrint Archive, Paper 2012/230, 2012. [Online]. Available: https://eprint.iacr.org/2012/230

[38] Y. Lindell, "Secure multiparty computation (MPC)," Cryptology ePrint Archive, Paper 2020/300, 2020. [Online]. Available: https://eprint.iacr.org/2020/300

[39] A. C.-C. Yao, "How to generate and exchange secrets," in 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). IEEE, pp. 162–167. [Online]. Available: http://ieeexplore.ieee.org/document/4568207/

[40] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, ser. STOC '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 218–229. [Online]. Available: https://doi.org/10.1145/28395.28420

[41] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries," Cryptology ePrint Archive, Paper 2007/060, 2007. [Online]. Available: https://eprint.iacr.org/2007/060

[42] A. Shamir, "How to share a secret," Commun. ACM, vol. 22, no. 11, p. 612–613, Nov. 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[43] O. Elharrouss, Y. Mahmood, Y. Bechqito, M. Serhani, E. Badidi, J. Riffi, and H. Tairi, Loss Functions in Deep Learning: A Comprehensive Review.

[44] S. J. Prince, Understanding Deep Learning. The MIT Press, 2023. [Online]. Available: http://udlbook.com

[45] T. Tieleman and G. Hinton, "Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude," http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, 2012, coursera: Neural Networks for Machine Learning.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," CoRR, vol. abs/1412.6980, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6628106

[47] E. Makri, D. Rotaru, N. P. Smart, and F. Vercauteren, "EPIC: Efficient private image classification (or: Learning from the masters)," Cryptology ePrint Archive, Paper 2017/1190, 2017. [Online]. Available: https://eprint.iacr.org/2017/1190

[48] L. Yin, L. Wang, S. Lu, R. Wang, Y. Yang, B. Yang, S. Liu, A. AlSanad, S. A. AlQahtani, Z. Yin, X. Li, X. Chen, and W. Zheng, "Convolution-transformer for image feature extraction," Computer Modeling in Engineering & Sciences, vol. 141, no. 1, pp. 87–106, 2024. [Online]. Available: https://www.sciopen.com/article/10.32604/cmes.2024.051083

[49] A. Mahendran and A. Vedaldi, " Understanding deep image representations by inverting them ," in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2015, pp. 5188–5196. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2015.7299155

[50] A. Dosovitskiy and T. Brox, "Inverting visual representations with convolutional networks," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4829–4837.

[51] H. O. Shahreza, A. George, and S. Marcel, "Face reconstruction from face embeddings using adapter to a face foundation model." [Online]. Available: http://arxiv.org/abs/2411.03960

[52] R. Xu, N. Baracaldo, and J. Joshi, "Privacy-preserving machine learning: Methods, challenges and directions." [Online]. Available: http://arxiv.org/abs/2108.04417

[53] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: applying neural networks to encrypted data with high throughput and accuracy," in Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ser. ICML'16. JMLR.org, 2016, p. 201–210.

[54] T. v. Elsloo, G. Patrini, and H. Ivey-Law, "SEALion: a framework for neural network inference on encrypted data." [Online]. Available: http://arxiv.org/abs/1904.12840

[55] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster CryptoNets: Leveraging sparsity for real-world encrypted inference." [Online]. Available: http://arxiv.org/abs/1811.09953

[56] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," Cryptology ePrint Archive, Paper 2017/452, 2017. [Online]. Available: https://eprint.iacr.org/2017/452

[57] C. Jin, M. Ragab, and K. M. M. Aung, "Secure transfer learning for machine fault diagnosis under different operating conditions," in Provable and Practical Security: 14th International Conference, ProvSec 2020, Singapore, November 29 – December 1, 2020, Proceedings. Berlin, Heidelberg: Springer-Verlag, 2020, p. 278–297. [Online]. Available: https://doi.org/10.1007/978-3-030-62576-4_14

[58] S. Lee, G. Lee, J. W. Kim, J. Shin, and M.-K. Lee, "HETAL: efficient privacy-preserving transfer learning with homomorphic encryption," in Proceedings of the 40th International Conference on Machine Learning, ser. ICML'23, vol. 202. JMLR.org, pp. 19 010–19 035.

[59] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in Proceedings of the 38th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8748–8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html

[60] Zama, "Concrete ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists," 2022, https://github.com/zama-ai/concrete-ml.

[61] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020. [Online]. Available: https://doi.org/10.1145/3372297.3417872

[62] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," in arXiv 2109.00984, 2021.

[63] Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang, "Deepfashion: Powering robust clothes recognition and retrieval with rich annotations," in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[64] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption security standard," HomomorphicEncryption.org, Toronto, Canada, Tech. Rep., November 2018.

[65] I. Damgard, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," Cryptology ePrint Archive, Paper 2011/535, 2011. [Online]. Available: https://eprint.iacr.org/2011/535

[66] I. Damgård, D. Escudero, T. Frederiksen, M. Keller, P. Scholl, and N. Volgushev, "New primitives for actively-secure MPC over rings with applications to private machine learning," Cryptology ePrint Archive, Paper 2019/599, 2019. [Online]. Available: https://eprint.iacr.org/2019/599

[67] L. van der Maaten and G. Hinton, "Viualizing data using t-sne," Journal of Machine Learning Research, vol. 9, pp. 2579–2605, 11 2008.