PriFT: Private Fine-Tuning using off-the-shelf MPC and HE libraries

Qiuxuan Ma Eleftheria Makri
Nusa Zidaric
LIACS, Leiden University, the Netherlands

Abstract

Privacy-Preserving Machine Learning (PPML) is a methodology designed to maintain data privacy throughout the machine learning pipeline. Although cryptographically protecting input privacy at the training phase is theoretically feasible, it remains computationally intensive in practice. As such, most recent works in this area focus on the inference phase. In contrast, we consider the training phase. Our goal is to enable machine learning engineers to use customer data earlier in the ML pipeline without compromising customer privacy or violating regulations. In this work, we introduce a framework named PriFT (Private Fine-Tuning), which leverages a transformer as a feature extractor and then performs training of a neural network on privacy-protected features. PriFT supports *fully-private* training, where the data is encrypted in the entire ML pipeline, as well as *semi-private* training, which balances privacy and performance by decrypting the true and predicted labels during training. PriFT can perform secure training both by means of Multiparty Computation (MPC) and based on Homomorphic Encryption (HE), which allows for a direct comparison of the two most prevalent cryptographic solutions for secure computation on a real-world use case. The codebase of our experiments is fully open-sourced and based on well-established libraries, namely Crypten and TenSEAL. Our experimental results show that the MPC approach largely outperforms the HE approach, especially in the semi-private setting. Furthermore, the MPC-based solution in the semi-private setting outperforms the fully-private training approximately by $3\times$, offering an adequate privacy-performance tradeoff. Our results show that both HE and MPC can achieve accuracy close to that of plaintext models.

1 Introduction

Today, many companies offer 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 ignored. The European General Data Protection Regulation (GDPR) requires organizations handling customer data to clearly state how the data will be used and prohibits its use for other purposes without obtaining additional consent [noab]. Similarly, the

Health Insurance Portability and Accountability Act (HIPAA) requires hospitals and insurance companies in the United States to keep patient medical records secure and private [Pub]. The California Consumer Privacy Act (CCPA), a state-level data privacy law, gives individuals the right to know what personal data is collected, request deletion of their data, and opt out of the sale of their data [noaa]. International companies serving customers around the world must comply with various regulations. They typically assert that they will not use customer data 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 phase in the production environment.

Recent research has made significant progress in Privacy-Preserving Machine Learning (PPML). However, most results in this field target the inference phase of the ML pipeline, while privacy-preserving training remains underexplored. Training models requires significantly more computational resources, and doing so in a privacy-preserving manner, makes it even more challenging [NRPH19]. While the feasibility of secure training has been shown by several works [NRPH19, BP23, MZ17, Chi, BV18], the target models are either (mostly) linear such as support vector machines (SVM), and logistic regression models, or shallow neural networks. These models are insufficient for many industry applications, where more complex architectures, such as transformers [VSP+17] are commonly used.

Instead of focusing on PPML for training an entirely new model from scratch, we investigate its application in transfer learning [PY10] for pre-trained models (PTMs). Our work aims to explore 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 [VSP+17]. It enables machine learning engineers to use customer data earlier in the ML pipeline without compromising customer privacy or violating regulations. It is important to involve actual data in the training phase, because fine-tuning a pre-trained model on target datasets has been shown to enhance model accuracy [YCBL14]. Consequently, fine-tuning a pre-trained model 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 deploy two secure computation technologies to the aforementioned architecture, namely Homomorphic Encryption (HE) and Multiparty Computation (MPC). The goal is to investigate how these approaches can be used during the training phase and to compare their performance and impact on model accuracy. Concretely, we make the following contributions:

- We design a framework that enables feature extraction based on transfer learning and then feeds the resulting features into a secure training platform, while allowing the selection of different *privacy levels*;
- We deploy the two most prominent secure computation technologies on a real-world use case;
- We implement our solutions and evaluate their performance, and perform a one-to-one comparison between the HE-based and the MPC-based approach;
- We provide an MPC approach that outperforms the HE approach, and we open-

source our implementation confirming this.

2 Preliminaries

In this section, we explain the secure computation techniques deployed: HE and MPC, and motivate our choice of schemes and implementation platforms. We also explain the deep learning concepts relevant to our work.

2.1 Homomorphic Encryption

HE is a cryptographic technique that allows computations to be performed directly on encrypted data without decrypting it [KSH20]. Concretely, a client can encrypt their data, outsource them to a server, and then the server can perform (meaningful) computations on this data, and return back the encrypted result to the client. As such, HE facilitates secure computation. There are different types of homomorphic encryption schemes. Partially Homomorphic Encryption (PHE) allows only one type of operation (addition or multiplication) to be performed on the encrypted data. Somewhat Homomorphic Encryption (SHE) can support a limited number of both operations; this number of operations is determined implicitly by several parameters of the cryptosystem that need to be defined. Similarly, Leveled Homomorphic Encryption [BGV11], supports a limited number of both operations, but unlike SHE, this maximum level L that the homomorphic circuit is able to evaluate is explicitly predefined as a parameter of the cryptosystem. Although Fully Homomorphic Encryption (FHE) supports an unlimited number of both operations, it introduces a significant performance penalty.

Given our application scenario, the most suitable type of homomorphic encryption is a leveled HE. In addition, our implementation is built on top of TenSEAL [BRCB21], which is based on SEAL [SEA23], and supports a leveled HE, namely the CKKS scheme [CKKS16]. The CKKS (Cheon-Kim-Kim-Song) encryption scheme is specifically designed to handle computations involving approximate, real-valued numbers. Like other HE schemes, such as BGV [BGV11] and BFV [FV12], CKKS supports Single Instruction Multiple Data (SIMD) operations, but provides approximate results that are valid up to a certain precision, meaning that computations lose accuracy incrementally [CKKS16]. Due to its native support for real-value plaintexts, CKKS is widely applied in PPML scenarios [CCH$^+$22].

The effectiveness and practicality of CKKS largely depend on selecting appropriate parameters. An important parameter is the polynomial modulus degree, N, which typically is a power of two. This polynomial defines a ring; 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 [LPR12], and improves the SIMD throughput. However, increasing N also increases the number of coefficients in plaintexts and ciphertexts, and as such decreases the efficiency of all involved operations (i.e., encryption, evaluation, decryption) [CKKS16].

Homomorphic operations, particularly multiplication, cause rapid noise growth; excessive noise can lead to decryption errors. To manage the growth of noise, Cheon

et al. [CKKS16] introduced a rescaling technique, which performs modulus reduction after multiplication operations. Let $R = \mathbb{Z}_q[X]/(X^N + 1)$, where q is the coefficient modulus. In the leveled implementation of CKKS, the modulus q is typically a product forming a moduli chain $q = q_0 \cdot q_1 \cdot \ldots \cdot q_L$, where each q_k is a small prime (e.g., $2^{30} + 7$), and L is the supported level of multiplication. Rescaling reduces this composite modulus q by dividing out one of these primes. HE schemes operate 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 for 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 4.3.

2.2 Multiparty Computation

Multiparty Computation (MPC) allows a set of mutually distrusting parties to jointly compute a function over their inputs, such that only the final result is revealed, and each participant's inputs remain private. No party can learn anything about the inputs of other parties, beyond what can be inferred by the output [Lin20].

MPC protocols can guarantee security under different adversarial models; e.g., an adversary corrupting protocol participants can be assumed to be semi-honest (aka honest-but-curious), where the corrupted parties are trying to learn as much information as possible about the private inputs, but they follow the protocol specification correctly, or an adversary can be active (aka malicious), where the corrupted parties can arbitrarily deviate from the protocol. Additionally, an MPC protocol can be secure against said adversaries, under different corruption thresholds; e.g., it can be secure against assuming an honest majority (i.e., the adversary can corrupt strictly fewer than half of the protocol participants), it can be full-threshold secure (i.e., secure as long as at least one party remains honest, thus $n-1$ out of the total n parties can be corrupted), or any other threshold $1 \leq t \leq n-1$ in between.

Perhaps the most prevalent cryptographic primitive to design MPC protocols is secret sharing. Secret sharing allows a secret to be distributed among a group of participants. Each participant holds a *share* of the original secret, which, on its own, does not reveal any information about the secret. An authorized subset of the participants can reconstruct the secret, while any unauthorized subset cannot learn anything about the secret, even if they collude. Linear secret sharing schemes are the most used in practice for MPC protocols, because, due to their linearity, they allow (linear) operations to be performed locally on the shares, while the result of these operations is also reflected on the secret upon reconstruction. The most notable linear secret sharing schemes are Shamir's secret sharing [Sha79] and additive secret sharing.

2.3 Deep Learning

Deep learning is a branch of machine learning that uses a layered neural network to learn hierarchical representations from data. It enables models to capture complex

patterns of data by optimizing large numbers of parameters through training. The training process is guided by task-specific loss functions. In our work, we use two loss functions: Mean Squared Error (MSE) and Categorical Cross-Entropy (CE). 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 [EMB$^+$].

Transfer learning serves as the fundamental methodology for our framework. It 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 [Pri23].

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. Transfer learning has been used as a preparation step before deploying PPML. EPIC [MRSV17], 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.

3 Related Work

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 [XBJ]; they also introduce a machine learning pipeline and different PPML approaches for different stages, including data preparation, model training and evaluation, model deployment, and model inference. To align with our research goal, we will primarily review the PPML studies for the model training and the inference phase.

Privacy-Preserving Machine Learning for inference. Inference is the phase in which a fully trained model processes user inputs and generates outputs, such as classification labels, regression values, or text. Inference primarily involves a forward pass through the network, without loss computation, regularization, and weight update [XBJ]. Numerous privacy preserving protocols and frameworks for model inference have been introduced. CryptoNets [DGBL$^+$16] in 2016, allows data owners to send their encrypted data to a cloud service, which uses the learned neural network without decryption to generate encrypted predictions, and send them back to data owners who have private keys that can decrypt the result. CryptoNets uses SHE to achieve privacy-preserving computation, and it achieves high accuracy (99%) on MNIST optical character recognition tasks. Two notable frameworks, inspired by CryptoNets, are SEALion [EPIL] and Faster CryptoNets [CBL$^+$]. They employ sparse representations and polynomial approximations for activation functions to accelerate homomorphic evaluations.

MiniONN [LJLA17] is a framework that transforms a trained model into an oblivi-

ous form, enabling it to generate outputs 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.

Privacy-Preserving Machine Learning for training. CryptoNets [DGBL$^+$16] identified challenges of PPML at the training phase. While theoretically feasible, it remains computationally intensive in practice. Training typically involves non-polynomial activation functions (e.g., ReLU, Softmax) and loss functions (e.g., cross-entropy). To accommodate HE, which only supports addition and multiplication, these functions are often approximated using polynomial expressions. Backpropagation, although consisting primarily of sums and products, becomes significantly slower when applied to encrypted data. In addition, the limited computation depth supported by SHE often proves infeasible for deep models, while FHE requires even more computational resources. Another issue is that encrypted training prevents direct inspection, thus making it difficult to correct errors [DGBL$^+$16]. One of the earliest notable works to train a neural network on encrypted data using HE is by Nandakumar et al. [NRPH19].Their approach leverages FHE for 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 neural network with two hidden layers, accuracy drops slightly from 96.4% to 96%. The trade-off is negligible compared to the security benefits of training on encrypted data.

Although Privacy Enhancing Technologies (PETs), such as FHE, improve data privacy, they also impose computational overhead 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. To mitigate these challenges, some research focuses on approaches like transfer learning, which leverages knowledge from well-labeled domains to support under-resourced ones [PY10]. PrivGD, introduced by Jin et al. [JRA20], 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. Similarly, Faster CryptoNets [CBL$^+$], a SHE-based solution, demonstrated the use of transfer learning employing delegated feature extraction technique offloaded to the client side, with encryption applied only to the evaluation of the fine-tuned layers on the server.

HETAL [LLK$^+$], 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. 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.

Another PET, MPC, is shown to be more time-efficient; however, it typically requires multiple rounds of communication among parties, leading to substantial over-

head [Lin20]. An earlier study by Mohassel and Zhang [MZ17] 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 SVM [BP23] and logistic regression [Chi, BV18].

4 Private Fine-Tuning Framework: PriFT

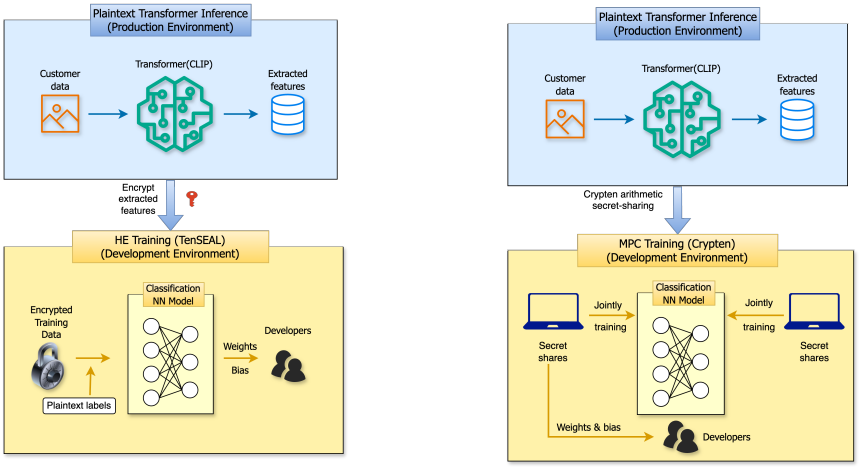
In this section, we detail our PPML pipeline. Specifically, we show how to integrate HE and MPC into the neural network training process, after the transfer learning-based feature extraction. We consider an application scenario, where the plaintext transformer inference takes place in the production 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., the extracted features), we treat this as the private input of the subsequent step in the pipeline, where the secure computation comes into play.

Our secure model architecture employs a single-layer perceptron, which is trained using the output from CLIP [RKH$^+$21], a multi-modal 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 in a privacy-preserving manner, because the transfer learning step happens in the production environment, where the customer data is already provided with consent to facilitate the inference step. 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 the privacy-preserving training, machine learning engineers can extract the weights and biases of the classification head for use during the inference stage (i.e., model serving). Fig. 1 gives a graphical representation of our framework, under both secure computation approaches, HE and MPC. We further elaborate on the indicated implementation platforms (TenSEAL and Crypten) in Section 4.2.

4.1 Privacy Level Definition

Several types of data are involved throughout the pipeline of our framework: input data, true labels, weights, biases, and output data (i.e., prediction labels). These data types require different levels of privacy protection, in the real-world application scenarios we consider. The input data requires the strongest privacy protection mechanisms. It is the main goal of our work to protect this data, hence it remains encrypted or secret shared during the training phase.

True labels and predicted labels are less privacy sensitive, compared to input data. While these labels can potentially reveal information about the input data, the information leakage is not direct, and encrypting them comes with a performance cost due to the additional computation required. As such, a trade-off between stronger privacy and more efficient training arises. To address this, we define two privacy levels, namely *fully private training*, and *semi-private training*, the workings of which are depicted in Fig. 2. It is in the discrepancy of the end user to decide how to tune this trade-off. In



(a) Implementation architecture of HE (b) Implementation architecture of MPC

Figure 1: PriFT Framework

addition, Fig. 2 demonstrates how backpropagation takes place in these two different privacy levels.

Fully Private Training. In this setting, all data involved in the pipeline remains encrypted. If all labels are 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. Fully private 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.

Semi-Private Training. In our HE experiment, we decrypt both the true labels and the predicted labels during training to maintain a balance between privacy and performance. In a practical deployment, the setup typically involves two parties: one party representing the production environment, *prod*, holds and encrypts the data, while the other party representing the development environment, *dev*, performs model training on the encrypted data. During training, *dev* evaluates the encrypted linear layers and transmits the resulting encrypted prediction vectors to *prod*, which decrypts these vectors to compute the loss and its corresponding output error scalars in plaintext. *prod* transmits only these plaintext loss gradient scalars back to *dev*. *dev* executes a hybrid ciphertext-plaintext multiplication against the encrypted input features to construct the encrypted weight gradients homomorphically. These calculated gradients are subsequently sent to *prod* for decryption, allowing *dev* to safely receive the plaintext gradients and update its parameters using the optimizer. Although this approach introduces additional communication overhead, it preserves the privacy of both input data and true labels throughout the training, while also reducing the computational burden associated with fully encrypted training.

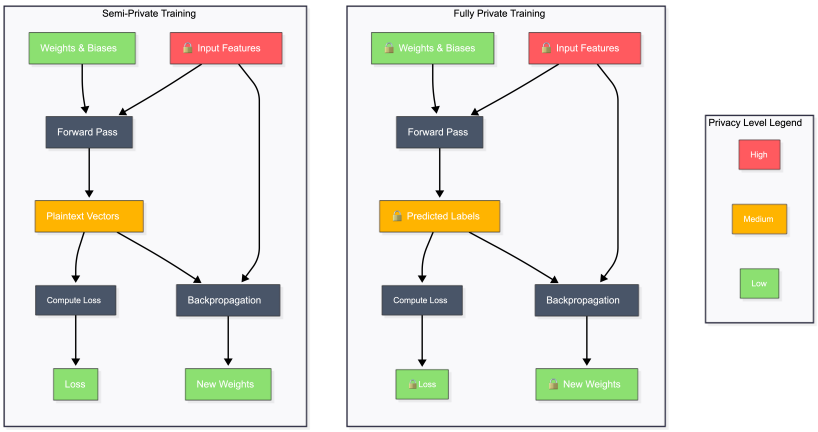


Figure 2: Backpropagation in both fully private and semi-private training. A lock icon indicates encrypted ciphertexts. In the semi-private setting, plaintext transitions occur twice: first to evaluate the intermediate error vector, and subsequently to decrypt the homomorphically computed gradients prior to local model parameter optimization.

4.2 Experiment Design

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. The experiment settings for HE and MPC are listed in Table 1.

There are various PPML frameworks designed for both HE and MPC. SEAL [SEA23] is a widely used HE library written in C++. An extension of SEAL, TenSEAL [BRCB21], enables operations on tensors, making it more suitable for deep learning applications. Another HE-based PPML library is Concrete-ML [Zam22], 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 are not optimized for training. In this work, we adopt TenSEAL for implementing the HE part of our framework, due to its native support for tensor computations and its flexibility for modifying and building a neural network training pipeline. For MPC, we considered MP-SPDZ [Kel20] as the currently most complete and well-documented framework, but we opted for Crypten [KVH+21], because it naturally supports PyTorch tensors, making it easier to build the private training process similar to regular PyTorch.

The baseline plaintext model for the MPC setting is implemented using PyTorch, while the HE baseline is developed from scratch, without using specialized frameworks. This ensures that both baselines are implemented in a manner that aligns with their respective encrypted counterparts. We perform the training using both CE and MSE as loss functions, with weight updates performed via the Adam optimizer. While CE is more suitable for multi-label classification tasks, MSE was included due to its linear gradient computation, which is more computationally efficient under encrypted settings.

Motivated by maintaining practicality in a real-world scenario, and based on our

privacy-level definition, plaintext data is used during loss evaluation and the final parameter optimization phase in the HE variant of our framework, with the transition to plaintext occurring after the encrypted weight gradients are computed and decrypted. 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.

The MPC variant of our framework employs the CrypTen library to handle both loss calculation and model updates. Although CrypTen supports both CE and MSE losses, it currently restricts weight updates to SGD. To align with this, our 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.

	HE	MPC
Library	TenSEAL	Crypten
GPU Used	No	No
Experiment System	MacOS M1	Amazon EC2 t2.micro
System Memory	16 GB	1 GB
CPU Cores	8	1
Deep learning framework	N/A	PyTorch
Epochs	20	20
Optimizer	Adam	SGD
Weight decay value	1e-5	N/A
Momentum	N/A	0.99
Batch size	64	64
Learning rate	1e-3	1e-1

Table 1: Training parameters for the HE and MPC experiments on the DeepFashion.

4.3 Homomorphic Encryption

TenSEAL supports operations using both CKKS vectors and tensors. In our implementation, we use `ckks_vector` as the primary operational unit. Although tensors typically offer higher computational efficiency, they require significantly more memory. Due to the limited RAM in our experimental setup environment, we cannot fully exploit the advantages of the CKKS tensor.

Encrypted computations in our setup are applied during both the forward and backward passes. Specifically, the forward pass dot product operation between input data and model parameters (weights and biases) is encrypted, as well as the initial phase of the backward pass, where a homomorphic ciphertext-plaintext multiplication is executed to derive the weight gradients. Once these encrypted weight gradients are obtained, the result is decrypted, as detailed in Section 4.1. Subsequent processes, such as loss evaluation and the final parameter optimization updates, occur entirely in plaintext.

Selecting suitable parameters for the CKKS scheme is a critical step in our design. TenSEAL provides three main parameters: `poly_mod_degree`, `coeff_mod_bit_sizes`,

and `global_scale`. The `poly_mod_degree` parameter denotes the degree of the polynomial modulus; higher degrees allow increased precision, but also cause an increase of the computation time. The `coeff_mod_bit_sizes` parameter 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. Lastly, `global_scale` is a scaling factor applied during encoding to map real or complex numbers to integers. It determines the precision of approximate arithmetic in CKKS.

According to the security requirements outlined in the Homomorphic Encryption Standard [ACC+18], 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, the total of `coeff_mod_bit_sizes` should not exceed 218, we choose configuration `[50, 40, 50]`, with a `global_scale` set to 40.

4.4 Secure Multiparty Computation

The MPC variant of our framework is implemented using CrypTen [KVH+21], which supports only the passive adversarial model. That is, we assume that an adversary can only passively corrupt the MPC parties, which then try to learn as much information as possible about the underlying secret data, but they correctly follow the protocol specification. Crypten implements arithmetic and binary secret sharing [DPSZ11, DEF+19, GMW87] 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, 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 adversaries (who, on top of what the passive ones can do, may arbitrarily deviate from the protocol specification). Therefore, CrypTen's support for passive security is sufficient for our purposes.

5 Performance Evaluation

In this section, we introduce the dataset used in our experiments. We detail the experimental setup for both the HE and the MPC variant of PriFT, and present the performance results of our experiments. Finally, we thoroughly compare the two variants we propose. We open source our implementation together with the designed experimental setup at: `https://github.com/h0rs3fa11/dl_pets`.

5.1 Datasets

We chose DeepFashion [LLQ+16] 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. 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. Concretely, zero-shot (i.e., without any fine-tuning) CLIP has an accuracy on the test dataset of only 0.0530. This 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, but rather 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 [MRSV17], although it focused on less complex operations, and inference instead of training.

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 2. Fig. 3 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 [vdMH08]. 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 2: Statistics of extracted features

5.2 Homomorphic Encryption

Performance Evaluation. The comparison of training times is presented in Table 3. Although we are in the semi-private setting, training with HE still requires significantly

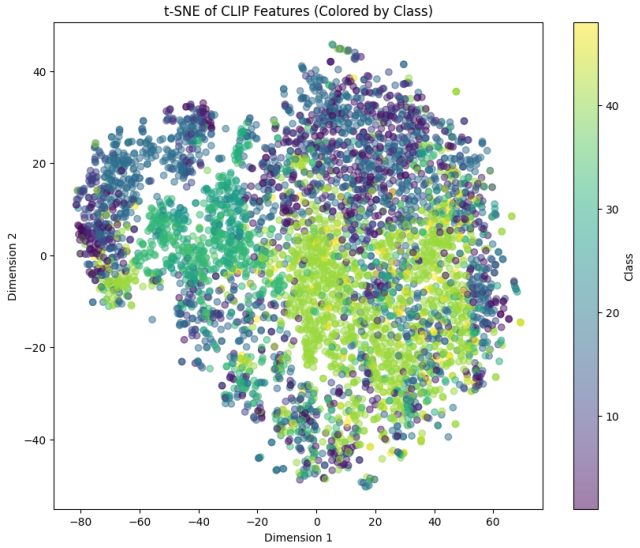


Figure 3: t-SNE of CLIP features

more time than the plaintext version. The training durations for CE and MSE complete in approximately 56 hours and 44 hours, respectively. While both methods share the computationally intensive encrypted forward pass and the hybrid ciphertext-plaintext multiplication during backpropagation, training with CE requires noticeably more time. This discrepancy arises because calculating the intermediate CE error scalars on the *prod* server requires computing complex non-linear functions, whereas deriving the MSE error scalars relies on highly efficient linear operations. In more complex scenarios (e.g., deeper neural network models), the training time using HE increases. Despite the significant overhead of the encrypted training, rapid response is usually crucial only in the inference phase, and most scenarios can tolerate a slower training.

	Avg Time/Epoch(s)	Total(s)	Total(h)	Accuracy
Plaintext (CE)	1.46	29.20	0.0081	0.6049
Plaintext (MSE)	1.37	27.32	0.0076	0.5891
Semi-private (CE)	10084.39	201687.77	56.02	0.6049
Semi-private (MSE)	8007.65	160153.04	44.48	0.5851

Table 3: Training time and accuracy (plaintext vs. HE encrypted) for the classification (20,895 samples).

Accuracy Evaluation. As shown in Fig. 4, the models using MSE initially have higher accuracy but quickly plateau, exhibiting slower improvement throughout the training process. In contrast, the CE 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 CE measures dis-

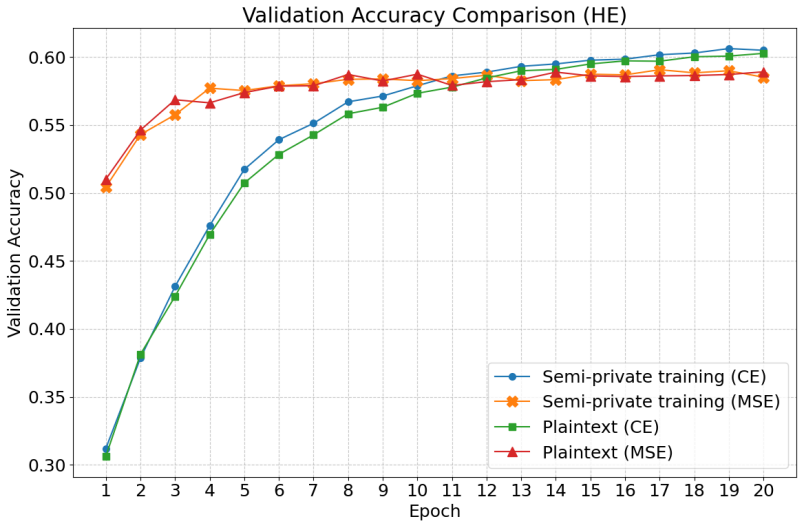


Figure 4: Accuracy of Homomorphic Encryption-based Training.

crepancies between probability distributions, making it particularly suitable for multiclass classification tasks. While CE clearly outperforms MSE in our multiclass classification experiment, MSE could be advantageous in other contexts.

The final validation accuracy after 20 epochs of training is summarized in Table 3. Notably, the plaintext model exhibits near-equivalent accuracy to the HE model, after 20 epochs. This highlights that HE models are able to perform accurate training, provided that the encryption parameters (e.g., CKKS parameters) are appropriately selected.

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, *prod*, who holds and encrypts the data, and could represent a server in a production environment, and a developer, *dev*, who performs the training on encrypted data. In our setup, the distributed workflow introduces distinct ciphertext and plaintext transmission steps across both the forward and backward passes. Specifically, we assume that *prod* is responsible for decrypting the predicted prediction vectors to compute the loss and its corresponding output error scalars in plaintext. These plaintext error scalars are sent back to *dev*, allowing the backward pass gradient multiplications to be initiated on the *dev* side.
The communication process for each batch consists of the steps depicted in Fig. 5:

1. *prod* sends the encrypted data and the evaluation key to *dev*. Using the evaluation key *dev* can perform homomorphic multiplications on ciphertexts.
2. *dev* sends the encrypted prediction vector to *prod*.
3. *prod* decrypts the raw prediction vectors, computes the loss, and transmits the

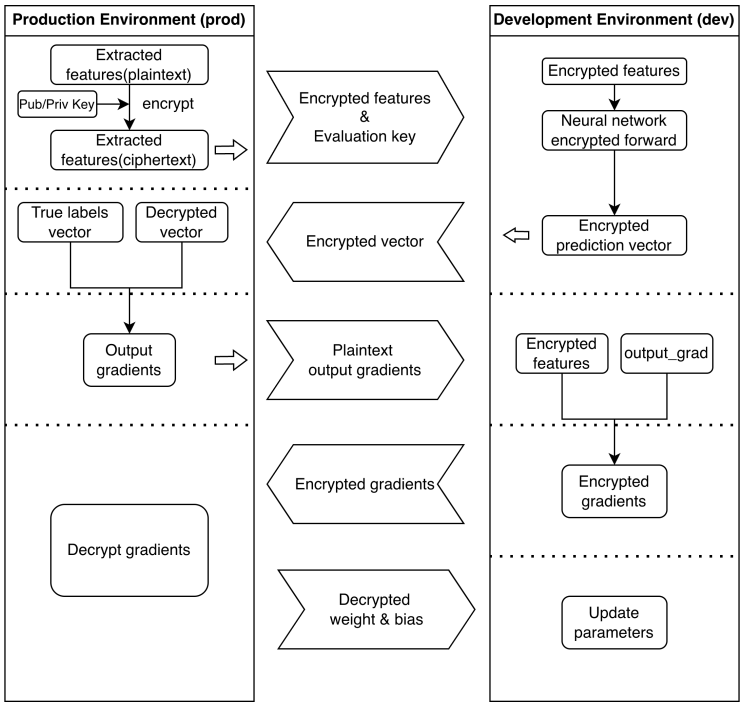


Figure 5: Communication process of the Homomorphic Encryption implementation. The diagram illustrates the process for a single batch.

plaintext loss gradient scalars back to *dev*.

4. *dev* executes a local homomorphic multiplication between its encrypted input features and the received error vector, returning the resulting encrypted weight gradients to *prod* for decryption.
5. *prod* transmits the decrypted, plaintext gradients back to *dev*, enabling the server to update the model parameters locally.

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 communication overhead is negligible because the transmitted error scalars are small plaintexts. However, Step 4 introduces a payload spike as *dev* sends the encrypted gradients to *prod*. Step 5 scales linearly with the dimensionality of the model parameters. 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, 50]`. Since the last modulus is used exclusively in key switching, it is not included in the ciphertext representation. Therefore, the ciphertext size is 90 bits. The bit size of a single polynomial's coefficient representation is the product of the polynomial modulus degree (8192) and the bit-length (90), resulting in 737,280 bits. Since a CKKS ciphertext consists of two polynomials [CKKS16], the

total size becomes 1,474,560 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 64×512 matrix, containing 32,768 values. Furthermore, because 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 1×512 vectors per batch. Therefore, the final ciphertext size is 94,371,840 bits, or approximately 11.08 MB. The evaluation key, which is 23 MB, needs to be transmitted only 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 prediction vector, which can be fit into a single ciphertext of 1,474,560 bits (i.e., 0.18 MB).

In Step 3, the transmission of the 32-bit plaintext error floats is negligible. In Step 4, transmitting the encrypted gradient vectors to *prod* repeats the ciphertext expansion profile observed in Step 2, requiring an additional 1,474,560 bits of bandwidth. Step 5 concludes with *prod* returning the 46×512 plaintext weight gradient elements to *dev* (0.094 MB). Accounting for this distributed round-trip, the total communication size is approximately 12.25 MB per batch. With 326 batches per epoch and 20 epochs in total, the overall transmission size amounts to 79,870 MB, plus a one-time transmission of evaluation keys, resulting in 79,893 MB. This estimate assumes no encoding methods (e.g., Protobuf, JSON) in place, and the overhead from transport protocols (e.g., HTTP, gRPC) is not accounted for.

Regarding network latency, we assume the same environment as used for MPC. 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 ms over 10 samples. Since `ping` measures round-trip time, and each communication round involves five steps, we approximate this as 2.5 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 135.73 ms. Over 20 epochs, the total communication time amounts to 884.95 seconds.

5.3 Secure Multiparty Computation

Similarly to the HE experiments, we compared accuracy and performance between private and plaintext training. Additionally, although fully private training (where the 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.

Performance Evaluation. The performance results are summarized in Table 4. As expected, MPC-based training is significantly slower than plaintext training. Among the loss functions, CE 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 CE 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.

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

Table 4: Training time (plaintext vs. MPC) for the classification (20,895 samples).

Accuracy Evaluation. Fig. 6 compares the validation accuracy over 20 epochs between plaintext models and fully private models using MPC, and Table 5 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 secret sharing has minimal impact on predictive performance. Additionally, models trained with CE consistently outperform those using MSE across all epochs.

Fig. 7 presents a comparison between the semi-private and fully private model. The two accuracy curves are nearly indistinguishable, indicating that secret sharing all components has a negligible effect on model accuracy, while incurring significantly higher computational overhead.

Communication Cost. We leverage Crypten's built-in functionality to measure the communication cost over 20 epochs of training. We summarize the results in Table 5, where we indicate the number of communication rounds, the total volume of data exchanged during the protocol (GB), and the time elapsed during communincation.

When using CE loss in both fully private and semi-private training settings, the total transmission volume is approximately 63.9 GB. In contrast, training with MSE results in significantly lower communication, around 14.1 GB, representing a reduction by a factor of about 4.5×. This substantial gap highlights that CE requires considerably more computation under secure settings, which in turn increases the number of communication rounds needed. Specifically, training with CE incurs roughly 1.13 million communication rounds, whereas MSE requires only about 58,000.

Although semi-private training generally incurs lower total communication cost and fewer 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 small.

In terms of communication time, semi-private training exhibits noticeable savings

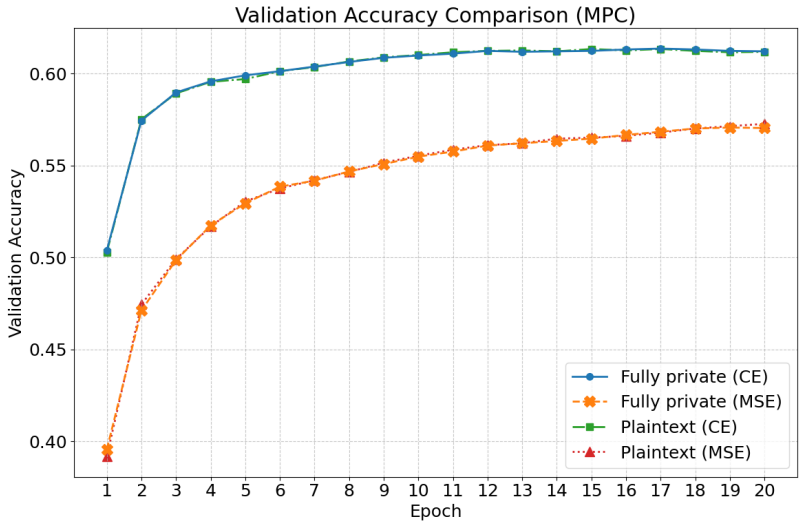


Figure 6: Accuracy of Secure Multiparty Computation.

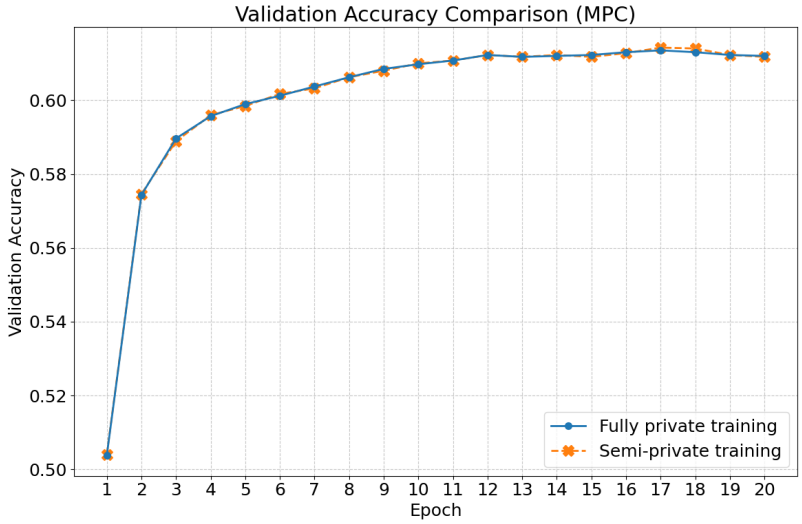


Figure 7: Accuracy Comparison: Fully private MPC vs. Semi-private MPC. Loss functions used: Cross-entropy

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 CE or MSE is used.

	Round	Gigabytes	Time(s)	Accuracy
Plaintext (CE)	N/A	N/A	N/A	0.6118
Plantext (MSE)	N/A	N/A	N/A	0.5726
Semi-private (CE)	1130851	63.91	2972.86	0.6118
Semi-private (MSE)	57211	14.11	183.26	0.5726
Fully private (CE)	1132111	63.96	9501	0.6120
Fully private (MSE)	58471	14.17	1660.37	0.5703

Table 5: Communication cost and validation accuracy of MPC.

5.4 Comparative Analysis: HE and MPC

Due to differences in the 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 Fig. 8. Therefore, only the communication cost is directly comparable between the two setups.

Performance. Although a direct performance comparison is not technically appropriate, given the implementation differences, we analyse and compare the trends exhibited by both approaches. As shown in Table 6a, the training time in both HE and MPC settings is significantly higher than that of its plaintext baseline. In semi-private training of HE, it increases to 56.02 hours, while in the case of MPC, under the semi-private setting, the total training time is 1.1663 hours. Lastly, a fully private setting is feasible in MPC, but was not implemented for HE, due to its high computational overhead.

Table 6a compares the performance of the two different loss functions we considered, CE and MSE, under the HE and MPC settings. For HE, as detailed in Section 5.2, training with MSE is notably faster than CE due to the computational efficiency of deriving linear error scalars during the plaintext evaluation phase. Similarly, under MPC, CE requires more training time than MSE across both semi-private and fully private settings, as the non-linear operations inherent to CE are computationally expensive to execute within secret sharing protocols. Finally, the table 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 CE.

In conclusion, our 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 even higher than the one we report in this work. While MPC also introduces computational overhead, it remains considerably more efficient than HE and is practical even under fully private conditions.

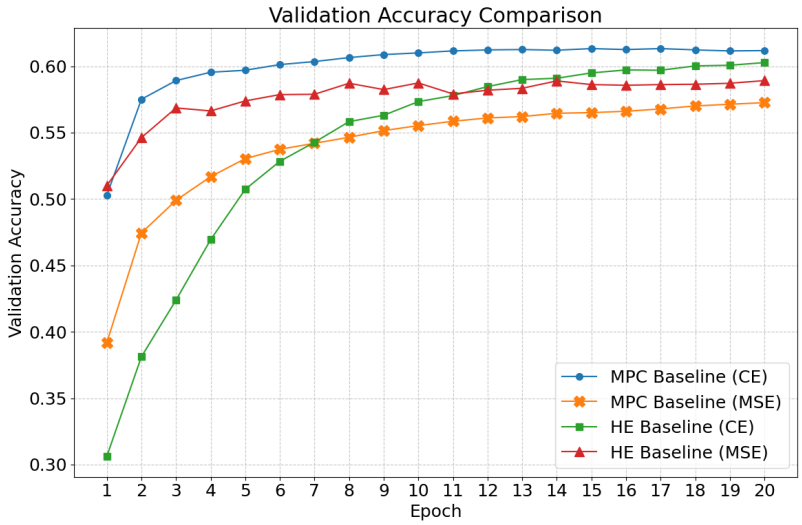


Figure 8: Comparison of accuracy between plaintext models in HE and MPC settings.

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 Fig. 4 and Fig. 6, the training curves for MPC are more distinct compared to those of HE, particularly between epochs 3 and 14. This is because with HE we cannot perform computations on floating-point numbers directly. Instead, such operations are approximated, and the encryption process introduces additional noise, contributing to loss of accuracy 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 [KVH+21].

	Plaintext (h)		Semi-private (h)		Fully private (h)	
Method	MSE	CE	MSE	CE	MSE	CE
HE	0.0076	0.0081	44.48	56.02	–	–
MPC	0.0002	0.0003	0.16	1.16	1.37	3.42

(a) Training time under different privacy settings

	Semi-private	
	Gigabytes	Time(s)
HE	79.89	884.95
MPC	64.61	2972.86

(b) Semi-private training

Table 6: Comparison of HE and MPC: (a) training time and (b) communication cost.

The accuracy between privacy training and plaintext training is almost identical for both HE and MPC, with the same result in semi-private and plaintext training, and only a slight difference between MPC 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.

Communication Cost. As discussed in Section 5.2 and 5.3, 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 6b.

While HE computations are largely executed on the server(i.e., dev), the semi-private distributed setup requires the client(i.e., prod) not only to receive the forward-pass predictions, but also to evaluate the loss in plaintext and exchange the resulting intermediate gradient matrices with the server during backpropagation.

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.

6 Conclusion

In this work, we focus 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, such as image classification tasks. To meet these demands, we designed and implemented a system, called PriFT, that uses features extracted from a transformer model as input to a deep neural network, and then feeds these features to the privacy-preserving part of our framework, which incorporates HE and MPC to complete the secure 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 neural network for 20 epochs using HE takes days, whereas the same task under MPC completes in about one hour, highlighting MPC's greater practicality in performance-critical applications.

Although MPC incurs a 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. In contrast, accelerating HE using more CPU and memory resources, 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.

References

[ACC+18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Paper 2011/277, 2011.

[BP23] Michael Brand and Gaëtan Pradel. Practical privacy-preserving machine learning using fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/1320, 2023.

[BRCB21] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. Tenseal: A library for encrypted tensor operations using homomorphic encryption, 2021.

[BV18] Charlotte Bonte and Frederik Vercauteren. Privacy-preserving logistic regression training. Cryptology ePrint Archive, Paper 2018/233, 2018.

[CBL+] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster CryptoNets: Leveraging sparsity for real-world encrypted inference.

[CCH+22] Anamaria Costache, Benjamin R. Curtis, Erin Hales, Sean Murphy, Tabitha Ogilvie, and Rachel Player. On the precision loss in approximate homomorphic encryption. Cryptology ePrint Archive, Paper 2022/162, 2022.

[Chi] John Chiang. Privacy-preserving logistic regression training on large datasets.

[CKKS16] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Paper 2016/421, 2016.

[DEF+19] Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. Cryptology ePrint Archive, Paper 2019/599, 2019.

[DGBL+16] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John 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*, ICML'16, page 201–210. JMLR.org, 2016.

[DPSZ11] I. Damgard, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Paper 2011/535, 2011.

[EMB+] Omar Elharrouss, Yasir Mahmood, Yassine Bechqito, Mohamed Serhani, Elarbi Badidi, Jamal Riffi, and H. Tairi. *Loss Functions in Deep Learning: A Comprehensive Review*.

[EPIL] Tim van Elsloo, Giorgio Patrini, and Hamish Ivey-Law. SEALion: a framework for neural network inference on encrypted data.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012.

[GMW87] 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*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.

[JRA20] Chao Jin, Mohamed Ragab, and Khin Mi Mi 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*, page 278–297, Berlin, Heidelberg, 2020. Springer-Verlag.

[Kel20] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[KSH20] Junsoo Kim, Hyungbo Shim, and Kyoohyung Han. *Comprehensive Introduction to Fully Homomorphic Encryption for Dynamic Feedback Controller via LWE-Based Cryptosystem*, pages 209–230. Springer Singapore, Singapore, 2020.

[KVH+21] B. Knott, S. Venkataraman, A.Y. Hannun, S. Sengupta, M. Ibrahim, and L.J.P. van der Maaten. Crypten: Secure multi-party computation meets machine learning. In *arXiv 2109.00984*, 2021.

[Lin20] Yehuda Lindell. Secure multiparty computation (MPC). Cryptology ePrint Archive, Paper 2020/300, 2020.

[LJLA17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. Cryptology ePrint Archive, Paper 2017/452, 2017.

[LLK+] Seewoo Lee, Garam Lee, Jung Woo Kim, Junbum Shin, and Mun-Kyu Lee. HETAL: efficient privacy-preserving transfer learning with homomorphic encryption. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *ICML'23*, pages 19010–19035. JMLR.org.

[LLQ+16] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou 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.

[LPR12] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Paper 2012/230, 2012.

[MRSV17] Eleftheria Makri, Dragos Rotaru, Nigel P. Smart, and Frederik Vercauteren. EPIC: Efficient private image classification (or: Learning from the masters). Cryptology ePrint Archive, Paper 2017/1190, 2017.

[MZ17] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. Cryptology ePrint Archive, Paper 2017/396, 2017.

[noaa] California consumer privacy act (CCPA) | state of california - department of justice - office of the attorney general.

[noab] 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.

[NRPH19] Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Shai Halevi. Towards deep neural network training on encrypted data. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 40–48, 2019.

[Pri23] Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023.

[Pub] Pub.L.No.104-191. HEALTH INSURANCE PORTABILITY AND ACCOUNTABILITY ACT OF 1996.

[PY10] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[RKH+21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 18–24 Jul 2021.

[SEA23] Microsoft SEAL (release 4.1). `https://github.com/Microsoft/SEAL`, January 2023. Microsoft Research, Redmond, WA.

[Sha79] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[vdMH08] Laurens van der Maaten and Geoffrey Hinton. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[XBJ] Runhua Xu, Nathalie Baracaldo, and James Joshi. Privacy-preserving machine learning: Methods, challenges and directions. `http://arxiv.org/abs/2108.04417`.

[YCBL14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[Zam22] Zama. Concrete ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists, 2022. `https://github.com/zama-ai/concrete-ml`.