FedML-HE: An Efficient Homomorphic-Encryption-Based Privacy-Preserving Federated Learning System

Weizhao Jin∗
University of Southern California
weizhaoj@usc.edu

Yuhang Yao∗
Carnegie Mellon University
yuhangya@andrew.cmu.edu

Shanshan Han
University of California Irvine
shanshan.han@uci.edu

Carlee Joe-Wong
Carnegie Mellon University
cjoe@andrew.cmu.edu

Srivatsan Ravi
University of Southern California
sravi@isi.edu

Salman Avestimehr
FedML Inc.
avestime@usc.edu

Chaoyang He
FedML Inc.
ch@fedml.ai

Abstract

Federated Learning trains machine learning models on distributed devices by aggregating local model updates instead of local data. However, privacy concerns arise as the aggregated local models on the server may reveal sensitive personal information by inversion attacks. Privacy-preserving methods, such as homomorphic encryption (HE), then become necessary for FL training. Despite HE’s privacy advantages, its applications suffer from impractical overheads, especially for foundation models. In this paper, we present FedML-HE, the first practical federated learning system with efficient HE-based secure model aggregation. FedML-HE proposes to selectively encrypt sensitive parameters, significantly reducing both computation and communication overheads during training while providing customizable privacy preservation. Our optimized system demonstrates considerable overhead reduction, particularly for large foundation models (e.g., ∼10x reduction for ResNet-50, and up to ∼40x reduction for BERT), demonstrating the potential for scalable HE-based FL deployment.

1 Introduction

Federated learning (FL) is increasingly popular in contemporary machine learning practices due to its ability to allow distributed clients to collectively train a global model without directly sharing data. Privacy preservation in standard federated learning systems depends on the distributed training process and the model aggregation function, such as FedAvg[McMahan et al. (2017)], FedSGD[Shokri & Shmatikov (2015)], and FedGAN[Rasouli et al. (2020)]. In FL, instead of uploading raw data to a central server for training, clients train models locally and share their models with the server, where the local models are aggregated based on the aggregation functions. While FL ensures that local raw data do not leave their original locations, it remains vulnerable to eavesdroppers and malicious FL servers that might exploit plaintext local models (or model updates) to reconstruct sensitive training data, i.e., data reconstruction attacks or gradient inversion attacks in literature Zhu et al. (2019);

*Equal Contribution

Figure 1: Data Reconstruction Attacks: an adversarial server can recover local training data from local model updates.

Existing defense methods that prevent privacy leakage from plaintext local models include differential privacy (DP) Truex et al. (2019a); Byrd & Polychroniadou (2020) and secure aggregation Bonawitz et al. (2017); So et al. (2022). DP adds noise to original models but may result in model performance degradation due to the privacy noises introduced. On the other hand, secure aggregation employs zero-sum masks to shield local model updates, ensuring that the details of each update remain private. However, secure aggregation demands additional interactive synchronization steps and is sensitive to client dropout, making it less practical in real-world FL applications, where the unstable environments of clients face challenges such as unreliable internet connections, and software crashes.

As shown in Table 1 compared to the non-HE FL solutions above, homomorphic encryption (HE) Pailier (1999); Gentry (2009); Fan & Vercauteren (2012); Brakerski et al. (2014); Cheon et al. (2017) offers a robust post-quantum secure solution that protects local models against attacks and provides stronger privacy guarantee while keeping the model aggregation with exact gradients. HE-based federated learning (HE-FL) encrypts local models on clients and performs model aggregation over ciphertexts on the server. This approach enables secure federated learning deployments with exactly the same model performance as vanilla FL and has been adopted by several FL systems Roth et al. (2022); IBM (2022); Zhang et al. (2020); Du et al. (2023) and a few domain-specific applications Stripelis et al. (2021); Yao et al. (2023).

Despite the advantages of homomorphic encryption, HE remains a powerful but complex cryptographic foundation with impractical overheads (as shown in Figure 2) for most real-world applications. Prior FL-HE solutions mainly employ existing generic HE methods without sufficient optimization for large-scale FL deployment Roth et al. (2022); IBM (2022); Zhang et al. (2020); Du et al. (2023). The scalability of encrypted computation and communication during federated training then becomes
Figure 2: Computational (left) and Computation (right) Overhead Comparison for Models of Different Sizes: Naive FedML-HE vs. Nvidia FLARE vs. Plaintext Aggregation. Due to TenSeal’s larger file sizes, FLARE did not finish the run on BERT on our 32GB memory machine.

Observation: As shown by the evaluation results in Figure 2, the computational and communication (package size) overheads introduced by HE is $O(n)$, both growing linearly with the input size $n$, which in our case the sizes of the models for aggregation. Although the unoptimized system is faster than Nvidia FLARE, the execution time and file size are still impractical, especially for large models.

To address these challenges, we propose FedML-HE, an efficient Homomorphic Encryption-based privacy-preserving FL system with Selective Parameter Encryption, designed for practical deployment across distributed edge devices. Our system significantly reduces communication and computation overheads, enabling HE-based federated learning to be more accessible and efficient in real-world scenarios (comparison with other popular HE-based FL work can be found in Table 2).

Table 2: Comparison with Existing HE-Based FL Systems. ✓ implies full support; ⊗ implies limited support: for Selective Parameter Encryption, FLARE offers the (random) partial encryption option which does not have clear indications on privacy impacts; for Encrypted Foundation Model Training, the other two platforms require massive resources to train foundation models in encrypted federated learning.

Key contributions:

• We propose FedML-HE, the first practical Homomorphic Encryption-based privacy-preserving FL system that supports encryption key management, encrypted FL platform deployment, encryption optimizations to reduce overhead, and is designed to support efficient foundation model federated training.

• We propose Selective Parameter Encryption that selectively encrypts the most privacy-sensitive parameters to minimize the size of encrypted model updates while providing customizable privacy preservation.
• Theoretical privacy analysis shows the HE system can ensure privacy under single-key and threshold adversaries and encrypting most sensitivity parameters provides orders-of-magnitude better privacy guarantees.
• Extensive experiments show that the optimized system achieves significant overhead reduction while preserving privacy against state-of-the-art ML privacy attacks, particularly for large models (e.g., ~10x reduction for HE-federated training ResNet-50 and up to ~40x reduction for BERT), demonstrating the potential for real-world HE-based FL deployments.

2 FedML-HE System Design

In this section, we first provide the overview of FedML-HE system in §2.1, define the threat model in §2.2, describe the algorithmic design of FedML-HE in §2.3, propose our efficient optimization method Selective Parameter Encryption after pinpointing the overhead bottleneck in §2.4, and explain how we integrate homomorphic encryption in federated learning from a software framework perspective in §2.5.

2.1 System Overview

As shown in Figure 3, our efficient HE-based federated training process at a high level goes through three major stages: (1) Encryption key agreement: the clients either use distributed threshold key agreement protocol or outsource a trusted key authority. We simplify the illustration here by abstracting the key pair of the public key and secret key (partial secret keys if using threshold protocol) as one key; in the Encryption Mask Calculation stage, clients use local datasets to calculate local model sensitivity maps which are homomorphically aggregated at the server to generate an encryption mask; in the Encrypted Federated Learning stage, clients use homomorphic encryption with encryption mask to protect local model updates where the server aggregates them but does not have access to sensitive local models.

Figure 3: FedML-HE System Pipeline: in the Encryption Key Agreement stage, clients can either use distributed threshold key agreement protocol or outsource a trusted key authority. We simplify the illustration here by abstracting the key pair of the public key and secret key (partial secret keys if using threshold protocol) as one key; in the Encryption Mask Calculation stage, clients use local datasets to calculate local model sensitivity maps which are homomorphically aggregated at the server to generate an encryption mask; in the Encrypted Federated Learning stage, clients use homomorphic encryption with encryption mask to protect local model updates where the server aggregates them but does not have access to sensitive local models.

2.2 Threat Model

We define a semi-honest adversary \( \mathcal{A} \) that can corrupt the aggregation server or any subset of local clients. \( \mathcal{A} \) follows the protocol but tries to learn as much information as possible. Loosely speaking, under such an adversary, the security definition requires that only the private information in local models from the corrupted clients will be learned when \( \mathcal{A} \) corrupts a subset of clients; no private information from local models nor global models will be learned by \( \mathcal{A} \) when \( \mathcal{A} \) corrupts the aggregation server.

When \( \mathcal{A} \) corrupts both the aggregation server and a number of clients, the default setup where the private key is shared with all clients (also with corrupted clients) will allow \( \mathcal{A} \) to decrypt local models from benign clients (by combining encrypted local models received by the corrupted server and the private key received by any corrupted client). This issue can be mitigated by adopting the threshold or multi-key variant of HE where decryption must be collaboratively performed by a certain number
of clients \cite{Aloufi2021, Ma2022, Du2023}. Since the multi-party homomorphic encryption issue is not the focus of this work, in the rest of the paper we default to a single-key homomorphic encryption setup, but details on threshold homomorphic encryption federated learning setup and microbenchmarks are provided in the appendix.

2.3 Algorithm for HE-Based Federated Aggregation

Privacy-preserving federated learning systems utilize homomorphic encryption to enable the aggregation server to combine local model parameters without viewing them in their unencrypted form by designing homomorphic encrypted aggregation functions. We primarily focus on FedAvg \cite{McMahan2017}, which has been proved as still one of the most robust federated aggregation strategies while maintaining computational simplicity \cite{Wang2022}.

\begin{algorithm}
\caption{HE-Based Federated Aggregation}
\begin{algorithmic}
\State $p$: the ratio of parameters for selective encryption;
\State $b$: (optional) differential privacy parameter.

\Comment{Key Authority Generate Key}
\State $(pk, sk) \leftarrow HE.KeyGen(\lambda)$;

\Comment{Local Sensitivity Map Calculation}
\For {each client $i \in [N]$ do in parallel}
\State $W_i \leftarrow Init(W)$;
\State $S_i \leftarrow Sensitivity(W, D_i)$;
\State $[S_i] \leftarrow Enc(pk, S_i)$;
\State Send $[S_i]$ to server;
\EndFor

\Comment{Server Encryption Mask Aggregation}
\State $[M] \leftarrow Select(\sum_{i=1}^{N} \alpha_i [S_i], p)$;

\Comment{Training}
\For {$t = 1, 2, \ldots, T$ do in parallel}
\If {$t = 1$}
\State Receive $[M]$ from server;
\State $M \leftarrow HE.Dec(sk, [M])$;
\EndIf
\If {$t > 1$}
\State Receive $[W_{glob}]$ from server;
\State $W_i \leftarrow HE.Dec(sk, M \odot [W_{glob}]) + (1 - M) \odot [W_{glob}]$;
\EndIf
\State $W_i \leftarrow Train(W_i, D_i)$;
\Comment{Additional Differential Privacy}
\If {Add DP}
\State $W_i \leftarrow W_i + Noise(b)$;
\EndIf
\State $[W_i] \leftarrow HE.Enc(pk, M \odot W_i) + (1 - M) \odot W_i$;
\State Send $[W_i]$ to server $S$;
\EndFor

\Comment{Server Model Aggregation}
\State $[W_{glob}] \leftarrow \sum_{i=1}^{N} \alpha_i [M \odot W_i] + \sum_{i=1}^{N} \alpha_i((1 - M) \odot W_i)$;
\EndFor
\end{algorithmic}
\end{algorithm}

Our HE-based secure aggregation algorithm, as illustrated in Algorithm 1, can be summarized as: given an aggregation server and $N$ clients, each client $i \in [N]$ owns a local dataset $D_i$ and initializes a local model $W_i$ with the aggregation weighing factor $\alpha_i$; the key authority or the distributed threshold key agreement protocol generates a key pair $(pk, sk)$ and the crypto context, then distributes it to clients and server (except the server only gets the crypto context which is public configuration). The
clients and the server then collectively calculate the encryption mask $M$ for **Selective Parameter Encryption** also using homomorphic encryption. At every communication round $t \in [T]$, the server performs the aggregation

$$[W_{\text{glob}}] = \sum_{i=1}^{N} \alpha_i [M \odot W_i] + \sum_{i=1}^{N} \alpha_i ((1 - M) \odot W_i),$$

where $[W_{\text{glob}}]$ is the partially-encrypted global model, $W_i$ is the $i$-th plaintext local model where $[]$ indicates the portion of the model that is fully encrypted, $\alpha_i$ is the aggregation weight for client $i$, and $M$ is the model encryption mask.

Note that the aggregation weights can be either encrypted or in plaintext depending on whether the aggregation server is trustworthy enough to obtain that information. In our system, we set the aggregation weights to be plaintext by default. We only need one multiplicative depth of HE multiplication in our algorithm for weighting, which is preferred to reduce HE multiplication operations. Our system can also be easily extended to support more FL aggregation functions with HE by encrypting and computing the new parameters in these algorithms (e.g. FedProx Li et al. (2020)). Additionally, in Algorithm 1 optional local differential privacy noise can be easily added after local models are trained if there is an extra desire for differential privacy.

We will explain in detail how the encryption mask $M$ is formalized in §2.4.

### 2.4 Efficient Optimization by Selective Parameter Encryption

![Diagram of Selective Parameter Encryption](image-url)

**Figure 4:** **Selective Parameter Encryption**: in the initialization stage, clients first calculate privacy sensitivities on the model using its own dataset and local sensitivities will be securely aggregated to a global model privacy map. The encryption mask will be then determined by the privacy map and a set selection value $p$ per overhead requirements and privacy guarantee. Only the masked parameters will be aggregated in the encrypted form.

Fully encrypted models can guarantee no access to plaintext local models from the adversary with high overheads. However, previous work on privacy leakage analysis shows that “partial transparency”, e.g. hiding parts of the models (Hatamizadeh et al., 2022; Mo et al., 2020), can limit an adversary’s ability to successfully perform attacks like gradient inversion attacks (Lu et al., 2022). We therefore propose **Selective Parameter Encryption** to selectively encrypt the most privacy-sensitive parameters in order to reduce impractical overhead while providing customizable privacy preservation; see Figure 4.

**Step 1: Privacy Leakage Analysis on Clients.** Directly performing a gradient inversion attack Wei et al. (2020) and evaluating the success rate of the attack can take much more time than the model training. We then adopt sensitivity Novak et al. (2018); Sokolić et al. (2017); Mo et al. (2020) for measuring the general privacy risk on gradients w.r.t. input. Given model $W$ and $K$ data samples with input matrix $X$ and ground truth label vector $y$, we compute the sensitivity for each parameter $w_m$ by $\frac{1}{K} \sum_{k=1}^{K} \|J_m(y_k)\|$, where $J_m(y_k) = \frac{\partial}{\partial y_k} \left( \frac{\partial \ell(X, y, W)}{\partial w_m} \right) \in R$, $\ell(\cdot)$ is the loss function given $X$, $y$ and $W$, and $\|\|$ calculates the absolute value. The intuition is to calculate how large the gradient of the parameter will change with the true output $y_k$ for each data point $k$. Each client $i$ then sends the encrypted parameters sensitivity matrix $[S_i]$ to the server.
As shown in Figure 5, different parts of a model contribute to attacks by revealing uneven amounts of information. Using this insight, we propose to only select and encrypt parts of the model that are more important and susceptible to attacks to reduce HE overheads while preserving adequate privacy.

**Step 2: Encryption Mask Agreement across Clients.** The sensitivity map is dependent on the data it is processed on. With potentially heterogeneous data distributions, the server aggregates local sensitivity maps to a global privacy map $\sum_{i=1}^{N} \alpha_i [S_i]$. The global encryption mask $M$ is then configured using a privacy-overhead ratio $p \in [0, 1]$ which is the ratio of selecting the most sensitive parameters for encryption. The global encryption mask is then shared among clients as part of the federated learning configuration.

### 2.5 Software Framework: Homomorphic Encryption In Federated Learning

In this part, we will illustrate how we design our HE-based aggregation from a software framework perspective.

![Framework Structure](image)

Figure 6 provides a high-level design of our framework, which consists of three major layers:

- **Crypto Foundation.** The foundation layer is where Python wrappers are built to realize HE functions including key generation, encryption/decryption, secure aggregation, and ciphertext serialization using open-sourced HE libraries;
• **ML Bridge.** The bridging layer connects the FL system orchestration and cryptographic functions. Specifically, we have ML processing APIs to process inputs to HE functions from local training processes and outputs vice versa. Additionally, we realize the optimization module here to mitigate the HE overheads;

• **FL Orchestration.** The FL system layer is where the key authority server manages the key distribution and the (server/client) managers and task executors orchestrate participants.

Our layered design makes the HE crypto foundation and the optimization module *semi-independent*, allowing different HE libraries to be easily switched into FedML-HE and further FL optimization techniques to be easily added to the system.

## 3 Privacy By Selective Parameter Encryption

In this section, we first provide proof to analyze the privacy of fully encrypted federated learning and then analyze the privacy guarantee of Selective Parameter Encryption.

### 3.1 Proof of Base Protocol

In this subsection, we prove the privacy of base protocol where homomorphic-encryption-based federated learning utilizes the full model parameter encryption (i.e., the selective parameter encryption rate is set to be 1). We define the adversary in Definition 3.1 and privacy in Definition 3.3.

**Definition 3.1 (Single-Key Adversary).** A semi-honest adversary $A$ can corrupt (at the same time) any subset of $n$ learners and the aggregation server, but not at the same time.

Note that the ref of the proof assumes the single-key setup and the privacy of the threshold variant of HE-FL (as shown in Definition 3.2) can be easily proved by extending the proofs of threshold homomorphic encryption [Boneh et al. (2006); Laud & Ngo (2008); Asharov et al. (2012)].

**Definition 3.2 (Threshold Adversary).** A semi-honest adversary $A_T$ can corrupt (at the same time) any subset of $n - k$ learners and the aggregation server.

**Definition 3.3 (Privacy).** A homomorphic-encryption federated learning protocol $\pi$ is simulation secure in the presence of a semi-honest adversary $A$, there exists a simulator $S$ in the ideal world that also corrupts the same set of parties and produces an output identically distributed to $A$’s output in the real world.

**Ideal World.** Our ideal world functionality $F$ interacts with learners and the aggregation server as follows:

- Each learner sends a registration message to $F$ for a federated training model task $W_{\text{glob}}$. $F$ determines a subset $N' \subset N$ of learners whose data can be used to compute the global model $W_{\text{glob}}$.
- Both honest and corrupted learners upload their local models to $F$.
- If local models $\tilde{W}$ of learners in $N'$ are enough to compute $W_{\text{glob}}$, $F$ sends $W_{\text{glob}} \leftarrow \sum_{i=1}^{N'} \alpha_i \tilde{W}_i$ to all learners in $N'$, otherwise $F$ sends empty message $\bot$.

**Real World.** In real world, $F$ is replaced by our protocol described in Algorithm 1 with full model parameter encryption.

We describe a simulator $S$ that simulates the view of the $A$ in the real-world execution of our protocol. Our privacy definition 3.3 and the simulator $S$ prove both confidentiality and correctness. We omit the simulation of the view of $A$ that corrupts the aggregation server here since the learners will not receive the ciphertexts of other learners’ local models in the execution of $\pi$ thus such a simulation is immediate and trivial.

**Simulator.** In the ideal world, $S$ receives $\lambda$ and $1^n$ from $F$ and executes the following steps:

1. $S$ chooses a uniformly distributed random tape $r$.
2. $S$ runs the key generation function to sample $pk$: $(pk, sk) \leftarrow HE.KeyGen(\lambda)$.
3. For a chosen $i$th learner, $S$ runs the encryption function to sample: $c_i \leftarrow HE.Enc(pk, r^iW_i)$.
4. $S$ repeats Step 3 for all other learners to obtain $\bar{c}$, and runs the federated aggregation function $f$ to sample: $(c_{\text{glob}}) \leftarrow HE.Eval(\bar{c}, f)$.

The execution of $S$ implies that:

$$\{ (c_i, c_{\text{glob}}) \} \equiv \left\{ \left( HE.Enc(pk_i, W), HE.Eval(\hat{W}, f) \right) \right\}$$

Thus, we conclude that $S$’s output in the ideal world is computationally indistinguishable from the view of $A$ in a real world execution:

$$\{ S(1^n, (\lambda)) \} \equiv \{ \text{view}^\pi(\lambda) \}$$

where $\text{view}$ is the view of $A$ in the real execution of $\pi$.

3.2 Proof of Encrypted Learning by DP Theory

**Definition 3.4 (Adjacent Datasets).** Two datasets $D_1$ and $D_2$ are said to be adjacent if they differ in the data of exactly one individual. Formally, they are adjacent if:

$$|D_1 \Delta D_2| = 1$$

**Definition 3.5 ($\epsilon$-Differential Privacy).** A randomized algorithm $M$ satisfies $\epsilon$-differential privacy if for any two adjacent datasets $D_1$ and $D_2$, and for any possible output $O \subseteq \text{Range}(F)$, the following inequality holds:

$$\frac{\Pr [M(D_1) \in O]}{\Pr [M(D_2) \in O]} \leq e^\epsilon$$

Smaller values of the privacy parameter $\epsilon$ imply stronger privacy guarantees.

**Definition 3.6 (Laplace mechanism).** Given a function $f : D \rightarrow \mathbb{R}$, where $D$ is the domain of the dataset and $d$ is the dimension of the output, the Laplace mechanism adds Laplace noise to the output of $f$.

Let $b$ be the scale parameter of the Laplace distribution, which is given by:

$$\text{Lap}(x \mid b) = \frac{1}{2b} e^{-\frac{|x|}{b}}$$

Given a dataset $D$, the Laplace mechanism $F$ is defined as:

$$M(D) = f(D) + \text{Lap}(0 \mid b)^d$$

**Definition 3.7 (Sensitivity).** To ensure $\epsilon$-differential privacy, we need to determine the appropriate scale parameter $b$. This is where the sensitivity of the function $f$ comes into play. The sensitivity $\Delta f$ of a function $f$ is the maximum difference in the output of $f$ when applied to any two adjacent datasets:

$$\Delta f = \max_{D_1, D_2 : |D_1 \Delta D_2| = 1} \| f(D_1) - f(D_2) \|_1$$

Based on Definition 3.4, 3.5, 3.6 and 3.7, we have

**Lemma 3.8 (Achieving $\epsilon$-Differential Privacy by Laplace Mechanism) [Dwork (2008), Abadi et al. (2016)].** To achieve $\epsilon$-differential privacy, we choose the scale parameter $b$ as:

$$b = \frac{\Delta f}{\epsilon}$$

With this choice of $b$, the Laplace mechanism $F$ satisfies $\epsilon$-differential privacy.

By adding noise $\text{Lap}(0 \mid b)^d$ on one parameter in the model gradient where $b = \frac{\Delta f}{\epsilon}$, we can achieve $\epsilon$-differential privacy. We then show homomorphic encryption provides a much stronger differential privacy guarantee.
Theorem 3.9 (Achieving 0-Differential Privacy by Homomorphic Encryption). For any two adjacent datasets \( D_1 \) and \( D_2 \), since \( M(D) \) is computationally indistinguishable, we have

\[
\Pr [M(D_1) \in O] \leq \Pr [M(D_2) \in O] \leq e^\epsilon.
\]

We then have \( \epsilon = 0 \) if \( O \) is encrypted.

In other words, \( A \) cannot retrieve sensitive information from encrypted parameters.

3.3 Proof of Selective Parameter Selection

Lemma 3.10 (Sequential Composition [Dwork (2008)]). If \( M_1(x) \) satisfies \( \epsilon_1 \)-differential privacy and \( M_2(x) \) satisfies \( \epsilon_2 \)-differential privacy, then the mechanism \( G(x) = (M_1(x), M_2(x)) \) which releases both results satisfies \( (\epsilon_1 + \epsilon_2) \)-differential privacy.

Based on Lemma 3.8, 3.10 and Theorem 3.9, we can now analyze the privacy of Selective Parameter Encryption

Theorem 3.11 (Achieving \( \sum_{i\in[N]/S} \frac{\Delta f_i}{b} \)-Differential Privacy by Partial Encryption). If we apply Homomorphic Encryption on partial model parameters \( S \) and Laplace Mechanism on remaining model parameters \( [N]/S \) with fixed noise scale \( b \). For each parameter \( i \in [N]/S \), we have \( \epsilon_i = \frac{\Delta f_i}{b} \). Such partial encryption satisfies \( \sum_{i\in[N]/S} \frac{\Delta f_i}{b} \)-differential privacy.

Let \( J = \sum_{i=1}^N \frac{\Delta f_i}{b} \) and assume \( \Delta f \sim U(0, 1) \) where \( U \) represents the uniform distribution, we can then show the privacy cost of adding Laplace noise on all parameters, random parameter encryption, and selective parameter encryption.

Remark 3.12 (Achieving \( J \)-Differential Privacy by Laplace Mechanism on All Model Parameters). If we add Laplace noise on all parameters with fixed noise scale \( b \), it satisfies \( J \)-differential privacy.

Remark 3.13 (Achieving \( (1-p)J \)-Differential Privacy by Random Selection). If we randomly select model parameters with probability \( p \) and homomorphically encrypt the remaining parameters, it satisfies \( (1-p)J \)-differential privacy.

Remark 3.14 (Achieving \( (1-p)^2J \)-Differential Privacy by Sensitive Parameter Selection). If we select the most sensitive parameters with ratio \( p \) and homomorphically encrypt the remaining parameters, it satisfies \( (1-p)^2J \)-differential privacy.

Key Observation: Selective Parameter Encryption requires \( (1-p)^2 \) times less privacy budget than random selection and complete differential privacy with the same privacy preservation.

4 Evaluation

In this section, we focus on the evaluation results to show how our proposed universal optimization scheme largely mitigates these overheads for real-world deployment but still guarantees adequate defense against privacy attacks. Note that additional experimental results regarding other FL system aspects are included in the appendix.

4.1 Experiment Setup

Models. We test our framework on models in different ML domains with different sizes including Llama-2 (7 billion) (more details in the appendix).

HE Libraries. We implement our HE core using both PALISADE and TenSEAL. Unless otherwise specified, our results show the evaluation of the PALISADE version.

Default Crypto Parameters. Unless otherwise specified, we choose the multiplicative depth of 1, the scaling factor bit digit of 52, an HE packing batch size of 4096, and a security level of 128 as our default HE cryptographic parameters during the evaluation.

Microbenchmark. For microbenchmarking HE overheads, we use an Intel 8-core 3.60GHz i7-7700 CPU with 32 GB memory and an NVIDIA Tesla T4 GPU on Ubuntu 18.04.6.
4.2 Optimizations

To mitigate the HE overhead surge, our optimization scheme **Selective Parameter Encryption** works by selecting sensitive portions of parameters for encrypted computation while leaving the rest in plaintext per desired overhead expectations and privacy promise. In this section, we first evaluate the overhead optimization from **Selective Parameter Encryption** and then use the state-of-the-art privacy attacks to evaluate the effectiveness of our selection defense during FL training. Note that other parameter efficiency techniques [Tang et al. (2019), Hu et al. (2021)] for both training-from-scratch and fine-tuning scenarios can also be applied in our system before **Selective Parameter Encryption** and efficiently reducing the sizes of shared models directly helps with HE computation and communication efficiency (we also include preliminary results on this part in the appendix).

4.2.1 Optimized Overheads

We first examine the overhead optimization gains from **Selective Parameter Encryption**. We examine the overhead change when parameters with high privacy importance are selected and encrypted. Figure 7 shows the overhead reduction from only encrypting certain parts of models, where both overheads are nearly proportional to the size of encrypted model parameters, which is coherent with the general relationship between HE overheads and input sizes. Note that after 10% encryption per our **Selective Parameter Encryption**, the overheads are close to the ones of plaintext aggregation.

![Figure 7: Computational (up) and Computation (down) Overhead Comparison For Models of Different Sizes (logarithmic scale): 10% Encryption is based on our selection strategy and 50% encryption is based on random selection.](image)

Figure 8: Time Distribution of A Training Cycle on ResNet-50: with a single AWS region bandwidth of 200 MB/s for plaintext FL (left), HE w/o optimization (middle), and HE w/ optimization (right). Optimization setup uses **DoubleSqueeze** [Tang et al. (2019)] with $k = 1,000,000$ and encryption mask with an encrypted ratio $s = 30\%$.

![Figure 8: Time Distribution of A Training Cycle on ResNet-50](image)
4.2.2 Effectiveness of Selection Defense

To evaluate the defense effectiveness of Selective Parameter Encryption, we first use privacy sensitivity to generate a privacy map (Figure 5) and then verify the effectiveness of selection by performing gradient inversion (DLG Zhu et al. (2019)). We also provide defense results with Language Model Inversion Attacks Fowl et al. (2022) on Bert.

Defense effectiveness on CV tasks. We use image samples from CIFAR-100 to calculate the parameter sensitivities of the model. In the DLG attack experiments, we use Multi-scale Structural Similarity Index (MSSSIM), Visual Information Fidelity (VIF), and Universal Quality Image Index (UQI) as metrics to measure the similarity between recovered images and original training images to measure the attack quality hence the privacy leakage. In Figure 9 compared to random encryption selection where encrypting 42.5% of the parameters can start to protect against attacks, our top-10% encryption selection according to the model privacy map only alone can defend against the attacks, meaning lower overall overhead with the same amount of privacy protection.

![Figure 9: Selection Protection Against Gradient Inversion Attack Zhu et al. (2019) On LeNet with the CIFAR-100 Dataset: attack results when protecting top-s sensitive parameters (left) vs protecting random parameters (right). Each configuration is attacked 10 times and the best-recovered image is selected.](image)

Defense effectiveness on NLP tasks. We use language samples from wikitext dataset in our experiment. As shown in Figure 10 with our sensitivity map indicating the top 30% privacy-sensitive parameters, our encryption mask can prevent inversion attacks that yields better defense results than randomly encrypting 75% of the model parameters.

![Figure 10: Language Model Inversion Attacks Fowl et al. (2022) on Bert with the wikitext Dataset: Red indicates falsely-inverted words and Yellow indicates correctly-inverted words.](image)

Empirical Selection Recipe. Our selection strategy works by first encrypting more important model parameters. Empirically, from our experimental investigation, encrypting top-30% most sensitive parameters, as well as the first and last model layers, tends to be robust to avoid information leakage Hatamizadeh et al. (2022) and attack defense (e.g. Figure 5), which can be used as a general guideline on top of model privacy maps.

---

2The image similarity metric library used is at [https://pypi.org/project/sewar/](https://pypi.org/project/sewar/)
5 Related Work

**Existing Privacy Attacks On FL.** Threats and attacks on privacy in the domain of Federated Learning have been studied in recent years [Mothukuri et al. (2021)]. General FL privacy attacks can be categorized into two types: inference attacks [Nasr et al. (2019); Wang et al. (2019); Truex et al. (2019b) and data leakage/reconstruction [Criswell et al. (2014); Bhowmick et al. (2018); Hitaj et al. (2017)]. Attacks are usually carried out on the models to retrieve certain properties of data providers or even reconstruct the data in the training datasets. With direct access to more fine-grained local models trained on a smaller dataset [Wang et al. (2019)], the adversary can have a higher chance of a successful attack. Moreover, further attacks can be performed using GAN-based attacks to even fully recover the original data [Hitaj et al. (2017)]. The majority of the privacy attacks can be traced back to the direct exposure of plaintext accesses to local models to other parties (usually the server).

**Existing Non-HE Defense Mechanism.** Local differential privacy has been adopted to protect local model updates by adding differential noise on the client side before the server-side aggregation [Truex et al. (2019a); Byrd & Polychroniadou (2020)] where privacy guarantee requires large-scale statistical noise on fine-grained local updates that generally degrades model performance by a large margin. On the other hand, other work proposes to apply zero-sum masks (usually pair-wise) to mask local model updates such that any individual local update is indistinguishable to the server [Bonawitz et al. (2017); So et al. (2022)]. However, such a strategy introduces several challenges including key/mask synchronization requirements and federated learner dropouts. Compared to these solutions providing privacy protection in FL, HE is non-interactive and dropout-resilient (vs. general secure aggregation protocols [Bonawitz et al. (2017); So et al. (2022)]) and it introduces negligible model performance degradation (vs. noise-based differential privacy solutions [Truex et al. (2019a); Byrd & Polychroniadou (2020)]).

**Existing HE-based FL Work.** Existing HE-based FL work either apply restricted HE schemes (e.g. additive scheme Paillier) [Zhang et al. (2020); Fang & Qian (2021); Jiang et al. (2021)] without extensibility to further FL aggregation functions as well as sufficient performance and security guarantee (due to Paillier) or provide a generic HE implementation on FL aggregation [Roth et al. (2022); IBM (2022); Jiang et al. (2021); Du et al. (2023); Ma et al. (2022)]. However, previous work still leaves the HE overhead increase issue as an open question. In our work, we propose a universal optimization scheme to largely reduce the overhead while providing promised privacy guarantees in a both systematic and algorithmic fashion, which makes HE-based FL viable in practical deployments.

6 Conclusion

In this paper, we propose FedML-HE, the first practical Homomorphic Encryption-based privacy-preserving FL system that supports encryption key management, encrypted FL platform deployment, encryption optimizations to reduce overhead, and is designed to support efficient foundation model federated training. We design Selective Parameter Encryption that selectively encrypts the most privacy-sensitive parameters to minimize the size of encrypted model updates while providing customizable privacy preservation. Future work includes quantitative and theoretical analysis of the trade-offs among privacy guarantee, system overheads, and model performance compared to other approaches (including difference privacy and secure aggregation approaches), and improving threshold-HE’s performance in the FL setting as well as supporting decentralized primitives such as Proxy Re-Encryption [Ateniese et al. (2006)].
References


A Preliminaries

A.1 Federated Learning

Federated learning is first proposed in [McMahan et al. (2017)], which builds distributed machine learning models while keeping personal data on clients. Instead of uploading data to the server for centralized training, clients process their local data and share updated local models with the server. Model parameters from a large population of clients are aggregated by the server and combined to create an improved global model.

The FedAvg [McMahan et al. (2017)] is commonly used on the server to combine client updates and produce a new global model. At each round, a global model $W_{\text{glob}}$ is sent to $N$ client devices. Each client $i$ performs gradient descent on its local data with $E$ local iterations to update the model $W_i$. The server then does a weighted aggregation of the local models to obtain a new global model, $W_{\text{glob}} = \sum_{i=1}^{N} \alpha_i W_i$, where $\alpha_i$ is the weighting factor for client $i$.

Typically, the aggregation runs using plaintext model parameters through a central server (in some cases, via a decentralized protocol), giving the server visibility of each local client’s model in plaintext.

A.2 Homomorphic Encryption

| Figure 11: General Scheme of Homomorphic Encryption |

Homomorphic Encryption is a cryptographic primitive that allows computation to be performed on encrypted data without revealing the underlying plaintext. It usually serves as a foundation for privacy-preserving outsourcing computing models. HE has generally four algorithms (KeyGen, Enc, Eval, Dec) as defined in Figure [11]. The fundamental concept is to encrypt data prior to computation, perform the computation on the encrypted data without decryption, and then decrypt the resulting ciphertext to obtain the final plaintext.

Since FL model parameters are usually not integers, our method is built on the Cheon-Kim-Kim-Song (CKKS) scheme [Cheon et al. (2017)], a (leveled) HE variant that can work with approximate numbers.

B Key Management And Threshold HE

Our general system structure assumes the existence of a potentially compromised aggregation server, which performs the HE-based secure aggregation. Alongside this aggregation server, there also exists a trusted key authority server that generates and distributes HE keys and related crypto context files to authenticated parties (as described previously in Algorithm 1 in the main paper. We assume there is no collusion between these two servers.

Moreover, secure computation protocols for more decentralized settings without an aggregation server are also available using cryptographic primitives such as Threshold HE [Aloufi et al. (2021)], Multi-Key HE [Aloufi et al. (2021)], and Proxy Re-Encryption [Ateniese et al. (2006); Jin et al. (2022)]. In such settings, secure computation and decryption can be collaboratively performed across multiple parties without the need for a centralized point. We plan to introduce a more decentralized version of FedML-HE in the future. Due to the collaborative nature of such secure computation, the key management will act more as a coordination point instead of a trusted source for key generation.
The threshold variant of HE schemes is generally based on Shamir’s secret sharing [Shamir (1979)] (which is also implemented in PALISADE). Key generation/agreement and decryption processes are in an interactive fashion where each party shares partial responsibility of the task. Threshold key generation results in each party holding a share of the secret key and threshold decryption requires each party to partially decrypt the final ciphertext result and merge to get the final plaintext result. We provide benchmarkings of the threshold-HE-based FedAvg implementation in Figure 12.

C Framework APIs and Platform Deployment

C.1 Framework APIs

Table 3 shows the framework APIs in our system related to HE.

<table>
<thead>
<tr>
<th>API Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( pk, sk = \text{key-gen}(\text{params}) )</td>
<td>Generate a pair of HE keys (public key and private key)</td>
</tr>
<tr>
<td>( 1d_local_model = \text{flatten}(\text{local}_model) )</td>
<td>Flatten local trained model tensors into a 1D local model</td>
</tr>
<tr>
<td>( \text{enc}_local_model = \text{enc}(pk, 1d_model) )</td>
<td>Encrypt the 1D model</td>
</tr>
<tr>
<td>( \text{enc}_global_model = \text{he_aggregate}(\text{enc}_models[n], \text{weight_factors}[n]) )</td>
<td>Homomorphically aggregate a list of 1D local models</td>
</tr>
<tr>
<td>( \text{dec}_global_model = \text{dec}(sk, \text{enc}_global_model) )</td>
<td>Decrypt the 1D global model</td>
</tr>
<tr>
<td>( \text{global}_model = \text{reshape}(\text{dec}_global_model, \text{model}_shape) )</td>
<td>Reshape the 1D global model back to the original shape</td>
</tr>
</tbody>
</table>

Table 3: HE Framework APIs

C.2 Deploy Anywhere: A Deployment Platform MLOps For Edges/Cloud

We implement our deployment-friendly platform such that FedML-HE can be easily deployed across cloud and edge devices. Before the training starts, a user uploads the configured server package and the local client package to the web platform. The server package defines the operations on the FL server, such as the aggregation function and client sampling function; the local client package defines the customized model architecture to be trained (model files will be distributed to edge devices in the first round of the training). Both packages are written in Python. The platform then builds and runs the docker image with the uploaded server package to operate as the server for the training with edge devices configured using the client package.

As shown in Figure 13, during the training, users can also keep tracking the learning procedure including device status, training progress/model performance, and FedML-HE system overheads (e.g., training time, communication time, CPU/GPU utilization, and memory utilization) via the web.
interface. Our platform keeps close track of overheads, which allows users to in real-time pinpoint HE overhead bottlenecks if any.

Figure 13: Deployment Interface Example of FedML-HE: Overhead distribution monitoring on each edge device (e.g. Desktop (Ubuntu), Laptop (MacBook), and Raspberry Pi 4), which can be used to pinpoint HE overhead bottlenecks and guide optimization.

### D Additional Experiments

<table>
<thead>
<tr>
<th>Model</th>
<th>Model Size</th>
<th>HE (s)</th>
<th>Non-HE (s)</th>
<th>Comp Ratio</th>
<th>Ciphertext</th>
<th>Plaintext</th>
<th>Comm Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Model</td>
<td>101</td>
<td>0.216</td>
<td>0.001</td>
<td>150.85</td>
<td>266.00 KB</td>
<td>1.10 KB</td>
<td>240.83</td>
</tr>
<tr>
<td>TimeSeries</td>
<td>5,609</td>
<td>2.792</td>
<td>0.233</td>
<td>12.00</td>
<td>532.00 KB</td>
<td>52.65 KB</td>
<td>10.10</td>
</tr>
<tr>
<td>Transformer</td>
<td>79,510</td>
<td>0.586</td>
<td>0.010</td>
<td>60.46</td>
<td>5.20 MB</td>
<td>311.98 KB</td>
<td>17.05</td>
</tr>
<tr>
<td>MLP (2 FC)</td>
<td>88,648</td>
<td>0.619</td>
<td>0.011</td>
<td>57.95</td>
<td>5.97 MB</td>
<td>349.52 KB</td>
<td>17.50</td>
</tr>
<tr>
<td>RNN (2 LSTM + 1 FC)</td>
<td>822,570</td>
<td>1.195</td>
<td>0.013</td>
<td>91.82</td>
<td>52.47 MB</td>
<td>3.14 MB</td>
<td>16.70</td>
</tr>
<tr>
<td>CNN (2 Conv + 2 FC)</td>
<td>1,663,370</td>
<td>2.456</td>
<td>0.058</td>
<td>42.23</td>
<td>103.15 MB</td>
<td>6.35 MB</td>
<td>16.66</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>12,556,426</td>
<td>19.950</td>
<td>1.100</td>
<td>18.14</td>
<td>796.70 MB</td>
<td>47.98 MB</td>
<td>16.61</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>21,797,672</td>
<td>37.555</td>
<td>2.925</td>
<td>12.84</td>
<td>1.35 GB</td>
<td>83.28 MB</td>
<td>16.60</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>25,557,032</td>
<td>46.672</td>
<td>5.379</td>
<td>8.68</td>
<td>1.58 GB</td>
<td>97.79 MB</td>
<td>16.58</td>
</tr>
<tr>
<td>GroupViT</td>
<td>55,726,609</td>
<td>86.098</td>
<td>19.921</td>
<td>4.32</td>
<td>3.45 GB</td>
<td>212.83 MB</td>
<td>16.61</td>
</tr>
<tr>
<td>Vision Transformer</td>
<td>86,389,248</td>
<td>112.504</td>
<td>17.739</td>
<td>6.34</td>
<td>5.35 GB</td>
<td>329.62 MB</td>
<td>16.62</td>
</tr>
<tr>
<td>BERT</td>
<td>109,482,240</td>
<td>136.914</td>
<td>19.674</td>
<td>6.96</td>
<td>6.78 GB</td>
<td>417.72 MB</td>
<td>16.62</td>
</tr>
<tr>
<td>Llama 2</td>
<td>6.74 B</td>
<td>13067.154</td>
<td>2423.976</td>
<td>5.39</td>
<td>417.43 GB</td>
<td>13.5 GB</td>
<td>30.92</td>
</tr>
</tbody>
</table>

Table 4: Vanilla Fully-Encrypted Models of Different Sizes: with 3 clients; Comp Ratio is calculated by time costs of HE over time costs of Non-HE; Comm Ratio is calculated by file sizes of HE over file sizes of Non-HE. CKKS is configured with default crypto parameters.

We evaluate the HE-based training overheads (without our optimization in place) across various FL training scenarios and configurations. This analysis covers diverse model scales, HE cryptographic parameter configurations, client quantities involved in the task, and communication bandwidths. This helps us to identify bottlenecks in the HE process throughout the entire training cycle. We also benchmark our framework against other open-source HE solutions to demonstrate its advantages.
D.1 Parameter Efficiency Techniques in HE-Based FL

Table 5 shows the optimization gains by applying model parameter efficiency solutions in HE-Based FL.

<table>
<thead>
<tr>
<th>Models</th>
<th>PT (MB)</th>
<th>CT (MB)</th>
<th>Opt (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18 (12 M)</td>
<td>47.98</td>
<td>796.70 MB</td>
<td>19.03</td>
</tr>
<tr>
<td>BERT (110 M)</td>
<td>417.72</td>
<td>6.78 GB</td>
<td>16.66</td>
</tr>
</tbody>
</table>

Table 5: Parameter Efficiency Overhead: PT means plaintext and CT means ciphertext. Communication reductions are 0.60 and 0.96.

D.2 Results on Different Scales of Models

We evaluate our framework on models with different size scales and different domains, from small models like the linear model to large foundation models such as Vision Transformer [Dosovitskiy et al. (2020)] and BERT [Devlin et al. (2018)]. As Table 4 shows, both computational and communicational overheads are generally proportional to model sizes.

Table 4 illustrates more clearly the overhead increase from the plaintext federated aggregation. The computation fold ratio is in general 5x ~ 20x while the communication overhead can jump to a common 15x. Small models tend to have a higher computational overhead ratio increase. This is mainly due to the standard HE initialization process, which plays a more significant role when compared to the plaintext cost. The communication cost increase is significant for models with sizes smaller than 4096 (the packing batch size) numbers. Recall that the way our HE core packs encrypted numbers makes an array whose size is smaller than the packing batch size still requires a full ciphertext.

<table>
<thead>
<tr>
<th>HE Batch Size</th>
<th>Scaling Bits</th>
<th>Comp (s)</th>
<th>Comm (MB)</th>
<th>Model Test Accuracy ∆ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>14</td>
<td>8.834</td>
<td>407.47</td>
<td>-0.28</td>
</tr>
<tr>
<td>1024</td>
<td>20</td>
<td>7.524</td>
<td>407.47</td>
<td>-0.21</td>
</tr>
<tr>
<td>1024</td>
<td>33</td>
<td>7.536</td>
<td>407.47</td>
<td>0</td>
</tr>
<tr>
<td>1024</td>
<td>40</td>
<td>7.765</td>
<td>407.47</td>
<td>0</td>
</tr>
<tr>
<td>1024</td>
<td>52</td>
<td>7.827</td>
<td>407.47</td>
<td>0</td>
</tr>
<tr>
<td>2048</td>
<td>14</td>
<td>3.449</td>
<td>204.50</td>
<td>-0.06</td>
</tr>
<tr>
<td>2048</td>
<td>20</td>
<td>3.414</td>
<td>204.50</td>
<td>-0.13</td>
</tr>
<tr>
<td>2048</td>
<td>33</td>
<td>3.499</td>
<td>204.50</td>
<td>0</td>
</tr>
<tr>
<td>2048</td>
<td>40</td>
<td>3.621</td>
<td>204.50</td>
<td>0</td>
</tr>
<tr>
<td>2048</td>
<td>52</td>
<td>3.676</td>
<td>204.50</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>14</td>
<td>1.837</td>
<td>103.15</td>
<td>-1.85</td>
</tr>
<tr>
<td>4096</td>
<td>20</td>
<td>1.819</td>
<td>103.15</td>
<td>0.32</td>
</tr>
<tr>
<td>4096</td>
<td>33</td>
<td>1.886</td>
<td>103.15</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>40</td>
<td>1.998</td>
<td>103.15</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>52</td>
<td>1.926</td>
<td>103.15</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6: Computational & Communicational Overhead of Different Crypto Parameter Setups: tested with CNN (2 Conv+ 2 FC) and on 3 clients; model test accuracy ∆s is the difference between the best plaintext global model and the best global encrypted global models.
D.3 Results on Different Cryptographic Parameters

We evaluate the impacts of variously-configured cryptographic parameters. We primarily look into the packing batch size and the scaling bits. The packing batch size determines the number of slots packed in a single ciphertext while the scaling bit number affects the “accuracy” (i.e., how close the decrypted ciphertext result is to the plaintext result) of approximate numbers represented from integers.

From Table 6, the large packing batch sizes in general result in faster computation speeds and smaller overall ciphertext files attributed to the packing mechanism for more efficiency. However, the scaling factor number has an almost negligible impact on overheads.

Unsurprisingly, it aligns with the intuition that the higher bit scaling number results in higher “accuracy” of the decrypted ciphertext value, which generally means the encrypted aggregated model would have a close model test performance to the plaintext aggregated model. However, it is worth mentioning that since CKKS is an approximate scheme with noises, the decrypted aggregated model can yield either positive or negative model test accuracy $\Delta s$, but usually with a negative or nearly zero $\Delta$.

D.4 Impact from Number of Clients

As real-world systems often experience a dynamic amount of participants within the FL system, we evaluate the overhead shift over the change in the number of clients. Figure 14a breaks down the cost distribution as the number of clients increases. With a growing number of clients, it also means proportionally-added ciphertexts as inputs to the secure aggregation function thus the major impact is cast on the server. When the server is overloaded, our system also supports client selection to remove certain clients without largely degrading model performance.

![Figure 14a](image)

(a) Step Breakdown of HE Computational Cost vs. Number of Clients (Up to 200): tested on fully-encrypted CNN

![Figure 14b](image)

(b) Impact of Different Bandwidths on Communication and Training Cycles on Fully-Encrypted ResNet-50: HE means HE-enabled training and Non means plaintext. Others include all other procedures except communication during training. Percentages represent the portion of communication cost in the entire training cycle.

Figure 14: Results on Different Number of Clients and Communication Setup

D.5 Communication Cost on Different Bandwidths

FL parties can be allocated in different geo-locations which might result in communication bottlenecks. Typically, there are two common scenarios: (inter) data centers and (intra) data centers. In this part, we evaluate the impact of the bandwidths on communication costs and how it affects the FL training cycle. We categorize communication bandwidths using 3 cases:
- Infiniband (IB): communication between intra-center parties. 5 GB/s as the test bandwidth.
- Single AWS Region (SAR): communication between inter-center parties but within the same geo-region (within US-WEST). 592 MB/s as the test bandwidth.
- Multiple AWS Region (MAR): communication between inter-center parties but across the different geo-region (between US-WEST and EU-NORTH). 15.6 MB/s as the test bandwidth.

As shown in Figure 14b, we deploy FedML-HE on 3 different geo-distributed environments, which are operated under different bandwidths. It is obvious that the secure HE functionality has an enormous impact on low-bandwidth environments while medium-to-high-bandwidth environments suffer limited impact from increased communication overhead during training cycles, compared to Non-HE settings.

D.6 Different Encryption Selections

Table 7 shows the overhead reductions with different selective encryption rates.

<table>
<thead>
<tr>
<th>Selection</th>
<th>Comp (s)</th>
<th>Comm</th>
<th>Comp Ratio</th>
<th>Comm Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enc w/ 0%</td>
<td>17.739</td>
<td>329.62 MB</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Enc w/ 10%</td>
<td>30.874</td>
<td>844.49 MB</td>
<td>1.74</td>
<td>2.56</td>
</tr>
<tr>
<td>Enc w/ 30%</td>
<td>50.284</td>
<td>1.83 GB</td>
<td>2.83</td>
<td>5.69</td>
</tr>
<tr>
<td>Enc w/ 50%</td>
<td>70.167</td>
<td>2.83 GB</td>
<td>3.96</td>
<td>8.81</td>
</tr>
<tr>
<td>Enc w/ 70%</td>
<td>88.904</td>
<td>3.84 GB</td>
<td>5.01</td>
<td>11.93</td>
</tr>
<tr>
<td>Enc w/ All</td>
<td>112.504</td>
<td>5.35 GB</td>
<td>6.34</td>
<td>16.62</td>
</tr>
</tbody>
</table>

Table 7: Overheads With Different Parameter Selection Configs Tested on Vision Transformer: “Enc w/ 10%” means performs encrypted computation only on 10% of the parameters; all computation and communication results include overheads from plaintext aggregation for the rest of the parameters.

D.7 Comparison with Other FL-HE Frameworks

We compare our framework to the other open-sourced FL frameworks with HE capability, namely NVIDIA FLARE (NVIDIA) and IBMFL.

Both NVIDIA and IBMFL utilize Microsoft SEAL as the underlying HE core, with NVIDIA using OpenMinded’s python tensor wrapper over SEAL and TenSEAL; IBMFL using IBM’s python wrapper over SEAL and HELayers (HELayers also has an HElib version). Our HE core module can be replaced

<table>
<thead>
<tr>
<th>Frameworks</th>
<th>HE Core (S)</th>
<th>Key Management</th>
<th>Comp (s)</th>
<th>Comm (MB)</th>
<th>HE Multi-Party Functionals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>PALISADE</td>
<td>✔️</td>
<td>2.456</td>
<td>105.72</td>
<td>PRE, ThHE</td>
</tr>
<tr>
<td>Ours (w/ Opt)</td>
<td>PALISADE</td>
<td>✔️</td>
<td>0.874</td>
<td>16.37</td>
<td>PRE, ThHE</td>
</tr>
<tr>
<td>Ours</td>
<td>SEAL</td>
<td>✔️</td>
<td>3.989</td>
<td>129.75</td>
<td></td>
</tr>
<tr>
<td>Nvidia FLARE (9a1b226)</td>
<td>SEAL</td>
<td>✔️</td>
<td>2.826</td>
<td>129.75</td>
<td></td>
</tr>
<tr>
<td>IBMFL (8c8ab11)</td>
<td>SEAL</td>
<td></td>
<td>3.955</td>
<td>86.58</td>
<td></td>
</tr>
<tr>
<td>Plaintext</td>
<td></td>
<td></td>
<td>0.058</td>
<td>6.35</td>
<td></td>
</tr>
</tbody>
</table>

Table 8: Different Frameworks: tested with CNN (2 Conv + 2 FC) and on 3 clients; Github commit IDs are specified. For key management, our work uses a key authority server; FLARE uses a security content manager; IBMFL currently provides a local simulator.
with different available HE cores, to give a more comprehensive comparison, we also implement a TenSEAL version of our framework for evaluation.

Table 8 demonstrates the performance summary of different FedML-HE frameworks using an example of a CNN model with 3 clients. Our PALISADE-powered framework has the smallest computational overhead due to the performance of the PALISADE library. In terms of communication cost, FedML-HE (PALISADE) comes second after IBMFL’s smallest file serialization results due to the efficient packing of HELayers’ Tile tensors [Aharoni et al., 2011].

Note that NVIDIA’s TenSEAL-based realization is faster than the TenSEAL variant of our system. This is because NVIDIA scales each learner’s local model parameters locally rather than weighing ciphertexts on the server. This approach reduces the need for the one multiplication operation usually performed during secure aggregation (recall that HE multiplications are expensive). However, such a setup would not suit the scenario where the central server does not want to reveal its weighing mechanism per each individual local model to learners as it reveals partial (even full in some cases) information about participants in the system.