SUPPLEMENTARY MATERIAL FOR SecureHDC-FL: ADDRESSING DATA HETEROGENEITY IN ENCRYPTED FEDERATED HYPERDIMENSIONAL COMPUTING

Xiangyu Hu[1,2,3], *Jingwei Chen*[1,2,3], *Wenyuan Wu*[1,2,3], *Yong Feng*[1,2,3]

[1] Chongqing Institute of Green and Intelligent Technology, CAS
[2] Chongqing School, University of Chinese Academy of Sciences
[3] Chongqing Key Laboratory of Secure Computing for Biology
{huxiangyu, chenjingwei, wuwenyuan, yongfeng}@cigit.ac.cn

1. BACKGROUND

In this section, we introduce the fundamental concepts of Federated Learning (FL), Hyperdimensional Computing (HDC), and Homomorphic Encryption (HE), which play an important role throughout this paper.

1.1. Federated Learning

Federated Learning (FL) is a decentralized machine learning paradigm that enables multiple data owners to collaboratively train a shared model without exposing raw data [1, 2]. FL maintains data locality and mitigates privacy risks associated with centralized data storage by performing training on each client device and exchanging only model updates (e.g., gradients or parameters). A typical FL training round consists of three stages [3]:

(1) each client trains a local model on its private data and computes updates;

(2) the server aggregates these updates, commonly using Federated Averaging, to form a new global model;

(3) the updated global model is distributed back to the clients for the next round of training.

Although FL aims to protect privacy, it remains vulnerable to various inference attacks, especially in the presence of semi-honest or malicious adversaries [4]. These limitations motivate the incorporation of cryptographic techniques such as homomorphic encryption into FL.

1.2. Hyperdimensional Computing

Hyperdimensional Computing (HDC) is a brain-inspired computational paradigm that uses high-dimensional holographic hypervectors to represent and process information[5]. The core idea is to map data into a very high-dimensional space (often with thousands of dimensions or more) and perform operations on vectors (e.g., addition, multiplication, and binding) to carry out computation. In supervised classification tasks, HDC encodes input data into hypervectors, learns prototype hypervectors for each class during training, and classifies new data by comparing the similarity (e.g., cosine similarity) between the query hypervector and the class prototypes[6]. The model performance can be further improved through retraining. Major advantages of HDC include computational efficiency, robustness to noise, suitability for low-power hardware (e.g., edge devices), competitiveness with deep learning in many tasks, as well as interpretability and ease of update.

Given a labeled dataset $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathcal{X} \subseteq \mathbb{R}^n$ and $\mathbf{y}_i \in \{\mathbf{c}_k\}_{k=1}^{K}$ denotes the class label, the standard HDC workflow typically comprises four key stages[6]: encoding, training, inference, and retraining.

Alternatively, given a labeled dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=0}^{N_{\text{data}}-1}$, where $x_i \in \mathbb{R}^n$ and $y_i \in \{c_k\}_{k=0}^{K-1}$, the HDC pipeline still follows these four stages[6]: encoding, training, inference, and retraining.

Encoding: The goal of encoding is to map each sample $\mathbf{x}_i$ into a hypervector $\phi(\mathbf{x}_i) \in \mathbb{R}^d$, where d is the dimensionality of the encoded hypervector. In practice, the following two encoding methods are commonly used, and are also adopted in this paper.

Random Projection Encoding[7]:

$$h_j = \operatorname{sign}(\mathbf{\Phi}_j \cdot \mathbf{x}_i), \quad j = 1, 2, \ldots, d \tag{1}$$

where $\cdot$ denotes the inner product and $\operatorname{sign}(\cdot)$ is the sign function. This results in a hypervector representation $\phi(\mathbf{x}_i) \in \mathcal{H} \subseteq \{-1, 1\}^d$ for each sample $\mathbf{x}_i$.

Nonlinear Encoding[8]: Sample row vectors $\mathbf{b}_j \in \mathbb{R}^n$, $j = 1, 2, \ldots, d$, from a Gaussian distribution $\mathcal{N}(0, 1)$ to construct a projection matrix $\boldsymbol{B} \in \mathbb{R}^{d \times n}$. For each j, also generate a random offset $\beta_j \in [0, 2\pi]$ from a uniform distribution. For a sample $\mathbf{x}_i$, the encoded hypervector $\phi(\mathbf{x}_i) = (h_1, h_2, \ldots h_d)$ is computed as:

$$h_j = \cos(\mathbf{b}_j \cdot \mathbf{x}_i + \beta_j), \quad j = 1, 2, \ldots, d \tag{2}$$

Thus, the hypervector representation is $\phi(\mathbf{x}_i) \in \mathcal{H} \subseteq [-1, 1]^d$.

Corresponding author.

Training: After encoding, training is performed by accumulating hypervectors belonging to the same class. Specifically, each class $\mathbf{c}_k$ is represented by a class hypervector $\phi(\mathbf{c}_k)$, obtained by aggregating all hypervectors of samples with label $\mathbf{y}_i = \mathbf{c}_k$:

$$\phi(\mathbf{c}_k) = \bigoplus_{i:\mathbf{y}_i=\mathbf{c}_k} \phi(\mathbf{x}_i) \tag{3}$$

where $\bigoplus$ denotes the accumulation operation in HDC, typically vector addition.

Inference: In the inference stage, the goal is to classify a query sample $\mathbf{x}_q$. First, encode $\mathbf{x}_q$ into a query hypervector $\phi(\mathbf{x}_q)$ using the same encoding method as in training. Then, determine its class by computing similarity with each class hypervector:

$$k^\star = \underset{k\in\{1,\ldots,K\}}{\operatorname{argmax}} \ \delta(\phi(\mathbf{x}_q), \phi(\mathbf{c}_k)) \tag{4}$$

The similarity measure $\delta(\cdot,\cdot)$ quantifies the closeness between two hypervectors. For example, cosine similarity is commonly used:

$$\delta(\phi(\mathbf{x}_q), \phi(\mathbf{c}_k)) = \frac{\phi(\mathbf{x}_q)\cdot\phi(\mathbf{c}_k)}{\|\phi(\mathbf{x}_q)\|\|\phi(\mathbf{c}_k)\|} \tag{5}$$

Retraining: Retraining is used to improve classification accuracy by dynamically adjusting class hypervectors. The idea is to strengthen the representation of the true class and weaken that of the misclassified class when misclassification occurs. Suppose a training sample $\mathbf{x}_i$ with true label $\mathbf{c}_k$ is misclassified as $\mathbf{c}_p$:

$$p = \arg\max_{l} \delta\left(\phi(\mathbf{c}_l), \phi(\mathbf{x}_i)\right) \tag{6}$$

where $\delta(\cdot,\cdot)$ measures hypervector similarity. To correct the error, update both the true class $\mathbf{c}_k$ and the predicted class $\mathbf{c}_p$ with a learning rate r as follows[9]:

$$\begin{aligned}\phi(\mathbf{c}_k) &\leftarrow \phi(\mathbf{c}_k) + r\cdot\left(1-\delta\left(\phi(\mathbf{c}_k),\phi(\mathbf{x}_i)\right)\right)\cdot\phi(\mathbf{x}_i),\\ \phi(\mathbf{c}_p) &\leftarrow \phi(\mathbf{c}_p) - r\cdot\left(1-\delta\left(\phi(\mathbf{c}_p),\phi(\mathbf{x}_i)\right)\right)\cdot\phi(\mathbf{x}_i).\end{aligned} \tag{7}$$

1.3. CKKS Scheme

Fully Homomorphic Encryption (FHE) is an encryption technique that allows computations to be performed directly on ciphertexts without decrypting the underlying data, thereby becoming a key technology for privacy-preserving computation [10].

Currently, most FHE schemes are based on the Learning with Errors (LWE) or Ring Learning with Errors (RLWE) problem, and they can be broadly divided into two categories: SIMD-style and single-value style. These two types differ significantly in how data is organized and in their computational characteristics.

- SIMD-style (BGV [11], BFV [12], CKKS [13]): These schemes support vectorized encryption, meaning that a single ciphertext can encrypt multiple data elements simultaneously while supporting element-wise operations (e.g., addition and multiplication). Among them, BGV and BFV are primarily designed for integer computations, while CKKS supports approximate computations and can efficiently handle floating-point numbers, making it particularly suitable for privacy-preserving machine learning and data analytics.
- Single-value style (FHEW [14], TFHE [15]): These schemes are mainly designed for Boolean or integer computations, where each ciphertext encrypts only a single value. While they enable efficient evaluations of logic circuits, they incur higher storage and computational overhead when processing high-dimensional data.

Since this work involves high-dimensional vector and matrix operations on ciphertexts, as well as support for floating-point arithmetic, we choose the CKKS scheme [13] as our FHE implementation. CKKS not only supports SIMD operations and approximate real (or complex) arithmetic, but also enables efficient real-number computations, thus reducing both computational and communication overhead while preserving privacy.

Ciphertext Structure: In CKKS, a ciphertext is represented as a pair $(b, a) \in R_Q^2$, where R is the polynomial ring $R_Q = \mathbb{Z}[X]/\langle X^N+1, Q\rangle$. Here, N denotes the *dimension* of the ring, and Q denotes the *ciphertext modulus*. The semantic security of CKKS is based on the RLWE assumption [16].

CKKS allows packing up to $\ell = N/2$ complex (or real) elements into a single ciphertext and supports SIMD-style homomorphic operations over them.

For $x = (x_i)_{0\le i<\ell}$ and $y = (y_i)_{0\le i<\ell}$, let $\mathsf{ct}.x$ and $\mathsf{ct}.y$ denote ciphertexts obtained by encrypting x and y under the same public key using CKKS. In addition to encryption (Enc) and decryption (Dec) as in conventional public-key encryption, CKKS supports several basic ciphertext operations, such as addition (Add), ciphertext-ciphertext multiplication (Mul), plaintext-ciphertext multiplication (CMul), and rotation (Rot).

- $\mathsf{Add}(\mathsf{ct}.x, \mathsf{ct}.y)$: $\mathsf{Dec}(\mathsf{Add}(\mathsf{ct}.x, \mathsf{ct}.y)) \approx x + y$.
- $\mathsf{Mul}(\mathsf{ct}.x, \mathsf{ct}.y)$: $\mathsf{Dec}(\mathsf{Mul}(\mathsf{ct}.x, \mathsf{ct}.y)) \approx x \circ y$, where $\circ$ denotes element-wise multiplication.
- $\mathsf{CMul}(m, \mathsf{ct}.x)$: $\mathsf{Dec}(\mathsf{CMul}(m, \mathsf{ct}.x)) \approx m \circ x$, where m is a message vector, with $m \in \mathbb{C}^\ell$.
- $\mathsf{Select}_{[i,j]}(\mathsf{ct}.x)$: transforms $\mathsf{ct}.x = \mathsf{Enc}(x_0, \ldots, x_{\ell-1})$ into a ciphertext encrypting $(\mathbf{0}, x_i, x_{i+1}, \ldots, x_j, \mathbf{0})$. This is essentially equivalent to $\mathsf{CMul}(\mathbf{m}, \mathsf{ct}.x)$, where $\mathbf{m} = (\mathbf{0} \in \mathbb{Z}^{i-1}, \mathbf{1} \in \mathbb{Z}^{j-i+1}, \mathbf{0} \in \mathbb{Z}^{\ell-j-1})$.

- $\mathsf{Rot}_k(\mathsf{ct}.x)$: transforms $\mathsf{ct}.x = \mathsf{Enc}(x_0, \ldots, x_{\ell-1})$ into a new ciphertext encrypting $(x_k, \ldots, x_{\ell-1}, x_0, \ldots, x_{k-1})$.

Leveled CKKS Scheme: To improve efficiency, CKKS employs the Residue Number System (RNS) to decompose the modulus Q into a product of smaller primes q_i: $Q_L = q_0 q_1 \ldots q_L$. This decomposition enables computations to be performed within smaller number ranges, thereby reducing the overhead of large integer arithmetic. Each homomorphic multiplication consumes one computation level, gradually decreasing the ciphertext modulus Q_L. When the modulus becomes too small to support further multiplications, a bootstrapping procedure can be applied to restore computational capacity.

2. DYNAMIC WEIGHTED AGGREGATION IN PLAINTEXT

Algorithm 1 (Dynamic Weighted Aggregation in Plaintext)

```
Input: Global model $G^{(t)} = (G_0^{(t)}, \ldots, G_{K-1}^{(t)}) \in \mathbb{R}^{d \times K}$, local model $\{L_i^{(t)}\}_{0 \le i < M}$ with each $L_i^{(t)} = (L_{i,0}^{(t)}, \ldots, L_{i,K-1}^{(t)}) \in \mathbb{R}^{d \times K}$, the number of samples of each client $\{n_i^{(t)}\}_{0 \le i < M}$, and two hyperparameters $\alpha$ and $\beta$.
Output: Updated global model $G^{(t+1)} = (G_0^{(t+1)}, \ldots, G_{K-1}^{(t+1)}) \in \mathbb{R}^{d \times K}$.
1:  for $j \in \{0, 1, \ldots, K-1\}$ do    ▷ for each class
2:      for $i \in \{0, 1, \ldots, M-1\}$ do    ▷ for each client
3:          $S_{i,j}^{(t)} \leftarrow \cos(L_{i,j}^{(t)}, G_j^{(t)})$ ▷ $S^{(t)} = (S_{i,j}^{(t)}) \in \mathbb{R}^{M \times K}$ is the similarity matrix.
4:      $C_j^{(t)} \leftarrow (L_{0,j}^{(t)}, L_{1,j}^{(t)}, \ldots, L_{M-1,j}^{(t)}) \in \mathbb{R}^{d \times M}$
        ▷ $C_j^{(t)}$ is a collection of the $j$-th category hypervectors from all clients.
5:  $S^{(t)} \leftarrow \mathrm{softmax}(S^{(t)})$    ▷ Act on the columns of $S^{(t)}$.
6:  $N^{(t)} \leftarrow \sum_{0 \le i < M} n_i^{(t)}$
7:  $n_{\text{norm}} \leftarrow \frac{1}{N^{(t)}} \cdot (n_0^{(t)}, \ldots, n_{M-1}^{(t)}) \in \mathbb{R}^{M \times 1}$ ▷ $n_{\text{norm}}$ is for data quantity weights.
8:  $N_{\text{norm}} \leftarrow (n_{\text{norm}}, \ldots, n_{\text{norm}}) \in \mathbb{R}^{M \times K}$
9:  $W \leftarrow \alpha \cdot N_{\text{norm}} + (1-\alpha) \cdot S^{(t)} \in \mathbb{R}^{M \times K}$
10: for $j \in \{0, 1, \ldots, K-1\}$ do
11:     $G_j^{(t+1)} \leftarrow C_j^{(t)} \cdot W_j$ ▷ $W_j \in \mathbb{R}^M$ is the $j$-th column of $W$
12: $G^{(t+1)} \leftarrow \beta \cdot G^{(t+1)} + (1-\beta) \cdot G^{(t)}$
13: return $G^{(t+1)}$
```

In Algorithm 1, we measure consistency between local client parameters and global model parameters using cosine similarity (Step 3), which is normalized via softmax (Step 5) to mitigate heterogeneity effects. The resulting $S^{(t)} = (S_{i,j}^{(t)})$ are the *similarity weights*. Additionally, we compute *data quantity weights* (Step 7) to balance contributions from each client. A tunable hyperparameter α is introduced in Step 9 to control the relative importance of these two weighting factors. Finally, the server aggregates local models through a weighted average using these combined weights, incorporating an EMA update (the global model from the previous round) [17] to reduce oscillations and prevent instability. The global model G evolves dynamically across rounds, with corresponding updates in similarity scores S and aggregation weights W. We refer to this process as *dynamic weighted aggregation*.

In *dynamic weighted aggregation*, α is used to balance the similarity weight and the data volume weight, while β controls the strength of the update for the global model. We perform a grid search over α and β within the interval $[0, 1]$, and select the optimal values based on validation performance. This procedure is consistent with the common practice of hyperparameter tuning in machine learning.

3. THEORETICAL ANALYSIS

3.1. HDC Federated Learning optimization framework

The training of the HDC model is equivalent to minimizing the empirical risk with a loss function, where optimization can be performed using SGD or its variants[18]. In the federated learning scenario, training data is distributed across M clients, and each client m can only access its local dataset $\mathcal{D}_m$. Denote the set of samples from client m belonging to class k as

$$\mathcal{D}_{m,k} = \{(\boldsymbol{x}_{m,k,j}, y_{m,k,j})\}_{j=1}^{n_{m,k}} \tag{8}$$

and define its local empirical risk as

$$F_{m,k}(\boldsymbol{\theta}) = \frac{1}{n_{m,k}} \sum_{j=1}^{n_{m,k}} \mathcal{L}\left(\boldsymbol{\theta}; (\boldsymbol{x}_{m,k,j}, y_{m,k,j})\right) \tag{9}$$

where

$$\mathcal{L}(\boldsymbol{\theta}; (\boldsymbol{x}_{m,k,j}, y_{m,k,j})) = \max\left(0, -y_{m,k,j}, \boldsymbol{\theta}^\top \boldsymbol{x}_{m,k,j}\right) \tag{10}$$

is the loss function and $\boldsymbol{\theta}$ denotes the model parameters. The global optimization objective aggregates the local losses from all clients and classes as

$$F(\boldsymbol{\theta}) = \sum_{m=1}^{M} \sum_{k=1}^{K} w_{m,k} F_{m,k}(\boldsymbol{\theta}) \tag{11}$$

where $w_{m,k} \ge 0$ is the aggregation weight for client m and class k, subject to

$$\sum_{m=1}^{M} \sum_{k=1}^{K} w_{m,k} = 1 \tag{12}$$

The goal is to minimize the weighted empirical risk, formulated as

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) \tag{13}$$

The aggregation method proposed in this paper defines the aggregation weight as:

$$w_{m,k} = \alpha p_{m,k} + (1-\alpha)s_{m,k} \tag{14}$$

where $\alpha \in (0,1)$ is a trade-off parameter. Here, $p_{m,k}$ denotes the normalized data size of client m for class k, and $s_{m,k}$, as defined in the algorithm 1, measures the similarity between the local and global model parameters for this class. For simplicity, we merge the client-class pair (m,k) into a single index $i = 1, \dots, N$, where $N = MK$, and rewrite the global objective as:

$$F(\boldsymbol{\theta}) = \sum_{i=1}^{N} w_i F_i(\boldsymbol{\theta}) \tag{15}$$

In addition, to enhance convergence stability and suppress oscillations, an EMA mechanism is further introduced after aggregation. The update formula is

$$G^{(t+1)} = \beta G^{(t+1)}_{\text{agg}} + (1-\beta)G^{(t)} \tag{16}$$

where $G^{(t+1)}_{\text{agg}}$ is the model obtained from the current round of weighted aggregation, and $\beta \in (0,1)$ is the EMA coefficient.

3.2. Theoretical Assumptions

Assume that each local objective function $F_k(\cdot)$ satisfies the following standard conditions:

- L-Lipschitz smoothness (L-smooth): There exists a constant $L > 0$ such that for any $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$, we have $\|\nabla F_k(u) - \nabla F_k(v)\| \le L\|u - v\|$.
- μ-strong convexity (μ-strong convex): There exists a constant $\mu > 0$ such that for any $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$, $F_k(\boldsymbol{u}) \ge F_k(\boldsymbol{v}) + \langle \nabla F_k(\boldsymbol{v}), \boldsymbol{u} - \boldsymbol{v}\rangle + \frac{\mu}{2}\|\boldsymbol{u} - \boldsymbol{v}\|^2$.
- Bounded variance: For each client-class pair (m,k), let $n_{m,k}$ denote the total number of local samples. In each round, $b_{m,k} = rn_{m,k}$ samples are randomly selected with a sampling rate r, denoted as $\{\xi_{m,k,j}\}_{j=1}^{b_{m,k}}$. Then, for all

$$\mathbb{E}\left\|\frac{1}{b_{m,k}}\sum_{j=1}^{b_{m,k}} \nabla F_{m,k}(\boldsymbol{\theta};\xi_{m,k,j}) - \nabla F_{m,k}(\boldsymbol{\theta})\right\|^2 \le \sigma^2_{m,k},$$
$$\sigma^2_{m,k} = \frac{\sigma^2}{b_{m,k}} \tag{17}$$

  where σ^2 is the per-sample gradient variance bound.

- Uniformly bounded gradient: The expected squared norm of mini-batch stochastic gradients is uniformly bounded: For all mini-batches $\{\xi_{m,k,j}\}_{j=1}^{b_{m,k}}$ at client-class pair (m,k) and for all $\boldsymbol{\theta} \in \mathbb{R}^d$

$$\mathbb{E}\left\|\frac{1}{b_{m,k}}\sum_{j=1}^{b_{m,k}} \nabla F_{m,k}(\boldsymbol{\theta};\xi_{m,k,j})\right\|^2 \le R^2 \tag{18}$$

3.3. Convergence Analysis

To systematically characterize the convergence behavior of the global model under a dynamically weighted aggregation strategy, we present the following theoretical result.

Theorem 1 *Under the assumptions given in section 3.2, after T executions of Algorithm 1, the global model satisfies*

$$\mathbb{E}[F(\boldsymbol{w}_T)] - F^* \le \frac{2\kappa}{\gamma + T}\left[\frac{B}{\mu} + \left(2L + \frac{E\mu}{4}\right)\|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2\right],$$

where F^* is the global optimal value, $\kappa = L/\mu$, $\gamma = \max\{8\kappa, E\}$, and

$$B = \frac{\beta^2}{2\beta - \beta^2}\left[\alpha^2 V_p + 2\alpha(1-\alpha)V_{ps} + (1-\alpha)^2 V_s\right] \tag{19}$$

$$+ 6L\Gamma + 8(E-1)^2R^2 + \frac{M - M'}{M-1}\frac{4}{M'}E^2R^2. \tag{20}$$

$$V_p = \sum_{i=1}^{N} p_i^2\sigma_i^2,\ V_{ps} = \sum_{i=1}^{N} p_i s_i \sigma_i^2,\ V_s = \sum_{i=1}^{N} s_i^2\sigma_i^2, \tag{21}$$

$$A_p = F^* - \sum_{i=1}^{N} p_i F_i^*,\ A_s = F^* - \sum_{i=1}^{N} s_i F_i^*, \tag{22}$$

$$\Gamma = \alpha A_p + (1-\alpha)A_s. \tag{23}$$

E is the number of local SGD steps per round, M is the total number of clients, and M' is the number of clients participated in the executions.

As shown by the above result, SecureHDC-FL achieves a convergence rate of $\mathcal{O}\left(\frac{1}{T}\right)$. Furthermore, the optimal interpolation coefficient α^* satisfies

$$\alpha^* = \frac{C(V_s - V_{ps}) + 3L(A_s - A_p)}{C(V_p - 2V_{ps} + V_s)}, C = \frac{\beta^2}{2\beta - \beta^2} \tag{24}$$

Furthermore, if $\frac{V_{ps} - V_p}{A_p - A_s} < \frac{3L}{C} < \frac{V_s - V_{ps}}{A_p - A_s}$, we have $\alpha^* \in (0,1)$, which theoretically achieves the optimal balance between variance and bias. This upper bound on the convergence rate is tighter than that of using either p_k or s_k alone as aggregation weights.

Since weighting by data volume usually outperforms uniform aggregation, and our approach further improves upon data-volume-only weighting, our method thus achieves a strictly better generalization error upper bound than uniform aggregation.

Further Analysis. For completeness, we now provide detailed analysis of the variance–bias trade-off, EMA smoothing, and optimality conditions that underpin Theorem 1. The following content was previously presented as supplementary material, but is integrated here for clarity.

3.3.1. *Inequalities for Aggregation Variance*

The aggregation variance can be written as

$$\mathrm{Var}(\alpha) = \sum_{i=1}^{N} w_i^2 \sigma_i^2, \tag{25}$$

where $w_i = \alpha p_i + (1-\alpha)s_i$, and p_i, s_i are two types of normalized weights. Expanding, we have

$$\mathrm{Var}(\alpha) = \alpha^2 V_p + 2\alpha(1-\alpha)V_{ps} + (1-\alpha)^2 V_s, \tag{26}$$

where

$$V_p = \sum_{i=1}^{N} p_i^2\sigma_i^2,\ V_{ps} = \sum_{i=1}^{N} p_i s_i \sigma_i^2,\ V_s = \sum_{i=1}^{N} s_i^2\sigma_i^2. \tag{27}$$

By the method of Lagrange multipliers, it can be shown that when $p_i = \frac{n_i}{N}$ (i.e., weights normalized by data quantity), the aggregation variance V_p is minimized among all normalized weights. For any similarity-based normalized weight $\mathbf{s} \neq \mathbf{p}$, by the Cauchy-Schwarz inequality, $V_p < V_{ps} < V_s$.

3.3.2. *Inequalities for Aggregation Bias*

Let

$$A_p = F^* - \sum_{i=1}^{N} p_i F_i^*,\ A_s = F^* - \sum_{i=1}^{N} s_i F_i^*, \tag{28}$$

where $F^* = \min_w F(w)$ and $F_i^* = \min_w F_i(w)$. Here, s_i denotes the normalized similarity between the local HDC class vector and the global class vector. The closer the local class vector is to the global optimum, the higher the similarity, and the local optimum F_i^* usually gets closer to the global optimum F^*. Therefore, s_i and $F^* - F_i^*$ are negatively correlated. By the rearrangement inequality, $A_s < A_p$,i.e., similarity-based weights s_i lead to smaller aggregation bias.

3.3.3. *The Variance Reduction Effect of EMA Smoothing*

Given aggregation weights $w_i = \alpha p_i + (1-\alpha)s_i$, introduce an EMA update after aggregation, with the update rule:

$$G^{(t+1)} = \beta G_{\text{agg}}^{(t+1)} + (1-\beta)G^{(t)}. \tag{29}$$

Assume $G_{\text{agg}}^{(t+1)}$ and $G^{(t)}$ are approximately independent. Recursively, the variance of $G^{(t+1)}$ is

$$\mathrm{Var}[G^{(t+1)}] = \beta^2 \mathrm{Var}[G_{\text{agg}}^{(t+1)}] + (1-\beta)^2 \mathrm{Var}[G^{(t)}]. \tag{30}$$

Let $\sigma_{\text{agg}}^2 = \mathrm{Var}[G_{\text{agg}}^{(t+1)}]$, then at steady state:

$$\mathrm{Var}_{\text{mom}} = \frac{\beta^2}{1-(1-\beta)^2}\sigma_{\text{agg}}^2 = \frac{\beta^2}{2\beta-\beta^2}\sigma_{\text{agg}}^2, \tag{31}$$

hence reducing variance and improving stability.

3.3.4. *Convergence and Optimal Trade-off of Dynamically Weighted Aggregation*

In federated learning with non-i.i.d. data and partial client participation, existing theory shows [19, 18]: Let

$$\kappa = \frac{L}{\mu}, \quad \gamma = \max\{8\kappa, E\}, \quad \eta_t = \frac{2}{\mu(\gamma+t)}. \tag{32}$$

The convergence rate satisfies:

$$\mathbb{E}[F(\boldsymbol{w}_T)] - F^* \leq \frac{2\kappa}{\gamma+T}\left[\frac{B}{\mu} + \left(2L + \frac{E\mu}{4}\right)\|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2\right], \tag{33}$$

where T is the number of communication rounds, and other symbols are as above. For dynamically weighted aggregation with $w_i = \alpha p_i + (1-\alpha)s_i$ and EMA smoothing, the overall variance term should be multiplied by $\frac{\beta^2}{2\beta-\beta^2}$. Thus,

$$B = \frac{\beta^2}{2\beta-\beta^2}\left[\alpha^2 V_p + 2\alpha(1-\alpha)V_{ps} + (1-\alpha)^2 V_s\right] + 6L\,\Gamma(\alpha) + 8(E-1)^2R^2 + \frac{N-K}{N-1}\frac{4}{K}E^2R^2, \tag{34}$$

where

$$\Gamma(\alpha) = \alpha A_p + (1-\alpha)A_s \tag{35}$$

The above bound is a strictly convex quadratic function of α:

$$B(\alpha) = C(V_p - 2V_{ps} + V_s)\alpha^2 + \left[C(2V_{ps} - 2V_s) + 6L(A_p - A_s)\right]\alpha + \text{others}. \tag{36}$$

The minimizer is

$$\alpha^* = \frac{C(V_s - V_{ps}) + 3L(A_s - A_p)}{C(V_p - 2V_{ps} + V_s)}, \quad C = \frac{\beta^2}{2\beta-\beta^2}. \tag{37}$$

If

$$\frac{V_{ps} - V_p}{A_p - A_s} < \frac{3L}{C} < \frac{V_s - V_{ps}}{A_p - A_s}, \tag{38}$$

then $\alpha^* \in (0,1)$, achieving an optimal trade-off. The interval endpoints are strictly ordered, i.e.,

$$\frac{V_{ps} - V_p}{A_p - A_s} < \frac{V_s - V_{ps}}{A_p - A_s}, \tag{39}$$

because $A_p > A_s$ and

$$(V_s - V_{ps}) - (V_{ps} - V_p) = V_s - 2V_{ps} + V_p = \sum_{i=1}^{N}(s_i - p_i)^2\sigma_i^2 > 0. \tag{40}$$

This condition is commonly satisfied by ensuring that the model's smoothness, variance, and bias terms are of similar order, which can be achieved by tuning loss weights and regularization terms.

4. ERROR BOUND FOR APPROXIMATION OF $1/X$

This section presents the mathematical derivation of the error bound for the quadratic polynomial approximation of the function $f(x) = 1/x$ as mentioned in the main text.

According to numerical analysis theory, for a function $f(x)$ on the interval $[a, b]$, the best n-th order polynomial approximation $P_n^*(x)$ has the following error bound:

$$\max_{x\in[a,b]} |f(x) - P_n^*(x)| \leq \frac{(b-a)^{n+1}}{2^{2n+1}(n+1)!} \max_{\xi\in[a,b]} |f^{(n+1)}(\xi)| \tag{41}$$

In our case, we use a quadratic polynomial ($n = 2$) to approximate the function $f(x) = 1/x$ over the interval $[a, b] = [\frac{M}{e}, Me]$.

First, we compute the third derivative of $f(x)$ (i.e., $n+1 = 3$):

$$f'''(x) = -\frac{6}{x^4}$$

Its absolute value, $|f'''(x)| = 6/x^4$, attains its maximum at the left endpoint $\xi = M/e$ within the interval $[\frac{M}{e}, Me]$:

$$\max_{\xi\in[\frac{M}{e},Me]} |f'''(\xi)| = \frac{6}{(M/e)^4} = \frac{6e^4}{M^4}$$

Substituting this result into the error bound formula, we obtain:

$$\begin{aligned}\left|\frac{1}{x} - P_2(x)\right| &\leq \frac{1}{2^2\cdot 3!}\cdot\left(\frac{6e^4}{M^4}\right)\cdot\left(\frac{Me - M/e}{2}\right)^3 \\ &= \frac{1}{32}\cdot\frac{e^4}{M}\cdot(e-e^{-1})^3\end{aligned}$$

This result shows that the approximation error bound is $\mathcal{O}(\frac{1}{M})$, i.e., inversely proportional to the number of clients M.

5. ALGORITHMS FOR EFFICIENT ENCRYPTED MATRIX-VECTOR MULTIPLICATION

This section provides the detailed algorithms for the two efficient encrypted matrix-vector multiplication schemes—Low-Bandwidth Mode and High-Bandwidth Mode.

5.1. Low-Bandwidth Environment.

In low-bandwidth environments, such as narrowband wireless or intermittent satellite links, minimizing communication is critical, so each client must upload as few ciphertexts as possible. Under Algorithm 1, the client i must send the ciphertexts of $L_i^{(t)} \in \mathbb{R}^{d\times K}$ to the server in every aggregation round. If CKKS supports packing ℓ real numbers into a single ciphertext, the client i needs to upload dK/ℓ ciphertexts per round.

Algorithm 2 explains how to compute Step 11 of Algorithm 1 in this setting. The basic idea is to rewrite matrix-vector multiplication Av as $\sum_j v_j \cdot A_j$, where v_j is the j-th entry of the vector v and A_j is the j-th column of the matrix A. Many HE-based applications, e.g., [20, 21], showed that it is efficient. In fact, Algorithm 2 requires at most M CMuls (Step 3), $M \cdot \log d$ Rots (Step 4), and M Muls (Step 6). Before execution of Algorithm 2, we need to prepare the input, i.e., extracting $C_j^{(t)}$ from the inputs $L_i^{(t)}$ (Step 4 of Algorithm 1). A naïve implementation first extracts $L_{i,j}^{(t)}$ from each $L_i^{(t)}$ and then rotates it into the correct slot. Performing this for all $j < K$ costs KM plaintext-ciphertext multiplications (CMul) and rotations (Rot), respectively. However, note that each $G_j^{(t+1)}$ is ultimately stored in the first d slots of the output. We can therefore rotate each $L_{i,j}^{(t)}$ into its target position first, and after execution of Algorithm 2, use a single CMul to extract the leading d slots. This optimization reduces the number of CMuls from KM to 1.

Algorithm 2 (Matrix-Vector Multiplication in Low-Bandwidth Environment)

```
Input: $C_j^{(t)} = [L_{0,j}^{(t)}, L_{1,j}^{(t)}, \ldots, L_{M-1,j}^{(t)}] \in \mathbb{R}^{d\times M}$ and $W_j = [w_0, w_1, \ldots, w_{M-1}] \in \mathbb{R}^M$.
Output: $G_j^{(t+1)} = C_j^{(t)} \cdot W_j \in \mathbb{R}^d$
1: $G_j^{(t+1)} \leftarrow 0 \in \mathbb{R}^d$
2: for $i = 0$ to $M-1$ do
3:     $a_i \leftarrow W_j \circ e_i$ ▷ $e_i \in \mathbb{R}^M$, the $i$-th element 1 and others 0.
4:     for $j = 1$ to $\log_2(d)$ do
5:         $a_i \leftarrow a_i + \mathsf{Rot}_{2^j}(a_i)$
6:     $G_j^{(t+1)} \leftarrow G_j^{(t+1)} + a_i \circ L_{i,j}^{(t)}$
7: return $G_j^{(t+1)}$
```

5.2. High-Bandwidth Environment.

In high-bandwidth environments, real-time computation takes precedence over communication efficiency. As shown in Algorithm 3, we adopt the diagonal encoding method [22] to improve the efficiency of ciphertext matrix–vector multiplication. Each client locally pre-encodes its parameter $L_{i,j}^{(t)}$ as $F_{i,j,k}^{(t)} = L_{i,j}^{(t)} \circ e_{i-k \bmod M}$, where e_k is constructed by concatenating $(0_k, 1, 0_{M-k-1})$ along dimension d. The server aggregates $F_{i,j,k}^{(t)}$ to obtain $d_k = \sum_i F_{i,j,k}^{(t)}$, and then computes $G_j^{(t+1)} = \sum_k d_k \odot \mathsf{Rot}_k(W_j)$. This method requires at most

$M + \log(d/M)$ Rots(a $\log d$ reduction over Algorithm 2) and M Muls, which reduces the server-side computational complexity at the cost of linearly increased communication. In future work, we will further optimize both computational and communication efficiency.

Algorithm 3 (Matrix-Vector Multiplication in High-Bandwidth Environment)

```
Input: $F_j^{(t)} = \{F_{i,j,k}^{(t)}\}_{0\le i<M}^{0\le k<M}$, where $F_{i,j,k}^{(t)} = L_{i,j}^{(t)} \circ e_{i-k \bmod M} \in \mathbb{R}^d$, and $e_k \in \mathbb{R}^d$ is obtained
       from $p = d/M$ times repeating of the vector $(0_k, 1, 0_{M-k-1}) \in \mathbb{R}^M$;
       $W_j = [w_0, w_1, \ldots, w_{M-1}] \in \mathbb{R}^M$.
Output: $G_j^{(t+1)} = C_j^{(t)} \cdot W_j \in \mathbb{R}^d$, where $C_j^{(t)} = [L_{0,j}^{(t)}, L_{1,j}^{(t)}, \ldots, L_{M-1,j}^{(t)}] \in \mathbb{R}^{d\times M}$.
1: $G_j^{(t+1)} \leftarrow 0 \in \mathbb{R}^d$
2: for $i = 0$ to $\lceil \log(d/M) \rceil - 1$ do ▷ Extend the $M$-dimensional $W_j$ to dimension $d$.
3:     $W_j \leftarrow W_j + \mathsf{Rot}_{2^i \cdot M}(W_j)$
4: for $k = 0$ to $M - 1$ do
5:     $d_k(C_j^{(t)}) \leftarrow \sum_{i=0}^{M-1} F_{i,j,k}^{(t)}$ ▷ $d_k(C_j^{(t)})$ is the $k$-th diagonal vector of $C_j^{(t)}$.
6:     $G_j^{(t+1)} \leftarrow G_j^{(t+1)} + d_k(C_j^{(t)}) \circ \mathsf{Rot}_k(W_j)$
7: return $G_j^{(t+1)}$
```

6. REFERENCES

[1] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[2] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, 2019.

[3] Peter Kairouz, H Brendan McMahan, Brendan Avent, et al., "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[4] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot, "When the curious abandon honesty: Federated learning is not private," in *Euro S&P*. IEEE, 2023, pp. 175–199.

[5] Pentti Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation*, vol. 1, pp. 139–159, 2009.

[6] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing, "A theoretical perspective on hyperdimensional computing," *Journal of Artificial Intelligence Research*, vol. 72, pp. 215–249, 2021.

[7] Mohsen Imani, Justin Morris, John Messerly, Helen Shu, Yaobang Deng, and Tajana Rosing, "BRIC: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *DAC '19*, 2019, pp. 1–6.

[8] Zhuowen Zou, Yeseong Kim, M Hassan Najafi, and Mohsen Imani, "ManiHD: Efficient hyper-dimensional learning using manifold trainable encoder," in *DATE 2021*. IEEE, 2021, pp. 850–855.

[9] Alejandro Hernández-Cano, Namiko Matsumoto, Eric Ping, and Mohsen Imani, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *DATE '21*. IEEE, 2021, pp. 56–61.

[10] Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank HP Fitzek, and Najwa Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.

[11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory*, vol. 6, no. 3, pp. 1–36, 2014.

[12] Junfeng Fan and Frederik Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[13] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT 2017*. 2017, pp. 409–437, Springer.

[14] Léo Ducas and Daniele Micciancio, "Fhew: bootstrapping homomorphic encryption in less than a second," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 617–640.

[15] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène, "Tfhe: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[16] Vadim Lyubashevsky, Chris Peikert, and Oded Regev, "On ideal lattices and learning with errors over rings," *Journal of ACM*, vol. 60, no. 6, pp. 43:1–35, 2013.

[17] Quanling Zhao, Kai Lee, Jeffrey Liu, Muhammad Huzaifa, Xiaofan Yu, and Tajana Rosing, "FedHD: Federated

learning with hyperdimensional computing," in *Mobi-Com '22*, 2022, pp. 791–793.

[18] Kazim Ergun, Rishikanth Chandrasekaran, and Tajana Rosing, "Federated hyperdimensional computing," *arXiv preprint arXiv:2312.15966*, 2023.

[19] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang, "On the convergence of fedavg on non-iid data," *arXiv preprint arXiv:1907.02189*, 2019.

[20] Wen-Jie Lu, Shohei Kawasaki, and Jun Sakuma, "Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data," in *NDSS '17*. The Internet Society, 2017.

[21] Jingwei Chen, Linhan Yang, Chen Yang, et al., "Secure transformer-based neural network inference for protein sequence classification," Cryptology ePrint 2024/1851, 2024.

[22] Shai Halevi and Victor Shoup, "Algorithms in HElib," in *CRYPTO 2014*. 2014, pp. 554–571, Springer.