Hyperdimensional Computing as a Rescue for Efficient Privacy-Preserving Machine Learning-as-a-Service

Jaewoo Park*, Chenghao Quan†, Hyungon Moon* and Jongeun Lee†
*Department of Computer Science and Engineering, †Department of Electrical Engineering
Ulsan National Institute of Science and Technology (UNIST), Ulsan, Korea
{hecate64,quanch,hyungon,jlee}@unist.ac.kr

***Abstract*—Machine learning models are often provisioned as a cloud-based service where the clients send their data to the service provider to obtain the result. This setting is commonplace due to the high value of the models, but it requires the clients to forfeit the privacy that the query data may contain. Homomorphic encryption (HE) is a promising technique to address this adversity. With HE, the service provider can take encrypted data as a query and run the model without decrypting it. The result remains encrypted, and only the client can decrypt it. All these benefits come at the cost of computational cost because HE turns simple floating-point arithmetic into the computation between long (of degree $\geq$ 1024) polynomials. Previous work has proposed to tailor deep neural networks for efficient computation over encrypted data, but already high computational cost is again amplified by HE, hindering performance improvement. In this paper we show hyperdimensional computing can be a rescue for privacy-preserving machine learning over encrypted data. We find that the advantage of hyperdimensional computing in performance is amplified when working with HE. This observation led us to design HE-HDC, a machine-learning inference system that uses hyperdimensional computing with HE. We carefully structure the machine learning service so that the server will perform only the HE-friendly computation. Moreover, we adapt the computation and HE parameters to expedite computation while preserving accuracy and security. Our experimental result based on real measurements shows that HE-HDC outperforms existing systems by $26 \sim 3000\times$ times with comparable classification accuracy.**

***Keywords*—Homomorphic encryption (HE), hyperdimensional computing (HDC), privacy-preserving machine learning (PPML)**

I. Introduction

Machine learning models are frequently deployed as cloud-backed services, wherein clients send their data to the model owner's service to obtain the inference result. This model provision method, known as Machine Learning-as-a-Service (MLaaS), is widespread for several reasons. Clients often prefer not to run the model locally due to limited computational resources or energy constraints. Additionally, model owners prefer to safeguard their models or any associated knowledge, such as training data, from potential leaks.

This prevalent service structure suffers from a critical drawback: the clients' data is vulnerable to exposure by the model owner. Despite employing contracts and security measures, such data is often considered at risk of being leaked. For instance, many tech companies prohibit their employees from using external MLaaS platforms due to concerns over highly confidential assets being compromised. To address these concerns, privacy-preserving machine learning (PPML) techniques [1]–[3] have emerged, with the primary goal of safeguarding the clients' data from being exposed to the model owner.

This work was supported in part by the Samsung Advanced Institute of Technology, Samsung Electronics Co., Ltd.; in part by IITP Grant through Artificial Intelligence Graduate School Program (UNIST, 2020-0-01336) and by National R&D Program through NRF of Korea (RS-2023-00258527) both funded by MSIT of Korea; and in part by the Free Innovative Research Fund of UNIST (1.170067.01).

J. Lee and H. Moon are the corresponding authors of this paper.

The PPML techniques for this problem—client data secrecy in MLaaS—can be classified into two groups. The first group [4] is the TEE (trust execution environment) or enclave approach, where MLaaS runs their model within a secure enclave on their cloud platform so that clients can attest that the enclave does not leak their data. This approach has performance advantages as it can perform inference over plaintext and leverage accelerators and protection mechanisms to enhance performance. However, the confidentiality of user data hinges upon the strength of the isolation mechanism or the secrecy of the cryptographic keys upon which the enclaves are built. While the enclave approach holds promise, no existing system is known to be completely secure against side-channel attacks [5]–[7].

The second group [2]–[4], [8], [9] is the cryptographic approach that utilizes *Homomorphic Encryption* (HE). The HE-based approach has a strong theoretical foundation that ensures data confidentiality on the fly; the only way to compromise data is by stealing the cryptographic key that is kept by the client locally. However, the primary obstacle preventing the widespread use of HE in various applications is its high computational cost, typically 5-6 orders of magnitude slower compared to plaintext operations. Most of the work is dedicated to the systems research [10], [11] while not much work focuses on developing HE-friendly algorithms. Thus in this paper, we present a novel strategy for utilizing HE in the context of PPML for classification applications, thereby addressing this challenge.

This paper presents HE-HDC, an efficient yet privacy-preserving ML framework that significantly expedites online MLaaS with HE and *hyperdimensional computing*. Hyperdimensional computing (HDC) [12] is an emerging ML paradigm with very efficient training and robustness to noise, in which a model is represented as a set of *hypervectors*. An HDC-based classification system, for instance, would first *encode* input data into a hypervector and compare it with model hypervectors. Then the class label associated with the most similar model hypervector is returned as inference result.

Our key observation is that when computed over HE, the performance advantage of HDC over DNN (deep neural network) can be greatly amplified. While the performance of DNN-based PPML is limited by excessive computational cost of HE especially for some essential operations (e.g., multiplication and non-linear operations in the case of CKKS), our HE-HDC framework very scarcely uses those operations that are expensive on HE while providing similar privacy guarantee. Also the application domain of HDC is fast expanding, including not just classification but also object recognition [13], natural language processing (NLP) [14], [15], reinforcement learning [16], and bio-informatics [17]–[19].

We implemented HE-HDC using the SEAL [20], which is one of the most widely used library that implements the core of CKKS scheme. The experiment running HE-HDC end-to-end with MNIST data set exhibits that HE-HDC outperforms the existing mechanisms [1], [2] running CKKS-tailored deep neural networks by

$26 \sim 3000\times$ with comparable classification accuracy.

This paper makes the following contributions:

- To the best of our knowledge, we are the first to propose and explore using hyperdimensional computing over HE for privacy-preserving MLaaS. Our study shows that our system, HE-HDC, performs image classification tasks with significantly low latency at comparable accuracy when compared to the state-of-the-art PPML techniques.
- We identify the inference using hyperdimensional computing requires nearly no multiplication, which is considerably more expensive when computed over HE. We further adapt the inference procedure to eliminate the multiplication operation, enabling to use the optimal parameters for HE.

II. Related Work

Here we briefly review previous work on PPML focusing on image classification task and on HE. However, none of the previous work considers combining HE and HDC, nor does any HDC work propose application of cryptography for PPML.

A. Privacy-Preserving Image Classification

Bourse et al. [9] present a framework for the homomorphic evaluation of neural networks named FHE-DiNN, in which each neuron's output is refreshed through bootstrapping. Juvekar et al. [8] develop Gazelle, a scalable and low-latency system for secure neural network inference, which contains a homomorphic encryption library, homomorphic linear algebra kernels, and optimized encryption switching protocols. Dowlin et al. [2] present CryptoNets, neural networks that can be applied to encrypted data to make accurate predictions while maintaining data privacy and security. Chou et al. [21] present Faster CryptoNets, which accelerate the homomorphic evaluation by developing a pruning and quantization approach that leverages sparse representations. Additionally, they approximate modern activation functions. Brutzkus et al. [22] present Low-Latency CryptoNets (LoLa), which change representations of the data throughout the computation by novel ways to reduce latency while maintaining accuracy and security. They also apply the method of transfer learning to provide private inference services. Boemer et al. [23] introduce nGraph-HE, a graph compiler, which enables the deployment of trained models with popular DL frameworks while simply treating HE as another hardware target. Furthermore, they develop HE-aware graph-compiler optimizations, both at compile-time and at run-time. Hesamifard et al. [24] design a privacy-preserving classification for convolutional neural networks over encrypted data, which replaces the commonly used activation functions in CNNs with low-degree polynomials. Benaissa et al. [25] present TenSEAL, a flexible open-source library, for doing encrypted tensor computation using homomorphic encryption. Lee et al. [1] reduce the runtime overhead of bootstrapping by a multiplexed packing method, which packs data of multiple channels into one ciphertext in a compact manner. Besides, they propose a faster multiplexed parallel convolution algorithm to reduce the number of required rotations using full ciphertext slots. Liu et al. [26] propose Trusted-DNN, an overall DNN model protection strategy based on TrustZone and encryption algorithms oriented to the security issues of embedded devices.

B. HDC

Zou et al. [27] develop NeuralHD, which is the first HDC algorithm with dynamic and regenerative encoder for adaptive learning. NeuralHD can enhance learning capability and robustness by identifying insignificant dimensions and regenerating those dimensions. Furthermore, they present a scalable learning framework to distribute NeuralHD computation over edge devices. In [28], the authors propose ManiHD which provides trainable encoding. It considers non-linear interactions between the features. ManiHD also supports online learning by sampling data and capturing important features in an unsupervised manner. Imani et al. [29] propose VoiceHD, an efficient and hardware-friendly speech recognition technique using HD computing. It maps preprocessed voice signals in the frequency domain to hypervectors and combines them to compute class hypervectors. Additionally, they extend VoiceHD to VoiceHD+NN which uses a single-layer neural network to improve the resolution of similarity measures. AdaptHD is proposed in [30], which is an adaptive retraining method for HD computing. AdaptHD introduces the definition of learning rate in HD computing and proposes a hybrid approach to update the learning rate considering both iteration and data dependency. Instead of naïve data accumulation during training, OnlineHD [31] updates the model differently depending on the model prediction result, which enables iterative training, potentially boosting performance of HDC models. Kim et al. [32] propose CascadeHD, an efficient many-class classification learning framework, which considers the inter-dependency of different classes by revising the iterative hypervector fine-tuning procedure and identifies confusing classes while learning a hierarchical inference structure by a meta-learning algorithm. Prive-HD [33] may be most similar to our work in that it considers HD computing from a privacy perspective. The authors observe that HD computing has no privacy due to its reversible computation, so they present Prive-HD to realize a differentially private HD model as well as to blur the information sent for cloud-hosted inference by quantization and pruning. However, in our work, the client data is encrypted, so there is no privacy issue.

III. Preliminary

A. Hyperdimensional Computing

The hyperdimensional computing (HDC) model is a machine learning paradigm that is built around vectors of large dimensions ($D \simeq 4000$ or more) called *hypervectors*. The HDC approach has important advantages including much simpler training (not based on back-propagation), superior generalization performance, and holographic representation promoting noise resilience [27], [34]–[36]. Here we briefly explain HDC using a simple image classification system example, which consists of an encoder and a classifier.

Encoding: The goal of encoding is to find a well-defined hypervector representation for input data. Though various encoding schemes have been proposed in the literature [27]–[32], they all share the common principle: (i) the generated hypervectors should be nearly orthogonal to one another, and (ii) different data points should result in different hypervectors. As an example, random projection encoding [31] transforms a flattened image vector $\vec{X}$ of length K into a hypervector $\vec{H}$ of length D as follows.

$$\vec{H} = \cos(B\vec{X} + \vec{b}) \otimes \sin(B\vec{X}) \tag{1}$$

where B is a $(D \times K)$-sized matrix, whose columns are random hence orthogonal *base hypervectors* drawn from a standard normal distribution; $\vec{b}$ is another random base hypervector of length D drawn from a uniform distribution in $[0, 2\pi]$; and $\otimes$ is element-wise multiplication.

Model and Training: A typical HDC-based image classifier model consists of a number of *class hypervectors* $\vec{C^l}$, one for each class (or label) l. A class hypervector is a generalized representation of each class. Due to the orthogonality of hypervectors, training, or computing class hypervectors, is extremely easy, and does not require error

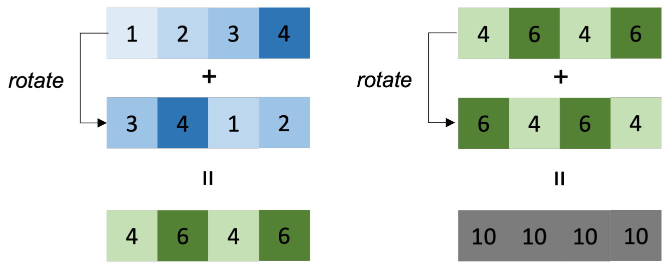

Fig. 1: Homomorphically summing all elements of a plaintext vector (of length-n) using $\log_2 n$ rotations and $\log_2 n$ additions.

back-propagation. Once input hypervectors are generated, a class hypervector is obtained by accumulating all hypervectors belonging to the class.

$$\vec{\mathcal{C}^l} = \sum_j \vec{\mathcal{H}^l_j} \tag{2}$$

Classifier and Inference: Classification of a query data is performed by first mapping the query data to a hypervector $\vec{\mathcal{H}_q}$, and then computing the similarity of the *query hypervector* with every class hypervector $\vec{\mathcal{C}^l}$. The label with the highest similarity is selected, which is the inference result. For the similarity measure (δ), cosine similarity is typically used:

$$\delta(\vec{\mathcal{H}_q}, \vec{\mathcal{C}^l}) = \frac{\vec{\mathcal{H}_q^\intercal} \cdot \vec{\mathcal{C}^l}}{\|\vec{\mathcal{H}_q}\| \cdot \|\vec{\mathcal{C}^l}\|} \tag{3}$$

where $\|\cdot\|$ represents the L_2 norm.

Iterative Training: Due to the limited performance (i.e., inference accuracy) of single-pass training, an iterative training method can be employed [31], whose idea is to examine mispredicted queries and update relevant class hypervectors (i.e., of correct and mispredicted labels). Given a query hypervector $\vec{\mathcal{H}_q}$ with the correct label l and the mispredicted label m, the class hypervectors are updated as follows (η is learning rate):

$$\vec{\mathcal{C}^l} = \vec{\mathcal{C}^l} + \eta(1 - \delta(\vec{\mathcal{H}_q}, \vec{\mathcal{C}^l}))\vec{\mathcal{H}_q} \tag{4}$$

$$\vec{\mathcal{C}^m} = \vec{\mathcal{C}^m} - \eta(1 - \delta(\vec{\mathcal{H}_q}, \vec{\mathcal{C}^m}))\vec{\mathcal{H}_q} \tag{5}$$

B. CKKS: an HE Scheme for Approximate Arithmetic

CKKS [37] is an HE scheme that is tailored for approximate arithmetic such as fixed-point operations. A ciphertext, represented as a pair of polynomials (c_0, c_1), each of which is of degree N, corresponds to a plaintext of a length-$N/2$ vector of integers. To represent fixed-point values as integers, scale factor (Δ) is used, which causes a problem with multiplication (note that both plaintext and ciphertext are represented as integer-coefficient polynomials), requiring a *rescaling operation* after each multiplication.

Other operations provided by CKKS include *bootstrapping* and *rotation*. Ciphertexts have *noise*, limiting the number of operations they can undergo before correct decryption is impossible. Multiplication increases noise significantly, hence *multiplicative depth*, or the number of homomorphic multiplications a ciphertext can go through, is an important parameter. The operation that effectively reduces this noise is called *bootstrapping*, which is however very expensive. To increase multiplicative depth (which is the length of modulus chain minus one), a large ciphertext modulus is required.

The homomorphic *rotation* operation effectively rotates the corresponding plaintext vector. In practice, we often need to rotate a vector so that we can perform an operation involving an element of a vector and another element of the same or a different vector. Fig. 1 illustrates how to perform a reduce-sum operation on a plaintext (here, a length-4 vector) using a series of homomorphic rotations and additions.

CKKS is configurable with two parameters (N, Q) that affect the level of security and computation performance (see Table I). These parameters must be chosen so that a certain level of security can be cryptographically provided. Table II lists the set of parameter combinations we consider in this paper. Increasing N results in slower computation over encrypted data due to longer ciphertexts. However, decreasing N mandates the sum of $\log_2 Q_i$ to be reduced in order to maintain the same level of security. Also, decreasing the sum of Q reduces the precision of computation, negatively impacting the quality of computation (e.g., inference accuracy). Throughput this paper we use $\log q$ to denote $\sum_i \log_2 Q_i$.

C. Threat Model

We assume a common threat model that privacy-preserving MLaaS considers [2], [9]. The MLaaS provider offers an image classification service and prefers not to reveal the model itself. For example, the provider aims to keep the class hypervectors as secrets. Clients want to use the classification service using private input data. They do not trust the MLaaS provider, i.e., they assume that the MLaaS provider may want to collect the private data if clients send them to the provider. We also follow a common assumption of the existing HE-based schemes by assuming that the MLaaS provider is honest but curious, i.e., it wants to steal clients' data but performs the requested computation honestly. This is a reasonable assumption in that dishonest computation in a service will simply reduce the quality of service, making it less attractive to clients.

IV. HE-HDC

A. Overview

Fig. 2 gives an overview of our HE-HDC approach. Similar to previous FHE-based PPML approaches, we assume that a fully trained hyperdimensional classifier model runs on a server, i.e., the server has unencrypted class hypervectors as their owner. The encoder of the HDC classifier is shared with the client device, which uses it to *encode* ① the user's (private) data into a query hypervector. The query hypervector is then *encrypted* ② into a ciphertext using the user's *secret key*. The encrypted query hypervector is then sent ③ to the server. The MLaaS provider homomorphically computes similarity scores ④ for all classes, resulting in L encrypted similarity scores ⑤ where L is the number of classes. These encrypted scores are sent back to the client, which decrypts ⑥ the scores by using the secret key. Note that the result of any homomorphic computation using encrypted data (e.g., the query hypervector) remains encrypted and only the client, who has the secret key, can decrypt it. As the last step, the client compares the similarity scores ⑦ to obtain the class result ⑧.

Rationale for Client-side Encoding: Exposing the encoder that consists of the base hypervectors does not enable the clients to learn about the provisioned model. These hypervectors are generated randomly in principle [12], and the choice of them does not affect the accuracy of the trained classification model significantly.[1] For this reason, revealing the base hypervectors does not help the clients to infer the class hypervectors, which is the trained model.

Computing Cosine Similarity over CKKS: The task left for the MLaaS provider, or the server, is to compute the cosine similarity

[1]We have generated 100 different encoders and used them to train HDC classifiers for MNIST, which have shown very uniform classification accuracy with variance of less than 10^{-3}%.

TABLE I: Key terms and symbols used for CKKS

Term	Meaning
Multiplicative Depth	The number of multiplications between ciphertexts that can be performed after one encryption.
Modulus Chain ($Q = (Q_1, \ldots, Q_l)$)	A sequence of moduli that a CKKS scheme uses for computation with ciphertexts. The length of chain determines the multiplicative depth and the sum the level of security.
N	The degree of ciphertext (which is represented as a pair of polynomials). N must be a power of an integer and a multiple of two. This parameter affects the level of security and latency of computation with ciphertexts.

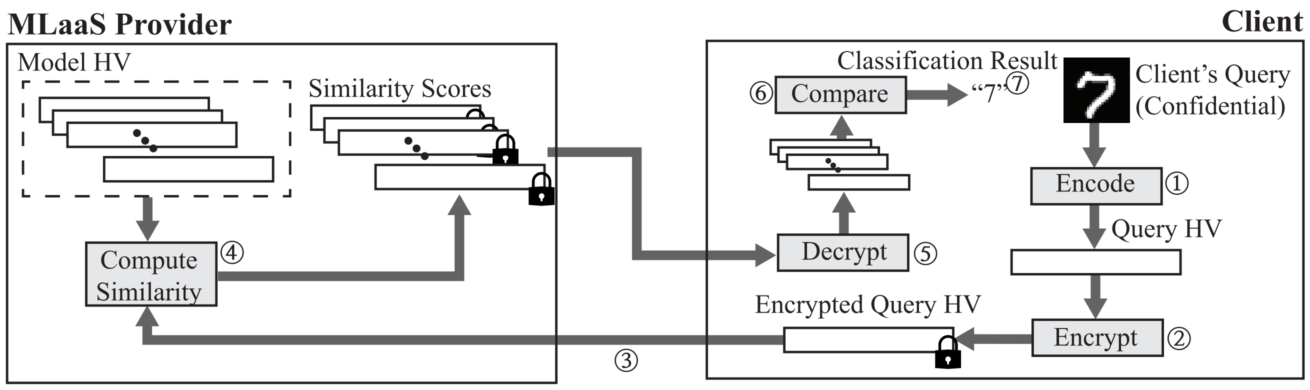

Fig. 2: An overview of HE-HDC. HV is short for hypervector.

TABLE II: Parameter Combinations Used in Experiments

Security Level (bits)	$\log_2 N$	$\log_2 q$	($\log_2 Q_i$)
128	11	54	(27, 27)
128	12	109	(54, 54)
128	13	218	(60, 60)
128	14	438	(60, 60)

between the query hypervector and each class hypervector, as (3) shows. Each cosine similarity computation requires one dot product between two hypervectors and two scalar divisions. Assuming the number of slots inside a single ciphertext is no less than the dimension of each hypervector, the dot product could be implemented as a single ciphertext-plaintext multiplication followed by $\log_2 D$ ciphertext rotations and $\log_2 D$ ciphertext-plaintext additions. Division by $\|\vec{\mathcal{C}^l}\|$ requires additional ciphertext-plaintext division. However the division by $\|\vec{\mathcal{H}_q}\|$ requires ciphertext-ciphertext division, which is not supported by CKKS, as well as many ciphertext-ciphertext multiplications and additions to compute L2-norm.

B. Optimizing Similarity Search on CKKS

Eliminating Homomorphic Division: Unlike homomorphic multiplication, homomorphic division is not supported natively by CKKS. To avoid division in similarity search, we perform these two optimizations.

First, we store class hypervectors in a normalized form $\vec{\mathcal{C}'^l} = \vec{\mathcal{C}^l}/\|\vec{\mathcal{C}^l}\|$, which can save one homomorphic multiplication as shown below.

$$\delta(\vec{\mathcal{H}_q}, \vec{\mathcal{C}^l}) = \frac{\vec{\mathcal{H}_q}^\intercal \cdot \vec{\mathcal{C}^l}}{\|\vec{\mathcal{H}_q}\| \cdot \|\vec{\mathcal{C}^l}\|} = \frac{\vec{\mathcal{H}_q}^\intercal \cdot \vec{\mathcal{C}'^l}}{\|\vec{\mathcal{H}_q}\|} \tag{6}$$

Second, we observe that the computation of $\|\vec{\mathcal{H}_q}\|$ is not strictly necessary for the purpose of classification. We only need to do comparison among labels, and $\|\vec{\mathcal{H}_q}\|$, being common for all labels, does not affect the comparison result, and therefore can be eliminated. After these two simplifications, similarity search is reduced to just one dot-product operation per class.

Matrix-Vector Multiplication Optimization: To expedite the computation of cosine similarity, we stack class hypervectors to form a class hypervector matrix M and perform a single matrix-vector multiplication (MVM) between the query hypervector and the class hypervector matrix as shown below.

$$M = \begin{bmatrix} \vec{\mathcal{C}'^1} & \vec{\mathcal{C}'^2} & \ldots & \vec{\mathcal{C}'^L} \end{bmatrix} \tag{7}$$

$$\vec{\delta}(\vec{\mathcal{H}_q}, M) = \vec{\mathcal{H}_q}^\intercal \cdot M \tag{8}$$

This allows us to leverage the highly optimized MVM library within CKKS. Given that MVM is one of the most widely used forms of computation, it has gained significant attention from the community for optimization. In particular, we employ the dense vector-row major matrix multiplication method proposed as proposed by Brutzkus et al. [22].

One hypervector is encrypted into one or more ciphertexts depending on the hypervector dimension and the number of slots in a single ciphertext ($N/2$).

Case 1: When the number of slots inside a single ciphertext ($N/2$) is no less than the hypervector dimension (D), each hypervector can be encrypted into a single ciphertext (or a plaintext in the case of $\vec{\mathcal{C}'^l}$). The dot-product between plaintext and ciphertext is evaluated by a ciphertext-plaintext multiplication followed by a reduce sum of the resulting ciphertexts as illustrated in Fig. 1.

Case 2: Otherwise, $k = \lceil \frac{2D}{N} \rceil$ ciphertexts must be used for a hypervector. We do element-wise multiplication of two hypervectors by first performing ciphertext-wise multiplications, generating k ciphertexts, which are added together to generate one ciphertext, to which a reduce sum is applied to obtain the final result. Note that doing ciphertext additions first then a reduce sum is significantly faster than the opposite case (i.e., reduce sums for all ciphertexts followed by ciphertext additions), since it involves fewer ciphertext rotations.

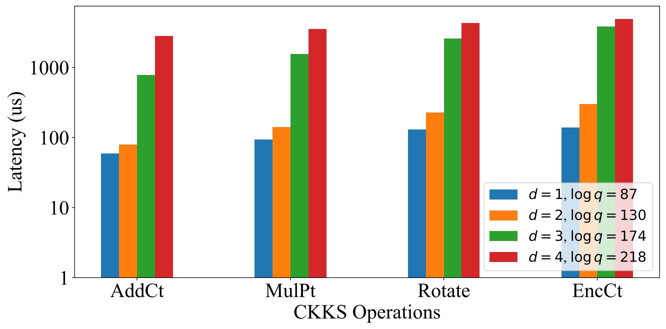

Fig. 3: Latency of CKKS operations for various multiplicative depths (d) when we choose N as 8192 and the largest Q that satisfies the 128-bit security.

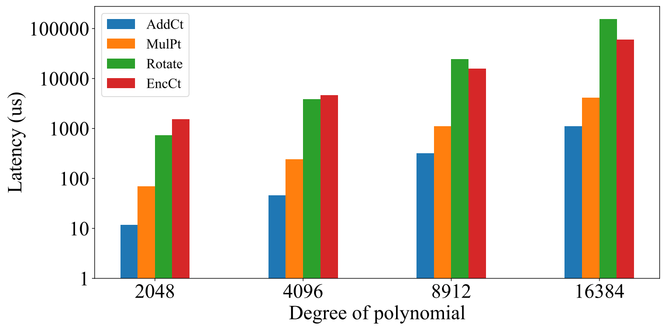

Fig. 4: The latency of CKKS operations with respect to the polynomial degree N.

Our inference latency result for the MNIST dataset is 57× faster than the DNN-based PPML approaches such as [22], even though we have not employed the state-of-the-art MVM algorithm (e.g., [38]).

C. Optimization of CKKS Parameters

As shown in Fig. 3, the latency of CKKS operations varies significantly depending on the length of modulus chain (Q), which is determined by the multiplicative depth d required by an application. While DNN-based PPML approaches require a multiplicative depth of up to 14 [1], our HE-HDC requires only a single ciphertext-plaintext multiplication in any data flow path. Moreover, this multiplication need not be followed by a rescaling operation, because the multiplication increases the scale factor of *all ciphertexts universally*. Therefore there is no need for rescaling if absence of overflow can be guaranteed, which is the case in our application. With this *rescale skipping* optimization, we can use the shortest modulus chain possible, with just two modulus integers. For a given polynomial degree (N), the runtime of CKKS operations is not affected by modulus bit-widths. Thus we choose the largest modulus chain bit-widths ($\log q$) that meet the 128-bit security standard.

D. Keeping the Polynomial Degree Low with Quantization

Fig. 4 shows that the latency of CKKS operations grows exponentially with respect to the degree of polynomials. To achieve better performance, it is desirable to use short polynomial degrees. However, using low-degree polynomials prevents us from choosing a large value for $\log q$, which affects the precision of computation with CKKS. For example, the $\log q$ must not be larger than 54 when the degree of polynomial (N) is 4096. The corresponding modulus chain Q with the highest arithmetic precision is (27, 27). Under a modulo integer of only 27 bits, the HDC classification accuracy is significantly low. To achieve both high model accuracy and high performance under limited arithmetic precision, we quantize the class hypervectors. Fig. 5 illustrates the effect of quantization on model accuracy for different modulo bit-widths for a hypervector dimension $D = 2048$. The encryption of floating-point vectors into CKKS ciphertexts already acts as an fixed-point quantization, multiplying the floating-point values by 2^{Δ} and rounding to the nearest integer. In HE-HDC, the CKKS scale is set to be greater than the quantization bit-precision of query hypervectors. Using an exhaustive search we determine the scale factor to have the best model accuracy.

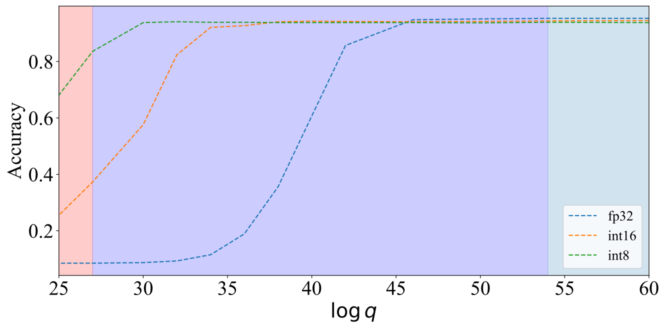

Fig. 5: Effect of quantization in accuracy with respect to modulus bit-length. The three colored regions indicate polynomial degrees ($N = 2^{11}, 2^{12}, 2^{13}$).

TABLE III: Impact of Design Decisions

Method	Inference Latency (ms)
Without Normalization and with Rescale†	257.37 (×1.30)
Without Normalization (Rescale Skipped)	214.75 (×1.09)
With Normalization (Rescale Skipped)	197.41 (×1)

Note. 1. $N = 8192$ and $D = 4096$.
2. †Requires a longer modulus chain.

V. Experiments

A. Experimental Setup

We have implemented our HE-HDC using the Microsoft SEAL library [20] to evaluate the inference latency and classification accuracy. Table II shows the CKKS parameters used in our experiments. As for the other configuration or parameters, we use the default values of the SEAL library. A single Intel i5-1038NG7 CPU without multithreading is used to measure the latency of both inference and encryption (on the client). The training and quantization of HE-HDC is implemented extending the official code of Online-HD [31].

B. Inference Latency

Table III summarizes the effect of our optimizations (class hypervector normalization and rescaling skipping). With the polynomial degree of 2^{13}, the inference latency of a naïve implementation takes around 257 ms. By removing the *rescaling* operation and therefore shortening the modulus chain, latency decreases by 21%. Further normalizing the class hypervectors improves the latency by an additional 9%.

The inference latency for different hypervector dimensions (D) and polynomial degrees (N) of ciphertexts is presented in Fig. 7. The latency is more sensitive to the polynomial degree than the hypervector dimension because the rotation is the most time-consuming operation

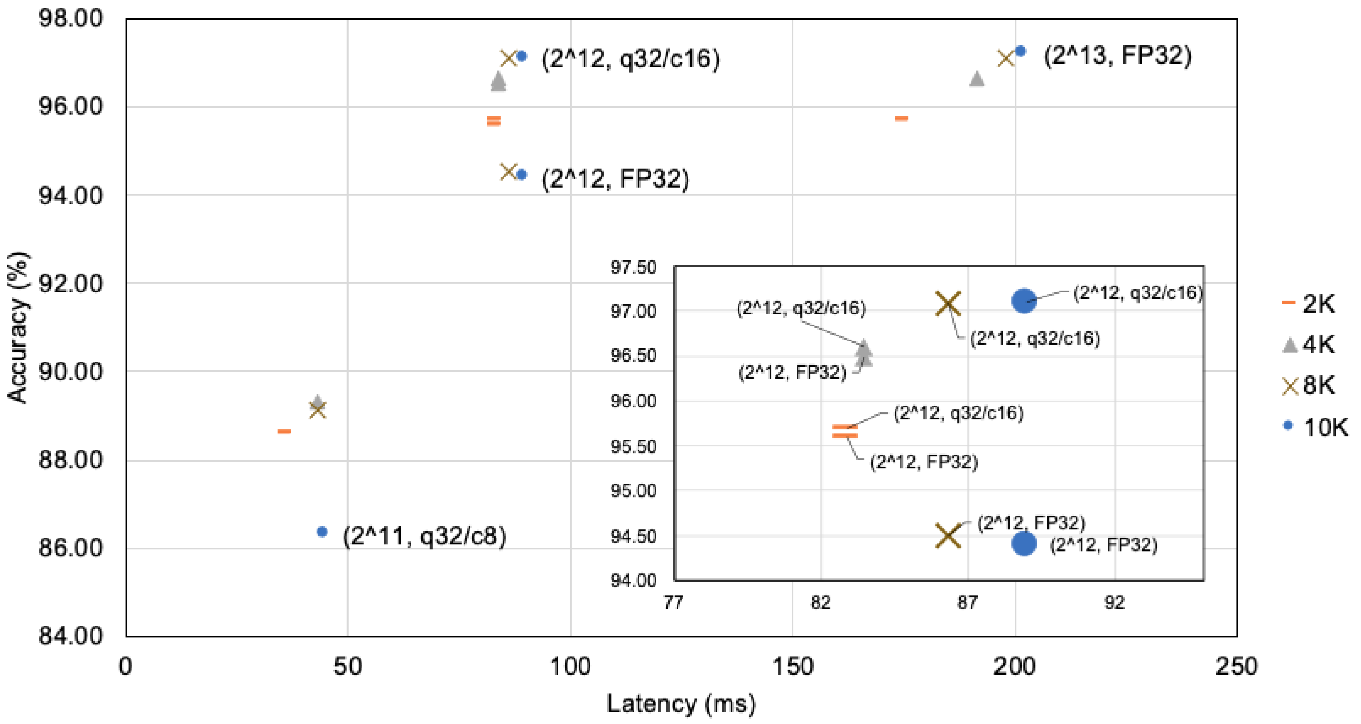

Fig. 6: The impact of quantization on classification accuracy when using various polynomial degrees ($N = 2^{11}, 2^{12}, 2^{13}$) and HV dimensions ($D$ = 2K, 4K, 8K, 10K, depending on the marker symbol). Inset shows the results of $N = 2^{12}$. q32/c16 means 32-bit query HV and 16-bit class HVs.

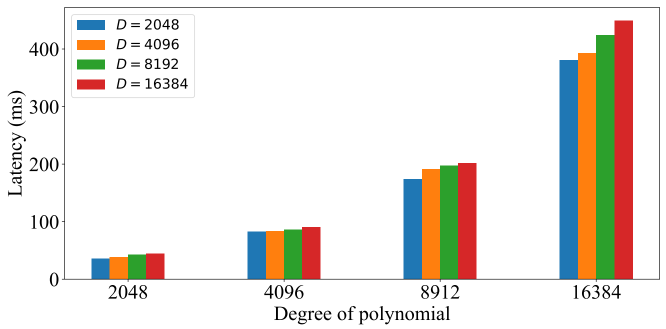

Fig. 7: Inference latency with various polynomial degrees (N) and hypervector dimensions (D).

whose performance is only affected by the polynomial degree. We observe that it is always better to lower the polynomial degree as long as the arithmetic precision is sufficient.

C. Effect of Quantization on Accuracy

To see the effect of quantization and the overall accuracy, results with three different quantization levels and various configurations are presented in Fig. 6. With a polynomial degree $N \geq 8192$, the encrypted inference shows no degradation in accuracy compared with the unencrypted case. Models with a polynomial degree of $N = 4096$ starts to lose accuracy compared to its unencrypted baseline. Though models with a larger hypervector dimension have a better baseline accuracy, more errors are accumulated during the dot-product evaluation, resulting in lower accuracy. The accuracy can be restored by quantizing the class hypervectors to lower precision (e.g., 16 bits). The model with $D = 4096$ and $N = 4096$ has the best accuracy of 97.10%. At a polynomial degree of $N = 2048$, only a modulus of bit-length 27 is supported, extremely constraining the precision. For every hypervector dimension value, floating-point models have an accuracy under 10%. Even though we have tried quantizing the class hypervectors to 8 bits, the accuracy still remains below 90%.

TABLE IV: Comparison with Previous Work

Method	Accuracy (%)	Message Size (B)	Latency (s) Encryption	Latency (s) Inference
CryptoNets [2]	98.95	367.5M†	44.5†	250†
LoLa [22]	98.95	11.72M*	1.42*	2.2 ‡
LoLa-Small [22]	96.92	11.72M*	1.42*	0.29 ‡
HE-HDC (Ours)	97.10*	96.16K*	0.011*^{4}	0.083*

Note. 1. †for a batch size of 4096 (other works use the batch size of 1).
2. ‡with multithreading enabled.
3. *Our own measurements.
4. ^{4}Including the hypervector encoding latency.

D. Comparison with Other HE-based PPML

Table IV compares our HE-HDC with previous HE-based PPML approaches. The table reports the inference and encryption latency, the message size (the size of ciphertexts sent to the server), and the model accuracy on the MNIST dataset. The encryption latency includes the hypervector encoding latency in the case of HE-HDC. For HE-HDC, we use the configuration of $N = 4096$ and $D = 8192$, which gives the best accuracy. Our results clearly demonstrate that our HE-HDC outperforms all the previous DNN-based methods by $26 \sim 3000\times$ in latency with marginal difference in accuracy. This is mainly because HE-HDC uses a polynomial degree of 2^{12} and the shortest modulo chain. It is also notable that HE-HDC has the smallest message size and extremely low encryption latency. HE-HDC requires $1 \sim 4$ ciphertexts depending on the hypervector dimension whereas LoLa [22] requires 25 ciphertexts to encrypt a single MNIST image, and CryptoNets [2] encrypts every single pixel into a single ciphertext. The latency of single inference on CryptoNets [2] is 250 seconds, which needs a batch size of 4096 for full utilization. This may not be realistic, since the entire batch must come from a single client sharing the same secret key.

VI. Conclusion

We presented a novel PPML scheme, HE-HDC, which uses HDC together with CKKS. Our experimental results based on a full implementation on top of the SEAL library and real measurements of actual CPU runtime clearly demonstrate that our HE-HDC is indeed CKKS-friendly, which has never been reported before. Our optimization methods and cryptography parameter tuning further expedite the encrypted inference procedure, leading HE-HDC to outperform previous state-of-the-art PPML methods by $26 \sim 3000\times$ times, without significantly sacrificing inference accuracy.

Acknowledgments

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2023-2021-0-01817) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation), and Samsung Electronics Co., Ltd.

References

[1] E. Lee *et al.*, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri *et al.*, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 12 403–12 422.

[2] R. Gilad-Bachrach *et al.*, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. PMLR, 20–22 Jun 2016, pp. 201–210.

[3] E. Hesamifard *et al.*, "Privacy-preserving machine learning as a service." *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 123–142, 2018.

[4] Z. Sun *et al.*, "ShadowNet: A secure and efficient on-device model inference system for convolutional neural networks," 2022.

[5] Z. Chen *et al.*, "Voltpillager: Hardware-based fault injection attacks against intel sgx enclaves using the svid voltage scaling interface." in *USENIX Security Symposium*, 2021, pp. 699–716.

[6] K. Murdock *et al.*, "Plundervolt: Software-based fault injection attacks against intel sgx," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1466–1482.

[7] G. Chen *et al.*, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.

[8] C. Juvekar *et al.*, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 1651–1669.

[9] F. Bourse *et al.*, "Fast homomorphic evaluation of deep discretized neural networks," in *Advances in Cryptology – CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III*. Berlin, Heidelberg: Springer-Verlag, 2018, p. 483–512.

[10] A. Feldmann *et al.*, "F1: A fast and programmable accelerator for fully homomorphic encryption (extended version)," *arXiv preprint arXiv:2109.05371*, 2021.

[11] J. Park *et al.*, "NTT-PIM: Row-centric architecture and mapping for efficient number-theoretic transform on pim," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023.

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

[13] P. Łuczak *et al.*, "Combining deep convolutional feature extraction with hyperdimensional computing for visual object recognition," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8.

[14] F. Liu *et al.*, "L3e-hd: A framework enabling efficient ensemble in high-dimensional space for language tasks," in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022, pp. 1844–1848.

[15] R. Thapa *et al.*, "Spamhd: Memory-efficient text spam detection using brain-inspired hyperdimensional computing," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 84–89.

[16] H. Chen *et al.*, "Darl: Distributed reconfigurable accelerator for hyperdimensional reinforcement learning," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[17] A. Burrello *et al.*, "One-shot learning for ieeg seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing," in *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2018, pp. 1–4.

[18] Y. Kim *et al.*, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 115–120.

[19] D. Ma *et al.*, "Molehd: Drug discovery using brain-inspired hyperdimensional computing," *arXiv preprint arXiv:2106.02894*, 2021.

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

[21] E. Chou *et al.*, "Faster CryptoNets: Leveraging sparsity for real-world encrypted inference," 2018.

[22] A. Brutzkus *et al.*, "Low latency privacy preserving inference," 2019.

[23] F. Boemer *et al.*, "nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data," 2019.

[24] E. Hesamifard *et al.*, "Deep neural networks classification over encrypted data," in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 97–108.

[25] A. Benaissa *et al.*, "Tenseal: A library for encrypted tensor operations using homomorphic encryption," 2021.

[26] Z. Liu *et al.*, "Trusted-DNN: A TrustZone-based adaptive isolation strategy for deep neural networks," in *ACM Turing Award Celebration Conference - China (ACM TURC 2021)*, ser. ACM TURC 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 67–71.

[27] Z. Zou *et al.*, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021.

[28] ——, "ManiHD: Efficient hyper-dimensional learning using manifold trainable encoder," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 850–855.

[29] M. Imani *et al.*, "VoiceHD: Hyperdimensional computing for efficient speech recognition," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, 2017, pp. 1–8.

[30] ——, "AdaptHD: Adaptive efficient training for brain-inspired hyperdimensional computing," in *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2019, pp. 1–4.

[31] A. Hernández-Cano *et al.*, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 56–61.

[32] Y. Kim *et al.*, "CascadeHD: Efficient many-class learning framework using hyperdimensional computing," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 775–780.

[33] B. Khaleghi *et al.*, "Prive-hd: Privacy-preserved hyperdimensional computing," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20. IEEE Press, 2020.

[34] A. Rahimi *et al.*, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–8.

[35] H. Li *et al.*, “Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition,” in *2016 IEEE International Electron Devices Meeting (IEDM)*, 2016, pp. 16.1.1–16.1.4.

[36] A. Thomas *et al.*, “A theoretical perspective on hyperdimensional computing,” *Journal of Artificial Intelligence Research*, vol. 72, pp. 215–249, 2021.

[37] J. H. Cheon *et al.*, “Homomorphic encryption for arithmetic of approximate numbers,” in *Advances in Cryptology – ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds. Cham: Springer International Publishing, 2017, pp. 409–437.

[38] Z. Huang *et al.*, “More efficient secure matrix multiplication for unbalanced recommender systems,” *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 551–562, 2023.