# Efficient Machine Learning on Encrypted Data using Hyperdimensional Computing

Yujin Nam*, Minxuan Zhou*, Saransh Gupta†, Gabrielle De Micheli*
Rosario Cammarota‡, Chris Wilkerson‡, Daniele Micciancio*, Tajana Rosing*
*Department of Computer Science and Engineering, UC San Diego, La Jolla, USA
† IBM Research, Santa Clara, USA, ‡ Intel Labs, Santa Clara, USA
{yujinnam, miz087, gdemicheli, dmicciancio, tajana}@ucsd.edu
saransh@ibm.com, {rosario.cammarota, chris.wilkerson}@intel.com

2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED) | 979-8-3503-1175-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISLPED58423.2023.10244262

***Abstract*—Fully Homomorphic Encryption (FHE) enables arbitrary computations on encrypted data without decryption, thus protecting data in cloud computing scenarios. However, FHE adoption has been slow due to the significant computation and memory overhead it introduces. This becomes particularly challenging for end-to-end processes, including training and inference, for conventional neural networks on FHE-encrypted data. Additionally, machine learning tasks require a high throughput system due to data-level parallelism. However, existing FHE accelerators only utilize a single SoC, disregarding the importance of scalability. In this work, we address these challenges through two key innovations. First, at an algorithmic level, we combine hyperdimensional Computing (HDC) with FHE. The machine learning formulation based on HDC, a brain-inspired model, provides lightweight operations that are inherently well-suited for FHE computation. Consequently, FHE-HD has significantly lower complexity while maintaining comparable accuracy to the state-of-the-art. Second, we propose an efficient and scalable FHE system for FHE-based machine learning. The proposed system adopts a novel interconnect network between multiple FHE accelerators, along with an automated scheduling and data allocation framework to optimize throughput and hardware utilization. We evaluate the value of the proposed FHE-HD system on the MNIST dataset and demonstrate that the expected training time is 4.7 times faster compared to state-of-the-art MLP training. Furthermore, our system framework exhibits up to 38.2 times speedup and 13.8 times energy efficiency improvement over the baseline scalable FHE systems that use the conventional data-parallel processing flow.**

## I. Introduction

Fully Homomorphic Encryption (FHE) is a family of encryption methods that enables computations on encrypted data. It allows data owners to securely outsource their models to servers without compromising data privacy. However, FHE faces challenges when it comes to easy deployment in production due to the significant performance degradation compared to computations on plaintext data [1]. This limitation restricts the usability of FHE in various machine learning scenarios, particularly during the training procedure. Previous research [2] has demonstrated that training a simple neural network on the MNIST dataset [3] using FHE requires over two months. Since many machine learning applications necessitate larger networks and datasets to achieve high accuracy, training with FHE becomes impractical. In addition to algorithm-level optimization, hardware acceleration plays a critical role in FHE programs because conventional systems fail to provide sufficient compute throughput and memory bandwidth for operations involving large polynomials (e.g., polynomials of degree greater than 32K). Although existing ASIC accelerators already promise to offer at least four orders of magnitude speedup compared to conventional systems [1], accelerating machine learning tasks on large datasets that require high throughput to exploit data-level parallelism remains challenging. Unfortunately, there is currently no prior work focusing on a high-throughput and scalable FHE acceleration system.

979-8-3503-1175-4/23/$31.00 ©2023 IEEE

In this work, we explore the combination of FHE with hyperdimensional Computing (HDC), a lightweight machine learning approach that has demonstrated promising performance and accuracy across a wide range of applications [4]. HDC operates by mapping data into a high-dimensional representation, leveraging this representation to enable learning capabilities. Compared to conventional training methods used in NN-based models (such as back-propagation), HDC training is considerably simpler and composed of friendly operations for FHE. Additionally, HDC models exhibit noise tolerance, which is crucial considering that FHE training introduces errors to the model. As a result, HDC offers advantages over NN-based models for FHE training. We propose both algorithmic and hardware optimizations for HDC based on FHE. We propose the first implementation of FHE for end-to-end HDC using the widely used CKKS FHE scheme [5], [6]. The CKKS scheme is commonly employed for learning tasks due to its support for real (and complex) number arithmetic and SIMD-style operations through vector encryption. Utilizing CKKS for HDC poses several challenges. Firstly, HDC requires several non-linear operations that are not directly supported by the limited operations of CKKS. Secondly, CKKS introduces approximation errors that can impact the training accuracy of HDC. To address these challenges, we introduce an efficient and noise-tolerant FHE-friendly HDC algorithm based on the CKKS scheme. In addition, we propose a novel system architecture consisting of multiple FHE ASIC chips and inter-ASIC connections. To optimize system throughput and memory utilization, we present an automated scheduling and data allocation framework. The contributions of this work are as follows:

- To the best of our knowledge, we provide the first FHE-based end-to-end HDC classification, covering all steps of HDC - encoding, training, inference, and retraining.

Authorized licensed use limited to: Access paid by The UC Irvine Libraries. Downloaded on October 31,2023 at 18:16:47 UTC from IEEE Xplore. Restrictions apply.

- We introduce an efficient FHE-friendly HDC model with good accuracy. The proposed model exhibits 4.7 times faster training compared to state-of-the-art FHE-based MLP models, while achieving over 1000 times faster inference than FHE-based RNN.
- We propose a novel FHE system with an automated scheduling and data allocation for high-throughput FHE learning. We demonstrate the efficiency of the proposed method on a scalable FHE system with state-of-the-art FHE accelerators, showing that the proposed system achieves up to 38.2 times speedup and 13.8 times energy efficiency improvement over baseline systems with conventional data-parallel processing flow.

## II. Background and Motivation

### A. Fully Homomorphic Encryption

In this work, we focus on the CKKS scheme [5]–[7] which supports processing in real numbers and SIMD packing.

#### 1) Basics of CKKS

A ciphertext in the CKKS scheme is given as a tuple $c = (b, a) \in R_Q^2$, where the polynomial rings considered are $R = \mathbb{Z}[X]/(X^N + 1)$ and $R_Q = R/QR$. For a given security parameter $\lambda$, CKKS sets the ring size $N$ and a ciphertext modulus $Q$. A ciphertext $c$ encrypts a vector $\mathbf{m}$ in $\mathbb{C}^{N/2}$ of up to $N/2$ real (or complex) elements. Applying residue number system (RNS), a large integer $Q_L$ can be decomposed into smaller primes $q_i$ where $i \in [0, L]$ and $q_0 q_1 ... q_L = Q_L$. RNS-CKKS is then a leveled homomorphic encryption method where each (ciphertext) multiplication consumes one level (or depth), thus reducing the ciphertext modulus from $Q_l$ to $Q_{l-1}$. When level 0 is reached, no further multiplications are possible. Bootstrapping [7] then allows to raise the modulus to further continue with operations.

Each homomorphic operation works in SIMD-style over the vector elements. The homomorphic operations supported by the CKKS scheme can be explained as follows, where $\circ$ denotes element-wise multiplication and $pk$ is a public key:

- $\texttt{HomAdd}\,(c_0, c_1) = \texttt{Enc}_{pk}(\mathbf{m}_0 + \mathbf{m}_1)$
- $\texttt{HomMult}\,(c_0, c_1) = \texttt{Enc}_{pk}(\mathbf{m}_0 \circ \mathbf{m}_1)$
- $\texttt{HomRot}\,(c_0, k) = \texttt{Enc}_{pk}((m_k, m_{k+1}, ..., m_{k-1}))$ circularly rotates the message by $k$ slots.

#### 2) On the security of CKKS

As this paper focuses mostly on efficiency and performance, we refer the readers to [8], [9] for discussions on the security of the CKKS scheme.

### B. Hyperdimensional Computing (HDC)

HDC based classification first maps the input data into hypervectors (high dimensional vectors). The rest of the steps in HDC - training, inference and retraining - use hypervectors.

#### 1) Encoding

The encoding procedure converts an input vector $F$ with $f$ features into a hypervector $HV$ of dimension $D$. We used a commonly used encoding, random projection with nonlinear activation [10]. A hypervector $HV$ can be evaluated as $HV = \sigma(A \cdot F)$, where $A$ is a randomly chosen matrix in $\{-1, 1\}^{D \times f}$ and $\sigma()$ applies a chosen nonlinear function to each element of the input vector.

#### 2) Training

After all the training data are mapped to high-dimensional space, training can be done by adding up the hypervectors that belong to the same class. We call the generated hypervectors as class hypervectors $classHV_i$.

#### 3) Inference

To infer a query, the input is first mapped to a query hypervector using the same encoding method. The result label is then decided as $\texttt{argmax}(\delta(queryHV, classHV_i))$, where $\delta(\cdot, \cdot)$ is a similarity metric between two hypervectors. We used cosine similarity as it is commonly used.

#### 4) Retraining

Retraining improves the accuracy of a model. When a train hypervector $trainHV$ with the correct label $c$ infers to a wrong label $w$, class hypervectors are updated as follows with learning rate $r$:

$$classHV_c \leftarrow classHV_c + r \cdot trainHV$$
$$classHV_w \leftarrow classHV_w - r \cdot trainHV$$

### C. FHE Accelerator

There have been various accelerator proposals for the CKKS FHE scheme [1], [11], [12]. These accelerators achieve significant performance improvement over conventional architectures by exploiting high-throughput function units, large on-chip storage, high-bandwidth memory, and algorithmic optimization. However, existing work focuses on single SoC, ignoring the scalability of FHE acceleration for applications with high data-level parallelism, like machine learning. There exist several challenges in designing a scalable FHE system, including 1) interconnect network to keep a high bandwidth when increasing FHE accelerators, 2) operation scheduling on different accelerators, and 3) data allocation on the distributed memory. Section IV introduces the details of our scalable FHE system.

## III. FHE-HD Algorithm

We propose FHE-friendly algorithm for HDC classification, FHE-HD. The overall algorithm is described in Figure 1.

### A. Data Encoding

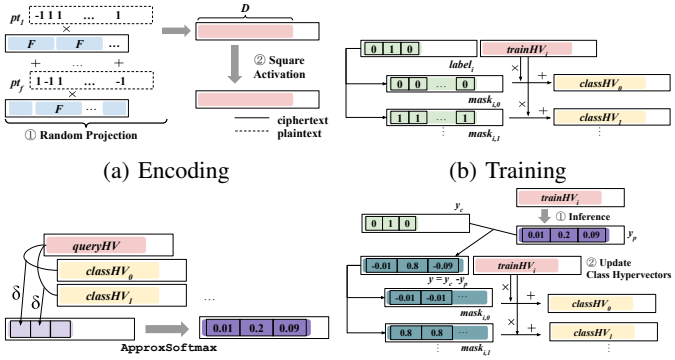
The data is first encoded into high dimensional space. We used random projection followed by an activation function.

#### 1) Random Projection

Random projection multiplies a random projection matrix $A = \{-1, 1\}^{D \times f}$ with a feature vector $F$ of size $f$. The random projection matrix does not need to be encrypted as it is generated randomly and contains no sensitive information. A naive method would be making $D$ different plaintext vectors and evaluating dot projects between an encrypted vector $F$ and plaintexts. However, this is inefficient as a dot product generates only one element of the result and it wastes the majority of available slots. Also because $D$ is large, many plaintext vectors and dot product operations are required.

We propose an efficient method for random projection that takes advantage of every slots in a ciphertext while requiring

Authorized licensed use limited to: Access paid by The UC Irvine Libraries. Downloaded on October 31,2023 at 18:16:47 UTC from IEEE Xplore. Restrictions apply.




(a) Encoding (b) Training

(c) Inference (d) Retraining

Fig. 1: **FHE-HD algorithm.**

minimal number of operations. The random projection process can be carried out by encrypting the repetitions of vector $F$ using every available slots in the ciphertext $c$. We evaluate rotations of the ciphertext, $\texttt{HomRot}(c, i)$, $i = 0, ...f-1$. Then we multiply each rotated ciphertext with a random plaintext vector and add up all the results. This process is described in Figure 1a. We further reduced the number of rotations using the baby-step-giant-step method [13].

*2) Activation Function*

The CKKS scheme can only evaluate linear functions because it supports limited arithmetic operations (e.g., addition and multiplication). Therefore, to evaluate a nonlinear function, a polynomial approximation of those functions is needed. Previous works approximated activation functions with low-degree polynomials, like sigmoid [14] or ReLu [15]. Other works [16] replaced activation functions with square activation, i.e., $f(x) = x^2$. Previous work [10] used $cos$ and $sign$ for activation in HDC. To minimize the depth consumption of the activation, we used square activation. We empirically found that square activation has similar accuracy to other nonlinear activations.

### B. Label Encoding

For data privacy, the labels should be encrypted. Labels for classification are typically integers from 0 to $l-1$, where $l$ is the total number of labels. However, directly encrypting these integer values into ciphertexts incurs comparisons between integer labels with high computational costs. We utilized one-hot encoding for the labels, where a label $i$ is represented as a vector of length $l$ whose the $i$-th element is 1 and others are 0. Then, using the SIMD style encryption of CKKS, we encrypted this one-hot encoded vector as one ciphertext. Later in the training process, we use the encrypted labels to create mask ciphertexts, which eliminates the need for comparison.

### C. Training

Once training data are mapped to hypervectors, we need to add up the train hypervectors for each class. We create mask ciphertexts by extending each slot of a one-hot encoded label ciphertext. This refers to copying a value from one slot to a encrypted hypervector's $D$ slots, using plaintext multiplication, rotations and additions. Our model is then trained by creating a $j$-th class hypervector performing $\sum_i mask_{i,j} \times trainHV_i$, where $mask_{i,j}$ is a mask generated by copying the $j$-th slot of the $i$-th data label. Due to the inability to execute different branches based on encrypted values, certain redundant operations with masks are introduced. Despite this, the proposed method avoids costly comparison operations that are not inherently supported by the CKKS scheme.

### D. Inference

We replaced `argmax` with `ApproxSoftmax`, which enabled us to compare values while avoiding expensive comparison operations. For the similarity metric, we used cosine similarity: $\delta(queryHV, classHV_i) = \langle queryHV, classHV_i \rangle / \|classHV_i\|$. Inferring a query includes computing a dot product, an inverse of square root, and `ApproxSoftmax`. Dot product can be computed easily using homomorphic multiplication, rotations and additions.

*1) Approximate Inverse and Inverse of Square Root*

The inverse function $f(x) = \frac{1}{x}$ and the inverse of square root function $f(x) = \frac{1}{\sqrt{x}}$ can be used for division and division by $l_2$-norm of class hypervectors. Because CKKS does not support nonlinear operations, we need to use an approximation approach. We used Goldschmidt's iterative method [17] for both functions.

*2) Approximate Softmax*

We proposed a hybrid approximation of `argmax` and `softmax`, in place of the `argmax` function of the initial HDC algorithm. For CNN models, Lee et al. [15] approximated `softmax` function. In the region of [-1,1], they used a degree-13 polynomial approximation of the exponent function. Cheon et al. [18] proposed `argmax` (or `MaxIdx`) procedure, which iteratively approximates $\frac{a_j^k}{a_1^k + \cdots + a_n^k}$ for large $k$. Both works used Goldschmidt's method for division operation.

We empirically verified that the `softmax` approximation from Lee et al. alone is insufficient to provide precision for the retraining procedure. On the other hand, Cheon et al. iteratively perform inverse operations, which uses up multiplication depths. As a result, we suggest a hybrid approach, `ApproxSoftmax`, that combines two earlier research. First, we evaluate the approximation of $e^{x/\lambda}$, using $\lambda$ to match the approximation region. Repeating square operations for $t$ times, we can evaluate $(e^{x/\lambda})^{2^t}$. We applied division $(a_j/(n \cdot \sum_{i=1}^{n} a_i))$ from Cheon et al. once in between the squaring to prevent the overflow. The result is obtained by dividing each value by the sum of the values.

*3) Packed Inference*

We made full use of SIMD operation by packing multiple hypervectors into one ciphertext and evaluating the dot product for cosine similarity. We also inferred multiple queries simultaneously by placing their cosine similarities in one ciphertext.

### E. Retraining

As stated in II-B4, the retraining is carried out by adjusting the class hypervectors with inference results. We can retrain the model using the predicted label and the ground truth label, which is an one-hot encoded label. To elaborate, let us assume that $y_c$ is the ground truth label, $y_p$ is the predicted label, and $y = y_c - y_p$. By performing

Authorized licensed use limited to: Access paid by The UC Irvine Libraries. Downloaded on October 31,2023 at 18:16:47 UTC from IEEE Xplore. Restrictions apply.

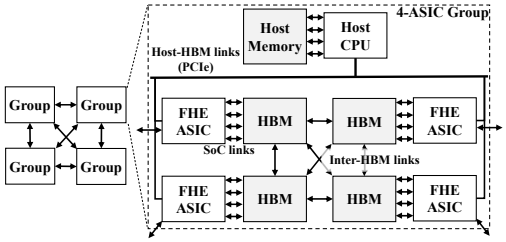



Fig. 2: **The scalable FHE system.**

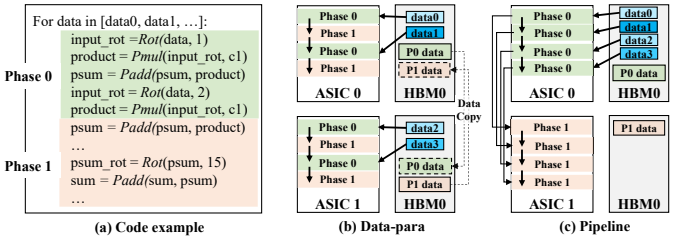



Fig. 3: **Scheduling and data allocation schemes.**

$classHV_j \leftarrow classHV_j + r \cdot y[j] \cdot trainHV_i$ for every possible $j$, we can retrain the class hypervectors. Note that multiplying by a single slot $y[j]$ can be achieved through the same mask generation process of the training step. Because we used `ApproxSoftmax` for inference, the prediction result $y_p$ is a vector of values in $[0, 1]$ rather than 0 or 1. This approach updates each class hypervector while taking into account their similarity with the query, as opposed to only updating two class hypervectors. Comparing this method to the original HDC algorithm, we confirmed that it still provides acceptable accuracy.

### *F. Computaion-Comunication Trade-Off*

We can consider different scenarios to reduce the server side computation. For example, encoding can be done in the client side in plaintext and the server can receive encrypted hypervectors. Also every training masks can be created on the client and sent to the server. This would raise the communication cost as more ciphertexts need to be sent while improving the computation efficiency on the server side by skipping the encoding or mask generation process.

## IV. Scalable FHE-HD System

Machine learning tasks, including inference and training, feature high data-level parallelism. Thus, the scalability of the underlying system is critical to the end-to-end performance of machine learning tasks. In this section, we first introduce a scalable FHE system with FHE accelerators. Then, we propose several operation scheduling and data allocation schemes that can be exploited by the runtime to achieve the best performance for different FHE-HD phases.

### *A. Scalable FHE System*

Figure 2 shows the scalable FHE systems, consisting of multiple FHE ASIC chips. Each FHE ASIC is connected to a High-Bandwidth Memory (HBM). For the scalable system, we connect all FHE ASICs in a dragonfly interconnect, similar to previous near-memory acceleration [19]. Therefore, all FHE ASICs are organized into multiple 4-ASIC groups, where four ASICs in each group are fully connected through the inter-memory links. Even though each ASIC can assess its local HBM through fast links, the ASIC HBM cannot hold the extremely large FHE-HD data, including the encrypted dataset and FHE keys. In this case, each group contains a host CPU and a host memory. The host stores the whole FHE-HD data and transfers the required data to ASIC HBM. The scalable system supports various types of FHE ASIC, which are treated as black boxes. Our contribution lies in the overall system design with efficient operation scheduling and data allocation schemes. In the following subsections, we introduce different methods of operation scheduling and data allocation that can fully utilize the scalable FHE system. Furthermore, we propose an automated framework to optimize the scheduling and data allocation in large-scale FHE systems for HDC.

### *B. Operation Scheduling*

The operation scheduling scheme indicates the way of distributing FHE-HD applications across all FHE ASICs in the system, critical to the system's efficiency. A straightforward operation scheduling is data-parallel, as shown in Figure 3(b) for an example code (Figure 3(a)). In this example, the algorithm processes different data points with identical operations, which is the common pattern in HD inference, training, and encoding phases. The data-parallel scheduling distributes the computation of different data points to different ASICs. In this case, each ASIC processes the end-to-end process for a subset of encrypted data points as well as FHE. The advantage of data-parallel scheduling is we can balance the workloads on different FHE ASICs. However, data-parallel scheduling requires each ASIC to load all FHE materials throughout the end-to-end process, significantly increasing the memory footprint, and hence increasing the host-ASIC data loading. The host-ASIC is slow due to the limited bandwidth of the host-ASIC link that is based on PCIE.

An alternative scheduling scheme is pipelining which allocates operations of phases to different FHE ASICs, shown in Figure 3(c). In the pipeline scheduling, each ASIC avoids loading all FHE materials to its local HBM. However, the pipeline scheduling requires data transfers between different FHE ASIC HBM. Furthermore, pipeline scheduling needs to carefully balance the memory footprint as well as operation latency among different ASICs. In general, the pipeline scheduling may suffer from significant inefficiency due to the overwhelming inter-ASIC transfers and unbalanced pipeline.

### *C. Data Allocation*

Another problem of running FHE-HD applications on scalable FHE systems is data allocation where we need to determine how to allocate data in different ASIC HBMs. The naive way is to treat all ASIC HBMs as a unified memory pool, where only one copy for each data can be loaded. For example, in Figure 3(b), P0 data is only stored in HBM0 so ASIC1 needs to load P0 data from ASIC0 to its on-chip memory (i.e., scratchpad) if needed. Such a no-copy method can minimize the memory footprint. However, it introduces a lot of inter-ASIC data transfers that can significantly slow down the system. Another data allocation method is to maintain multiple

Authorized licensed use limited to: Access paid by The UC Irvine Libraries. Downloaded on October 31,2023 at 18:16:47 UTC from IEEE Xplore. Restrictions apply.

copies of data in the AISCs that need the data. In this case, each ASIC can access its local HBM for all needed data. However, the data copy introduces coherence issues as well as increases the memory footprint, increasing the inter-ASIC and host-ASIC transfers. We propose a hybrid data allocation scheme that only allows the data copy for read-only data with a high reuse rate. For example, all the key-switching keys, including homomorphic rotation and multiplication, are read-only and highly reusable among different ASICs. With the hybrid data allocation, we can avoid a large number of inter-ASIC and host-ASIC data transfers while maintaining a relatively low memory footprint.

### D. System Support

We implement an offline exploration framework to determine the best combination of scheduling and data allocation for each FHE-HD step. In addition to the high-level exploration, the framework optimizes the detailed scheme for pipeline and hybrid data allocation. For the pipeline scheduling, we need to optimize the pipeline phase segmentation to balance the overall latency. For a given program on one data point, consisting of a sequence of FHE instructions, we use a random search with trimming to approximately optimize the pipeline. For each searched pipeline, we run a simulation to determine its goodness. For hybrid data allocation, our framework checks all data in the input program and determines which data can be duplicated. Our framework runs in the offline stage and will not affect the production runtime for FHE-HD applications because the optimized processing can be used for any new data points without re-optimization.

## V. Results

### A. Experimental Setup

We implemented our framework using OpenFHE [20] library. Our implementation was tested with Intel Core i7-8700K processor and 64GB memory. We evaluated our algorithm and hardware design on MNIST [3] and ISOLET [21] datasets with hypervectors of size 8192. We chose a minimum power of 2 hypervector dimensionality that gives an acceptable accuracy to maximize the utilization of packing method. For CKKS parameters, we used uniform ternary secret distribution following the Standard [22], ring dimension $N = 2^{16}$, ciphertext prime $logQ = 1770$ and 53-bit scaling factor, which satisfies $\lambda = 128$-bit security [23]. We used the iterative bootstrapping of the library for extra precision of our workload. We used ARK [1], which is the state-of-the-art FHE accelerator, as the FHE ASIC in our scalable system. Specifically, each ARK ASIC has a 2048-lane processing pipeline for FHE operations and 512MB on-chip scratchpads to buffer the data. Each ARK ASIC is connected to two HBM2 stacks which have a total of 16GB capacity and 512GB/s bandwidth. The host-ASIC and inter-ASIC connection follows the dragonfly interconnect network, similar to previous near-data processing acceleration [19]. We assume a total 32GB/s PCIe bandwidth (shared by all hosts) and 40GB/s inter-ASIC bandwidth for each link. For the latency, energy, and power evaluation, we use the reported values from previous work [1], [19] for different operations. For evaluation, we implement an in-house simulator that can simulate instruction traces of FHE operations. We implement all hardware components and corresponding mechanisms to accurately calculate the number of operations, including multiplications, NTTs, scratchpad load/store, HBM load/store, and inter-ASIC communication.

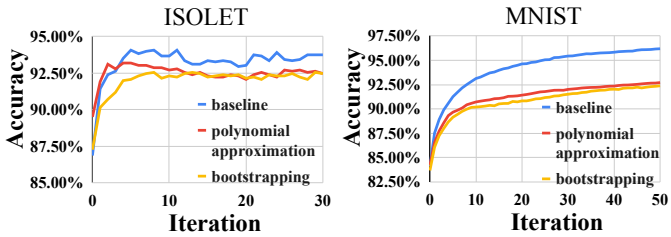



Fig. 4: **Model accuracy by retraining epochs. The baseline HDC is the initial algorithm described in Section II-B with square activation. "polynomial approximation" uses proposed approximations, and "bootstrapping" adds bootstrapping error to the "polynomial approximation".**

TABLE I: **Amortized latency of each step per data point.**

| **Latency** (s) on 48-thread CPU | | | |
|---|---|---|---|
| **Encoding** | **Training** | **Retraining** | **Inference** |
| ISOLET / MNIST<br>4.67 / 5.76 | 6.28 | 16.59 | 10.31 |

### B. Accuracy

The accuracy degradation of the model was evaluated in plaintext. We empirically measured the magnitude of bootstrapping error and simulated it. Figure 4 shows the model's accuracy by retraining epochs. For the ISOLET dataset, the model only lost around 1% accuracy compared to the baseline HDC algorithm with both polynomial approximation and bootstrapping errors. Because of the noise robustness of the HDC algorithm, we were able to achieve minimal accuracy degradation with FHE-HD algorithm. For the MNIST dataset, the accuracy loss was about 3% with polynomial approximation and 3.8% with bootstrapping error. Most of the accuracy loss was due to polynomial approximation rather than the bootstrapping error. This can be improved by a better approximation of `argmax`. As we are focusing on the efficiency of the task in this paper, we leave it as further work.

### C. Latency

Table I reports the latency of each step in our HDC model. Note that the encoding latency differs as it depends on $\sqrt{f}$ in random projection. Other operations are the same because they operate in the same high dimensional space. Table II compares our training latency with previous FHE-based training models, which train the entire model without other techniques like transfer learning. Compared to the previous MLP works, our HDC trains 4.7× and 5.8× faster. We assumed that the data encoding and mask generation for training was done by the client. But even when assuming they are done by the server, it only increases the total training time by 1.5 days. Also compared to the FHE-based RNN model which takes 49 minutes to inference an image, our work takes 2.58 seconds.

Authorized licensed use limited to: Access paid by The UC Irvine Libraries. Downloaded on October 31,2023 at 18:16:47 UTC from IEEE Xplore. Restrictions apply.

TABLE II: **Latency comparison with previous works using MNIST dataset on CPU (48 threads). Inference latency is an amortized result.**

| Work | #epoch | Latency | | |
|---|---|---|---|---|
| | | 1-epoch | Total training | Inference |
| MLP 1 [2] | 5 | 14 days | 69.9 days | - |
| MLP 2 [24] | 5 | 17.4 days | 86.8 days | - |
| **FHE-HD** | 5 | 2.9 days | 15 days | 2.58 s |
| RNN [25] | - | - | - | 49 min |

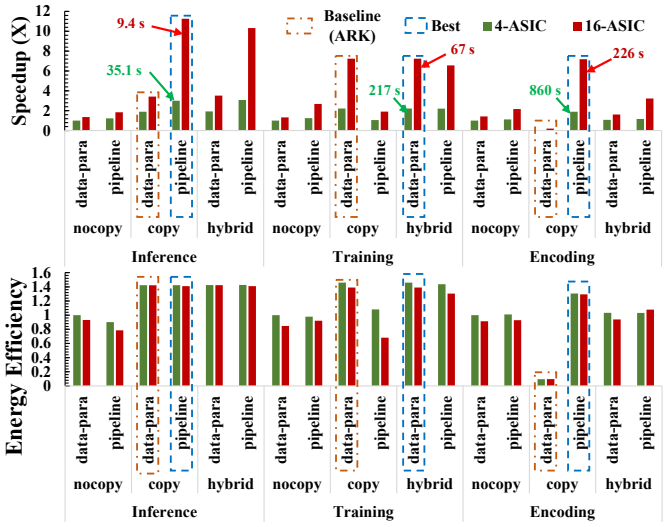



Fig. 5: **The performance and energy efficiency of various hardware configurations. The data-para with a "copy" data allocation is the conventional way of utilizing multiple accelerators for data-parallel execution, which is similar to the straightforward usage of multiple ARK chips [1].**

### *D. Hardware Acceleration*

Figure 5 shows the performance and energy efficiency results of FHE-HD hardware acceleration using various scheduling and data allocation schemes. In the 4-ASIC system, the best scheme, explored by our framework, is 3.1×, 2.2×, and 1.9× faster while consuming 1.4×, 1.5×, and 1.3× less energy than the naive data-parallel processing with no-copy data allocation on HDC inference, training, and encoding respectively. We also observe good scalability when we increase the number of ASICs from 4 to 16. Specifically, the best scheme on the 16-ASIC system is 3.7×, 3.3×, and 3.8× faster than the 4-ASIC system, achieving almost linear speedup with the number of ASICs. As compared to the baseline (data-para copy (ARK) [1]), the best solution achieves 1.6×, 1.1×, and 38.2× speedup. The reason for the extremely slow encoding of the baseline is encoding requires a large number of rotation keys, where the copy-based data allocation significantly increases the memory footprint, thus introducing large host-ASIC communication overhead. Overall, the proposed system can finish FHE-HD inference, training, and encoding on 60,000 MNIST data points in 35.1 (9.4) s, 217 (67) s, and 860 (226) s on the 4-ASIC (16-ASIC). As for energy, the proposed system consumes up to 13.8× less energy than the baseline system.

## VI. Conclusion

We proposed an efficient algorithm-hardware optimization, FHE-HD, for FHE-based HDC classification. First, we suggested the first FHE-based HDC model using polynomial approximation and packing method. Our model shows 4.7× faster training and over 1000× faster inference compared to the state-of-the-art works. Furthermore, we suggested a scalable FHE accelerator system considering the scheduling and data allocation of HD workload, which achieved up to 38.2× speedup compared to a state-of-the-art.

## Acknowledgment

This work was supported in part by PRISM and CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA, SRC Global Research Collaboration (GRC) grants, and NSF grants #1826967, #1911095, #2003279, #2052809, #2112665, #2112167, and #2100237.

## References

[1] J. Kim *et al.*, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *MICRO '22*.
[2] Q. Lou *et al.*, "Glyph: Fast and accurately training deep neural networks on encrypted data," *NeurIPS '20*, vol. 33, pp. 9193–9202, 2020.
[3] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.
[4] M. Imani *et al.*, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *HPCA '21*.
[5] J. H. Cheon *et al.*, "Homomorphic encryption for arithmetic of approximate numbers," in *Asiacrypt '17*. Springer, 2017, pp. 409–437.
[6] J. H. Cheon *et al.*, "Bootstrapping for approximate homomorphic encryption," in *Eurocrypt '18*. Springer, 2018, pp. 360–384.
[7] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," in *CT-RSA'20*.
[8] B. Li and D. Micciancio, "On the security of homomorphic encryption on approximate numbers," in *EUROCRYPT '21*. Springer.
[9] B. Li *et al.*, "Securing approximate homomorphic encryption using differential privacy," in *CRYPTO '22*. Springer, 2022, pp. 560–589.
[10] Z. Zou *et al.*, "Manihd: Efficient hyper-dimensional learning using manifold trainable encoder," in *DATE '21*. IEEE, 2021, pp. 850–855.
[11] N. Samardzic *et al.*, "Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data," in *ISCA '22*, 2022.
[12] R. Cammarota, "Intel HERACLES: homomorphic encryption revolutionary accelerator with correctness for learning-oriented end-to-end solutions," in *CCSW*, 2022.
[13] S. Halevi and V. Shoup, "Faster homomorphic linear transformations in helib," in *CRYPTO '18*. Springer, 2018, pp. 93–120.
[14] K. Han *et al.*, "Logistic regression on homomorphic encrypted data at scale," in *AAAI'19*.
[15] J.-W. Lee *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, 2022.
[16] A. Brutzkus *et al.*, "Low latency privacy preserving inference," in *ICML '19*.
[17] R. E. Goldschmidt, "Applications of division by convergence," Ph.D. dissertation, Massachusetts Institute of Technology, 1964.
[18] J. H. Cheon *et al.*, "Numerical method for comparison on homomorphically encrypted numbers," in *Asiacrypt '19*.
[19] J. Ahn *et al.*, "A scalable processing-in-memory accelerator for parallel graph processing," in *ISCA '15*, 2015, pp. 105–117.
[20] A. A. Badawi *et al.*, "Openfhe: Open-source fully homomorphic encryption library," Cryptology ePrint Archive, Paper 2022/915, 2022.
[21] D. Dua and C. Graff, "Uci machine learning repository," 2017.
[22] M. Albrecht *et al.*, "Homomorphic encryption security standard," HomomorphicEncryption.org, Toronto, Canada, Tech. Rep., November 2018.
[23] B. R. Curtis and R. Player, "On the feasibility and impact of standardising sparse-secret lwe parameter sets for homomorphic encryption," Cryptology ePrint Archive, Paper 2019/1148.
[24] K. Nandakumar *et al.*, "Towards deep neural network training on encrypted data," in *CVPR Workshops*, June 2019.
[25] J. Jang *et al.*, "Privacy-preserving deep sequential model with matrix homomorphic encryption," in *ASIA CCS '22*, 2022, pp. 377–391.

Authorized licensed use limited to: Access paid by The UC Irvine Libraries. Downloaded on October 31,2023 at 18:16:47 UTC from IEEE Xplore. Restrictions apply.