A Framework for Collaborative Learning in Secure High-Dimensional Space

Mohsen Imani*, Yeseong Kim*, M. Sadegh Riazi, John Messerly, Patric Liu, Farinaz Koushanfar, Tajana Rosing
University of California San Diego, La Jolla, CA 92093, USA
{moimani, yek048, mriazi, jmesserl, ptliu, fkoushanfar, tajana}@ucsd.edu

***Abstract*—As the amount of data generated by the Internet of the Things (IoT) devices keeps increasing, many applications need to offload computation to the cloud. However, it often entails risks due to security and privacy issues. Encryption and decryption methods add to an already significant computational burden. In this paper, we propose a novel framework, called SecureHD, which provides a secure learning solution based on the idea of high-dimensional (HD) computing. We encode original data into secure, high-dimensional vectors. The training is performed with the encoded vectors. Thus, applications can send their data to the cloud with no security concerns, while the cloud can perform the offloaded tasks without additional decryption steps. In particular, we propose a novel HD-based classification algorithm which is suitable to handle a large amount of data that the cloud typically processes. In addition, we also show how SecureHD can recover the encoded data in a lossless manner. In our evaluation, we show that the proposed SecureHD framework can perform the encoding and decoding tasks 145.6× and 6.8× faster than a state-of-the-art encryption/decryption library running on the contemporary CPU. In addition, our learning method achieves high accuracy of 95% on average for diverse practical classification tasks including cloud-scale datasets.**

***Keywords*-Machine learning; Secure learning; Brain-inspired computing; Hyperdimensional computing; Distributed learning**

I. Introduction

Internet of Things (IoT) applications often analyze collected data using machine learning algorithms. As the amount of the data keeps increasing, many applications send the data to powerful systems, e.g., data centers, to run the learning algorithms [1], [2], [3]. On the one hand, sending the original data is not desirable due to privacy and security concerns [4], [5], [6], [7]. On the other hand, many machine learning models require unencrypted (plaintext) data, e.g., original images, to train models and perform inference. When offloading the computation tasks, sensitive information is exposed to the *untrustworthy cloud system* which is susceptible to internal and external attacks [8], [9], [10] . In many IoT systems, the learning procedure should be performed with the data that is held by a large number of *user devices* at the edge of the Internet. These users may be unwilling to share the original data with the cloud and other users [11], [12], [13], [14].

An existing strategy applicable to this scenario is exploiting Homomorphic Encryption (HE). HE enables encrypting the raw data and allowing certain operations to be performed directly on the ciphertext without decryption [15]. However, this approach significantly increases computation burden in addition to the costly learning procedures. For example, in our evaluation, with Microsoft SEAL, a state-of-the-art homomorphic encryption library [16], it takes around 14 days to encrypt all of the 28x28 pixel images in the entire MNIST dataset, and increases the data size by 28 times. More recently, Google presented a protocol for secure aggregation of high-dimensional data that can be used in *federated* learning [17]. This approach enables training Deep Neural Networks (DNN) when data is distributed over different users. In this technique, the users' devices need to perform the DNN training task locally to update the global model. However, IoT edge devices often do not have enough computation resources to perform complex DNN training.

This paper proposes a novel secure technique that enables efficient, scalable, and secure collaborative learning in high-dimensional space. Instead of using conventional machine learning algorithms, we exploit High-Dimensional (HD) computing [18] to perform the learning tasks in a secure domain. HD computing does not require complete knowledge for the original data that the conventional learning algorithms need – it runs with a mapping function that encodes a given data to a high-dimensional space that mimics massive numbers of neurons and synapses in brains. The original data cannot be reconstructed from the mapped data without knowing the mapping function, since a single value can be represented with extremely huge possibilities in the high-dimensional space.

Along with the attractive properties for secure learning, HD computing also offers additional benefits. HD provides an efficient learning strategy without complex computations such as back propagation in neural networks. In addition, the HD-based learning models are extremely robust in the presence of hardware failures due to the independence of dimensions in the computation. Prior work shows that HD computing can be applied to many cognitive tasks such as analogy-based reasoning [19], latent semantic analysis [20], language recognition [21], [22], and speech/object recognition [23], [24].

We address several technical challenges to enable HD-based *trustworthy, collaborative* learning. For example, to map the original data into high-dimensional vectors, called *hypervectors*, it exploits a set of *base hypervectors* which

* Mohsen Imani and Yeseong Kim contributed equally to this research.

are randomly generated. Since the base hypervectors can be used to estimate the original data, every user has to have different base hypervectors to ensure the confidentiality of the data. However, in this case, the HD computation cannot be performed with the data provided by different users. The other challenge is that the existing HD learning methods do not scale with the size of data. This highly limits the applicability to HD-based computing on cloud-scale services.

In this paper, we design a novel framework, called SecureHD, which fills the gap between the existing HD computing and trustworthy, collaborative learning. We present the following contributions:

i) We design a novel *secure collaborative learning protocol* that securely generates and distributes public and secret keys. SecureHD utilizes Multi-Party Computation (MPC) techniques which are proven to be secure when each party is untrusted [25]. With the generated keys, the user data are not revealed to the cloud server, while the server can still learn a model based on the data encoded by users. Since MPC is an expensive protocol, we carefully optimize it by replacing a part of tasks with two-party computation. In addition, our design leverages MPC only for a one-time key generation operation. The rest of the operations such as encoding, decoding, and learning are performed without using MPC.

ii) We propose a new encoding method that maps the original data with the secret key assigned to each user. Our encoding method significantly improves classification accuracy as compared to the state-of-the-art HD work [22], [26]. Unlike existing HD encoding functions, the proposed method encodes both the data and the metadata, e.g., data types and color depths, in a recover-friendly manner. Since the secret key of each user is not disclosed to anyone, although one may know encoded data of other users, they cannot be decoded.

iii) SecureHD provides a robust decoding method for the authorized user who has the secret key. We show that the cosine similarity metric widely used in HD computing is not suitable to recover the original data. We propose a new decoding method which recovers the encoded data in a lossless manner through an iterative procedure.

iv) We present scalable HD-based classification methods for many practical learning problems which need the collaboration of many users, e.g., human activity and face image recognition. We propose two collaborative learning approaches, cloud-centric learning for the case that end-node devices do not have enough computing capability, and edge-based learning that all the user devices participate in secure distributed learning.

v) We also show a hardware accelerator design that significantly minimizes the costs paid for security. This enables secure HD computing on less-powerful edge devices, e.g., gateways, which are responsible for data encryption/deception.

We design and implement the proposed SecureHD framework on diverse computing devices in IoT systems, including a gateway-level device, a high-performance system, and our proposed hardware accelerator. In our evaluations, we show that the proposed framework can perform the encoding and decoding tasks 145.6× and 6.8× faster than a state-of-the-art homomorphic encryption library when both are running on the Intel i7-8700K. The hardware accelerator further improves the performance efficiency by 35.5× and 20.4× as compared to the CPU-based encoding and decoding of SecureHD. In addition, our classification method presents high accuracy and scalability for diverse practical problems. It successfully performs learning tasks with 95% average accuracy for six real-world workloads, ranging from datasets collected in a small IoT network, e.g., human activity recognition, to a large dataset which includes hundreds of thousands of images for the face recognition task. Our decoding method also provides high quality in the data recovery. For example, SecureHD can recover the encoded data in a lossless manner, where the size of the encoded data is 4 times smaller than the one encrypted by the state-of-the-art homomorphic encryption library [27].

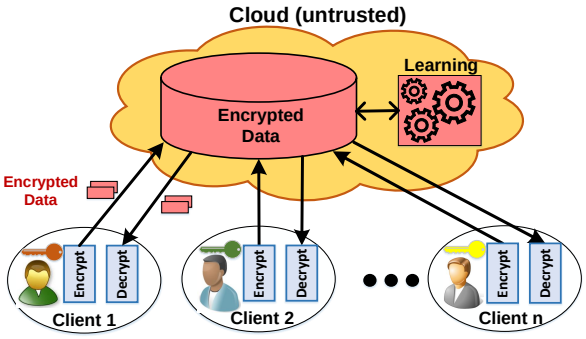


Figure 1. Motivational scenario

II. Motivation and Background

A. Motivational Scenario

Figure 1 shows the scenario that we focus in this paper. The clients, e.g., user devices, send either their sensitive data or partially trained models in an encrypted form to the cloud. The cloud performs a learning task by collecting the encrypted information received from multiple clients. In our security model, we assume that a client cannot trust the cloud as well as other clients. When requested by the user, the cloud sends back the encrypted data to clients. The client then decrypts the data with its private key.

As an existing solution, homomorphic encryption enables processing on the encrypted version of data [15]. Figure 2 shows the execution time of a state-of-the-art homomorphic encryption library, Microsoft SEAL [16], for MNIST training dataset, which includes 60000 images of 28×28 pixels. We execute the library on two platforms that a client in IoT systems may use, a high-performance computer (Intel i7-8700K) and a Raspberry Pi 3 (ARM Cortex A53). The result shows that, even with the simple dataset of 47 MBytes, it takes significantly large execution time, e.g., more than 13 days on ARM to encrypt.

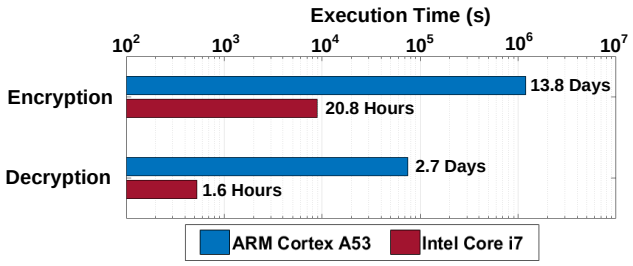


Figure 2. Execution time of homomorphic encryption and decryption over MNIST dataset

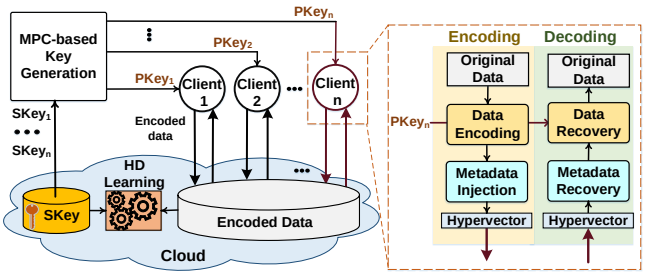


Figure 3. Overview of SecureHD

Another approach is to utilize secure Multi-Party Computation (MPC) techniques [25], [28]. In theory, any function, which can be represented as a Boolean circuit with inputs from multiple parties, can be evaluated securely without disclosing each party's to anyone else. For example, by describing the machine learning algorithm as a Boolean circuit with learning data as inputs to the circuit, one can securely learn the model. However, such solutions are very costly in practice and are computation and communication intensive. In SecureHD, we only use MPC to securely generate and distribute users' private keys which is orders of magnitude less costly than performing the complete learning task using MPC. The key generation step is a one-time operation so the small cost associated with it is quickly amortized over time for future tasks.

B. Background: HD Computing

Brain-inspired high-dimensional (HD) computing performs cognitive tasks using ultra-wide words – that is, with very high-dimensional vectors, also known as *hypervectors* [18], [29]. Hypervectors are holographic and (pseudo)random with independent identically distributed (i.i.d.) components. A hypervector may contain multiple values by spreading them across its components in full holistic representation. In this case, no component in a hypervector is more responsible for storing any piece of information than others. These unique features make hypervectors robust against errors in their components.

Hypervectors are implemented with high-dimensional operations, such as *binding* that forms a new hypervector which associates two hypervectors, and *bundling* that combines hypervectors into a single composite hypervector. (i) The binding of two hypervectors A and B is denoted as $A*B$. The result of the operation is a new hypervector that is dissimilar to its constituent vectors. For bipolar hypervectors ($\{-1,+1\}^D$), the component-wise multiplication performs the binding operation. For example, let us consider two hypervectors randomly generated, i.e., each component has either -1 or +1 with 50:50 chance. Since they are near-orthogonal, their binding has approximately zero similarity each other, i.e., $\delta(A*B,A) \approx 0$ where δ is a function that computes the cosine similarity. (ii) Bundling operation is denoted as $A \odot B$ and preserves similarity to its component hypervectors. The component-wise addition implements the bundling for bipolar hypervectors. For example, the bundling of two random hypervectors keeps the information, i.e., $\delta(A \odot B,A) \approx cos(\pi/4)$.

Many learning tasks can be implemented in the HD domain. To implement a simple classification algorithm, a data point in a given training dataset is first converted into a hypervector. This step is often referred to as *encoding*. The encoding exploits a set of orthogonal hypervectors, called *base hypervectors*, to map each data point into the HD space. Then, it bundles the encoded hypervectors for each class. The reasoning (inference) can be performed by choosing the class whose bundled hypervector presents the highest similarity to an unseen hypervector [18].

III. Secure Learning in HD Space

A. Security Model

In SecureHD, we consider the server and other clients to be untrusted. More precisely, we consider Honest-but-Curious (HbC) adversary model where each party, server or a client, is untrusted but follows the protocol. Both the server and other clients are not able to extract any information based on the data that they receive and send during the secure computation protocol. For the task of key generation and distribution, we utilize a secure MPC protocol which is proven to be secure in the HbC adversary model [25]. We also use two-party Yao's Garbled Circuits (GC) protocol which is also to be secure in the HbC adversary model as well [30]. The intermediate results are stored as additive unique shares of PKey by each client and the server.

B. Proposed Framework

In this section, we describe the proposed SecureHD framework which enables trustworthy, collaborate HD computing. Figure 3 illustrates the overview of SecureHD. The first step is to create different keys for each user and cloud-based on an MPC protocol. As discussed in Section II-B, to perform a HD learning task, the data are encoded with a set of base hypervectors. The MPC protocol creates the base hypervectors for the learning application, called global keys (*GKeys*). Instead of sharing the original GKeys with clients, the server distributes permutations of each GKey, i.e., a hypervector whose dimensions are randomly shuffled. Since each user has different permutations of GKeys, called personal keys (*PKeys*), no one can decode encoded data of others. The cloud has dimension indexes used in the GKey shuffling, called shuffling keys (*SKeys*). Since the cloud does

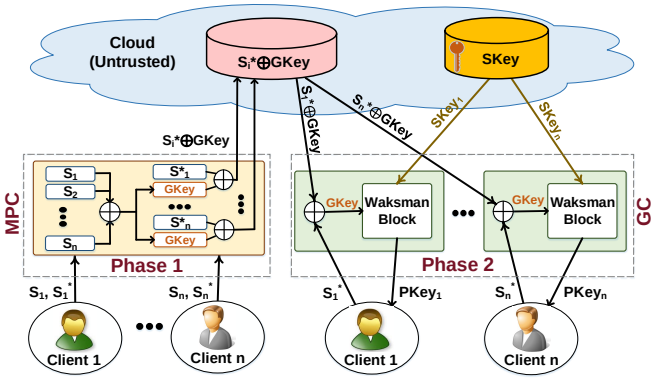


Figure 4. MPC-based key generation

not have the GKeys, it cannot decrypt the encoded data of clients. This MPC-based key generation runs only once.

After the key generation, each client can encode their data with its PKeys. SecureHD securely injects a small amount of information into the encoded data. We exploit this technique to store the metadata, e.g., data types, which are important to recover the entire original data. Once the encoded data is sent to the cloud, the cloud reshuffles the encoded data with the *SKeys* for the client. This allows the cloud to perform the learning task with no need for accessing GKeys and PKeys. With the SecureHD framework, the client can also decode the data from the encoded hypervectors. For example, once a client fetches the encoded data from the cloud storage service, it can exploit the framework to recover the original data using its own PKeys. Each client may also utilize the specialized hardware to accelerate both the encoding and decoding procedures.

C. Secure Key Generation and Distribution

Figure 4 illustrates how our protocol securely create the key hypervectors. The protocol runs two phases: *Phase 1* that all clients and the cloud participate, and *Phase 2* that two parties, a single client and cloud, participate. Recall that in order for the cloud server to be able to learn the model, all have to be projected based on the same base hypervectors. Given the base hypervector and the encoded result, one can reconstruct the plaintext data. Therefore, all clients have to use the same key without anyone having access to the base hypervectors. We realize these two constraints at the same time with a novel hybrid secure computation solution.

In the first phase, we generate the base hypervectors, which we denote by *GKey*. The main idea is that the base hypervectors are generated collaboratively inside the secure Multi-Party Computation (MPC) protocol. At the beginning of the first phase, each party i inputs two sets of random strings called S_i and S_i^*. Each stream length is D, where D is the dimension size of a hypervector. The MPC protocol computes element-wise XOR ($\oplus$) of all the provided bitstreams, and the substream of D elements represent the global base hypervector, i.e., GKey. Then, it performs XOR for the GKeys again with S_i^* provided by each client. At the end of the first MPC protocol phase, the cloud receives $S_i^* \oplus GKey$ corresponding to each user i and stores these secret keys. Note that since S_i and S_i^* are inputs from each user to the MPC protocol, it is not revealed to any other party during the joint computation. It can be seen that the server has a unique XOR-share of the global key *GKey* for each user. This, in turn, enables the server and each party to continue their computation in a point-to-point manner without involving other parties during the second phase.

Our approach has a strong property that even if all other clients are dishonest and provide zero vectors as their share to generate the Gkey, the security of our system is not hindered. The reason is that the Gkey is generated with XOR of S_i for *all* clients. That is, if one generates its seed randomly, the global key will have a uniform random distribution. In addition, the server only receives an XOR-share of the global key. The XOR-sharing technique is equivalent to One-Time Pad encryption and is information-theoretic secure which is superior to the security against computationally-bounded adversaries in standard encryption schemes such as Advanced Encryption Standard (AES). We only use XOR gates in MPC which are considerably less costly than non-XOR gates [31].

In the second phase, the protocol distributes the secret key for each user. Each party engages in a two-party secure computation using the GC protocol. Server's inputs are $SKey_i$ and $S_i^* \oplus GKey$, while the client's input is S_i^*. The global key *GKey* is securely reconstructed inside the GC protocol by XOR of the two shares: $GKey = S_i^* \oplus (S_i^* \oplus GKey)$. The global key is then shuffled based on the unique permutation bits held by the server ($SKey_i$). In order to avoid costly random accesses inside the GC protocol, we use the Waksman permutation network with $SKey_i$ being the permutation bits [32]. The shuffled global key is sent back to the user, and we perform a single rotational shift for the GKey to generate the next base hypervector. We repeat this n times where n is the required number of base hypervectors, e.g., the feature size. The permuted base hypervectors serve as user's personal keys, called *PKey*, for the projection. Once a user performs the projection with *PKey*, she can send the result to the server, and the server permutes back based on the $SKey_i$ for the learning process.

IV. SecureHD Encoding and Decoding

Figure 5 shows how the SecureHD framework performs the encoding and decoding of a client with the generated PKeys. The example has been shown for an image input data with n pixel values, $\{f_1, \ldots, f_n\}$. Our design encodes each input data into a high-dimensional vector from the feature values (Ⓐ). It exploits the PKeys, i.e., a set of the base hypervectors for the client, where 0 and 1 in the PKeys correspond to -1 and 1 to form a bipolar hypervector ($\{-1,+1\}^D$). We denote them by $PKeys = \{\mathbf{B}_1, \ldots, \mathbf{B}_n\}$. To store the metadata with negligible impact on the encoded hypervector, we devise a method which injects several metadata to small segments of an encoded hypervector. This method exploits another set of base vectors, $\{\mathbf{M}_1, \ldots, \mathbf{M}_k\}$ (Ⓑ). We call them as

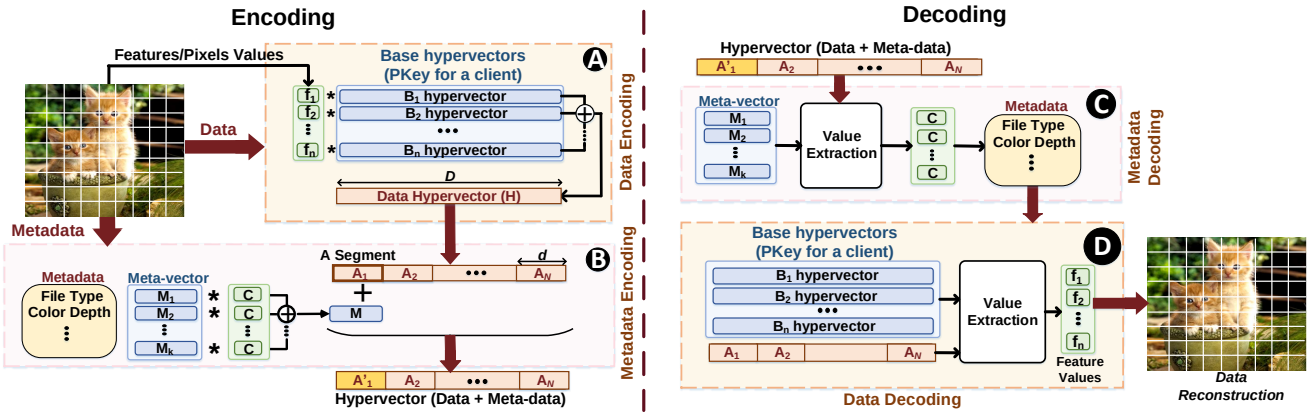


Figure 5. Illustration of SecureHD encoding and decoding procedures

metavector. The encoded data are sent to the cloud to perform HD learning.

Once the encoded data is received from the cloud, SecureHD can also decode them back to the original domain. This is useful for other cloud services, e.g., cloud storage. This procedure starts with identifying the injected metadata (C). Based on the injected metadata, it figures out the base hyperevectors that will be used in the decoding. Then, it reconstructs the original data from the decoded data (D). The key of the data recovery procedure is the value extraction algorithm, which retrieves both metadata and data.

A. Encoding in HD Space

1) Data Encoding: The first step of SecureHD is to encode input data into hypervector, where an original data point has n features. We associate each feature with a hypervector. The features can have discrete value (e.g., alphabets in the text), in which we perform a straight mapping to hypervectors, or they can have a continuous range, in which case the values can be quantized and then mapped similar to discrete features. Our goal is to encode each feature vector to a hypervector that has D dimensions, e.g. $D = 10,000$.

To differentiate each feature, we exploit a PKey for each feature value, i.e., $\{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n\}$, where n is the feature size of an original data point. Since the PKeys are generated from the random bit streams, the similarity of different base hypervectors are nearly orthogonal [29]:

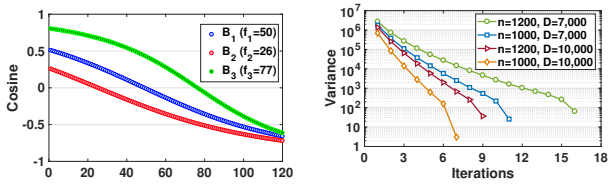
$$\delta(B_i,\ B_j) \simeq 0 \qquad (0 < i, j \leq n,\quad i \neq j).$$

The orthogonality of feature hypervectors is ensured as long as the hypervector dimension, D, is large enough compared to the number of features ($D >> n$) in the original data.

Different features are combined by multiplying feature values with the corresponding base hypervector, $\mathbf{B}_i \in \{-1, +1\}^D$ and adding them for all the features. For example, where f_i is a feature value, the following equation represents the encoded hypervector, $\mathbf{H}$:[1]

$$H\ = f_1 * \mathbf{B}_1\ +\ \ f_2 * \mathbf{B}_2 + \ldots\ +\ f_n * \mathbf{B}_n.$$

[1]The scalar multiplication, denoted by *, can make a hypervector that has integer elements, i.e., $\mathbf{H} \in \mathbb{N}^D$.



(a) (b)

Figure 6. Value extraction example

If two original feature values are similar, their encoded hypervectors are also similar, thus providing the learning capability for the cloud without any knowledge for the PKeys. Please note that, with this encoding scheme, although an attacker intercepts sufficient hypervectors, the upper bound of the information leakage is the distribution of the data. It is because the hypervector does not preserve any information of the feature order, e.g., pixel positions in an image, and there are extremely large combinations of values in hypervector elements which exponentially grow as n increases. In the case that n is small, e.g., $n < 20$, we can simply add extra features drawn from a uniform random distribution, and it does not affect the data recovery accuracy and HD computation results.

2) Metadata Injection: A client may receive an encoded hypervector where SecureHD processes multiple data types. In this case, to identify base hypervectors used in the prior encoding, it needs to embed additional information of the data identifier and metadata, such as data type (e.g., image or text) and color depth. One naive way is to store this metadata as attached bits to the original hypervector. However, this does not keep the metadata secure.

To embed the additional metadata into hypervectors, we exploit the fact that HD computing is robust to small modification of hypervector elements. Let us consider a data hypervector as a concatenation of several partial vectors. For example, a single hypervector with the D dimension can be viewed as the concatenation of different d-dimensional vectors, $\mathbf{A}_1, \ldots, \mathbf{A}_N$:

$$\mathbf{H} = \mathbf{A}_1 \frown \mathbf{A}_2 \frown \cdots \frown \mathbf{A}_N$$

where $D = N \times d$, and each $\mathbf{A}_i$ vector is called as a *segment*.

We inject the metadata in a minimal number of segments.

Figure 5 shows the concatenation of a hypervector to $N = 200$ segments with $d = 50$ dimensions. We first generate a random d dimensional vector with bipolar values, $\mathbf{M}_i$, i.e., *metavector*. A metavector corresponds to a metadata type. For example, $\mathbf{M}_1$ and $\mathbf{M}_2$ can correspond to the image and text types, while $\mathbf{M}_3$, $\mathbf{M}_4$, and $\mathbf{M}_5$ correspond to each color depth, e.g., 2-bit, 8-bit, and 32-bit. Our design injects each $\mathbf{M}_i$ into one of the segments in the data hypervector. We add the metavector multiple times to better distinguish it against the values already stored in the segment. For example, if we inject the metavector in the first segment, the following equation denotes the metadata injection procedure:

$$\mathbf{A}'_1 = \mathbf{A}_1 \; + \; C * \mathbf{M}_1 \; + \; C * \mathbf{M}_2 \; + \; \ldots \; + \; C * \mathbf{M}_k$$

where C is the number of injections for each metavector.

B. Decoding in HD Space

1) Value Extraction: In many of today's applications, the clouds are used as a storage, so the clients should be able to recover the original data from encoded ones. The key component of the decoding procedure is a new data recovery method that extracts the feature values stored in the encoded hypervectors. Let us consider an example of $\mathbf{H} = f_1 * \mathbf{B}_1 + f_2 * \mathbf{B}_2 + f_3 * \mathbf{B}_3$, where $\mathbf{B}_i$ is a base hypervector with D dimensions and f_i is a feature value. The goal of the decoding procedure is to find a f_i for a given $\mathbf{B}_i$ and $\mathbf{H}$. A possible way is to exploit the cosine similarity metric, δ. For example, if we measure the cosine similarity of $\mathbf{H}$ and $\mathbf{B}_1$ hypervectors, $\delta(\mathbf{H}, \mathbf{B}_1)$, the higher δ value represents higher chance of the existence of $\mathbf{B}_1$ in $\mathbf{H}$. Thus, one method may iteratively subtracts one instance of $\mathbf{B}_1$ from $\mathbf{H}$ to check when the cosine similarity is zero, i.e., $\delta(\mathbf{H}', \mathbf{B}_1)$ where $\mathbf{H}' = \mathbf{H} - m * \mathbf{B}_1$.

Figure 6a shows an example of the cosine similarity for each $\mathbf{B}_i$ when $f_1 = 50$, $f_2 = 26$ and $f_3 = 77$ and m changes from 1 to 120. The result shows that the similarity decreases as subtracting more instances of $\mathbf{B}_1$ from $\mathbf{H}$. For example, the similarity is zero when m is close to f_i as expected, and it gets negative values for further subtractions, since $\mathbf{H}'$ has the term of $-\mathbf{B}_1$. Regardless of the initial similarity of $\mathbf{H}$ with $\mathbf{B}$, the cosine similarity is around zero when m is close to each feature value f_i.

However, there are two main issues in the cosine similarity-based value search. First, finding the feature values in this way needs iterative procedures, slowing down the runtime of data recovery. In addition, it is more challenging when feature values are represented in floating points. Second, the cosine similarity metric may not give accurate results in the recovery. In our earlier example, the similarity of each f_i is zero, when m_i is 49, 29 and 78 respectively.

To efficiently estimate f_i values, we exploit another approach that utilizes the random distribution of the hypervector elements. Let us consider the following equation:

$$\mathbf{H} \cdot \mathbf{B}_i = f_i * (\mathbf{B}_i \cdot \mathbf{B}_i) + \sum_{j, \forall j \neq i} f_j * (\mathbf{B}_i \cdot \mathbf{B}_j).$$

$\mathbf{B}_i \cdot \mathbf{B}_i$ is D since each element of the base hypervector is either 1 or -1, while $\mathbf{B}_i \cdot \mathbf{B}_j$ is almost zero due to their near-orthogonal relationship. Thus, we can estimate f_i with the following equation, called *value discovery metric*:

$$f_i \simeq \mathbf{H} \cdot \mathbf{B}_i / D.$$

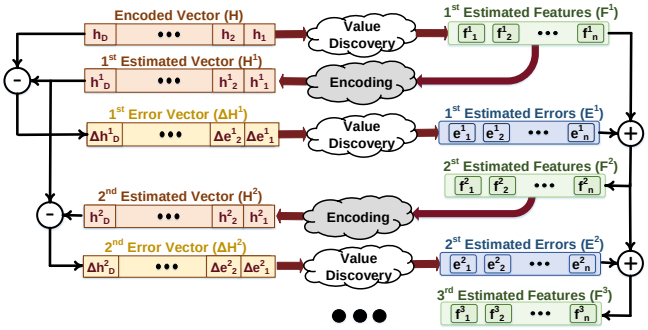


Figure 7. Iterative error correction procedure

This metric yields an initial estimate of all feature values, say $\mathbf{F}^1 = \{f_1^1, ..., f_n^1\}$. Starting with the initial estimation, SecureHD minimizes the error through an iterative procedure. Figure 7 shows the iterative error correction mechanism. We encode the estimated feature vector, $\mathbf{F}^1$, into the high dimensional space, $\mathbf{H}^1 = \{h_1^1, ..., h_D^1\}$. We then compute $\Delta\mathbf{H}^1 = \mathbf{H} - \mathbf{H}^1$, and apply the value extraction metric for $\Delta\mathbf{H}^1$. Since this yields the estimated error, E^1, in the original domain, we add it to the estimated feature vector for the better estimate of the actual features, i.e., $\mathbf{F}^2 = \mathbf{F}^1 + \mathbf{E}^1$. We repeat this procedure until the estimated error converges. To determine the termination condition, we compute the variance of the *error hypervector*, $\Delta\mathbf{H}^i$, at the end of each iteration. Figure 6b shows the variance changes when decoding four example hypervectors. For this experiment, we used two feature vectors whose size is either $n = 1200$ or 1000, where the feature values are uniform-randomly generated. We encoded each feature vector to two hypervectors with either $D = 7,000$ or $D = 10,000$. As shown in the results, the iterations required for accurate recovery depends on both the number of features in the original domain and hypervector dimensions. In the rest of the paper, we use the ratio of the hypervector dimension to the number of features in the original domain, i.e., $R = D/n$, to evaluate the quality of the data recovery for different feature sizes. The larger R ratio, the larger the retraining iterations are expected to sufficiently recover the data.

2) Metadata Recovery: We utilize the value extraction method to recover the metadata. We calculate how many times each metavector $\{\mathbf{M}_1, ..., \mathbf{M}_k\}$ presents in a segment. If the extracted instances of metavector are similar to the actual C value that we injected, such metavector is considered to be in the segment. However, since the metavector has a small number of elements, i.e., $d << D$ dimensions, it might have a large error in finding the exact C value. Let's assume that, when injecting a metavector C times, the value extraction method identifies a value, $\widehat{C}$, in a range of $[C_{min}, C_{max}]$. The range also includes C. If the metavector does not exist, the

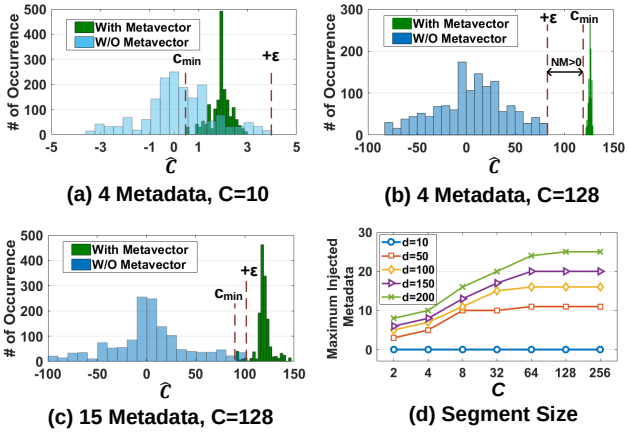


Figure 8. Relationship between the number of metavector injections and segment size

value $\widehat{C}$ will be approximately zero, i.e., a range of $[-\varepsilon, \varepsilon]$. The amount of ε depends on the other information stored in the segment.

Figure 8a shows the distribution of extracted values, $\widehat{C}$, when injecting 5 metadata 10 times ($C = 10$) into a single segment of a hypervector. These distributions are reported using a Monte Carlo simulation with 1500 randomly generated metavectors. The results show that the distributions of the existing and non-existing cases are overlapped, making the estimation difficult. However, as shown in Figure 8b, when using $C = 128$, there is a clear margin between these two distributions which identify the existence of a metadata. Figure 8c shows the distributions when we inject 8 metadata into a single segment with $C = 128$. In that case, two distributions overlap, i.e., there are a few cases when we cannot fully recover the metadata.

We determine C so that the distance between C_{min} and ε is larger than 0. We define the distance as the noise margin, $NM = C_{min} - \varepsilon$. Figure 8d shows how many metavectors can be injected for different C values. The results show that the number of meta vectors that we can inject saturates for larger C values. Since the large number of C and segment size, d, also have a higher chance to influence on the accuracy of the data recovery, we choose $C = 128$ and $d = 50$ for our evaluation. In Section VI-E, we present a detailed evaluation for different settings of the metavector injection.

3) Data Recovery: After recovering the metadata, SecureHD can recognize the data types and choose the base hypervectors for decoding. We subtract the metadata from the encoded hypervector and start decoding the main data. SecureHD utilizes the same value extraction method to identify the values for each base hypervector. The quality of data recovery depends on the dimension of hypervectors in the encoded domain (D) and the number of features in the original space (n), i.e., $R = D/n$ defined in Section IV-B1. Intuitively, with the larger the R value, we can achieve a higher accuracy during the data recovery at the expense of the size of encoded data. For instance, when storing an image with $n = 1000$ pixels in a hypervector with $D = 10,000$ dimensions ($R = 10$), it is expected to achieve high accuracy for the data recovery. In our evaluation, we observed that, with $R = 7$, it is enough to ensure lossless data recovery in the worst case. In Section VI-D, we explore more detailed discussion about how R impacts on the accuracy of the recovery procedure.

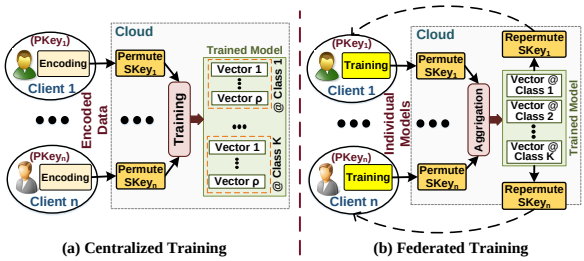


Figure 9. Illustration of the classification in SecureHD

V. Learning in HD Space

A. Secure Collaborative Learning

Figure 9 shows the HD-based collaborative learning in the high-dimensional space. In this paper, we show two training approaches, *centralized* and *federated* training, which performs classification learning with a large amount of data provided by many clients. The cloud can perform the training procedures using the encoded hypervectors without explicit decoding. It only needs to permute the encoded data using the SKey of each client. Note that the permutation aligns the encoded data on the same GKey base, even though the cloud does not have the GKeys. It reduces the cost of the learning procedure, and the data can be securely classified even on the untrustworthy cloud. The training procedure creates multiple hypervectors as the trained model, where each hypervector represents the pattern of data points in one class. We refer them to *class hypervectors*.

1) Approach 1: Centralized Training: In this approach, the clients send the encoded hypervectors to the cloud. The cloud permutes them with the SKeys, and a trainer module combines the permuted hypervectors. The training is performed with the following sub-procedures.

(i) Initial training: At the initial stage, it creates the class hypervectors for each class. As an example, for a face recognition problem, SecureHD creates two hypervectors representing "face" and "non-face". These hypervectors are generated with element-wise addition for all encoded inputs which belong to the same class, i.e., one for "face" and the other one for "non-face".

(ii) Multivector expansion: After training the initial HD model, we expand the initial model with cross-validation, so that each class has multiple hypervectors of the size of ρ. The key idea is that, when training with larger data, it may need to capture more distinct patterns with different hypervectors. To this end, we first check cosine similarity for each encoded hypervector again to the trained model. If an encoded data does not correctly match with its corresponding class, it means that the encoded hypervector has a distinct

pattern as compared to the majority of all the inputs in the class. For each class, we create a set that includes such mismatched hypervectors and the original model. We then choose two hypervectors, whose similarity is the highest among all pairs in the set, and update the set by adding the selected two into a new hypervector. This is repeated until the set includes only ρ hypervectors.

(iii) Retraining: As the last step, we iteratively adjust the HD model over the same dataset to give higher weights for misclassified samples that may often happen in a large dataset. We check the similarity for each encoded hypervector again with all existing classes. Let us assume that $\mathbf{C}_k^p$ is one of the class hypervectors belonging to k^{th} class, where p is the index of multiple hypervectors in the class. If an encoded hypervector $\mathbf{Q}$ belonging to i^{th} class is incorrectly classified to $\mathbf{C}_j^{miss}$, we update the model by

$$\mathbf{C}_j^{miss} = \mathbf{C}_j^{miss} - \alpha\mathbf{Q} \quad \text{and} \quad \mathbf{C}_i^{\tau} = \mathbf{C}_i^{\tau} + \alpha\mathbf{Q}$$

where $\tau = \text{argmax}_t\, \delta(\mathbf{C}_i^t, \mathbf{Q})$ and α is a learning rate in a range of [0.0, 1.0]. In other words, in the case of misclassification, we subtract the encoded hypervector from the class which it is incorrectly classified to, while adding it to the class hypervector which has the highest similarity in the correct class. This procedure is repeated for predefined iterations, and the final class hypervectors are used for the future inference.

2) Approach 2: Federated Training: The clients may not have enough network bandwidth to send every encoded hypervector. To address this issue, we present the second approach, called *federated* training, as an edge computing. In this approach, the clients individually train initial models, i.e., one hypervector for each class, only using their own encoded hypervectors. Once the cloud receives the initial models of all the clients, it permutes the models with the SKeys and performs element-wise additions to create a global model, $\mathbf{C}_k$, for each k^{th} class.

Since the cloud only knows the initial models for each client, the multivector expansion procedure is not performed in this approach, but we can still execute the retraining procedure explained in Section V-A1. To this end, the cloud re-permutes the global model and sends it back to each client. With the global model, each client performs the same retraining procedure. Let us assume that $\widetilde{\mathbf{C}}_k^i$ is the retrained model by the i^{th} client. After the cloud aggregates all $\tilde{\mathbf{C}}_k^i$ with the permutation, it updates the global models by $\mathbf{C}_k = \sum_i \widetilde{\mathbf{C}}_k^i - (n-1) * \mathbf{C}_k$. This is repeated for the predefined iterations. This approach allows the clients to send the trained class hypervectors only for each retraining iteration, thus significantly reducing the network usage.

B. HD Model-Based Inference

With the class hypervectors generated by either approach, we can perform the inference in any device including the cloud and clients. For example, the cloud can receive an encoded hypervector from a client, and permute the dimension with the SKey in the same way to the training procedure. Then, it checks cosine similarity of the permuted hypervector to all trained class hypervectors to label with the corresponding class to the most similar class hypervector. In the case of the client-based inference, once the cloud sends re-permuted class hypervectors to a client, the client can perform the inference for its encoded hypervector with the same similarity check.

VI. Evaluation

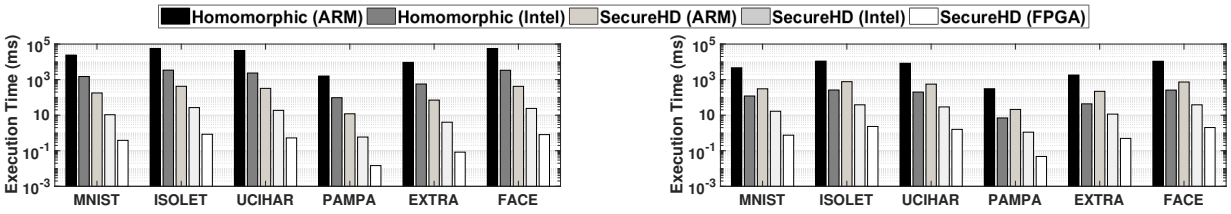
A. Experimental Setup

We have implemented the SecureHD framework including encoding, decoding, and learning in high-dimensional space using C++. We evaluated the system on three different platforms: Intel i7 7600 CPU with 16GB memory, Raspberry Pi 3, and Kintex-7 FPGA KC705. We also exploit a network simulator, NS-3 [33], for large-scale simulation. We verify the FPGA timing and the functionality of the encoding and decoding by synthesizing Verilog using Xilinx Vivado Design Suite [34]. The synthesis code has been implemented on the Kintex-7 FPGA KC705 Evaluation Kit. We compare the efficiency of the proposed SecureHD with SEAL, the state-of-the-art C++ implementation of a homomorphic library, Microsoft SEAL [27]. For SEAL, we used the default parameters: polynomial modulus of $n = 2048$, coefficient modulus of $q = 128 - bit$, plain modulus of $t = 1 << 8$, noise standard deviation of 3.9, and decomposition bit count of 16. We evaluate the proposed SecureHD framework with real-world datasets including human activity recognition, phone position identification, and image classification. Table I summarizes the evaluated datasets. The tested benchmarks range from relatively small datasets collected in a small IoT network, e.g., PAMAP2, to a large dataset which includes hundreds of thousands of images of facial and non-facial data. We also compare the classification accuracy of SecureHD for the datasets with the state-of-the-art learning models shown in the table.

B. Encoding and Decoding Performance

As explained in Section III-C, SecureHD performs a one-time key generation to distribute the PKeys to each user using the MPC and GC protocols. Table II listed the number of required logic gates evaluated in the protocol and the amount of required communication between clients. This overhead comes mostly from the first phase of the protocol, since the second phase has been simplified with the two-party GC protocol. The cost of the protocol is dominated by network communication. In our simulation conducted under our in-house network of 100 Mbps, it takes around 9 minutes to create $D = 10,000$ keys for 100 participants. Note that the runtime overhead is negligible since the key generation happens only once before all future computation.

We have also evaluated the encoding and decoding procedure running on each client. We compare the efficiency of SecureHD with the Microsoft SEAL [27]. We run both the SecureHD framework and homomorphic library on ARM Cortex 53 and Intel i7 processors. Figure 10 shows the



(a) Encoding (b) Decoding

Figure 10. Comparison of SecureHD efficiency to homomorphic algorithm in encoding and decoding

Table I
DATASETS (n: FEATURE SIZE, K: NUMBER OF CLASSES)

	n	K	Data Size	Train Size	Test Size	Description/State-of-the-art Model
MNIST	784	10	220MB	60,000	10,000	Handwritten Recognition/DNN [35], [36]
ISOLET	617	26	19MB	6,238	1,559	Voice Recognition/DNN [37], [38]
UCIHAR	561	12	10MB	6,213	1,554	Activity recognition(Mobile)/DNN [38], [39]
PAMAP2	75	5	240MB	611,142	101,582	Activity recognition(IMU)/DNN [40]
EXTRA	225	4	140MB	146,869	16,343	Phone position recognition/AdaBoost [41]
FACE	608	2	1.3GB	522,441	2,494	Face recognition/Adaboost [42]

Table II
OVERHEAD FOR KEY GENERATION AND DISTRIBUTION

Phases		**Phase 1**			**Phase 2**
# of Clients		*10*	*50*	*100*	
D=1000	*# of Gates*	11K	51K	101K	8.9K
	Communication	7.1MB	160MB	650MB	284MB
D=5000	*# of Gates*	55K	255K	505K	56.4K
	Communication	35MB	813MB	3.24GB	1.8MB
D=10,000	*# of Gates*	110K	510K	101K	122.9K
	Communication	70.34MB	1.64GB	6.46GB	3.93MB

execution time of the SecureHD and homomorphic library to process a single data point. For SecureHD, we used $R = 7$ to ensure 100% data recovery rate for all benchmark datasets. Our evaluation shows that SecureHD achieves on average 133× and 14.7× (145.6× and 6.8×) speedup for the encoding and decoding, respectively, as compared to the homomorphic technique running on the ARM architecture (Intel i7). The encoding of SecureHD running on embedded devices (ARM) is still 8.1× faster than the homomorphic encryption running on the high-performance client (Intel i7). We also compare the SecureHD efficiency on the FPGA implementation. We observe that the encoding and decoding of SecureHD achieve 626.2× and 389.4× (35.5× and 20.4×) faster execution as compared to the SecureHD execution on the ARM (Intel i7). For example, the proposed FPGA implementation is able to encode 2,600 data points and decode 1,335 for the MNIST images in a second.

C. Evaluation of SecureHD Learning

1) Learning Accuracy: Based on the proposed SecureHD, clients can share the information with the cloud in a secure way, such that the cloud cannot understand the original data while still performing the learning tasks. Along with the proposed two learning approaches, we also evaluate the state-of-the-art HD classification approach, called *one-shot HD model*, which trains the model using a single hypervector per class with no retraining [22], [26]. For the centralized training, we trained two models, one that has 64 class hypervectors for each class and the other one that has 16 for each class. We call them as *Centralized-64* and *Centralized-16*. The retraining procedure was performed for 100 times with $\alpha = 0.05$, since the classification accuracy was converged with this configuration for all the benchmarks.

Figure 11 shows the classification accuracy of the SecureHD for the different benchmarks. The results show that the centralized training approach achieves high classification accuracy comparable to the state-of-the-art learning methods such as DNN models. We also observed that, by training more hypervectors per class, it can provide higher classification accuracy. For example, for the federated training approach, which does not use multivectors, the classification accuracy is 90% on average, which is 5% lower than the Centralized-64. As compared to the state-of-the-art one-shot HD model which does not retrain models, Centralized-64 achieves 15.4% higher classification accuracy on average.

2) Scalability of SecureHD Learning: As discussed in Section V-A, the proposed learning method is designed to effectively handle a large amount of data. To understand the scalability of the proposed learning method, we evaluate how the accuracy is changed when the training data are come from different numbers of clients, with simulation on NS-3 [33]. In this experiment, we exploit three datasets, PAMAP2, EXTRA, and FACE, which include information of where data points are originated. For example, PAMAP2 and EXTRA are gathered from 7 and 56 individual users. Similarly, the FACE dataset includes 100 clients that have different facial images with each other. Figure 12a and b show the accuracy changes for the centralized and federated training approaches. The result shows that increasing the number of clients improves classification accuracy by training with more data. Furthermore, as compared to the one-shot HD model, the two proposed approaches show better scalability in terms of accuracy. For example, the accuracy difference between the proposed approach and the one-shot model grows as more clients engage in the training. Considering the centralized training, the accuracy difference for the FACE dataset is 5% when trained with one client, while it is 14.7% for the 60-client case. This means that the multivector expansion and retraining techniques are effective to learn with a large amount of data.

We also verify how the SecureHD learning methods work with constrained network conditions that often happen in IoT systems. In our network simulation, we assume the worst-case network condition, i.e., all clients share the bandwidth of a standard WiFi 802.11 network. Note that it is a worst-case scenario and in practice, each embedded device may not share the same network. Figure 12c shows that the network bandwidth limits the number of hypervectors that can be sent for each second as multiple clients involve the learning task.

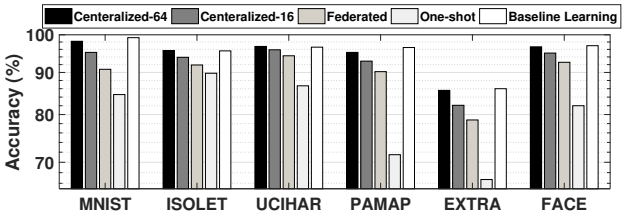
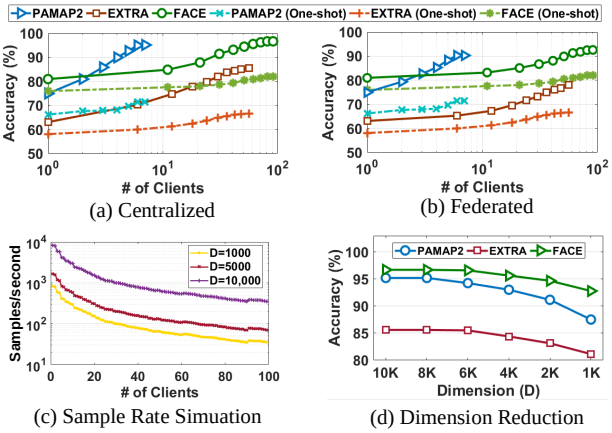


Figure 11. SecureHD classification accuracy



(a) Centralized (b) Federated

(c) Sample Rate Simuation (d) Dimension Reduction

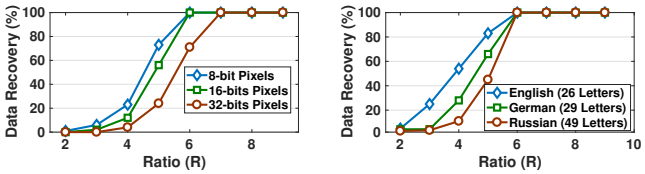
Figure 12. Scalability of SecureHD classification

For example, a network with 100 clients can send the lower number of hypervectors by 23.6× than a single-client case.

As discussed before, the federated learning can be exploited to overcome the limited network bandwidth at the expense of the accuracy loss. Another solution is to use a reduced dimension in the centralized learning. As shown in Figure 12c, when $D = 1,000$, clients can send the data to the cloud with 353 samples per second, which is 10 times higher than the case of $D = 10,000$. Figure 12d shows how learning accuracy changes for different dimension settings. The results show that reducing the hypervector dimensions to 4000 and 1000 dimensions has less than 1.4% and 5.3% impact on the classification accuracy. This strategy gives another choice of the trade-off between accuracy and network communication cost.

D. Data Recovery Trade-offs

As discussed in Section IV-B3, the proposed SecureHD framework provides a decoding method for the authorized user that has the original Pkeys used in the encoding. Figure 13a shows the data recovery rate on images with different pixel sizes. To verify the proposed recovery method in the worst case scenario, we created 1000 images whose pixel values are randomly chosen, and report the average error when we map the 1000 images to $D = 10,000$ dimension. The x-axis shows the ratio R, i.e., D/n where the number of hypervector dimension (D) to the number of pixels (n) in an image. The data recovery rate depends on the precision of the pixel values. Using high-resolution images, SecureHD requires a larger R value to ensure 100% accuracy. For instance, for images with 32-bit pixel resolution, SecureHD can achieve 100% data recovery using $R = 7$, while lower



(a) Image (b) Text

Figure 13. Data recovery accuracy of SecureHD

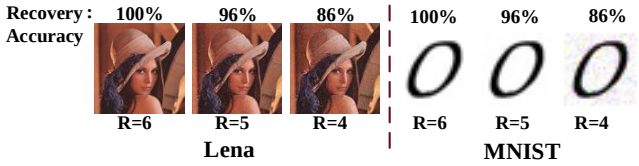


Figure 14. Example of image recovery

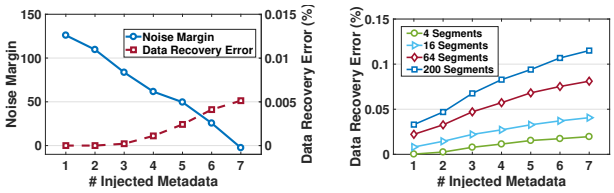
resolution images (e.g., 16 and 8-bits) requires $R = 6$ to ensure 100% data recovery. Our evaluation shows that our method can decode any input image with 100% data recovery rate using $R = 7$. This means that we can securely encode data with 4× smaller size compared to the homomorphic encryption library which increases the data size by 28 times through the encryption.

We also evaluate the SecureHD framework with a text dataset written in three different European languages [43]. Figure 13b shows the accuracy of data recovery for the three languages. The x-axis is the ratio between the length of hypervectors to the number of characters in the text when $D = 10,000$. Our method assigns a single value to each alphabet letter and encodes the texts with the hypervectors. Since the number of characters in these languages is less than 49, we require at most 6 bits to represent each alphabet. In terms of the data recovery, it is equal to encoding the same size image with the 6-bit pixel resolution. Our evaluation shows that SecureHD can provide 100% data recovery rate with $R = 6$.

Figure 14 shows the quality of the data recovery for two example images. The Lena and MNIST image have 100×100 pixels and 28×28 pixels, respectively. Our encoding maps the input data to hypervectors with different dimensions. For example, the Lena image with $R = 6$ means that the image has been encoded with $D = 60,000$ dimensions. Our evaluation shows that SecureHD can achieve lossless data recovery on Lena photo when $R \geq 6$, while using $R = 5$ and $R = 4$ the data recovery rates are 93% and 68%. Similarly, $R = 5$ and $R = 4$ provide 96% and 56% data recovery for the MNIST images.

E. Metadata Recovery Trade-offs

As discussed in Section IV-B2, the metadata injection method needs to be performed such that it ensures 100% metadata recovery and it has minimal impacts on the original hypervector for the learning and data recovery. The solid line in Figure 15a shows the noise margin when injecting multiple metavectors into a single segment of hypervector when the number of elements in the segment is chosen by $50 (= d)$. We report the results based on the worst case for



(a) Single Segment (b) Multiple Segments

Figure 15. Data recovery rate for different settings of metavector injection

5000 Monte Carlo simulation. The results show that each segment can store 6 metavectors at most to take a positive noise margin that ensures 100% metadata recovery. The dotted line shows the data recovery error rate for different numbers of metavectors injected into a single segment. Our evaluation shows that adding 6 metavectors has less than 0.005% impact on the data recovery rate.

Since the number of metavectors which can be injected in one segment is limited, we may need to distribute the metadata in different segments. Figure 15b presents the impact of the metadata injection on the data recovery error rate. When we inject 6 metadata into each of all 200 segments, i.e., 1200 metavectors in total, the impact on the recovery accuracy is still minimal, i.e., less than 0.12%.

VII. Related Work

Secure Learning Privacy-preserving deep learning and classification has been an active research area in recent years [17], [44], [45], [46], [47], [48], [49]. Shokri and Shmatikov [45] have proposed a solution for collaborative deep learning where the training data is distributed among many parties. Each party locally trains her model and sends the parameter updates to the server. However, it has been shown that Generative Adversarial Networks (GANs) can be used to attack this method [50].

SecureML [47] is a framework for secure training of machine learning models. All of the computation of SecureML is performed by the two servers using MPC protocols, whereas, SecureHD only relies on the MPC protocol for secure key generation and distribution. Chameleon [44] is a privacy-preserving machine learning framework that utilizes different cryptographic protocols for different operations within the machine learning task. In contrast to SecureML and Chameleon, our solution does not require two non-colluding servers and only involves one server.

Google has also proposed a federated learning approach [17] for collaborative learning. In their approach, each client needs to learn the local model based on the private training data to update the central model in the cloud. However, our solution is more light-weight to be run on less-powerful IoT devices and also applicable to other cloud-oriented tasks, e.g., data storage services.

Hyperdimensional Computing Since the Finnish computational neuroscientist P. Kanerva introduced the field of hyperdimensional computing [18], prior research have applied the idea into diverse cognitive tasks, such as analogy-based reasoning [19], latent semantic analysis [20], language recognition [21], speech/object recognition [23], [24], activity recognition [51], [52], and clustering [53] . However, most of the existing works assume that the HD learning tasks are performed on a single system. To the best of our knowledge, this paper is the first work that securely performs the HD learning tasks on a cloud scale. In addition, we also focus on how to accurately recover the encoded data in the HD space.

Some existing works have presented the hardware accelerator design to efficiently perform HD tasks [26], [54], [55]. Work in [56] proposed a framework for enabling model sparsity in HD computing. Several works showed new memory architectures that perform the HD operations inside the memory array [26], [57], [58]. Our design is orthogonal in this view that it can exploit any of these hardware for hardware acceleration.

VIII. Conclusion

In this paper, we propose a novel framework, called SecureHD, which provides secure data encoding and learning solution based on HD computing. With our framework, clients can securely send their data to untrustworthy clouds, while the cloud can perform the learning tasks without the knowledge of the original data. Our proof-of-concept implementation demonstrates that the proposed SecureHD framework successfully performs the encoding and decoding tasks with high efficiency, e.g., 145.6× and 6.8× faster than the state-of-the-art encryption/decryption library. Our learning method achieves accuracy of 95% on average for diverse practical learning tasks. In addition, SecureHD provides lossless data recovery with 4× reduction in the data size compared to the existing encryption solution.

Acknowledgements

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, Office of Naval Research (N00014-17-1-2500), MURI (FA9550-14-1-0351), Semiconductor Research Corporation (2016-TS-2690), and also NSF grants #1730158, #1527034, and #1619261. Authors would like to thank Kazim Ergun from UCSD for his help on the network simulation.

References

[1] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server." in *OSDI*, vol. 14, 2014.
[2] J. Andriessen, M. Baker, and D. D. Suthers, *Arguing to learn: Confronting cognitions in computer-supported collaborative learning environments*. Springer Science & Business Media, 2013, vol. 1.
[3] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan, "Big data analytics over encrypted datasets with seabed." in *OSDI*, 2016.
[4] S. Suthaharan, "Big data classification: Problems and challenges in network intrusion prediction with machine learning," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, 2014.
[5] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, 2016.
[6] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.
[7] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, 2015.

[8] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009.

[9] M. Taram, A. Venkat, and D. Tullsen, "Context-sensitive fencing: Securing speculative execution via microcode customization," in *International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019.

[10] M. Taram, A. Venkat, and D. Tullsen, "Mobilizing the micro-ops: Exploiting context sensitive decoding for security and energy efficiency," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018.

[11] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, 2015.

[12] G. Zhao, C. Rong, J. Li, F. Zhang, and Y. Tang, "Trusted data sharing over untrusted cloud storage providers," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010.

[13] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten, "Sporc: Group collaboration using untrusted cloud resources." in *OSDI*, vol. 10, 2010.

[14] S. Choi, G. Ghinita, H.-S. Lim, and E. Bertino, "Secure knn query processing in untrusted cloud environments," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, 2014.

[15] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010.

[16] H. Chen, K. Han, Z. Huang, A. Jalali, and K. Laine, "Simple encrypted arithmetic library v2.3.0," in *Microsoft*, 2017.

[17] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *CCS*. ACM, 2017.

[18] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, 2009.

[19] P. Kanerva, "What we mean when we say "whats the dollar of mexico?": Prototypes and mapping in concept space," in *AAAI Fall Symposium: Quantum Informatics for Cognitive, Social, and Semantic Processes*, 2010.

[20] P. Kanerva, J. Kristofersson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proceedings of the 22nd annual conference of the cognitive science society*, vol. 1036. Citeseer, 2000.

[21] M. Imani, J. Hwang, T. Rosing, A. Rahimi, and J. M. Rabaey, "Low-power sparse hyperdimensional encoder for language recognition," *IEEE Design & Test*, vol. 34, 2017.

[22] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 2016.

[23] M. Imani, C. Huang, D. Kong, and T. Rosing, "Hierarchical hyperdimensional computing for energy efficient classification," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018.

[24] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *Rebooting Computing (ICRC), 2017 IEEE International Conference on*. IEEE, 2017.

[25] A. Ben-Efraim, Y. Lindell, and E. Omri, "Optimizing semi-honest secure multiparty computation for the internet," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2016.

[26] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017.

[27] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library-seal v2. 1," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017.

[28] W. Du and M. J. Atallah, "Secure multi-party computation problems and their applications: a review and open problems," in *Proceedings of the 2001 workshop on New security paradigms*. ACM, 2001.

[29] P. Kanerva, J. Kristofersson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proceedings of the 22nd annual conference of the cognitive science society*, vol. 1036. Citeseer, 2000.

[30] A. C.-C. Yao, "How to generate and exchange secrets," in *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 1986.

[31] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *Automata, Languages and Programming*. Springer, 2008.

[35] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, 1998.

[32] A. Waksman, "A permutation network," *Journal of the ACM (JACM)*, vol. 15, 1968.

[33] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, 2008.

[34] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.

[36] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*. IEEE, 2012.

[37] "Uci machine learning repository," http://archive.ics.uci.edu/ml/datasets/ISOLET.

[38] M. S. Razlighi, M. Imani, F. Koushanfar, and T. Rosing, "Looknn: Neural network with no multiplication," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017.

[39] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International workshop on ambient assisted living*. Springer, 2012.

[40] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *Wearable Computers (ISWC), 2012 16th International Symposium on*. IEEE, 2012.

[41] Y. Vaizman, K. Ellis, and G. Lanckriet, "Recognizing detailed human context in the wild from smartphones and smartwatches," *IEEE Pervasive Computing*, vol. 16, 2017.

[42] Y. Kim, M. Imani, and T. Rosing, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*. IEEE, 2017.

[43] U. Quasthoff, M. Richter, and C. Biemann, "Corpus portal for search in monolingual corpora," in *Proceedings of the fifth international conference on language resources and evaluation*, vol. 17991802, 2006.

[44] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *ASIACCS*. ACM, 2018.

[45] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *CCS*. ACM, 2015.

[46] M. S. Riazi and F. Koushanfar, "Privacy-preserving deep learning and inference," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, 2018.

[47] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning." in *IEEE S&P*, 2017.

[48] M. S. Riazi, B. D. Rouhani, and F. Koushanfar, "Deep learning on private data," in *IEEE Security and Privacy Magazine*. IEEE, 2019.

[49] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," *USENIX Security*, 2019.

[50] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the GAN: information leakage from collaborative deep learning," in *CCS*. ACM, 2017.

[51] M. Imani, J. Messerly, F. Wu, W. Pi, and T. Rosing, "A binary learning framework for hyperdimensional computing," in *Design Automation and Test in Europe Conference (DATE)*. IEEE/ACM, 2019.

[52] M. Imani *et al.*, "Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2019.

[53] M. Imani, Y. Kim, T. Worley, S. Gupta, and T. Rosing, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *Design Automation and Test in Europe Conference (DATE)*. IEEE/ACM, 2019.

[54] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM/SIGDA, 2019.

[55] M. Imani *et al.*, "Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity," in *ASP-DAC*. IEEE, 2019.

[56] M. Imani *et al.*, "Sparsehd: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *IEEE FCCM*. IEEE, 2019.

[57] T. Wu, P. Huang, A. Rahimi, H. Li, J. Rabaey, P. Wong, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *IEEE Intl. Solid-State Circuits Conference (ISSCC)*. IEEE, 2018.

[58] H. Li, T. F. Wu, A. Rahimi, K.-S. Li, M. Rusch, C.-H. Lin, J.-L. Hsu, M. M. Sabry, S. B. Eryilmaz, J. Sohn *et al.*, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *Electron Devices Meeting (IEDM), 2016 IEEE International*. IEEE, 2016.