Private and Efficient Learning with Hyperdimensional Computing

Behnam Khaleghi*, Xiaofan Yu*, Jaeyoung Kang, Xuan Wang, Tajana Rosing, *Fellow, IEEE*

***Abstract*—Machine learning algorithms, especially deep neural networks (DNNs), are computationally expensive and vulnerable to privacy breaches. Hyperdimensional Computing (HDC) is a brain-inspired machine learning paradigm which offers a potential alternative for efficient and private learning. HDC encodes dense, sensory inputs into sparse, high-dimensional vectors of 10,000 bits or more, and learns using well-defined and lightweight operations on these vectors. However, naïve HDC is vulnerable to privacy attacks as the encoding process can be reversely determined. In this paper, we systematically study the fundamental privacy challenges of HDC due to reversibility and propose an end-to-end private training and inference framework for HDC with efficient hardware implementation. We first show that HDC is not naturally private as the original data and encoding parameters can be reconstructed from the encoded vector, regardless of the encoding methods applied. We term this attack as the feature extraction attack. We then propose a locally sparse encoding method to obfuscate the encoded information for protecting inference privacy, leading to computational and communication savings. We further propose Privé-HDnn which uses a hybrid architecture of Convolutional Neural Network (CNN) as feature extractors and private-preserved HDC classifier for complex tasks such as image classification. Thanks to the noise tolerance of HDC, Privé-HDnn enables private single- and multi-pass training of HDC by differential privacy-guaranteed noise injection. Results show that our new encoding method defends against decoding attacks with an RMSE deviation of 0.42, while also speeding up encoding by 2.6–5.3× (19–79×) compared to the FPGA (CPU) baseline. For privacy-preserved training, Privé-HDnn achieves comparable or up to 5.8% better accuracy than DNNs with up to 1231× faster training time on GPU as compared to the state-of-the-art NNs.**

***Index Terms*—Privacy-Preserved Learning, Hyperdimensional Computing.**

I. Introduction

In recent years, the number and diversity of applications benefiting from machine learning (ML) have grown significantly [1], [2]. A typical and widely used ML model is the deep neural network (DNN). However, training DNNs consumes vast hardware resources due to large model sizes, substantial data requirements, and the nature of backpropagation [3], [4]. Another challenge for DNNs is preserving privacy against various attacks, such as membership inference [5]. To protect training privacy, existing works have utilized *differential privacy* (DP) [6], [7], which adds carefully designed noise to randomize the output. However, this approach can impair training convergence and reduce accuracy [7]. Therefore, despite their outstanding learning performance, DNNs face significant challenges in practical deployment due to resource constraints and vulnerability to malicious attacks.

Hyperdimensional Computing (HDC) - a new brain-inspired learning paradigm that is very lightweight to compute and robust to hardware errors, has emerged as a potential solution. HDC encodes raw input data to hyperspace points, represented by vectors of thousands of bits called hypervector. HDC achieves memorization (training) through element-wise addition of encoded vectors in an operation called *bundling*, where the resulting vector has much higher similarity to the original vectors than to any random hypervector. We can associate two different pieces of information using an HDC operation called *binding*, which involves element-wise multiplication, producing a new hypervector that is orthogonal to the original vectors. Simple and parallelizable coordinate-wise computation makes HDC implementations more than two orders of magnitude energy-efficient compared to DNNs, especially with specialized hardware like FPGAs and GPUs [8], [9]. We can use these HDC operations to train ML models. For example, we can train a classifier by summing the high-dimensional vectors with the same label to create class vectors. The encoded query can then be compared with the class vectors using simple metrics such as Hamming distance for binary vectors or dot product, to predict a class for the query. HDC is well understood formally [10], and has been successfully applied to various applications and circuit systems in recent years [11]–[16].

However, HDC does not inherently preserve privacy. The key procedure of HDC, encoding, is reversible and thus can return the original data as a part of the decoding process [17], [18]. In practical use cases, HDC-encoded models are often exchanged between clients and servers, such as in cloud-hosted inference [19] or federated learning [20]–[22], as shown in Fig. 1. Despite not sharing raw data samples, there remains a risk of eavesdroppers uncovering the encoding parameters and subsequently accessing the raw data. While some literature has addressed these privacy challenges of HDC and proposed strategies for improvement [17], [18], these works have often focused on limited types of HDC encoding, assuming the adversary knows the encoding parameters, or only ensuring privacy guarantees in single-pass training. In contrast, our research aims to mitigate privacy breaches in all popular HDC encoding methods during inference and iterative training processes.

In this paper, we comprehensively study the reversibility challenges of HDC and propose an end-to-end framework, Privé-HDnn, to achieve private training and inference using

All authors are with the Department of Computer Science and Engineering, UC San Diego, USA. Emails: {bkhaleghi, x1yu, j5kang, xuw009, tajana}@ucsd.edu

*Both authors contributed equally to this research.

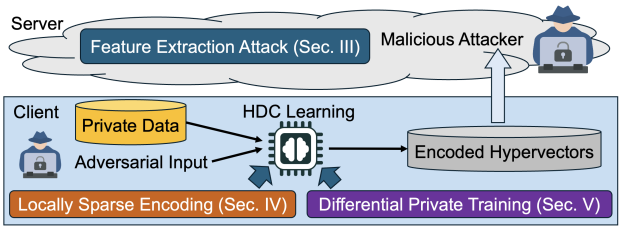


Fig. 1: Privé-HDnn offers three key contributions in typical cloud-client scenarios where malicious attackers may exist within the cloud networks and have access to the encoded hypervectors.

HDC. Fig. 1 provides an overview of Privé-HDnn in a typical client-server scenario. We assume that malicious attackers may reside within cloud networks and gain access to encoded HDC hypervectors as the HDC client exchanges models with other entities. We begin by demonstrating the reversibility challenge in HDC, where attackers can reconstruct the original features by applying adversarial inputs on the client, even without prior knowledge of HDC encoding parameters. We refer to this as the *feature extraction attack*. To protect privacy against the feature extraction attack during HDC inference, we propose locally-sparse encoding by obfuscating the information of the transferred data to untrustworthy hosts, along with an efficient FPGA implementation. We further present Privé-HDnn, a framework ensuring differential private training across single-pass and iterative scenarios. Privé-HDnn employs a pretrained CNN as a feature extractor and a trainable HDC classifier guided by private training techniques. Inspired from differential privacy [6], we show that the superior error tolerance of HDC [23], [24] can be prudently leveraged to enhance the privacy by noise/error injection.

In summary, we make the following novel contributions in this paper:

(1) We thoroughly study the reversibility of HDC with all common-used encoding methods. We further show that the adversary can infer the original features using adversarial inputs without prior knowledge, which we refer to as the feature extraction attack.

(2) We propose an approximate locally-sparse encoding to improve the HDC inference privacy, safeguarding against the feature extraction attack. Such technique also saves huge computational and communication costs with an efficient FPGA implementation.

(3) We propose Privé-HDnn for privacy-preserved training in complex HDC classification tasks, such as image classification, using an architecture of CNN feature extractor and HDC classifier. Privé-HDnn is designed with differentially private one-pass and iterative training.

(4) We conduct comprehensive evaluation of the proposed techniques. The new sparse encoding method increases the normalized RMSE between the original and decoded data from 0.15 to 0.42, in average, while enhancing encoding performance by up to 5.3x on FPGA and 79x on CPU baselines. Privé-HDnn attains comparable or superior accuracy gains of up to 5.8% against DNN-only solutions, while drastically reducing GPU training time by 9.2–1231× compared to state-of-the-art methods.

The rest of the paper is organized as follows: Section II provides an overview of prior research in HDC. In Section III, we present a thorough analysis on the decoding and feature extraction attack to HDC encoding. In Section IV, we propose a new approximate locally-sparse encoding method which preserves privacy of HDC inference even if the HDC models are exchanged with untrustworthy hosts. Section V presents a novel differential private training framework, Privé-HDnn, for both one-pass and iterative HDC training. Comprehensive experiments are conducted and results are reported in Section VI. Section VII concludes the paper.

II. Related Work

A. HDC Designs for Privacy

SecureHD [19] appeared as one of the first works exploring secure HDC learning in a collaborative learning setting with an untrustworth cloud. Each client has distinct base vectors for encoding the data, which are generated by shuffling a seed base. The cloud has only the shuffling keys (not the bases) to reshuffle the clients' encoded vectors; hence, after reshuffling, all the vectors are encoded by the same bases, and HDC learning becomes possible. However, such obscurity *does not* guarantee privacy as we show in Section III. Recent studies have explored data encryption within cloud-based systems, such as Homomorphic Encryption [25]–[27]. By encrypting data locally without decryption in the cloud, users' privacy is guaranteed. HDC has the potential to enhance the efficiency of these encryption techniques on local clients.

Focusing on HDC privacy on local clients, **PRID [18]** proposed model inversion attack and defense techniques of the HDC models. Given a query, PRID attempted to reconstruct the training data by replacing a subset of input query features (e.g., pixels of an image) with the features of the decoded class so that the similarity score increased. To reduce information leakage and protect HDC models, PRID used noise injection and model quantization techniques. **PP-HDC [28]** proposed a privacy-preserving inference framework using a novel hash encoding method and multi-model inference.

The most related work is **Prive-HD [17]**, which trained a differential private HDC model by adding Gaussian noise to the final model. Prive-HD assumed that trained models of adjacent datasets differ by only one vector (i.e., the encoded vector of the extra record) and adjusted noise injection accordingly. While effective for one-pass training, this approach may fail in iterative training, where updates depend on the initial model. Adding just one sample can alter the initial model, leading to significant differences in subsequent updates. Furthermore, considering only a *single* vector difference significantly underestimates the required noise.

Our work significantly improves previous state-of-the-art privacy-aware HDC from the following aspects: (1) We demonstrate the reversibility of all common encoding methods while Prive-HD [17], PRID [18] and PP-HDC [28] only consider one specific encoding. (2) Prive-HD [17] is built on the assumption that the adversary knows the encoding parameters. In this work, we show how the adversary can extract these parameters (for various encodings) without prior knowledge by using adversarial inputs. (3) Prive-HD only

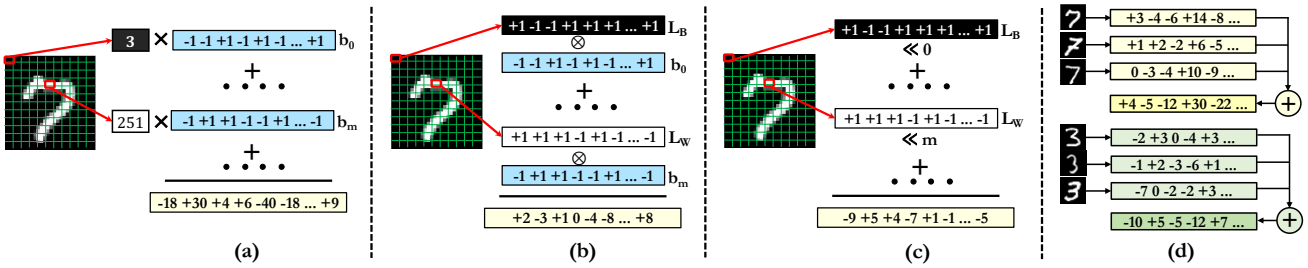


Fig. 2: (a) Random projection, (b) base-level, and (c) permutation encoding. (d) Training by bundling.

devises single-pass private HDC training, while our work proposes single-pass and iterative differential private HDC training with a stringent privacy budget. (4) In this work, for the first time, we propose Privé-HDnn, a combination of CNN feature extractor and HDC classifier for complex classification tasks (e.g., image classification) with privacy guarantees. (5) We also propose a novel sparse encoding methods to combat the reversibility challenges along with a computationally and communication efficient FPGA implementation.

B. HDC Designs for Cyber Attacks

Another body of research has studied various attacks on HDC. **Yang *et al.*** [29] used genetic algorithms to generate adversarial images with minimal perturbation that leads to misclassification by the HDC model. **HDTest [30]** applied random row/column mutation, shift, and noise to generate adversarial images for HDC classification. **PoisonHD [31]** aims to degrade the performance of the HDC model by injecting false data with flipped labels. A recent study on Hyperdimensional Data Poisoning Attacks **(HDPA) [32]** introduced an HDC-based approach to generate adversarial samples by perturbing within the HD space. From a hardware perspective, **HyperAttack [33]** presented an efficient attacker targeting HDC models by maliciously flipping an extremely small number of bits within the associative memory. We refer the readers to Ma *et al.* [34] for a comprehensive survey on the robustness of HDC against cyber attacks and hardware errors. Contrary to focusing on a specific type of attack, Privé-HDnn addresses the broader privacy challenge posed by reversibility in HDC and ensures differential privacy during HDC training regardless of the encoding method.

III. Reversibility of HDC and the Feature Extraction Attack

In this section, we explore the privacy challenges of using HDC models in networked applications, as illustrated in Fig. 1. Malicious attackers within the network can compromise privacy by merely accessing shared HDC models (i.e., class hypervectors) and generating adversarial inputs on local clients. We demonstrate that HDC encoding parameters (base vectors) can be uncovered using adversarial inputs across all encoding methods, a vulnerability not identified in previous work [17], [18]. Furthermore, we reveal that *every encoding method* faces a privacy risk, where the original data can be reconstructed from the encoded vectors. We term this privacy vulnerability as the *feature extraction attack*. Such finding poses a significant privacy challenge to HDC training and inference, particularly when the application necessitates a shared HDC model via untrustworthy links to the server.

A. Encoding and Decoding

Figs. 2(a)-(c) illustrate the common HDC encoding techniques, each of which preserves a particular distance in the hyperspace [10] and is suitable for particular type of data [9]. We represent input samples as a d element feature vector $V = \langle v_0, v_1, \cdots, v_{d-1} \rangle$, and we use capital D for encoded vectors dimensionality. We use vector symbol $\vec{}$ to represent vectors in the hyperspace. The encoded vector is represented as $\vec{\mathcal{H}}$.

(1) Random projection (RP) encoding, shown in Fig. 2(a), multiplies each feature with a D-dimensional base vector associated with that index. $\vec{\mathcal{B}}_k$s are constant orthogonal vectors called *base* or *id* vectors that are used to preserve the spatiotemporal relations of the features. RP encoding can be transformed to a matrix-vector multiplication by laying the d base vectors as the columns of a $D \times d$ matrix $\mathcal{B}^T$.

$$\vec{\mathcal{H}} = \sum_{k=0}^{d-1} v_k \times \vec{\mathcal{B}}_k = \mathcal{B}^T \times V. \tag{1}$$

Thus, we can decode $\vec{V}$ in the following way:

$$(\mathcal{B}^T)^\dagger \times \vec{\mathcal{H}} = (\mathcal{B}^T)^\dagger \times \mathcal{B}^T \times V \Rightarrow V \simeq (\mathcal{B}^T)^\dagger \times \vec{\mathcal{H}}.$$

Since $\mathcal{B}$ is non-square, we use Moore-Penrose pseudo-inverse [35] to estimate $\mathcal{B}^\dagger$.

(2) Base-level (a.k.a. *id-level*) encoding uses a set of high-dimensional level vectors $\vec{\mathcal{L}}$ to represent values, and a set of base vectors $\vec{\mathcal{B}}$ to identify the index of features, i.e., the position of the pixels in Fig. 2 (b). Both the base and level vectors have bipolar components, i.e., $\{+1, -1\}^D$. The base vectors $\vec{\mathcal{B}}_0, ..., \vec{\mathcal{B}}_{d-1}$ are generated randomly. For a total of q level vectors, $\vec{\mathcal{L}}_0$ is generated randomly, then every $\vec{\mathcal{L}}_k$ is generated by flipping $\frac{D}{2q}$ bits of $\vec{\mathcal{L}}_{k-1}$. Therefore, closer features have more similar levels and vice versa; particularly, $\vec{\mathcal{L}}_0 \cdot \vec{\mathcal{L}}_{q-1} \simeq 0$ (e.g., white and black pixels have the most different level vectors). The number of level vectors is limited, so input values are quantized to q bins to obtain the right level of each value. Equation (2) formulates the base-level encoding, where $\vec{\mathcal{L}}(v_k)$ denotes the level vector matched by v_k. The two D-dimentional vector $\vec{\mathcal{L}}(v_k)$ and $\vec{\mathcal{B}}_k$ are bound ($\otimes$, element-wise multiplication) together, producing a vector that is dissimilar to both.

$$\vec{\mathcal{H}} = \sum_{k=0}^{d-1} \vec{\mathcal{L}}(v_k) \otimes \vec{\mathcal{B}}_k. \tag{2}$$

We can decode the base-level encoding index by index. To obtain a feature v_i, we first bind $\vec{\mathcal{H}}$ with $\vec{\mathcal{B}}_i$ so we have:

$$\vec{\mathcal{B}}_i \otimes \vec{\mathcal{H}} = \vec{\mathcal{B}}_i \otimes \sum_{k=0}^{d-1} \vec{\mathcal{L}}(v_k) \otimes \vec{\mathcal{B}}_k = \vec{\mathcal{L}}(v_k) + \sum_{\substack{k=0 \\ k \neq i}}^{d-1} \vec{\mathcal{L}}(v_k) \otimes \vec{\mathcal{B}}_k \otimes \vec{\mathcal{B}}_i.$$

Then, we try all possible level vectors $\vec{\mathcal{L}}_x \in \{\vec{\mathcal{L}}_0, ..., \vec{\mathcal{L}}_{q-1}\}$ on the resultant vector to find out the one with highest dot-product:

$$\vec{\mathcal{L}}_x \cdot (\vec{\mathcal{B}}_i \otimes \vec{\mathcal{H}}) = \vec{\mathcal{L}}_x \cdot \vec{\mathcal{L}}(v_i) + \overbrace{\vec{\mathcal{L}}_x \cdot \sum_{\substack{k=0 \\ k \neq i}}^{d-1} \vec{\mathcal{L}}(v_k) \otimes \vec{\mathcal{B}}_k \otimes \vec{\mathcal{B}}_i}^{noise(\approx 0)}. \quad (3)$$

Since base vectors are random, the dot product between two random vectors is close to zero. The expected value of the right-hand side of the above equation is ≈ 0, and $\vec{\mathcal{L}}_x \cdot \vec{\mathcal{L}}(v_i)$ becomes maximum when $x = i$. Once we obtained $\vec{\mathcal{L}}(v_i)$, we can infer v_i from the associated level.

(3) Permutation uses a different way to preserve the spatiotemporal information in time-series data [14], [36]. Instead of using base vectors, it applies $P^{(k)}$ on the level vector of each feature v_k, where $P^{(k)}(\vec{\mathcal{L}})$ denotes circular shift of $\vec{\mathcal{L}}$ by k indexes. Given the pre-generated level vectors , permutation can be formulated as follows:

$$\vec{\mathcal{H}} = \sum_{k=0}^{d-1} P^{(k)}(\vec{\mathcal{L}}(v_k)) \quad (4)$$

A graphical explanation is displayed in Fig. 2 (c).

Decoding the permuting encoding can be done in a similar way to base-level decoding, except for the first step. Instead of binding $\vec{\mathcal{H}}$ by $\vec{\mathcal{B}}_i$, we permute $\vec{\mathcal{H}}$ by $-i$ indexes, then try to find the $\vec{\mathcal{L}}_x$ that maximizes the dot-product.

B. Feature Extraction Attack

In the previous subsection, we show how the vectors of common encoding methods can be decoded to the original data. To decode these vectors, one requires an understanding of the HDC *parameters*, i.e., the base and/or level vectors. Intriguingly, we showcase how these parameters can be extracted using adversarial inputs. Unlike previous research [17], [18], which focused only on decoding random projection encodings and assumed prior knowledge of encoding parameters, our work is the first to introduce various decoding techniques and show how to extract both the parameters and the original data, even without prior knowledge of the encoding parameters.

Our approach operates within a gray-box model, where the input format and encoding algorithm is known, but not its parameters. This setup enables us to manipulate the inputs and observe the corresponding encoded outputs. Such a scenario mirrors situations encountered in federated learning [37] and cloud inference [19]. Note, that our approach can also decode parameters for combined encoding methods, such as RP and Base-Level with permutation, both of which are commonly used in prior HDC studies [14], [36]. The proof is omitted due to space constraints.

(1) RP feature extraction. Random projection is a simpler encoding as it directly projects each scalar input feature by multiplying it with the associated base vector. By feeding an adversary input $V = e^{(i)}$, i.e., a null input vector except $v_i = 1$ in Equation (1), we have:

$$\vec{\mathcal{H}} = \sum_{k=0}^{d-1} v_k \times \vec{\mathcal{B}}_k = 1 \times \vec{\mathcal{B}}_i + \sum_{\substack{k=0 \\ k \neq i}} 0 \times \vec{\mathcal{B}}_k = \vec{\mathcal{B}}_i$$

That is, to infer the i^{th} base $\vec{\mathcal{B}}_i$, we only need to generate an adversary input where the i^{th} feature is 1 and the others are 0. The observed encoded output $\vec{\mathcal{H}}$ is the same as $\vec{\mathcal{B}}_i$. Our approach requires d adversary inputs to extract all d base vectors.

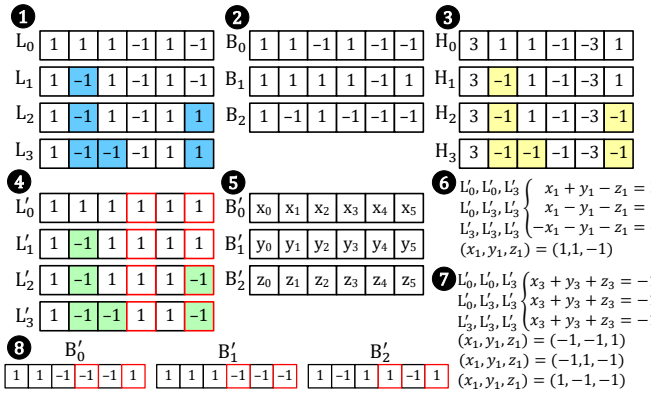


Fig. 3: An example of extracting the level and base vectors.

(2) Base-level feature extraction. Here we deal with two unknowns, level and base vectors. We aim to first infer the level vectors, followed by the base vectors. We denote the inferred level and base vectors by $\vec{\mathcal{L}}'$ and $\vec{\mathcal{B}}'$. Base-level extraction does not have a unique solution, so we set $\vec{\mathcal{L}}'_0 = \{1\}^D$.

To extract the level vectors, we use an adversary $V = \{\alpha\}^d$ and increase α until $\vec{\mathcal{H}} = \sum_{k=0}^{d-1} \vec{\mathcal{L}}(\alpha) \times \vec{\mathcal{B}}_k$ changes. Once α falls in the next bin ($\vec{\mathcal{L}}_1$), a j^{th} component of encoded $\vec{\mathcal{H}}$ changes only if the same component is different in $\vec{\mathcal{L}}^j_0$ versus $\vec{\mathcal{L}}^j_1$. As we set $\vec{\mathcal{L}}'_0 = \{1\}^D$, we observe the changed components of $\vec{\mathcal{H}}$ and set those of $\vec{\mathcal{L}}'_1$ to -1. We keep increasing the α to create $\vec{\mathcal{L}}'_2$ upon the next change of $\vec{\mathcal{H}}$, so on and so forth.

Fig. 3 shows an example with $D = 6$ dimensions, $d = 3$ features/bases and $m = 4$ levels/bins. ❶ and ❷ are the original level and base vectors. In ❸, we have observed all the $m = 4$ encoded vectors by using inputs in the form of $V = \{\alpha, \alpha, \alpha\}$. The yellow elements show changes of $\vec{\mathcal{H}}$ as α increases. In ❹, we set $\vec{\mathcal{L}}'_0 = \{1\}^6$ and generate the rest of the levels by comparing the $\vec{\mathcal{H}}_1$, $\vec{\mathcal{H}}_2$ and $\vec{\mathcal{H}}_3$ with $\vec{\mathcal{H}}_0$. The inferred $\vec{\mathcal{L}}'$ is different from $\vec{\mathcal{L}}$ in dimensions shown in red.

After inferring the $\vec{\mathcal{L}}'$ vectors, we create a system of linear equation for each index, with $\vec{\mathcal{L}}'$ and the observed $\vec{\mathcal{H}}$ as constant, and $\vec{\mathcal{B}}'$ as variable. Fig. 3❻ shows the equations corresponding to $\vec{\mathcal{B}}'^1$ (index 1 of bases). We decided to use d adversary inputs in the form of $V_1 = \{v_{min}, \cdots, v_{min}, v_{max}\}$, $V_2 = \{v_{min}, \cdots, v_{max}, v_{max}\}$, $\cdots$, and $V_d = \{v_{max}, \cdots, v_{max}, v_{max}\}$ to assure the rows are linearly independent. We need only d adversary inputs to solve the linear equations, thus infer all base vectors. Fig. 3❽ shows

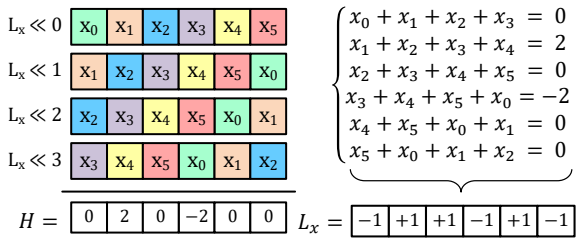


Fig. 4: An example of inferring an unknown level vector $\vec{\mathcal{L}}_x$ in permutation encoding by setting all four features v_k to $\vec{\mathcal{L}}(v_k) = \vec{\mathcal{L}}_x$.

the inferred base vectors $\vec{\mathcal{B}}'$, which are different from $\vec{\mathcal{B}}$, but $\{\vec{\mathcal{L}}', \vec{\mathcal{B}}'\}$ returns the same encoding as the original $\{\vec{\mathcal{L}}, \vec{\mathcal{B}}\}$.

(3) Permutation feature extraction. Similar as the base-level case, we start with inferring the number and size (length) of the quantization bins. We create an all-equal-features adversary input $V = \{\alpha\}^d$ starting with the lowest value for α based on the application. By gradually increasing α, the encoded vector $\vec{\mathcal{H}} = \sum_{k=0}^{d-1} P^{(k)}(\vec{\mathcal{L}}_0)$ remains the same until α falls in the next bin ($\vec{\mathcal{L}}_1$). By continually increasing the α, we can obtain the number and length of the bins.

Then, to extract the x^{th} level vector, we generate an adversary input in which all the features fall into the x^{th} bin. Suppose $\vec{\mathcal{L}}_x = \{x_0, \cdots, x_{D-1}\}$ is the x^{th} level vector that we want to infer, where $x_0, ..., x_{D-1}$ are the D-dimensional elements that are either -1 or 1. Fig. 4 shows an example where $d = 4$ features and $D = 6$ dimensions. For instance, the 4th element in $\vec{\mathcal{H}}$ is computed as $\mathcal{H}^4 = \vec{\mathcal{L}}_x^4 + \vec{\mathcal{L}}_x^5 + \vec{\mathcal{L}}_x^0 + \vec{\mathcal{L}}_x^1 = x_4 + x_5 + x_0 + x_1$. In general, it creates a system of D linear equations with D unknowns, where each equation has d coefficients of 1, and the rest $D-d$ of 0. The linear equations have a unique solution given that the rows of the co-efficient matrix are linearly independent. We need only one adversary input to infer each level vector. While we assumed a typical binding where the level vector at index k is permuted by k, our approach works for arbitrary permutations as we will always get a system of D linear equations.

IV. Privacy-Preserved HDC Inference in Privé-HDnn

The reversibility of HDC encoding that we demonstrated in Section III poses a serious privacy challenge when we need to exchange the trained HDC model with other entities over untrustworthy communication links, such as in cloud-hosted inference [19] and federated learning [20]–[22]. To protect HDC inference against the feature extraction attack, we propose a novel locally sparse encoding technique to obfuscate the information of encoded vectors, so that even if the encoded vectors are exchanged with malicious hosts, the reconstructed data is indiscernible (Section IV-A). Note, that we can also encode the labels into vectors and use the same technique to protect label information. Our essential idea is to make the vectors highly sparse by keeping the most effective elements only, while maintaining high accuracy by leveraging the noise tolerance property of HDC [23], [24]. More concretely, each segment of the vectors is sparsified independently, which also contributes to the compression of the simple index-based model to save communication costs (Section IV-B). We realize the proposed technique with an efficient hardware implementation on FPGAs (Section IV-C). To assist understanding, important notations used in the paper are summarized in Table I.

TABLE I: List of important notations used in the paper.

Symbol	Meaning
d	Dimension of feature vector
D	Dimension of the HD encoded vector
v_i	The ith feature in the feature vector
x_i	The ith dimension in the HD encoded vector
$\vec{\mathcal{H}}$	Encoded HD vector
$\vec{\mathcal{C}}$	Trained class vectors
$\vec{\mathcal{B}}_k$	The kth base vectors during encoding
$\vec{\mathcal{L}}_k$	The kth level vectors during encoding
$P^{(k)}$	Permutation shift by k indexes during encoding
m	Sparse ratio in locally sparse encoding
n	Column split of matrix in locally sparse encoding
w	Number of bits to represent each HD dimension
ϵ	Probability of privacy leakage
δ	Probability of breaking the differential privacy
σ	Noise variance
$\Delta_{\mathcal{M}}$	Sensitivity of a randomized algorithm $\mathcal{M}$
κ	Encoding l_2 norm limit
q	Fraction of data samples in one batch
N	Dataset size
L	Batch size
E	Number of epochs
T	Number of training iterations

A. Locally Sparse Encoding

In order to mitigate the potential exposure of information resulting from encoding through adversary links or cloud services, we employ a sparse representation approach for HDC without sacrificing significant accuracy. The key idea is to leverage HDC's tolerance to large noise levels compared to neural networks [23], [24]. This resilience stems from HDC's distributed representation, where all dimensions in a hypervector contribute equally, each containing low-precision values such as bipolar or integer components. Consequently, when certain dimensions are zeroed out in an HDC-encoded vector, the overall accuracy remains relatively stable. Leveraging this insight, our design employs sparsification as a lossy compression technique that diminishes recoverable information while maintaining accuracy and reducing communication costs.

An effective method for sparsification involves retaining the top-k components within the encoded vector while zeroing out the remainder. This approach is reasonable due to the higher contribution of these components to the similarity score, as measured by the dot-product. The top-k mechanism also aligns with the sensory systems of numerous organisms, including the olfactory system of fruit flies, which executes a winner-take-all thresholding mechanism on the expanded representation of input odors [10], [38].

Our objective revolves around achieving sparsification in a manner that optimizes both hardware efficiency and communication efficacy. Although selecting the top k components from the encoded vector may seem straightforward in a processor through sorting operations, this method is irregular and not

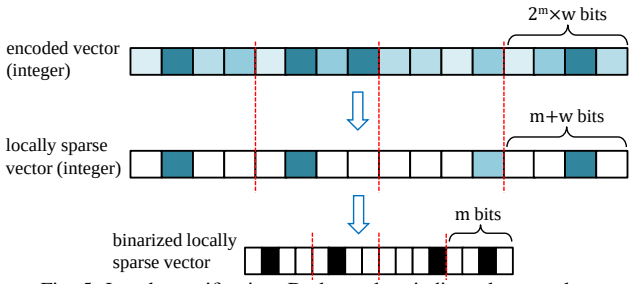


Fig. 5: Local sparsification. Darker colors indicate larger values.

efficiently executed in low-power devices [39]. Instead of targeting the top-k percentage of components across the entire vector, we opt for a local sparsification strategy involving the identification of local maxima. This concept is illustrated in Figure 5. For a designated non-sparse rate of $\frac{1}{2^m}$, we segment the encoded vector into units comprising 2^m elements. We then replace all elements within each segment, except the maximum one, with zeros. Alternatively, bipolar notation can be employed, which utilizes -1's instead of 0's. This design approximates the top-k component selection within a vector without necessitating a full sorting procedure. As a result, we bypass resource-intensive computations on low-power devices while maintaining efficiency in implementation.

B. Communication Compression

An advantage inherent in our approach lies in the reduction of communication overhead. As shown in Figure 5, instead of transmitting $2^m \times w$ bits for each segment (consisting of 2^m components, each with w bits), we only require $m+w$ bits. This allocation consists of m bits to denote the nonzero index and w bits for its associated value. Given that HDC encodings exhibit compatibility with quantization [40], we can further compress the encoding vectors to binary. In this case, only m bits per segment are needed.

Contrastingly, conventional top-k sparsification entails additional complexities due to the requirement of sorting to identify the top-k elements. Moreover, $\log(D)$ bits per nonzero component are essential to represent the index, along with a cumulative $k \cdot w$ bits to record the values of nonzero components. In binary vector scenarios, the top-k sparsification needs $\log(D)$ bits per non-zero components. We remind the readers that D denotes the dimension of the encoded vectors. Note that bit reduction is independent of the underlying encoding hardware, so communication improvement holds for all devices.

C. Efficient FPGA Implementation

Since we are interested only in the *index* of the maximum components rather than their exact values, we can use more efficient approximate encoding methods to obtain the components. The efficiency gain of using approximate encoding can be realized and maximized on FPGA. For *base-level* and *permutation* encoding methods that vastly use binary popcount for bundling, previous work [41] used an approximate binary popcounts that simplify these encoding methods by replacing part of the sum operations with the majority

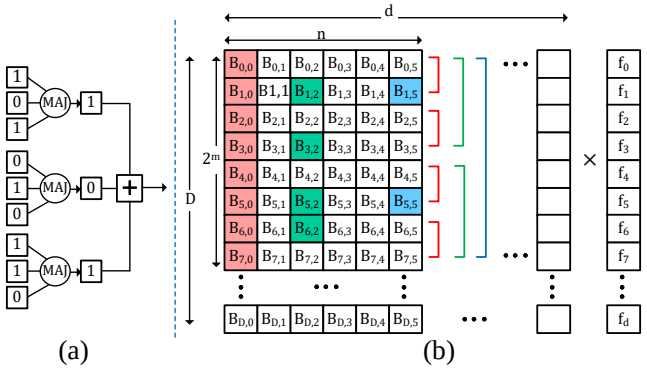


Fig. 6: (a) The approximate binary popcounts in previous work [41] that is implemented by the majority functions, suitable for sparse base-level and permutation encoding, (b) our novel approximate locally-sparse random projection encoding that is friendly for low-power hardware (e.g., FPGAs).

functions (Fig. 6(a)). While employing the majority function significantly reduces FPGA resource usage by up to 80%, and despite the approximative nature of the encoding components, they are only suitable in the context of sparsification where the relative value of components matters.

In contrast, here in Privé-HDnn, we focus on a novel approximate implementation for locally-sparse *random projection* (RP) encoding. Fig. 6(b) shows our FPGA-friendly implementation of the proposed sparse RP encoding. RP involves the multiplication of the projection matrix with the input vector, whereby the multiplication of an entire row of the projection matrix with the input vector yields one output component. Our key idea is to bypass some matrix elements when generating the localized sparsified encoding output, as outlined in Section IV-A. We split the original matrix into partitions of 2^m rows and n columns each, with $\frac{1}{2^m}$ non-sparse rate. Among these 2^m rows, the output of only one of them will be 1 and the remaining will be zeroed out. At each step, one bit of a partition's column is multiplied with the corresponding feature. The partial encoding result compares the adjacent values, which belong to adjacent rows, and allows the maximum one to proceed to the next step. In Fig. 6(b), the red bits are for the first step, active in all rows, while the green bits are activated for the rows that passed to the second step. Accordingly, m steps are needed to finalize the encoding.

We choose $m \leq n$, which means that even for a row that proceeds to till the last step, $n-m$ bits of it still remains unused. That is, the projection matrix is also sparse. The sparsity pattern varies among the partitions so none of the matrix columns is entirely zero. This approach does not require local sorting as, at each step, adjacent outputs are compared and the maximum index is passed to to the next steps. This helps to realize a deterministic and hardware (e.g. FPGA) friendly data flow.

V. Differential Private HDC Training in Privé-HDnn

Ensuring privacy in HDC training is essentially challenging, given that data inputs are mapped onto reversible vectors. As we showed in Section III, feature extraction can infer the encoding parameters using adversarial inputs across all common encoding techniques. Previous work of Prive-HD [17] targeted

private iterative HDC training by post-training noise injection. However, Prive-HD [17] only applies to one-pass training and substantially underestimates the amount of additive noise, as we show more details in Section V-C. In fact, post-training noise injection is not a viable approach for iterative training.

In this section, we introduce the training design of Privé-HDnn, which enables differentially private yet accurate HDC classification. We start with the necessary background for differential privacy (DP) in Section V-A. We then show how DP can be applied to protect the data privacy of HDC for both one-pass and iterative (multi-pass) training using state of the art HDC encoding techniques in Section V-B and V-C. While most classification applications have excellent accuracy when encoding raw data into HDC vectors, our recent work showed that for complex data like images, such as CIFAR-100 [42], we need to use a subset of a pretrained neural network as a feature extractor prior to HDC-based encoding and training [43]. In order to handle complex datasets, Privé-HDnn adopts a hybrid CNN-HDC approach to facilitate accurate classification as well as privacy guarantees (Section V-D). Table I lists the key notations.

A. Differential Privacy (DP)

Differential Privacy serves as a widely adopted privacy standard across various applications [6], [17]. DP enforces strict indistinguishability, ensuring that an algorithm does not reveal patterns in the original data, even down to the difference of a single record. As a robust privacy protocol, DP effectively prevents attackers from probing the original data. Specifically, a randomized algorithm $\mathcal{M}$ with domain $\mathcal{D}_{\mathcal{M}}$ is (ϵ, δ)-differentially private if for any two adjacent datasets $\mathcal{D}, \mathcal{D}_{-1} \in \mathcal{D}_{\mathcal{M}}$ that differ in one record (an input sample), for all output subset $\mathcal{S}$ of $\mathcal{M}$ the following holds [6]:

$$Pr[\mathcal{M}(\mathcal{D}) \in \mathcal{S}] \leq e^{\epsilon} Pr[\mathcal{M}(\mathcal{D}_{-1}) \in \mathcal{S}] + \delta, \tag{5}$$

which means that observing $\mathcal{D}$ after $\mathcal{D}_{-1}$ increases the probability of an event by no more than e^{ϵ}. The additive term δ allows breaking the DP by a probability of δ, which is usually selected to be smaller than $\frac{1}{|\mathcal{D}|}$.

A common approach to satisfy DP is applying a Gaussian noise proportional to the *sensitivity* of the algorithm, defined as the maximum amount of change of the output if the input differs in one element. For Gaussian noise, we use ℓ_2 norm for sensitivity, i.e., $\Delta_{\mathcal{M}} = \|\mathcal{M}(\mathcal{D}) - \mathcal{M}(\mathcal{D}_{-1})\|_2$. Thus,

$$\mathcal{M}_{DP}(\mathcal{D}) = \mathcal{M}(\mathcal{D}) + \mathcal{N}(0, \Delta_{\mathcal{M}} \cdot \sigma), \tag{6}$$

in which $\Delta_{\mathcal{M}} \cdot \sigma$ is the standard deviation of the noise. For a fixed δ and a privacy target ϵ, the parameter σ can be obtained to satisfy $\delta \geq \frac{4}{5} e^{-\frac{(\sigma\epsilon)^2}{2}}$ [6]. A smaller ϵ or δ (i.e., less likelihood of leakage) demands larger noise parameter σ.

B. Private One-pass Training

Suppose $\vec{\mathcal{H}}_i$ and y_i are the encoded vectors and true label of the ith sample. One-pass HDC training simply accumulates the encoded vectors of the same label to form class vectors, as visualized in Fig. 2 (d) and as follows:

$$\vec{\mathcal{C}}_j = \sum_{i \text{ s.t.} y_i = j} \vec{\mathcal{H}}_i \tag{7}$$

When a raw input V is discarded from the dataset, only one of the class vectors is affected, where $\vec{\mathcal{C}}_{-1} = \vec{\mathcal{C}} - \vec{\mathcal{H}}(V)$. Thus, the sensitivity of one-pass model is $\Delta_{\mathcal{M}} = \| \vec{\mathcal{H}}(V) \|_2$. This is a valuable property as it shows the sensitivity of one-pass training does not depend on the dataset size. We can train a one-pass HDC model on arbitrary datasets and add a noise based on $\Delta_{\mathcal{M}_{max}}$, which is independent of the encoding and the dataset. For inference, suppose $\vec{\mathcal{C}}$ are the original class vectors while $\vec{\mathcal{C}}'$ are class vectors with additive noise $\vec{\mathcal{N}}$, we will predict the class $\tilde{y}$ as:

$$\tilde{y} = \underset{i \in C}{\operatorname{argmax}} \vec{\mathcal{C}}'_i \cdot \vec{\mathcal{H}} = \underset{i \in C}{\operatorname{argmax}} \vec{\mathcal{C}}_i \cdot \vec{\mathcal{H}} + \vec{\mathcal{N}} \cdot \vec{\mathcal{H}} \tag{8}$$

A large noise can impact the value of $\vec{\mathcal{N}} \cdot \vec{\mathcal{H}}$ and change the scores rank. Each encoded component $\vec{\mathcal{H}}^j$ belongs to $[-d, d]$ (with $v_{max} = 1$). This implies a sensitivity of $\Delta_{\mathcal{M}_{max}} = d\sqrt{D}$ that demands a significant amount of noise $\mathcal{N}\big(\mu = 0, \sigma = \frac{\Delta_{\mathcal{M}_{max}}}{\epsilon} \sqrt{2 \log(\frac{1.25}{\delta})}\big)$ which can be as large as the class values themselves. This is because DP considers the worst case of $\vec{\mathcal{H}}$ where the components are $\{\pm d\}$.

To solve this issue, we need to ensure that $\vec{\mathcal{H}}$ does not exceed an expected bound, so we can use a smaller noise in accordance. Thus, we *normalize* and *clip* the encoded vectors by $\vec{\mathcal{H}}' = \vec{\mathcal{H}} / \max(1, \frac{\|\vec{\mathcal{H}}\|_2}{\kappa})$ which ensures $\|\vec{\mathcal{H}}'\|_2 \leq \kappa$. By choosing, e.g., $\kappa = 1$, we achieve $\Delta_{\mathcal{M}_{max}} = 1$ which shrinks the variance of noise. By normalizing the encoded vectors, the values of class vectors also scale accordingly. Therefore, normalization alone cannot mitigate the impact of noise. Clipping bounds the values and helps avoid the worst-case scenario of paying extra sensitivity ($\Delta_{\mathcal{M}}$) cost.

C. Private Iterative Training

Iterative (multi-pass) HDC training improves the HDC accuracy by evaluating the model on the training data. In case of wrong prediction, the encoded vector is subtracted from the mispredicted class to make them less similar, and is added to the expected class once again. Let $\vec{\mathcal{C}}_f$ represent the mispredicted class vector, and $\vec{\mathcal{C}}_t$ denote the ground-truth class vector. The iterative training operations for an incorrectly predicted sample $\vec{\mathcal{H}}$ are outlined below.

$$\begin{aligned} \vec{\mathcal{C}}_f &= \vec{\mathcal{C}}_f - \vec{\mathcal{H}}, \\ \vec{\mathcal{C}}_t &= \vec{\mathcal{C}}_t + \vec{\mathcal{H}}, \end{aligned} \tag{9}$$

Privacy of iterative training cannot simply be obtained by finding the sensitivity and adding the post-training noise. In iterative training, a discarded or extra input affects the model updates in the subsequent epochs. Normalizing the encoded vectors guarantees $\Delta_{\mathcal{M}_{max}} = \|\vec{\mathcal{H}}\|_2 \leq \kappa$ for one-pass learning, but it does not hold for iterative training. The work in [17] was built on this assumption. For the datasets of Section VI, after normalizing the vectors to $\Delta_{\mathcal{M}_{max}} = \| \vec{\mathcal{H}} \|_2 \leq 1$, we observe that excluding only one input makes $\Delta_{\mathcal{M}}$ of $[\sim 20 - 43]$ which makes the required noise σ intractable.

To address this challenge and realize differentially private multi-pass training, we propose to use the composition property of DP: a mechanism with a series of ϵ_i-private steps is $\sum \epsilon_i$-differentially private [44]. Thus, we can preserve privacy at the batch level, where the total privacy cost of

Algorithm 1 Differentially private iterative HDC training

Inputs: Dataset $\mathcal{D} = \{\vec{V}_1, \cdots \vec{V}_N\}$, batch size L, noise variance σ^2, encoding l_2 norm limit κ
Output: Class vectors $\vec{\mathcal{C}} = \{\vec{\mathcal{C}}_1, \cdots \vec{\mathcal{C}}_C\}$

1: $\vec{\mathcal{H}}_\mathcal{D} \leftarrow$ enc($\mathcal{D}$) // encode the train data
2: $\vec{\mathcal{C}} \leftarrow 0$
3: **for** t in $[1:T]$ **do**
4: $\quad\vec{\mathcal{G}}[1:C] = \{0\}^D$ // initialize class update vectors per iteration
5: $\quad L_t \leftarrow$ random_select($\vec{\mathcal{H}}_\mathcal{D}$, L) // $L = 1$ in normal training
6: $\quad$**for** $\vec{\mathcal{H}}$ in L_t **do**
7: $\quad\quad\ell = \text{argmax}_i\, \vec{\mathcal{C}}_i \cdot \vec{\mathcal{H}}$ // inference on the train sample
8: $\quad\quad$**if** $\ell \neq \ell_\mathcal{H}$ **then**
9: $\quad\quad\quad\vec{\mathcal{H}}' \leftarrow \vec{\mathcal{H}} / \max(1, \frac{\sqrt{2}\,\|\vec{\mathcal{H}}\|_2}{\kappa})$ // bound the sensitivity
10: $\quad\quad\quad\vec{\mathcal{G}}[\ell] \leftarrow \vec{\mathcal{G}}[\ell] - \vec{\mathcal{H}}'$ // subtract from mispredicted class ℓ
11: $\quad\quad\quad\vec{\mathcal{G}}[\ell_\mathcal{H}] \leftarrow \vec{\mathcal{G}}[\ell_\mathcal{H}] + \vec{\mathcal{H}}'$ // add to the golden class $\ell_\mathcal{H}$
12: $\quad\quad$**end if**
13: $\quad$**end for**
14: $\quad\vec{\mathcal{G}} \leftarrow \vec{\mathcal{G}}/L + \vec{\mathcal{N}}(0, \kappa{\cdot}\sigma)$ // average the updates and add noise
15: $\quad\vec{\mathcal{C}} \leftarrow \vec{\mathcal{C}} + \vec{\mathcal{G}}$ // update the classes
16: **end for**
17: **return** $\vec{\mathcal{C}}$

training is the total cost of the batches and the total amount of required noise is smaller. This new design involves the injection of additive noise during each epoch, allowing the model to simultaneously undergo fine-tuning in response to the introduced noise. Specifically, if the training is divided into batches of $L = qN$ randomly selected samples (for N being the dataset size), following the privacy amplification theorem, each iteration is $(q\epsilon, q\delta)$-private with respect to the whole dataset [6], [45]. For T training iteration ($E = q \times T$ epochs), existing DP theory [6] provides a tighter bound of $(q\epsilon\sqrt{T}, \delta)$ on privacy of the composite model with the following additive noise:

$$\sigma \geq c\frac{q}{\epsilon}\sqrt{T\ln(1/\delta)} \tag{10}$$

Algorithm 1 details the proposed private iterative learning for HDC. The lines with comments in red are exclusive to private training. During each of the T training iterations, a random batch of encoded data is selected (line 5). Similar to DNN training, a forward inference pass is run over the batch (lines 6–7). For the data with mispredicted labels (line 8), the normalized vectors are saved (similar to gradients of different samples in neural networks) and averaged followed by noise injection (line 14). The algorithm is fairly similar to normal HDC training, except that the updates are made per batch for L samples, as opposed to per sample, and the encoded vectors are normalized to bound the worst-case sensitivity as discussed in subsection V-B. Our new design enables differential private iterative HDC training which has not been addressed in previous work [17].

D. Private HDC Classification on Complex Datasets

HDC achieves poor accuracy for complex image datasets when using state of the art HDC encoding directly on the raw image data. A way to resolve this issue is to train a Convolutional Neural Network (CNN)-based feature extractor and use HDC for classification over the extracted features [20], [43]. The hybrid architecture is called HDnn. The conventional

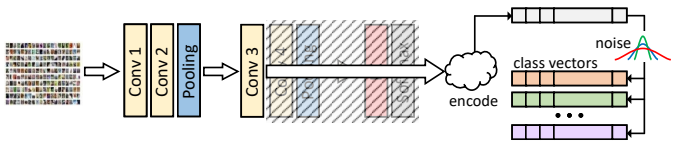


Fig. 7: Privé-HDnn for private training on complex datasets such as images. A shallow CNN extracts the features (only once) as raw inputs for HDC, which performs private training according to Algorithm 1. A random projection encoder is used for HDC.

HDnn flow consists of (i) extracting image features using a subset of a pretrained CNN, (ii) training an HDC model over the extracted features, and (iii) attaching the trained HDC to the CNN and finetuning the CNN to compensate potential accuracy loss. HDnn offers other advantages such as few-shot learning [16] due to the capability of HDC in learning from fewer data, and more straightforward in-field learning as only the HDC head needs to be updated.

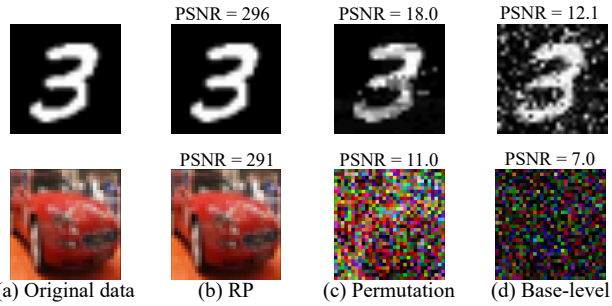
In Privé-HDnn, we leverage the HDnn structure for private learning on complex datasets like images, as shown in Fig. 7. A major issue with CNNs is their intolerance to additive noise due to the highly sensitive training process using gradient descent. Previous studies on privacy-preserving training of CNNs have shown low accuracy, e.g. 72% on CIFAR-10 [46]. Some of the previous works use simple networks and even keep some of the layers fixed [6]. We bypass the noise intolerance of CNNs by using transfer learning over a *public dataset* from a different distribution. That is, we keep the pre-trained CNN feature extractor unchanged in Privé-HDnn. We *do not* finetune it on the target dataset after attaching the HDC. Hence, it acts as a constant function and we do not need to add noise on the CNN part as it does not learn nor expose any information of the target dataset. In other words, the CNN feature extractor retains its exact functionality.

On the other hand, the HDC classifier head is trained on the extracted features of the target dataset with differential privacy using the proposed iterative method of Algorithm 1. The extracted features act as raw data for HDC algorithm and are fed to a random projection encoder. With Privé-HDnn, the features (i.e., the outputs of CNN) are secure, thus the original data that generates those features is also protected.

VI. Evaluation

A. Experimental Setup

(1) Benchmarks: We use standard domain benchmarks to evaluate privacy training and inference: `FACE` detection with face versus non-face labels [47], `UCIHAR` user activity recognition using smartphone (five activities such as sitting, standing, etc.) [48], `PAMAP2` physical activity monitoring using inertial measurement and heart rate monitor (12 activities such as cycling, running, etc.) [49], and `ISOLET` voice of English alphabet recognition [50]. We also use `MNIST` [51], `CIFAR-10` [52] and `CIFAR-100` [52] images to evaluate Privé-HDnn that requires a pre-trained CNN feature extractor. We would like to emphasize that HDnn, the combination of HDC and CNNs achieves the same or better performance compared to DNNs [43] in non-private setting. However, private training is more challenging because one also needs to safeguard against potential privacy breach. For example, previous DNN studies



(a) Original data (b) RP (c) Permutation (d) Base-level

Fig. 8: Examples of decoded data from MNIST and CIFAR-10 under different encodings.

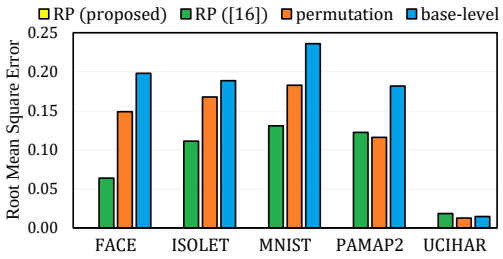


Fig. 9: RMSE between the original and the decoded data for different encodings (RP, base-level, permutation) along with comparison with analytical RP decoding of [17]. Our method of RP decoding perfectly reconstructs the original data and achieves an average RMSE of ≈ 0 among all benchmarks.

have demonstrated lower accuracies, such as a mere 72% on CIFAR-10 [46].

(2) Implementation: We implement Privé-HDnn with Python 3.9 running on a Linux machine with an Intel Core i7-12700 CPU. We use accuracy, decoding RMSE, and error resiliency as major performance metrics. Note, that all datasets are normalized to [0, 1], thereby bounding the RMSE values within [0, 1]. If not specified otherwise, we utilize an HD dimension of $D = 5000$.

To evaluate the efficiency of the proposed locally sparse encoding, we accelerate it on a Xilinx Kintex-7 KC705 FPGA Kit, where we estimated the power using Xilinx Power Estimator (XPE) [53]. For CPU, we estimate the power consumption using CPU Energy Meter [54]. For the case study of Privé-HDnn, in addition to CPU implementation, we use a CUDA implementation on NVIDIA's GTX 1080 Ti GPU for a fair comparison of its training time with previous DNN works as they mainly use GPUs. We emphasize that efficient HDC libraries can support implementation on specific hardware, such as F5-HD [55] for FPGAs, OpenHD [56] for GPUs and HDCC [57] for CPUs.

In Privé-HDnn, we use ResNet-18 or ResNet-50 network [58] pretrained on ILSVRC 2012 dataset [59] as the constant feature extractor. For privacy purposes, we do not finetune the public model after attaching the HDC head in place of the last layers. Note, that Privé-HDnn is a general private learning methodology, thus ResNet can be replaced by any pretrained feature extractor identified by the user.

(3) Baselines: For decoding performances, we compare with the state-of-the-art HDC privacy works, **Prive-HD** [17] and **PRID** [18] in terms of HDC reconstruction error.

B. Feature Extraction Attack Performance

In Section III, we provide a list of new decoding methods (i.e., feature extraction attack) to decode HDC model and reconstruct the original data, showing that HDC models are reversible across all major encoding methods. This implies significant privacy challenges to HDC training. To visualize the proposed attack, Fig. 8 shows decoded examples of a handwritten digit from MNIST and a car image from CIFAR-10 under different encoding techniques. The reconstructed images closely resemble the original data, confirming the effectiveness of our proposed attack. Particularly, the proposed feature extraction attack proves more effective on simpler datasets; for instance, MNIST is easier to reconstruct than CIFAR-10. Among the three encoding methods, RP exhibits the least privacy and the highest reconstruction similarity, while permutation and base-level present greater difficulty in reconstruction, attributed to the complexities involved in solving linear equations.

Root Mean Square Error (RMSE): To assess the efficacy of our feature extraction attack, we measure the RMSE between the original and decoded samples across all investigated encoding methods (RP, base-level, permutation). All RMSEs are bounded by [0, 1] thus are comparable. A lower RMSE indicates a closer resemblance between the decoded and original samples, signifying a more successful attack. Additionally, a comparison is drawn with the state-of-the-art RP decoding technique from earlier research [17]. Our method of decoding random projection (RP) exactly reconstructs the original data with an average RMSE of approximately 0, while prior work [17] uses an analytical approach to decode the RP and achieves an average RMSE of 0.09. PRID [18] also decodes RP encoding and reports a PSNR of $\approx$51 dB for MNIST, whereas our approach achieves an average PSNR of $\approx$250 dB.

The average RMSE of decoding the permutation and base-level is relatively higher, i.e., 0.126 and 0.164 respectively. The higher RMSE of these methods compared to RP decoding is mainly due to the noise term in Equation (3) that affects finding the right $\vec{\mathcal{L}}_x$ that achieves the maximum score. However, even with an RMSE of 0.23 for MNIST base-level decoding, which is the highest RMSE among benchmarks, the constructed image is still recognizable. The quality of decoding is determined by the scale of the noise term in Equation (3), which is directly related to the number of features (d in Equation (3)). UCIHAR has the least number of features per sample ($d = 27$), so its decoding achieves a small RMSE of < 0.02. The other datasets have 561 to 784 features, and thus they have larger RMSEs.

C. Inference Privacy

We evaluate the impact of the locally sparse encoding which aims at preserving the privacy during HDC inference. In the previous subsection we observed that RP encoding exhibits the highest exposure of data. Thus, for brevity, we focus on the results of obfuscating the RP encoding. Fig. 10 illustrates reconstructed examples of a sinusoidal wave, a handwritten digit (from MNIST), and a car image (from CIFAR-10) following locally sparse encoding, exemplifying

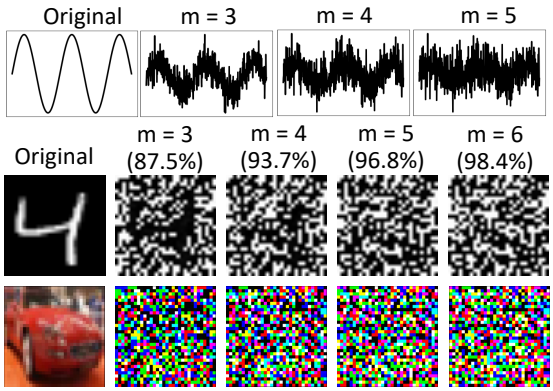


Fig. 10: Examples of decoding the sparse vectors of RP encoding on a sinsoid wave, MNIST and CIFAR-10.

TABLE II: Accuracy and decoding RMSE for locally sparse encoding. The RMSE serves as a measure of the effectiveness of the feature extraction attack, with a higher RMSE indicating stronger privacy protection.

Sparsity	FACE (95%)	UCIHAR (95%)	PAMAP2 (94.5%)	ISOLET (94%)	MNIST (95.5%)
87.5% (m = 3)	-0.3%	0.0%	-0.1%	-0.2%	-0.1%
93.7% (m = 4)	-1.1%	-1.6%	-1.7%	-1.3%	-2.2%
96.8% (m = 5)	-2%	-2.9%	-5.5%	-3.0%	-7%
98.4% (m = 6)	-3.3%	-20.0%	-10.0%	-5.8%	-9.5%
RMSE	0.173–0.184	0.282–0.346	0.72–0.735	0.616–0.624	0.316–0.332

the raw information leakage during the feature extraction attack. In simpler data samples like the sinusoidal wave and handwritten digit, reconstruction becomes increasingly difficult as higher sparsity (larger m) is enforced during encoding. The more complex dataset CIFAR-10 makes reconstruction nearly impossible after sparse encoding. Quantitatively, the average PSNR among all the inputs drops below 12 dB, and RMSE of the sinusoidal wave increases to 0.7 (versus ~ 0 of original RP). These results collectively demonstrate that our proposed sparse encoding technique substantially improves privacy and provides robust protection against the feature extraction attack.

RMSE and accuracy trade-off: Table II summarizes the accuracy of benchmarks for different local sparsity rates, where, e.g., $m = 4$ means only one dimension out of $2^4 = 16$ consecutive dimensions remains one, so the sparsity rate is $1 - \frac{1}{16} = 93.7\%$. The first row of the table reports the baseline non-sparse accuracy (e.g., 95.0% for FACE). Increasing the sparsity ratio improves computational efficiency and enhances privacy, albeit at the expense of accuracy. However, thanks to HDC's noise resilience property, the decrease in accuracy is modest: the average accuracy loss of $m = 3$ ($m = 4$) is only 0.14% (1.6%) among all benchmarks.

The last row of the Table II reports the RMSE *range* of the decoded benchmarks, while the detailed RMSE on three of the datasets are plotted in Fig. 11. Without sparsification, the RMSEs of all decoded benchmarks are ~ 0 as discussed in the previous subsection. As reported in Table II and Fig. 11, when $m = 3$, a marginal accuracy decrease of merely 0.14% leads to an elevation of the RMSE to 0.421. This implies that the extracted features deviate by 0.421 on average from their original values, with features constrained between 0 and 1. Increasing m further amplifies the RMSE, signifying a greater deviation during reconstruction and thus better preservation of privacy. Therefore, our locally sparse encoding successfully

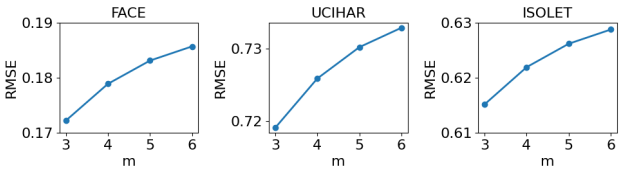
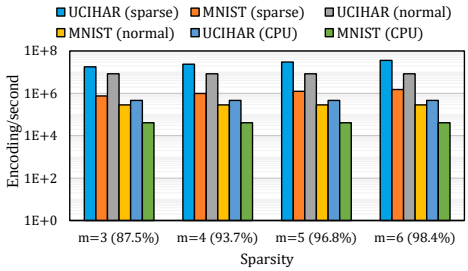
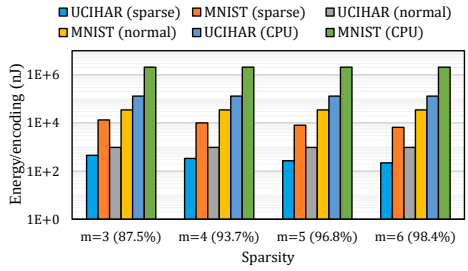


Fig. 11: The RMSE between the original and decoded samples (via feature extraction attack) under various sparsity rates of m.



(a) Encoding performance



(b) Energy consumption

Fig. 12: Comparing the encoding performance of the approximate locally sparse encoding (FPGA) versus normal encoding on FPGA and CPU.

obfuscate the HDC encoded vectors, protecting the inference privacy against decoding attacks, while sacrificing negligible accuracy. Such result is notable especially when the features are independent measurements such as a vector of temperature, heart rate, and inertial measurements (as in PAMAP2), so the exact values cannot be recovered.

D. Inference Efficiency and Error Resiliency

(1) FPGA performance: Fig. 12 (a) compares the encoding performance of the approximate locally sparse encoding implemented on FPGA versus normal (non-sparse) encoding on FPGA and CPU. We use $D = 4K$ and $n = 2m$ (n determines the matrix sparsity; see Fig. 6(b)). Note that the performance of non-sparse models is constant for different sparsity levels. We choose UCIHAR and MNIST as these benchmarks contain the least and highest number of features respectively.

The sparse FPGA implementation of MNIST has yielded a performance increase of 2.6× (5.3×) when compared to the non-sparse FPGA, at sparsity levels of $m = 3$ ($m = 6$). Similarly, for UCIHAR, the performance gains range from 2.1× ($m = 3$) to 4.3× ($m = 6$). This variation is somewhat less pronounced in the case of UCIHAR due to its fewer features, which leads to a slightly reduced savings due to the control overhead.

Compared to the CPU implementation, our sparse FPGA approach showcases a performance boost of 38× ($m = 3$) to 79× ($m = 6$) for UCIHAR, and 19× ($m = 3$) to 37× ($m = 6$) for MNIST dataset. The greater FPGA enhancements observed

TABLE III: Execution time (ms) for decoding

Benchmark	RP	Permutation	Base-Level
FACE	0.015	0.86	1.58
ISOLET	0.015	1.14	1.49
MNIST	0.019	1.32	1.99
PAMAP2	0.013	1.06	1.38
UCIHAR	0.002	0.09	1.01

TABLE IV: Communication bit reduction.

Encoding	m = 3 (87.5%)	m = 4 (93.7%)	m = 5 (96.8%)	m = 6 (98.4%)
Local Sparsity	63%	75%	84%	91%
Top-k	0%	25%	63%	81%

in UCIHAR can be attributed to its smaller number of features, magnifying the influence of costly data movement in CPU.

Table III reports the execution time in milliseconds for decoding each input sample. In the case of RP decoding, each feature requires D multiplications, resulting in an average decoding time of 0.013 mSec per input sample. On the other hand, both permutation and base-level encodings involve $D \times \mathcal{L}$ multiplications (with $\mathcal{L}$ denoting the number of levels), yet the decoding process for these methods remains efficient, taking 0.90 mSec for permutation and 1.49 mSec for base-level encoding.

(2) FPGA energy: Fig. 12 (b) compares the energy consumption of the proposed approximate sparse encoding (FPGA) with the non-sparse FPGA and CPU encoding. Since the power consumption of the sparse and normal FPGA implementation is similar (8 W for UCIHAR and 10 W for MNIST), the energy saving of sparse implementation over the non-sparse FPGA is similar to the speedup numbers, e.g., 2.6× (5.3×) with $m = 3$ ($m = 6$) sparsity level for MNIST. CPU consumes 60 W for UCIHAR and 85 W for MNIST. Accordingly, the proposed sparse FPGA implementation achieves 289× ($m = 3$) to 590× ($m = 6$) higher performance for UCIHAR, and 157× ($m = 3$) to 315× ($m = 6$) for MNIST, compared to the CPU implementation.

Trade-offs between Accuracy and Efficiency: Table II and Fig. 8 show the accuracy and computational efficiency under locally sparse encoding, with sparsity rates from $m = 3$ to 6. In practice, the sparsity rates act as a knob to control the trade-offs between accuracy and efficiency. A higher sparse ratio results in faster encoding speed but at the cost of slighted degraded accuracy.

(3) Communication efficiency: The conservation of data communication energy depends on the transmission distance [60]. Nevertheless, due to the linear correlation between transfer energy and the number of bits, we can estimate the transmission energy based on the quantity of transmitted bits.

Table IV reports the communication bit reduction achieved through the proposed local sparsity approach, comparing it against the top-k method which demands $\log D$ bits for each non-zero component, as elucidated in Section IV-B, where $D = 4000$ in this setting. With $m = 3$ (corresponding to 87% sparsity), the transmit energy experiences a 62% reduction (compared to 0% reduction in the case of top-k). This reduction further escalates to 91% for $m = 6$. It is worth noting that for $m = 3$, representing the encoded vector of the top-k method along with its index information leads to an increase in the number of bits when contrasted with a simple binary representation. Consequently, for $m = 3$, the top-k method transmits the information in its original format, resulting in a compression rate of 0%.

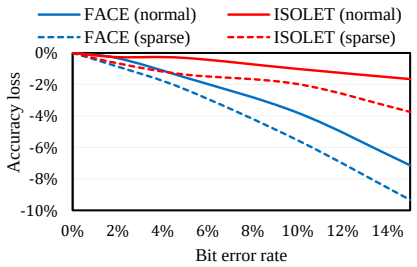


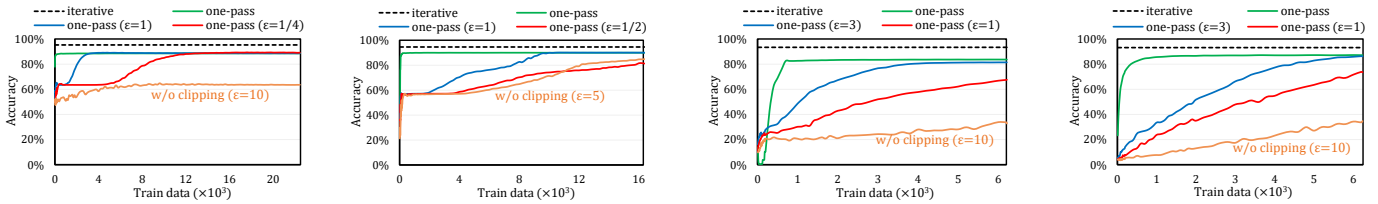
Fig. 13: Impact of bit error on the accuracy of conventional and sparse encoding. $D = 4000$ for both datasets.

(4) Error resiliency. In the context of normal HDC encoding, there is an equal expectation of zeros and ones within the encoded components. Conversely, in sparse encoding, the majority of bits take on the value of zero. Consequently, a bit-flip that toggles encoded components from zero to one, and vice versa, exerts a more substantial influence on the similarity (dot-product) outcome. Fig. 13 compares the impact of bit error on the accuracy of models using the conventional non-sparse and locally sparse encodings. This analysis considers bit errors on the final encoded vectors, so the result is independent of the underlying hardware. Such errors could stem from hardware-related issues, such as emerging memory cell anomalies [61], or communication bit errors. We adopt the communication error model outlined in [24], which involves the adjustment of parameters within the WiFi protocol stack (802.11n). Additionally, the model alters the distance between the transmitter and receiver, gathering signal-to-noise ratio (SNR) data via the Friis propagation loss model.

We show the results on FACE and ISOLET datasets that have the smallest and largest number of classes, respectively. From Fig. 13, we can see that while normal encoding exhibits slightly better robustness than using sparse encoding, sparse encoding equally demonstrates remarkable error robustness. With 2% bit error rate (equivalent to 180 meter-distant transfer), the average accuracy loss of sparse encoding is 0.77% (versus 0.30% of normal encoding). With ∼6.5% bit error (250 meter distance), the accuracy loss is 2.37% (versus 1.32% of normal encoding). At these error rates, the accuracy of conventional ML algorithms such as logistic regression and support vector machine drops to zero (random) [24].

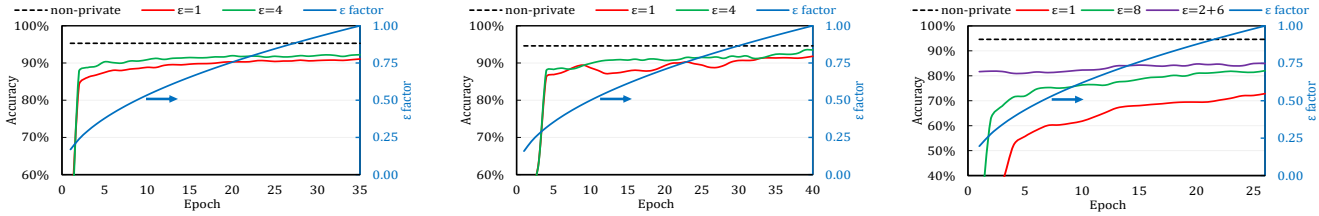
E. One-pass Training Privacy

In Section V, we propose differentially private one-pass and iterative training in Privé-HDnn. Fig. 14 shows the results of private one-pass training. We set δ=10^{-5} ($< \frac{1}{|\mathcal{D}|}$ for all datasets) and use D=$4K$ dimensions. At we feed more training data, we evaluate the accuracy using the entire test set for inference. The shown results are the average of 20 runs. The dashed curve (labeled *iterative*) shows the ultimate accuracy of non-private iterative HDC training, which indicates the accuracy upper bound for all private training approaches. The green curve (labeled *one-pass*) shows the typical non-private non-pass HDC training which achieves 87.4% accuracy on average (−6.8% versus non-private iterative training). The sharp spike of one-pass learning curves demonstrates the fast



(a) FACE detection (b) UCIHAR activity recognition (c) PAMAP2 physical activity recog. (d) ISOLET speech recognition

Fig. 14: Accuracy of differentially private one-pass HDC training ($\delta = 10^{-5}$, $D = 4000$).



(a) FACE detection (b) UCIHAR activity recognition (c) PAMAP2 physical activity recognition

Fig. 15: Accuracy of differentially private iterative HDC training ($\delta = 10^{-5}$, $D = 4{,}000$).

learning capability of HDC from limited data. Specifically, in FACE and UCIHAR, one-pass learning reaches close-to-optimal accuracy by learning from only 2% of the data.

The private one-pass training on FACE and UCIHAR benchmarks yields the same accuracy of non-private one-pass with $\epsilon = 1$ which is a tight privacy constraint. We remind the readers that, as shown in Equation (6), ϵ indicates the likelihood of privacy leakage. A smaller value of ϵ corresponds to a stronger the privacy guarantee, requiring a greater magnitude of Gaussian noise (indicated by a larger noise parameter σ) to attain this level of privacy preservation, in accordance with the principles of differential privacy theory [6]. In Fig. 14, FACE can obtain the non-private one-pass training accuracy with $\epsilon = \frac{1}{4}$ as well, though requiring more training data, while UCIHAR could not converge with $\epsilon = \frac{1}{2}$. This can be attributed to the fact that FACE has more data and fewer labels (two labels versus five as in UCIHAR) which makes it more resilient to noise. The magnitude of noise for one-pass learning depends on the sensitivity, $\|\vec{\mathcal{H}}\|_2$, which is independent of the dataset size. Since the size of the dataset affects the scale of class vectors, with more vector accumulation in datasets with more samples, the $\vec{\mathcal{C}}_i \cdot \vec{\mathcal{H}}$ score in Equation (8) grows versus the noise term $\vec{\mathcal{N}} \cdot \vec{\mathcal{H}}$ in larger datasets. As a result, the other two benchmarks PAMAP2 and ISOLET, each featuring 12 and 26 classes, yet possessing a smaller training dataset, could converge to their maximum accuracy at larger $\epsilon = 3$ (less noise).

In Section V-B, we propose to normalize and clip the vectors to reduce the HDC sensitivity, and consequently, the amount of noise. The "*w/o clipping*" curves of Fig. 14 display the accuracy of private learning without normalization and clipping. Accordingly, only a lower level of privacy ($\epsilon = 5$ or $\epsilon = 10$) could be attained, accompanied by a 32.7% decrease in accuracy compared to the results obtained from one-pass private training, which benefits from the normalization and clipping approaches.

F. Iterative Training Privacy

Fig. 15 compares the accuracy of the benchmarks trained with our new differentially private iterative HDC training procedure in Privé-HDnn. With $\epsilon = 1$ ($\epsilon = 4$), private FACE benchmark achieves 91.1% (92.2%) accuracy, which is 2.5% (3.6%) higher than one-pass training. Such outcome verifies that our iterative training design effectively unleashes the power of iterative training without violating the privacy bounds. Likewise, private UCIHAR attains a 1.8% (3.5%) advantage in accuracy over the one-pass approach. It is worth highlighting that, at $\epsilon = 4$, the differentially private UCIHAR exhibits a mere 1% decline in accuracy compared to its non-private counterpart. On the other hand, PAMAP2 presents a distinct scenario with its greater number of classes and reduced training data. As a result, even with $\epsilon = 8$ (smaller noise), it achieves an accuracy of 82%, exhibiting only marginal improvement over the one-pass private training scenario. This outcome mirrors the same observation made in the one-pass training – the disparity between similarity score ($\vec{\mathcal{C}}_i \cdot \vec{\mathcal{H}}$) and noise ($\vec{\mathcal{N}} \cdot \vec{\mathcal{H}}$) is enlarged by the limited training set of PAMAP2. Notably, the greater number of classes within PAMAP2 exacerbates the impact of noise, as the increased class count contributes to a reduction in the margin between the class vectors. For a quick verification, we first train a one-pass private model with $\epsilon = 2$ on PAMAP2 for a faster initial convergence, followed by an iterative training process with $\epsilon = 6$. The resultant model ($\epsilon = 2 + 6$ in Fig. 15) obtains 3% higher accuracy compared to the $\epsilon = 8$ iterative training from scratch.

To fulfill Equation (10) for a given target (ϵ, δ), various combinations of σ, q, and T can be employed. Opting for a smaller noise parameter σ can yield enhanced accuracy within fewer training epochs. Therefore, we set $T = 16 \times 10^4$ and $q = \frac{0.1}{\sqrt{T}}$, a configuration that results in a unique and small σ_{min} to satisfy the (ϵ, δ) requirements. Also, as training progresses, the total number of epochs ($E = q \times T$) signifies the point at which the privacy objective ϵ is attained. The ϵ factor curve in Fig. 15 (the secondary Y-axis) indicates the ϵ value at each epoch. Notably, privacy becomes tighter before reaching the final epoch, albeit at the cost of lower accuracy. For instance, in Fig. 15 (a), at the 10^{th} epoch, the ϵ factor is approximately 0.5, implying that the green curve setting (ultimately achieving $\epsilon = 4$) could have achieved a preferable ϵ of 2 if the training had terminated at that point.

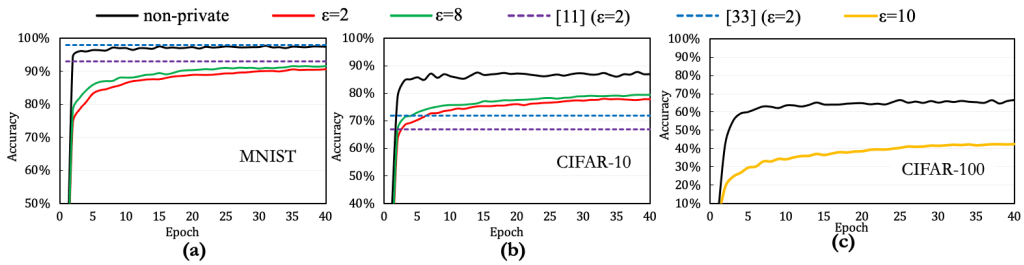


Fig. 16: Private training on images with Privé-HDnn.

TABLE V: Training time (minutes) of Privé-HDnn versus previous studies.

Dataset	**Privé-HDnn GPU**	**Privé-HDnn CPU**	[6] (low)	[6] (high)	[46] (low)	[46] (high)
MNIST	1.1 (1.0×)	15.73 (14.3×)	12 (9.2×)	533 (410×)	28 (22×)	160 (123×)
CIFAR-10	1.2 (1.0×)	10.43 (8.69×)	40 (31×)	467 (359×)	280 (215×)	1600 (1231×)
CIFAR-100	4.3 (1.0×)	34.0 (7.92×)	-	-	-	-

G. Case Study: Image Classification

(1) Accuracy: Figure 16 illustrates the accuracy outcomes of both the baseline non-private HDnn and Privé-HDnn models when evaluated on the MNIST, CIFAR-10 and CIFAR-100 datasets. The non-private HDnn model attains accuracy rates of 97.8% for MNIST and 87.0% for CIFAR-10. It is important to mention that the baseline HDnn model employed in this context utilizes a ResNet-18 (for MNIST and CIFAR-10) and ResNet-50 (for CIFAR-100) network that was pretrained on the ILSVRC 2012 dataset as its feature extractor. However, it was not fine-tuned specifically for the target MNIST and CIFAR datasets. Consequently, this approach results in an accuracy decrease when compared to a fine-tuned HDnn model (which otherwise achieves accuracy comparable to or exceeding that of DNN [43]) or a baseline ResNet-18 model. The non-private training achieves 99%, 90% and 66% of accuracy on MNIST, CIFAR-10 and CIFAR-100, after 40 epochs.

With $\epsilon = 2$, Privé-HDnn achieves 90.6% on MNIST. In previous works [46], [62], the accuracy numbers for for differentially-private MNIST DNNs vary, ranging from 90% [62] to 98.1% [46] (with $\epsilon = 2.93$). Notably, the latter work constructs new models from scratch, incorporating minimal parameters to enhance the DNN's tolerance to noise. Regarding CIFAR-10, the investigation outlined in [46] records a peak accuracy of 72% with $\epsilon = 2$, surpassing the pioneering study by Abadi *et al.* [6] by 5%. In contrast, Privé-HDnn achieves a 77.8% accuracy on the same CIFAR-10 dataset, a 5.8% improvement over [46] with the same $\epsilon = 2$. This showcases that the accuracy of Privé-HDnn is either on par or exceeds the performance of prior deep learning investigations incorporating differential privacy. Since previous studies [46], [62] did not include results for CIFAR-100. We exclusively present Privé-HDnn's outcomes, demonstrating its ability to ensure privacy with $\epsilon = 10$, albeit at a 40% accuracy trade-off after 40 epochs. The accuracy of Privé-HDnn can be further improved by adopting larger CNN backbones, such as ResNet-50, or by leveraging more relevant public datasets for pretraining the feature extractor, like a more pertinent subset of ILSVRC 2012.

(2) Execution time: Unlike studies like [46] that built a custom model, Privé-HDnn leverages existing models for feature extraction and only trains HDC on the extracted features. Due to the fast convergence of HDC and its simplicity, Privé-HDnn requires only 40 training epochs (Fig. 16). Table V compares the training time of Privé-HDnn with DNN training in [6] and [46] on MNIST, CIFAR-10. The training time of Privé-HDnn on CIFAR-100 is also reported as a reference, which is slower than CIFAR-10 due to our use of ResNet-50 instead of ResNet-18 as the feature extractor. The terms *low* and *high* denote the minimal and maximal configurations of prior works, established according to their respective privacy goals and hyperparameters. For instance, in the *high* configuration of a previous study [46], which is tailored to the CIFAR-10 dataset, 400 epochs were utilized with a batch size of 256, each epoch taking 240 seconds. The GPU implementation of Privé-HDnn demonstrates remarkable runtime improvement, achieving speeds that are 9.2–410× faster than [6], and 22–1231× faster than [46]. Even the CPU-based training of Privé-HDnn outperforms previous DNN methods designed for GPU acceleration. The CPU training time is reduced by a factor of 2.1–92× compared to [6], and 4.8–276× compared to [46]. This acceleration is underpinned by HDC's computational efficiency. Privé-HDnn effectively addresses privacy concerns inherent to HDC, establishing itself as an ideal framework for efficient and privacy-aware learning.

H. Overhead Analysis

(1) Computational overhead: The proposed privacy-enhancing techniques impose minimal computational overhead. The accuracy drop of the local sparsification is only 0.14% for 87% sparsification rate. This rate of sparsification lowers the energy cost of communicating data by 63% and improves performance by 2.6× (38×) over non-sparse FPGA (CPU). The noise is applied on top of the model, so the model size and training time do not change. In particular, for images, Privé-HDnn leverages the readily available pre-trained CNNs (unlike previous studies that train from scratch). Consequently, the training is very fast as only the HDC part is updated, taking only 1.3 minutes as compared to 26 hours for the CNN, with better accuracy. The computational time for noise injection is less than 30% of the differential-private iterative HDC pipeline in Algorithm 1. This HDC pipeline does not include the neural network feature extraction part in Privé-HDnn, thus is extremely lightweight. When including the extraction of HDnn features, the overhead of noise injection is less than 1% on Privé-HDnn (GPU), and would be even smaller on Privé-HDnn (CPU).

(2) Memory overhead: Privé-HDnn largely reduces the memory costs compared to DNN baselines due to the nature of HDC. In Privé-HDnn, the shallow CNN feature extractor remains frozen, necessitating training solely for the HDC classifier. Consequently, Privé-HDnn boasts minimal memory consumption in comparison to DNN training, where gradients and intermediate activations must be stored. For instance, in the CIFAR-100 case study utilizing ResNet-50 as a pretrained feature extractor, Privé-HDnn requires 25.5MB, 2.56MB, and 1MB to store all the parameters in ResNet-50, the random projection matrix, and the trained class vectors, respectively. In contrast, training the entire ResNet-50 model may demand over 10GB of memory [63]. Privé-HDnn thus achieves a memory cost reduction exceeding 300x.

VII. Conclusion

In this paper, we leverage the noise robustness of HDC to address the inherent privacy challenges of HDC for efficient privacy-preserved training and inference. We first thoroughly study the feature extraction challenge that can infer the important encoding parameters (e.g., base vectors) and reconstruct the original data in HDC. To improve inference privacy, we propose local sparsification with efficient FPGA implementation, which deviates the decoded values by a normalized RMSE of 0.42 versus the original values, and improves the encoding performance by up to 5.3× over non-sparse FPGA implementation, and 79× over CPU implementation. For training, we propose Privé-HDnn with differential private one-pass and iterative training on complex applications such as image classification. Privé-HDnn achieves comparable or better accuracy by up to 5.8% versus DNN-only solutions, while reducing the training time on GPU 9.2–1231× over the state of the art [6], [46].

Acknowledgements

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

References

[1] A. Gupta and P. Nahar, "Classification and yield prediction in smart agriculture system using iot," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–10, 2022.

[2] P. Boobalan *et al.*, "Fusion of federated learning and industrial internet of things: A survey," *Computer Networks*, vol. 212, p. 109048, 2022.

[3] E. Wang *et al.*, "Deep neural network approximation for custom hardware: Where we've been, where we're going," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–39, 2019.

[4] I. Gim and J. Ko, "Memory-efficient dnn training on mobile devices," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pp. 464–476, 2022.

[5] Y. Mao, W. Hong, B. Zhu, Z. Zhu, Y. Zhang, and S. Zhong, "Secure deep neural network models publishing against membership inference attacks via training task parallelism," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 3079–3091, 2021.

[6] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 308–318, 2016.

[7] F. Mireshghallah *et al.*, "Privacy in deep learning: A survey," *arXiv preprint arXiv:2004.12254*, 2020.

[8] S. Datta, R. A. Antonio, A. R. Ison, and J. M. Rabaey, "A programmable hyper-dimensional processor architecture for human-centric iot," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 439–452, 2019.

[9] B. Khaleghi, J. Kang, H. Xu, J. Morris, and T. Rosing, "Generic: highly efficient learning engine on edge using hyperdimensional computing," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1117–1122, 2022.

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

[11] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *2017 IEEE international conference on rebooting computing (ICRC)*, pp. 1–8, IEEE, 2017.

[12] Y. Kim, M. Imani, and T. S. Rosing, "Efficient human activity recognition using hyperdimensional computing," in *Proceedings of the 8th International Conference on the Internet of Things*, pp. 1–6, 2018.

[13] Y. Yao, W. Liu, G. Zhang, and W. Hu, "Radar-based human activity recognition using hyperdimensional computing," *IEEE Transactions on Microwave Theory and Techniques*, vol. 70, no. 3, pp. 1605–1619, 2021.

[14] A. Moin *et al.*, "A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition," *Nature Electronics*, vol. 4, no. 1, pp. 54–63, 2021.

[15] W. Chen and H. Li, "Adversarial attacks on voice recognition based on hyper dimensional computing," *Journal of Signal Processing Systems*, vol. 93, no. 7, pp. 709–718, 2021.

[16] W. Xu, J. Kang, and T. Rosing, "Fsl-hd: Accelerating few-shot learning on reram using hyperdimensional computing," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2023.

[17] B. Khaleghi, M. Imani, and T. Rosing, "Prive-hd: Privacy-preserved hyperdimensional computing," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2020.

[18] A. Hernández-Cano, R. Cammarota, and M. Imani, "Prid: Model inversion privacy attacks in hyperdimensional learning systems," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 553–558, IEEE, 2021.

[19] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 435–446, IEEE, 2019.

[20] R. Chandrasekaran, K. Ergun, J. Lee, D. Nanjunda, J. Kang, and T. Rosing, "Fhdnn: Communication efficient and robust federated learning for aiot networks," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 37–42, 2022.

[21] Q. Zhao, K. Lee, J. Liu, M. Huzaifa, X. Yu, and T. Rosing, "Fedhd: federated learning with hyperdimensional computing," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pp. 791–793, 2022.

[22] S. Zhang, D. Ma, S. Bian, L. Yang, and X. Jiao, "On hyperdimensional computing-based federated learning: A case study," in *2023 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2023.

[23] S. Zhang, R. Wang, J. J. Zhang, A. Rahimi, and X. Jiao, "Assessing robustness of hyperdimensional computing against errors in associative memory," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 211–217, IEEE, 2021.

[24] J. Morris, K. Ergun, B. Khaleghi, M. Imani, B. Aksanli, and T. Simunic, "Hydrea: Utilizing hyperdimensional computing for a more robust and efficient machine learning system," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 6, pp. 1–25, 2022.

[25] J. Park, C. Quan, H. Moon, and J. Lee, "Hyperdimensional computing as a rescue for efficient privacy-preserving machine learning-as-a-service," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–8, IEEE, 2023.

[26] Y. Nam, M. Zhou, S. Gupta, G. De Micheli, R. Cammarota, C. Wilkerson, D. Micciancio, and T. Rosing, "Efficient machine learning on encrypted data using hyperdimensional computing," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, IEEE, 2023.

[27] X. Yu, M. Zhou, F. Asgarinejad, O. Gungor, B. Aksanli, and T. Rosing, "Lightning talk: Private and secure edge ai with hyperdimensional computing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–2, IEEE, 2023.

[28] R. Wang, W. Wen, K. Juretus, and X. Jiao, "Pp-hdc: A privacy-preserving inference framework for hyperdimensional computing," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2024.

[29] F. Yang and S. Ren, "Adversarial attacks on brain-inspired hyperdimensional computing-based classifiers," *arXiv preprint arXiv:2006.05594*, 2020.

[30] D. Ma, J. Guo, Y. Jiang, and X. Jiao, "Hdtest: Differential fuzz testing of brain-inspired hyperdimensional computing," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 391–396, IEEE, 2021.

[31] R. Wang and X. Jiao, "Poisonhd: poison attack on brain-inspired hyperdimensional computing," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 298–303, IEEE, 2022.

[32] H. Kasyap and S. Tripathy, "Beyond data poisoning in federated learning," *Expert Systems with Applications*, vol. 235, p. 121192, 2024.

[33] F. Liu, H. Li, Y. Chen, T. Yang, and L. Jiang, "Hyperattack: An efficient attack framework for hyperdimensional computing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.

[34] D. Ma, S. Zhang, and X. Jiao, "Robust hyperdimensional computing against cyber attacks and hardware errors: A survey," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pp. 598–605, 2023.

[35] P. Courrieu, "Fast computation of moore-penrose inverse matrices," *arXiv preprint arXiv:0804.4809*, 2008.

[36] A. Menon *et al.*, "Efficient emotion recognition using hyperdimensional computing with combinatorial channel encoding and cellular automata," *Brain informatics*, 2022.

[37] L. Lyu, H. Yu, J. Zhao, and Q. Yang, "Threats to federated learning," *Federated Learning: Privacy and Incentive*, pp. 3–16, 2020.

[38] Y. Shen *et al.*, "Algorithmic insights on continual learning from fruit flies," *arXiv preprint arXiv:2107.07617*, 2021.

[39] S. Salamat, A. Haj Aboutalebi, B. Khaleghi, J. H. Lee, Y. S. Ki, and T. Rosing, "Nascent: Near-storage acceleration of database sort on smartssd," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 262–272, 2021.

[40] M. Imani, S. Bosch, S. Datta, S. Ramakrishna, S. Salamat, J. M. Rabaey, and T. Rosing, "Quanthd: A quantization framework for hyperdimensional computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2268–2278, 2019.

[41] B. Khaleghi, S. Salamat, A. Thomas, F. Asgarinejad, Y. Kim, and T. Rosing, "Shear er: highly-efficient hyperdimensional computing by software-hardware enabled multifold approximation," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 241–246, 2020.

[42] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[43] A. Dutta, S. Gupta, B. Khaleghi, R. Chandrasekaran, W. Xu, and T. Rosing, "Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, pp. 281–286, 2022.

[44] C. Dwork, G. N. Rothblum, and S. Vadhan, "Boosting and differential privacy," in *2010 IEEE 51st annual symposium on foundations of computer science*, pp. 51–60, IEEE, 2010.

[45] S. P. Kasiviswanathan *et al.*, "What can we learn privately?," *SIAM Journal on Computing*, 2011.

[46] N. Papernot *et al.*, "Making the shoe fit: Architectures, initializations, and tuning for learning with privacy," 2019.

[47] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[48] D. Anguita *et al.*, "A public domain dataset for human activity recognition using smartphones," in *ESANN'13*.

[49] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *2012 16th international symposium on wearable computers*, pp. 108–109, IEEE, 2012.

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

[51] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE signal processing magazine*, 2012.

[52] "The cifar dataset." https://www.cs.toronto.edu/~kriz/cifar.html.

[53] "Xilinx power estimator user guide ug440." User Guide, April 2022.

[54] "CPU energy meter," 2020.

[55] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 53–62, 2019.

[56] J. Kang *et al.*, "Openhd: A gpu-powered framework for hyperdimensional computing," *IEEE Transactions on Computers*, 2022.

[57] P. Vergés *et al.*, "Hdcc: A hyperdimensional computing compiler for classification on embedded systems and high-performance computing," *arXiv preprint arXiv:2304.12398*, 2023.

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[59] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[60] M. Abo-Zahhad *et al.*, "An energy consumption model for wireless sensor networks," in *ICEAC'15*.

[61] T. F. Wu, H. Li, P.-C. Huang, A. Rahimi, G. Hills, B. Hodson, W. Hwang, J. M. Rabaey, H.-S. P. Wong, M. M. Shulaker, *et al.*, "Hyperdimensional computing exploiting carbon nanotube fets, resistive ram, and their monolithic 3d integration," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 11, pp. 3183–3196, 2018.

[62] C. Chen and J. Lee, "Stochastic adaptive line search for differentially private optimization," in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 1011–1020, IEEE, 2020.

[63] "Pytorch forums." https://discuss.pytorch.org/t/resnet-50-takes-10-13gb-to-run-with-batch-size-of-96/117402.