UC Riverside

UC Riverside Previously Published Works

Title

Probabilistic Unlearning-Based Defense for Robust and Efficient Edge Learning

Permalink

https://escholarship.org/uc/item/7b50q6h6

Journal

Electronics, 15(20)

ISSN

1450-5843

Authors

Emami, Ava

Zhao, Quanling

Yang, Jiaying

et al.

Publication Date

2026-10-21

DOI

10.3390/electronics15204879

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

electronics

MDPI

Article

Probabilistic Unlearning-Based Defense for Robust and Efficient Edge Learning

Ava Emami [1], Quanling Zhao [1], Jiaying Yang [1], Flavio Ponzina [2], Fatemeh Asgarinejad [3,*], Tajana Rosing [1,*] and Baris Aksanli [2,*]

[1] Department of Computer Science and Engineering, University of California San Diego, San Diego, CA 92093, USA; seemami@ucsd.edu (A.E.); quzhao@ucsd.edu (Q.Z.); jiy018@ucsd.edu (J.Y.)
[2] Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA 92182, USA; fponzina@sdsu.edu
[3] Department of Electrical and Computer Engineering, University of California Riverside, Riverside, CA 92521, USA
* Correspondence: fasga001@ucr.edu (F.A.); tajana@ucsd.edu (T.R.); baksanli@sdsu.edu (B.A.)

Abstract

Hyperdimensional computing (HDC), a brain-inspired paradigm, encodes information as high-dimensional vectors and forms associations through bundling. Yet, the ability to selectively forget information—critical for privacy, security, and adaptability—remains unexplored in HDC. We show that unlearning in HDC is difficult due to its memorization nature: vectors are directly stored in associative memory and persist, making it hard to forget specific data, while naive removal causes severe accuracy loss. We propose `ClearAttack`, a Computationally light and effective unlearning framework for HDC. `ClearAttack` tracks the influence of encoded vectors and offsets the impact of forgotten data with suitable substitutes, enabling in situ unlearning. To further mitigate accuracy loss, it incorporates selective retraining on sensitive samples. Experimental results show that `ClearAttack` unlearns data more effectively and efficiently than prior methods, achieving up to 3.6× higher forget score, accuracy within 0.2–0.6% of the oracle model, and up to 5.2× faster execution. Moreover, `ClearAttack` successfully eliminates the effect of poisoned data, demonstrating its utility for both accuracy and security. Because the complete unlearning procedure costs a fraction of a single retraining epoch and adds only 6.4% memory overhead, deletion requests and poisoning recovery can be handled directly on resource-constrained devices rather than requiring a full retraining pass.

Keywords: hyperdimensional computing; machine unlearning; adversarial robustness; data poisoning defense; edge intelligence; privacy-preserving learning; trojan attack mitigation

Check for updates

Academic Editor: Firstname Lastname

Received: 10 August 2026
Revised: 28 September 2026
Accepted: 14 October 2026
Published:

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Machine learning is increasingly used in security-critical edge applications (e.g., self-driving cars, malware detection, drones) [1], but conventional models struggle with resource constraints. Lightweight methods like hyperdimensional computing (HDC) address this with high-dimensional, low-precision, and error-resilient representations [2–5]. However, HDC remains vulnerable to adversarial perturbations [6,7].

Many machine learning applications rely on continuously collected data [8–10], where individual samples can propagate through model parameters and affect future predictions. Therefore, it is critical to remove perturbed or poisoned data to improve robustness, defend

against attacks, and comply with data deletion requests. Restoring the model to a state that is free of perturbed, poisoned, or unwanted samples must happen without retraining the entire model from scratch or losing critical information.

Traditional defenses, such as adversarial retraining, can mitigate attacks but are computationally expensive, require full dataset access, and often assume prior knowledge of attack strategies [11]. Furthermore, adversarial attacks can be difficult to detect. In addition, models may need to remove specific data and their propagated effects while maintaining model integrity and efficiency [12–14].

Unlearning is a lightweight, targeted approach to remove adversarial or unwanted samples without full retraining the entire model from scratch. It is essential for security, since adversaries can poison training to manipulate predictions or implant trojans that can perturb the model's performance at runtime [15]. It is equally critical for privacy, as legislation such as the General Data Protection Regulation (GDPR) [16] grants the "right to be forgotten" [17]. Unlearning in HDC is particularly advantageous because the model stores information directly in high-dimensional vectors. Compared to retraining, unlearning improves efficiency as it allows the selective removal of perturbed or unwanted data while preserving the integrity of previously learned data with low computing overhead.

Once a sample is identified for removal, unlearning is challenging because its influence is embedded in model parameters and propagates through training. Removing many samples can lead to catastrophic forgetting [13], while fine-tuning introduces an efficiency–accuracy trade-off. In HDC, data are encoded as high dimensional vectors and directly aggregated into associative memory [2,4,18]. This enables precise sample removal without recomputing unrelated information, offering efficient privacy and adversarial defense benefits.

In this work, we propose the first study of unlearning in HDC and an efficient unlearning framework for HDC classifiers. We also propose a method to identify perturbed and poisoned data. We demonstrate unlearning as a defense against adversarial attacks and data poisoning. While unlearning has been explored for neural networks and other ML models [12–14], these approaches are not well-suited for HDC due to its unique training process. Unlike neural networks, where sample information is distributed through gradient-based updates, HDC *directly*bundles data into high-dimensional vectors. Consequently, a sample's influence can persist when training continues without it, and this direct association increases vulnerability to membership-inference and decoding attacks [19,20]. As further analyzed in Section 2.4, pruning, perturbation, and retain-set retraining may not reliably eliminate this influence. Thus, removing sensitive or adversarial data requires dedicated unlearning mechanisms. Although unlearning in HDC may appear straightforward by removing encoded vectors, naive removal can severely degrade performance. To address this, we propose `ClearAttack` as an efficient, low-cost unlearning framework. `ClearAttack` operates in both proactive and reactive modes to identify and remove adversarial or perturbed samples.

(i) We show that information from past data can persist in HDC even after its exclusion, while direct deletion from associative memory can cause severe performance degradation. We further demonstrate that this memorization makes HDC vulnerable to adversarial and backdoor Trojan attacks.
(ii) Framework-level contributions: We propose `ClearAttack`, an end-to-end unlearning framework that detects perturbed samples, tracks and exactly removes their contributions, substitutes suitable retain samples to mitigate accuracy loss, and selectively retrains on sensitive low-margin samples.
(iii) Mechanism-level contributions: We design an optimized updates table that reduces memory overhead from 65% to 6.4% by storing only effective sample contributions.

We also introduce a bounded substitute-search mechanism that limits the search to a small set of same-label retain vectors, substantially reducing computational cost while maintaining unlearning quality.

(iv) We evaluate `ClearAttack` in terms of unlearning quality, accuracy, efficiency, and robustness. It achieves up to 3.6× higher forget score, remains within 0.2–0.6% of oracle accuracy, and provides up to 5.2× speedup over retraining-based baselines. It also reduces Trojan attack success from 91–100% to below 3% at substantially lower cost.

2. Background and Motivation

2.1. Hyperdimensional Computing

HDC operates by encoding input samples to high-dimensional vectors and combining them to form an associative memory of class vectors. `ClearAttack` is architecturally compatible with encoding schemes that produce hypervectors aggregated into an associative memory, although we evaluate only random projection encoding. In this work, we use random projection (RP) encoding due to its simplicity and overall performance [5]. RP ties an ith feature in a data sample $\mathcal{F} = \langle f_1, \cdots, f_d \rangle$ with a $D \approx 10k$ dimensional binary base vector to generate the encoded vector $\vec{H}$ by $\vec{H} = \sum_{i=1}^{d} f_i \vec{B}^i$.

In single-pass HDC training, we create the class vector $\vec{C}^\ell$ for label ℓ by accumulating the related encoded vectors as $\vec{C}^\ell = \sum_{j: y_j = \ell} \vec{H}^j$. For higher accuracy, we adopt iterative training from [21], which adaptively adjusts update rates to the associative memory. In each iteration, if an encoded sample $\vec{H}$ with true label **t** is misclassified as **f**, the class vectors are updated as $\vec{C}^t = \vec{C}^t + (1 - s_t)\vec{H}$ and $\vec{C}^f = \vec{C}^f - (1 - s_f)\vec{H}$, where $s_{t/f} = \mathbf{\Phi}(\vec{H}, \vec{C}^{t/f})$ is the cosine similarity. For inference (also used during training), the predicted label is $\ell^* = \arg\max_{k=1,\ldots,n_c} \mathbf{\Phi}(\vec{H}, \vec{C}^k)$.

2.2. Adversarial Attacks

2.2.1. Threat Model

Attacker's knowledge: We consider both white-box and gray-box models. In the former, the attacker has full access to the HDC model, including input data, base vectors, and trained class vectors. In the gray-box model, the attacker can query the model and adapt based on observed outputs, following a threat model similar to [22].

2.2.2. Defense Model

Defender's Goal: The defender aims to mitigate both targeted and untargeted attacks. For targeted attacks, the goal is to prevent forced outputs; for untargeted attacks, it is to maintain accuracy and prevent misclassification.

Applicable Modes: In proactive mode, the detector screens incoming samples before they update an online HDC model. Rejected samples are not bundled into the associative memory and therefore require no unlearning. In reactive mode, corrupted or unwanted samples have already influenced the class vectors. Detection or an external deletion request identify these samples, after which `ClearAttack` removes their recorded contributions. If the forget set is already known, the detection stage is skipped.

Recent studies in other application domains provide complementary perspectives on efficient and robust learning. The multi-scale feature-fusion approach in [23] preserves discriminative information while suppressing noisy or redundant features, conceptually aligning with `ClearAttack`'s identification of anomalous samples and selective retraining of low-margin samples. Similarly, the joint security optimization in [24] balances robustness and resource cost under dynamic uncertainty, supporting `ClearAttack`'s objective of lightweight defense on resource-constrained devices. These studies address different tasks but share the principles of selective information processing and security-efficiency optimization.

2.3. Machine Unlearning

An unlearning algorithm $\mathcal{U}$ takes as input $\mathcal{M}(\mathcal{D})$ (a model trained on dataset $\mathcal{D}$ with algorithm $\mathcal{M}$), a forget set $\mathcal{F} \subseteq \mathcal{D}$, and optionally $\mathcal{D}$, and outputs a model with $\mathcal{F}$ removed. The forget set $\mathcal{F}$ is not used for training; it only specifies which samples to remove and is later used to evaluate forgetting quality. The quality of forgetting is evaluated against $\mathcal{M}(\mathcal{D}\backslash\mathcal{F})$, i.e., retraining on the retain set.

Inspired by differential privacy, we adopt the following definition of unlearning [13,25]. Given dataset $\mathcal{D}$, forget set $\mathcal{F} \subseteq \mathcal{D}$ and algorithm $\mathcal{M}$ (e.g., HDC), $\mathcal{U}$ achieves (ϵ, δ)-unlearning if, for all output regions, $R \subseteq \mathcal{R}$ (parameter space),

$$\begin{aligned} Pr[\mathcal{M}(\mathcal{D}\backslash\mathcal{F}) \in R] &\leq e^{\epsilon} Pr[\mathcal{U}(\mathcal{M}(\mathcal{D}), \mathcal{F}, \mathcal{D}) \in R] + \delta \\ Pr[\mathcal{U}(\mathcal{M}(\mathcal{D}), \mathcal{F}, \mathcal{D}) \in R] &\leq e^{\epsilon} Pr[\mathcal{M}(\mathcal{D}\backslash\mathcal{F}) \in R] + \delta \end{aligned} \tag{1}$$

This requires the distributions (Pr) of the oracle model $\mathcal{M}(\mathcal{D}\backslash\mathcal{F})$ and the unlearned model $\mathcal{U}(\mathcal{M}(\mathcal{D}), \mathcal{F}, \mathcal{D})$ to be nearly indistinguishable.

Comparisons of distributions can be done in weight or output space but typically require multiple retrainings. To make this tractable, we follow [25] and compare the outputs of the oracle and unlearned models on the forget set as unlearning is more effective when these outputs are indistinguishable from one another. For a forget sample $s \in \mathcal{F}$, let $\mathcal{R}^s$ and $\mathcal{U}^s$ be the distributions of *logits* from the oracle and unlearned models, each run N times with different initialization: $\mathcal{R}^s = f(R_1(s)), \ldots, f(R_N(s))$ and $\mathcal{U}^s = f(\mathcal{U}_1(s)), \ldots, f(\mathcal{U}_N(s))$, where f computes logit-scaled confidences. Statistical or classification rules (e.g., logistic regression) are then applied to decide if a sample $x \in \mathcal{R}^s \cup \mathcal{U}^s$ comes from $\mathcal{R}^s$ or $\mathcal{U}^s$. The best rule in distinguishing the two distributions is then used to calculate ϵ^s for sample s:

$$\epsilon^s = \max\{\log \frac{1-\delta-\text{FPR}}{\text{FNR}}, \log \frac{1-\delta-\text{FNR}}{\text{FPR}}\} \tag{2}$$

FPR and FNR are the false positive and false negative rates. FPR and FNR are the error rates of the logistic-regression rule fitted to distinguish the two distributions, and δ follows the convention of [26], with the value now stated. Higher FPR/FNR indicate greater indistinguishability and lower ϵ^s. Note that ϵ is exponential; for example, $\epsilon = 1.5$ is $e^{3.2-1.5} = 5.4\times$ more effective than $\epsilon = 3.2$. The overall forget score is computed by aggregating ϵ across all samples (see Section 4).

2.4. Why HDC Unlearning Is Non-Trivial?

Mathematical basis of catastrophic forgetting: As described in Section 2.1, each sample is encoded as $\vec{H}^i = \sum_{k=1}^{d} f_k^i \vec{B}^k$, and HDC training accumulates these sample hypervectors into each class vector:

$$\vec{C}^c = \sum_{i=1}^{N} \alpha_{i,c} \vec{H}^i = \sum_{i=1}^{N} \alpha_{i,c} \sum_{k=1}^{d} f_k^i \vec{B}^k, \tag{3}$$

where $\alpha_{i,c}$ is the training-dependent scalar contribution of sample i to class c. In single-pass training, $\alpha_{i,c} = 1$ for the true class and zero otherwise, whereas iterative training produces signed, order-dependent contributions. Removing a forget set $\mathcal{F}$ therefore changes the class vector by

$$\Delta\vec{C}^c = \sum_{i \in \mathcal{F}} \alpha_{i,c} \vec{H}^i. \tag{4}$$

For a query $\vec{q}$, this removal changes both the numerator and normalization of the cosine similarity $\Phi(\vec{q}, \vec{C}^c)$. Consequently, even a small forget set can substantially shift decision margins when its hypervectors align with a query or class direction. This additive structure explains both why sample contributions can be removed exactly and why uncompensated removal can cause severe accuracy loss. Due to the associative nature of HDC, one might

nevertheless expect HDC to be more unlearning-friendly than gradient-based models. To isolate this effect, Figure 2.4 is constructed without using `ClearAttack`. We track sample impact under single-pass and iterative training, then remove random clean samples one by one to evaluate performance (Figure 2.4a,b; see Section 3 for details).

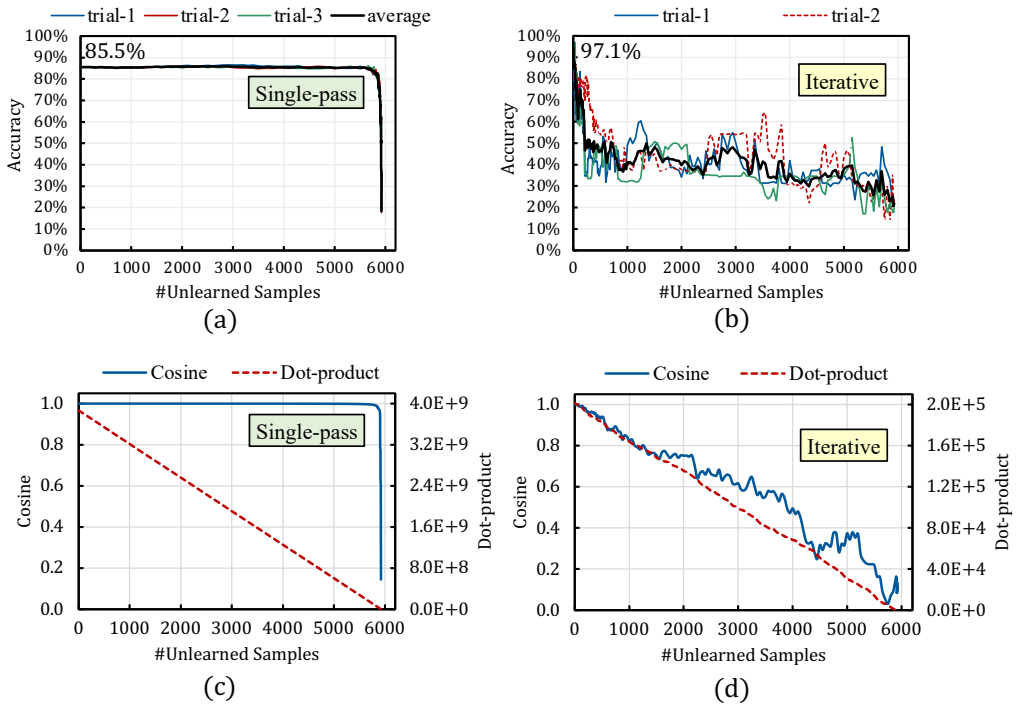


Figure 1. (**a**,**b**) show the test accuracy of unlearning an HDC model (on UCIHAR dataset [21]) trained as (**a**) single-pass (simple bundling) and (**b**) iterative (multiple epochs) for three different trials (i.e., random initializations and data order). The charts are created by tracking the impact of each train sample on the associative memory during training, then removing the samples' impact one by one, and performing a test after each removal. (**c**,**d**) show the cosine similarity and dot-product of the classes (after each data removal) against the original classes.

Catastrophic forgetting: Due to the associative nature of HDC, one might expect HDC to be more unlearning-friendly than gradient-based models. To isolate this effect, Figure 2.4 is constructed *without* using `ClearAttack`. We track sample impact under single-pass and iterative training, then remove random clean samples one by one to evaluate performance (Figure 2.4a,b; see Section 3 for details).

In single-pass training (Figure 2.4a), unlearning up to 90% of samples has minimal impact on accuracy (<0.2%). Performance quickly saturates, and cosine similarity between original and unlearned class vectors remains above 0.99 even after removing ∼98% of samples (Figure 2.4c), preserving query similarity. However, single-pass yields lower accuracy than iterative training (e.g., 11.4% drop on UCIHAR [21]), since bundling forms centroids but weak decision boundaries. Iterative retraining, akin to the perceptron [26], increases class margins and improves classification.

This motivates iterative training for higher accuracy. However, it is stochastic: sample order and updates strongly affect learning. As shown in Figure 2.4b, removing just a few samples can cause large degradation (e.g., 9% loss from 10 samples). Figure 2.4d explains this, as cosine similarity drops rapidly. Unlearning is thus unstable and order-sensitive, as seen from trial variations.

Persistent impact: While pruning [27] and weight perturbation [**?**] are effective in deep learning, Figure 2 shows their limitations for HDC. Using UCIHAR [29], we trained

with both retain (90%) and forget (10%) subsets for 30 epochs, then sparsified 95% of class vectors and continued training with only the retain data. A perfectly unlearned model would treat forget and test data similarly in accuracy and loss [13,25]. Instead, the model retained better performance on the forgotten data.

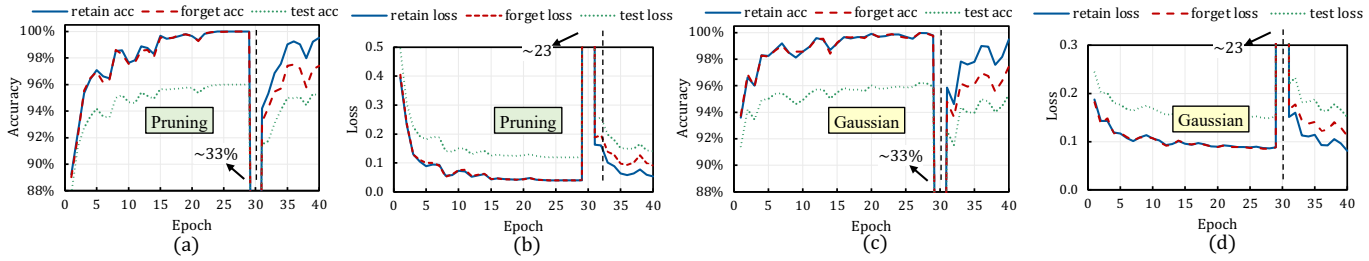


Figure 2. Accuracy and cross-entropy loss of the HDC model once it begins unlearning after 30 epochs of training. (**a**,**b**) correspond to pruning, while in (**c**,**d**), a Gaussian noise is applied to the associative memory. Both retain data and forget data (10% of total data) are used in the first 30 epochs. During unlearning (i.e., after the epoch 30th), the forget data are discarded from training. Note that right after unlearning, the accuracy drops to ∼33%.

A similar result occurs when Gaussian noise is added (Figure 2c,d). Even after perturbation and retraining, traces of the forgotten data remain detectable. These findings indicate that partial removal through pruning or noise is insufficient. Exact removal of forget vectors is required to mitigate their persistent influence while avoiding catastrophic performance collapse.

3. `ClearAttack` Framework

`ClearAttack` has four components: (A) probabilistic detection of perturbed versus clean inputs, (B) tracking and removing the forget set, (C) compensating for memory loss, and (D) selective fine-tuning after unlearning, all designed for efficiency. Component A is only used when corrupted data are unknown; it is skipped if such data are already identified. This compatibility follows from the iterative training rule in Equation (5): the updates table records each sample's scalar contribution to the class vectors, which Equation (6) removes regardless of how the hypervector was encoded. Systematic evaluation across encoding schemes and dimensionalities remains future work.

The remainder of this section follows the four components of `ClearAttack`. Section 3.1 presents probabilistic detection of perturbed samples. Section 3.2 describes how individual sample contributions are tracked during training. Section 3.3 explains exact contribution removal and substitute-based replacement, and, selective retraining on sensitive samples.

3.1. Probabilistic Classification of Adversarial Data

State of the art adversarial defenses often use compute-intensive neural-network-based methods like autoencoders [30,31]. For binary classification of perturbed versus clean [32] calculates average difference between the denoising autoencoder input and reconstructed output from the autoencoders. Such methods, when applied in the HDC framework, are only adapted to HDC and do not consider inherent characteristics of hyperdimensional computing.

`ClearAttack` uses a Chebyshev-based metric to compute a confidence score for each query vector q with respect to all classes. For each class c and feature dimension $j \in 1, \ldots, D$, we compute the normalized deviation as $z_{c,j} = \frac{|q_j - \mu_{c,j}|}{\sigma_{c,j}}$, where $q \in \mathbb{R}^D$ is the query hypervector, and $\mu_c, \sigma_c \in \mathbb{R}^D$ are the mean and standard deviation vectors estimated from training samples of class c. According to Chebyshev's inequality, large deviations within a

class are unlikely; hence, we define the weighted confidence score as $\text{conf}(q, c) = \sum_{j=1}^{D} \frac{1}{z_{c,j}^2 + \varepsilon}$, where $\varepsilon > 0$ is a small constant for stability. Higher confidence indicates that q is statistically consistent with class c.

A query vector is classified as *clean* if its maximum class confidence exceeds a threshold τ and as *perturbed* otherwise:

$$\hat{y}(q) = \begin{cases} \text{clean}, & \max_c \text{conf}(q, c) \geq \tau, \\ \text{perturbed}, & \text{otherwise.} \end{cases} \tag{5}$$

In our implementation, τ is derived empirically from the confidence-score distribution of clean validation samples. Figure 3 shows that clean and perturbed samples form separable confidence distributions, and we set τ according to this observed separation (shown as the red dashed line).

For perturbation experiments, we consider two types of noise:

Gaussian noise: Each input is corrupted by additive Gaussian noise with standard deviation σ. Figure 3a shows the confidence distributions for Gaussian-noise-corrupted Fashion-MNIST dataset. **Neural-network-based perturbation (PGD):** We also evaluate against adversarial inputs generated using projected gradient descent (PGD) [33]. PGD iteratively updates a clean input x to maximize the model's prediction error while constraining the perturbation within an ℓ_∞-norm ball:

$$x_{t+1} = \text{clip}_{x,\epsilon}\Big(x_t + \alpha \cdot \text{sign}\big(\nabla_x \mathcal{L}(f(x_t), y)\big)\Big), \tag{6}$$

where α is the step size, ϵ bounds the perturbation, and $\text{clip}_{x,\epsilon}(\cdot)$ keeps the adversarial example within the ϵ-ball around x. Multiple PGD steps increase attack strength. PGD is used as a stronger alternative to Gaussian noise to rigorously evaluate the robustness of `ClearAttack`. Figure 3b shows the confidence distribution for PGD-perturbed Fashion-MNIST [34].

Precision results: Table 1 summarizes the precision of detecting clean samples under both noises using Chebyshev-based classifier.

Table 1. Detection precision of clean samples under Gaussian and PGD perturbations.

Dataset	Gaussian Noise	PGD Perturbation
Fashion-MNIST [34]	0.9765	0.9979
ISOLET [35]	0.9421	0.8696

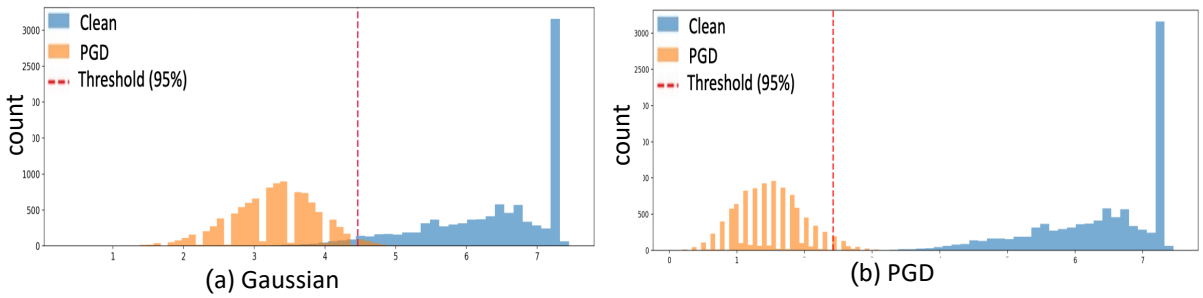


Figure 3. Chebyshev-based confidence distributions for clean versus perturbed samples. Panel (**a**) shows Gaussian noise ($\sigma = 0.05$) on Fashion-MNIST [34], while (**b**) shows PGD perturbation ($\epsilon = 0.2$, 20 steps). The vertical red dashed line indicates threshold τ, derived from clean samples, to separate clean and perturbed inputs. Well-separated distributions show `ClearAttack`'s effectiveness in detecting both Gaussian and NN-based perturbations.

As shown, our Chebyshev-based confidence metric reliably separates clean from perturbed samples. Gaussian noise leads to slightly lower precision than PGD on Fashion-MNIST, while ISOLET [35] is more sensitive to PGD attacks, as reflected by the reduced precision.

Figure 4 presents the `ClearAttack` baseline after identifying a data point as perturbed or aiming to remove unwanted data by the user. During training, an *updates table* records each sample's contribution to the class vectors, specifying which entries (indexes) are influenced and by what magnitude (*scores*). For unlearning, `ClearAttack` inputs the idx-th sample, retrieves its scores and indexes from the table, then searches the training data to find the most similar encoded vector (idx_max-th sample). This vector substitutes the removed sample in the associative memory, weighted by the original scores. The *updates table* is updated accordingly.

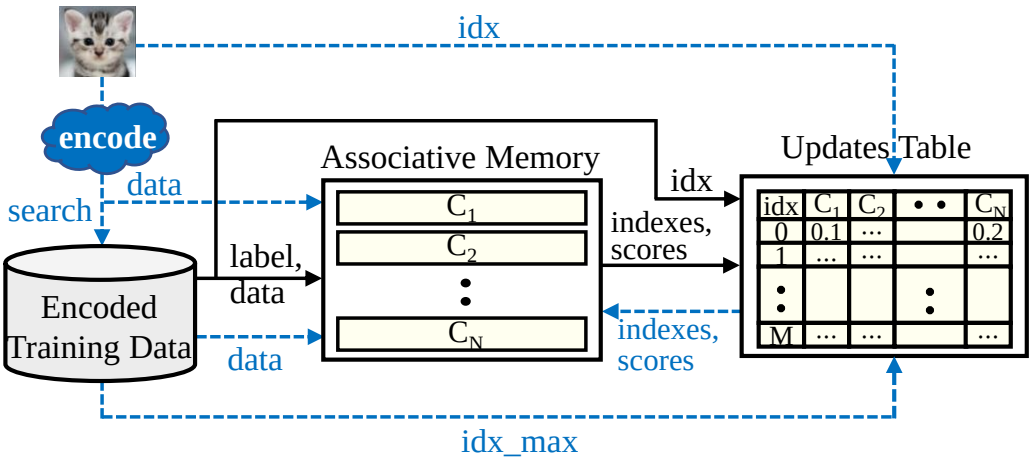


Figure 4. Overview of `ClearAttack`. During HDC training, the framework records each sample's contribution in an updates table. When a sample is selected for removal (e.g., perturbed or in a forget set), `ClearAttack` (B) retrieves its updates, (C) removes its contribution from class memories, (D) substitutes it with a similar retain sample to mitigate forgetting and performs selective retraining. Black arrows denote training flow; dashed arrows indicate unlearning.

We use per-dimension means and standard deviations to keep detection practical on resource-constrained devices; learned detectors would add substantial overhead. Although these statistics effectively separate Gaussian- and PGD-perturbed samples (Table 1 and Figure 3), robustness to higher-order distribution shifts remains a limitation and direction for future work.

3.2. Tracking Training Impact

We construct an *updates table* with N_T rows (the number of training samples) and N_C columns (the number of classes). Each entry *updates[i][j]* records the impact of the ith sample on class j during training. When processing the ith sample (according to Equation (7)), the updates are applied as

$$\begin{aligned} updates[i][t] &= updates[i][t] + (1 - s_t) \\ updates[i][f] &= updates[i][f] - (1 - s_f) \end{aligned} \tag{7}$$

where **t** and **f** denote the true and misclassified labels.

Across benchmarks, we find that only 24% of the rows in *updates* become non-zero since most samples are classified correctly during training, leaving their rows untouched. Furthermore, in 91% of the non-zero rows, only two columns are non-zero because a sample is typically misclassified only once, so only the **t** and **f** columns are updated.

To make the structure more hardware-friendly, we enforce exactly two non-zero values per non-zero row. Suppose a sample i was misclassified as $\mathbf{f}_0$ in earlier epochs (leading to updates of $\vec{C}^{\mathbf{t}}$ and $\vec{C}^{\mathbf{f}_0}$) but is later misclassified as $\mathbf{f}_1$. We suppress the update to $\vec{C}^{\mathbf{f}_1}$ and allow only $\vec{C}^{\mathbf{t}}$ to be updated. Therefore, the optimized *updates table* keeps keys only for training samples that impacted the classes. Each such row stores two pairs: the two affected classes ($\mathbf{t}$ and $\mathbf{f}_0$) and their corresponding update values ($\sum(1-s_t)$ and $\sum(1-s_f)$). Section 4 provides quantitative results on the memory saving of this optimized structure. Once the update map is constructed, removing the influence of the ith training sample from memory is straightforward:

$$\begin{aligned} row &= updates[i] \\ \vec{C}^{row[0]} &= \vec{C}^{row[0]} - row[1] \times \vec{H}^i \\ \vec{C}^{row[2]} &= \vec{C}^{row[2]} - row[3] \times \vec{H}^i \end{aligned} \tag{8}$$

Here, row[0] and row[2] represent the non-zero class indices ($\mathbf{t}$ and $\mathbf{f}$), while row[1] and row[3] contain the aggregated updates. To mitigate the memory loss and catastrophic effect of forgetting an ith sample, we find $\vec{H}^{max}$, the most similar encoded vector to $\vec{H}^i$ from the *retain* data use its factor to update $\vec{C}^{\mathbf{t}}$ and $\vec{C}^{\mathbf{f}}$. These operations are combined in lines 13–14 of Algorithm 1.

Replacing a sample with a substitute also requires adjustments to the *updates table*. This may cause the substitute to be linked with more than two classes if it had already influenced others, a case handled by the condition in line 10 of Algorithm 1. Updating the table also ensures that if the substitute later becomes part of a forget set, its effect on the model will be correctly tracked.

Figure 5 shows `ClearAttack`'s impact when unlearning is applied at the 30th epoch. After unlearning, the accuracy and loss curves of the forget and test sets overlap, showing that the forget data are effectively treated as unseen. Unlike simple vector removal—which dropped accuracy to 33% (previously seen in Figure 2)—`ClearAttack` maintains 93.1% accuracy immediately after unlearning, with no retraining. With just one epoch of retraining on the *retain* data, accuracy further improves to 95.0%, only 1.1% below the original.

Algorithm 1 `ClearAttack` unlearning procedure.

Input: $\mathcal{R}, \mathcal{F}$: retain/forget indices; $\vec{C}, \vec{H}$: class memory, encoded vectors; *updates*: updates map; *y_train*: train labels
Output: $\vec{C}$: unlearned model

for $idx_f \in \mathcal{F}$ **do**
  if $idx_f \notin updates$ **then continue**
  $row \leftarrow updates[idx_f]$
  $score_{max} \leftarrow -\infty$
  for $idx_r \in \mathcal{R}$ **do**
    if $y_train[idx_r] \neq y_train[idx_f]$ **then continue**
    $score \leftarrow \vec{H}^{idx_f} \cdot \vec{H}^{idx_r}$
    if $score > score_{max}$ **and**
      $(idx_r \notin updates$ **or** $updates[idx_r][0,2] = row[0,2])$ **then**
      $score_{max} \leftarrow score;\ idx_{max} \leftarrow idx_r$
    end if
  end for
  $\vec{C}^{row[0]} \leftarrow \vec{C}^{row[0]} + row[1]\,(\vec{H}^{idx_{max}} - \vec{H}^{idx_f})$
  $\vec{C}^{row[2]} \leftarrow \vec{C}^{row[2]} + row[3]\,(\vec{H}^{idx_{max}} - \vec{H}^{idx_f})$
  delete $updates[idx_f]$
  if $idx_{max} \notin updates$ **then** $updates[idx_{max}] \leftarrow row$
  else $updates[idx_{max}][1,3]$ += $row[1,3]$
  end if
end for
return $\vec{C}$

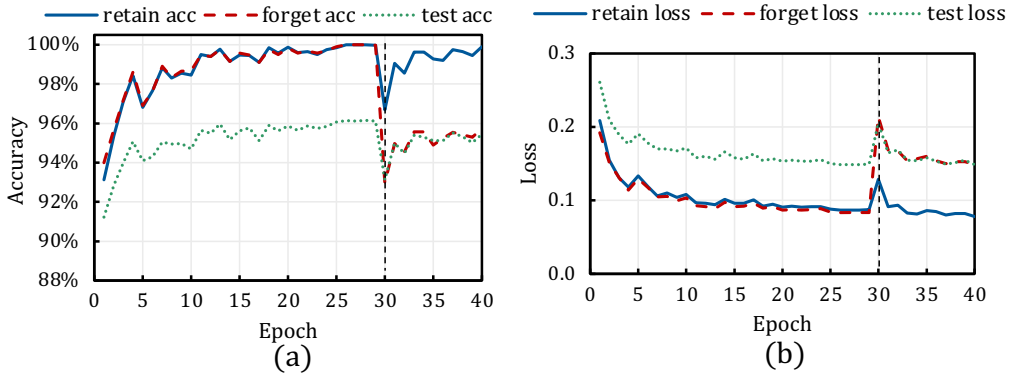


Figure 5. (**a**) Accuracy and (**b**) cross-entropy loss of the HDC model unlearned by `ClearAttack` on the 30th epoch. Forget data comprise 10% of total data.

Search complexity: For a given forget sample, the basic form of the proposed method requires exhaustively searching all candidate vectors of the same label to identify a suitable substitute. The expected number of searches is

$$N_F \times \frac{|updates|}{N_T} \times \frac{N_R}{N_C} \approx N_F \times 0.24 \times \frac{N_T - N_F}{N_C} \tag{9}$$

where N_T, N_R, N_F denote the sizes of the train, retain, and forget data, respectively. A search is performed only when the row is non-zero, i.e., $\frac{|updates|}{N_T}$ of cases (∼24% of cases across the benchmarks). The term $\frac{N_R}{N_C}$ represents the average number of retain samples sharing the same label as the forget sample, i.e., the substitute candidates. On the other hand, the number of inference searches in a retrain epoch is $N_R \times N_C = (N_T - N_F) \times N_C$. By dividing Equation (9) by the above equation, we derive a relative complexity of $\propto N_F/N_C^2$, showing that `ClearAttack`'s search cost may exceed the retraining (used in oracle/baselines) when the forget set is large or datasets have few classes, making retraining cheaper. To mitigate

this, we observed that a smaller set of representative retain vectors via K-medoids [36] or random sampling yields satisfactory results. On the UCIHAR dataset with a 10% forget ratio, exhaustive search examines over 800 vectors per sample and achieves an average similarity score of 0.88, whereas random search over only 25 vectors still attains 0.85.

3.3. Selective Retraining

In HDC training, memory updates occur only when a sample is misclassified. Consequently, ∼90% of inferences in the first epoch and ∼99% in later epochs are ineffectual. When `ClearAttack` performs unlearning through substitution, retain accuracy remains high (e.g., 96.7% on UCIHAR in Figure 5a), meaning that less than 4% of computation during tuning on retain data is useful. Thus, instead of retraining on the entire retain set, we focus on the most sensitive examples. Empirically, selecting 10% of samples with the lowest $\Delta s = s_t - s_{f_{max}}$ (where s_t is the similarity score of the true class, and $s_{f_{max}}$ is the highest competing score) achieves comparable accuracy. These low-margin samples are more prone to misclassification and therefore provide the greatest impact when the model is adjusted after unlearning. The selected subset is roughly an order of magnitude smaller than the retain set. Hence, the combination of a full unlearning and tuning for a sample stays well inside the compute budget of a single conventional retraining epoch, which is what makes on-device deletion viable in the first place.

4. Experiments

Experimental Setup: We consider three scenarios: (1) reactive defense on clean samples requiring removal; (2) proactive defense by identifying and unlearning unknown perturbed samples; and (3) proactive unlearning of known adversarial (Trojan) samples.

4.1. Privacy-Legislation-Driven Unlearning of Training Samples

First, we consider `ClearAttack` in a privacy-preserving reactive setting, where the model is already trained with unwanted data and the user requests their removal. We do not assume knowledge of whether the data are perturbed. We implement `ClearAttack` and two baselines—pruning [27] and perturbation [**?**]—in Python. 3 All methods are evaluated under the same pipeline, using identical datasets and splits. We use $D = 10k$-dimensional vectors and 30 training epochs, achieving nearly 100% training accuracy. Thus, performance differences reflect the unlearning strategy rather than model or data variations. For unlearning, we limit retraining to five epochs to keep overhead low. In the pruning baseline, 95% of the smallest-magnitude elements are sparsified to balance forgetting and performance. In the perturbation baseline, Gaussian noise is added proportional to the mean magnitude of model elements. Dataset statistics are summarized in Table 2; varying dataset sizes and class counts enable evaluation under different conditions. For runtime evaluation, we run `ClearAttack` and baselines on a Linux system with an Intel Core i7-8700K CPU and 64 GB RAM.

Table 2. Details of the datasets used in the evaluation.

Dataset	Classes	Train Data	Test Data	Description
UCIHAR [29]	12	6213	1554	Smartphone activity recognition
ISOLET [35]	26	6238	1559	Speech recognition
Fashion-MNIST [34]	10	60,000	10,000	Clothing image classification

Evaluation metrics: We evaluate the forget score of `ClearAttack` against baseline techniques using forget ratios of 5% (moderate) and 20% (high). As described in Section 2.3, we first compute ϵ^s for each forget sample s by comparing the distributions of N oracle models and unlearned models on the same data. We set $N = 256$, which strikes a balance

between computation load and evaluation reliability (i.e., reliable sample size to estimate the output distribution with high fidelity). Following [25], each sample is then assigned a forget quality $\mathcal{H}(s)$ based on its ϵ^s bin:

$$\mathcal{H}(s) = \begin{cases} 1 & \text{if } 0 \leq \epsilon^s < 0.5, \\ 0.5 & \text{if } 0.5 \leq \epsilon^s < 1.0, \\ \dots \end{cases} \quad Q_F = \frac{1}{N_F} \sum_{s \in \mathcal{F}} \mathcal{H}(s) \tag{10}$$

The overall forget score S_F also incorporates the accuracy of the unlearned model relative to the oracle, as defined in Equation (11), where RA and TA denote accuracies on retain and test data, and $\mathcal{M}_\mathcal{U}$ and $\mathcal{M}_\mathcal{R}$ represent the unlearned and oracle (retrained) models:

$$S_F = Q_F \times \frac{RA^{\mathcal{M}_\mathcal{U}}}{RA^{\mathcal{M}_\mathcal{R}}} \times \frac{TA^{\mathcal{M}_\mathcal{U}}}{TA^{\mathcal{M}_\mathcal{R}}} \tag{11}$$

We further analyze the runtime and complexity of `ClearAttack` compared with the baselines. Finally, we demonstrate an additional practical utility of `ClearAttack` by illustrating how its unlearning capability can mitigate the influence of a Trojan injected into the model.

Forget score: Figure 6 compares the forget score of `ClearAttack` with prior techniques under one and five retraining epochs. Substitute searches of `ClearAttack` are limited to 100 vectors. As described in Section 3, `ClearAttack` retrains only on a selected subset of retain data, whereas the baselines use the full dataset. Even with a single tuning epoch, `ClearAttack` clearly outperforms both one- and five-epoch baseline settings. At a 5% forget ratio with one retraining epoch, `ClearAttack` achieves a 3.02× higher score than the best baseline; with five epochs, it remains 1.89× better. At a 20% forget ratio, baseline scores drop by 4.2–16.5%, while `ClearAttack` slightly improves, reaching 3.58× and 2.39× higher scores for one- and five-epoch retraining, respectively. Compared to exhaustive search, the selective search (limited to 100 vectors) reduces the forget score by only 9.2% and 1.4% for one- and five-epoch retraining, while cutting search overhead by an order of magnitude as demonstrated in the next subsections.

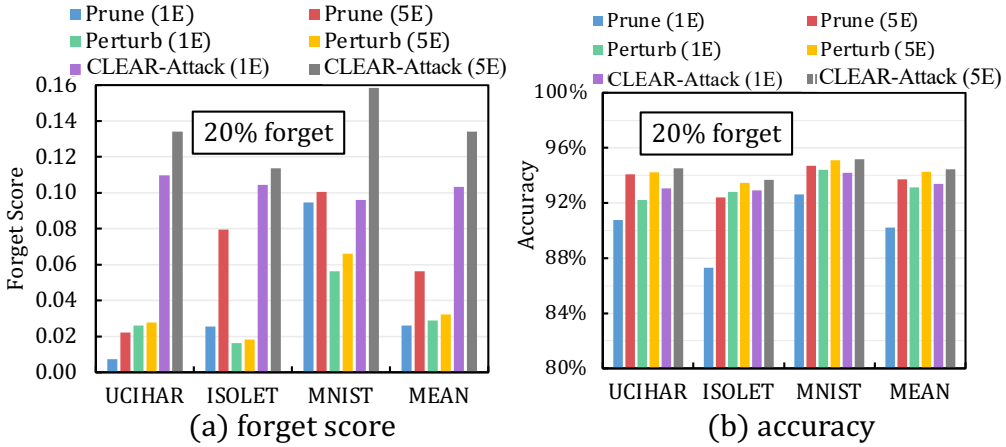


Figure 6. Forget score `ClearAttack` (with substitute searches limited to 100 vectors) and accuracy (using the proposed optimized table with a maximum of two non-zero entries per sample) compared with pruning [27] and perturbation [**?**]. (1 E) and (5 E) indicate the number of tuning (retraining) Epochs.

Accuracy: Figure 6b shows test accuracy after unlearning. `ClearAttack` (with an optimized table of at most two non-zero entries per sample) consistently outperforms baselines across all forget ratios and epochs as it introduces less noise than pruning or perturbation [27**?**].

At 5% forget ratio, `ClearAttack` with one retraining epoch already exceeds baselines trained for five epochs, achieving 1.37% and 0.80% higher average accuracy with one and five epochs, respectively. At higher forget ratios, all methods show slight degradation due to larger perturbations. Still, at 20% forget ratio, `ClearAttack` maintains a 0.23% and 0.19% advantage over the baselines. Compared to the oracle model retrained from scratch, `ClearAttack`'s accuracy drops by only 0.15% at 5% forget ratio and 0.62% at 20%.

Efficiency: ***Table overhead:*** Figure 7 compares the memory overhead of the *updates table*relative to the HDC model. The "Table (optimized)" configuration omits ineffectual samples and enforces that each sample updates only two classes. Each row stores a 2-byte sample index plus one byte each for the class index and value associated with the impacted **t** and **f** classes. Storing the full table (all indexes) results in 65% overhead, whereas our optimized version requires only 6.4% additional memory, on average, relative to the HDC model size. This overhead is what keeps the mechanism deployable: at 6.4% of the model size, tracking is a negligible addition that preserves HDC's lightweight footprint, whereas retaining the full training set for retraining would not.

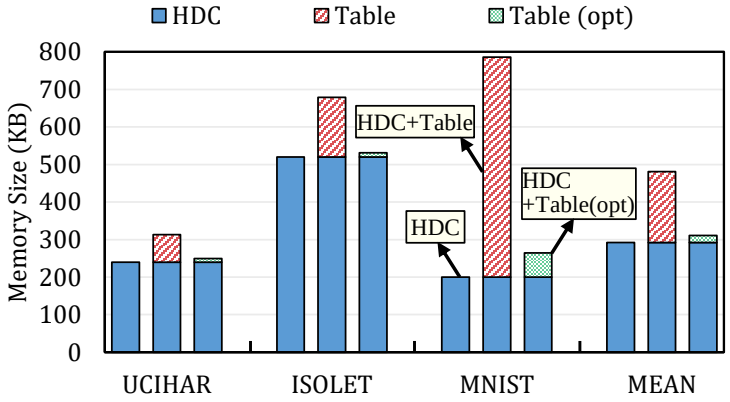


Figure 7. Comparing the overhead of *updates table* versus the model size. Optimized table (opt), on average, incurs only 6.4% overhead.

Efficiency: Search complexity:In `ClearAttack`, unlearning requires finding a suitable substitute for each forget sample, yielding a complexity of $O(N_F \times N_R)$, where N_F and N_R are the sizes of the forget and retain sets. As either grows, proportional to the overall training set size, the overhead can exceed the cost of full or partial retraining, as used in the oracle and baselines. Figure 8 compares the total per-epoch searches over the retain set with those required by `ClearAttack`. For a high forget ratio (20%), exhaustive search costs 3.5× more than retraining. Restricting the search to 100 (50) randomly chosen same-label vectors reduces this to 0.4× (0.2×) of retraining. With a low forget ratio (5%), costs further drop to 0.08× (0.04×).

Importantly, substitute searches in `ClearAttack` are performed only once, whereas retraining requires multiple epochs. Moreover, these searches involve simple dot-products of binary-encoded vectors, while training searches require dot-products between binary-encoded vectors and higher-bit class vectors, leading to proportionally higher resource and bandwidth cost.

Efficiency: Execution time:Figure 9 compares `ClearAttack` (search sizes 50 and 100) with baselines under one and five retraining epochs. Oracle retraining, pruning [27], and perturbation [**?**] have similar runtimes—dominated by retraining samples—so they are grouped as baselines. Leveraging selective retraining (Section 3), `ClearAttack` achieves 1.13–4.93× speedup (1 epoch) and 2.23–5.19× (5 epochs). The advantage decreases with larger forget ratios and search sizes. Search accounts for 13% of runtime in (50, 5) and up to 82% in (100,1). While implemented as int8 dot-products on CPU, hardware acceleration

(e.g., FPGA [37], in-memory [38]) can further reduce this overhead. The measured 5.2× speedup changes the unlearning from a scheduled maintenance operation into an event-driven one: a deletion request or a poisoning alert can be addressed as it arrives, rather than deferred until a full retraining pass can be afforded.

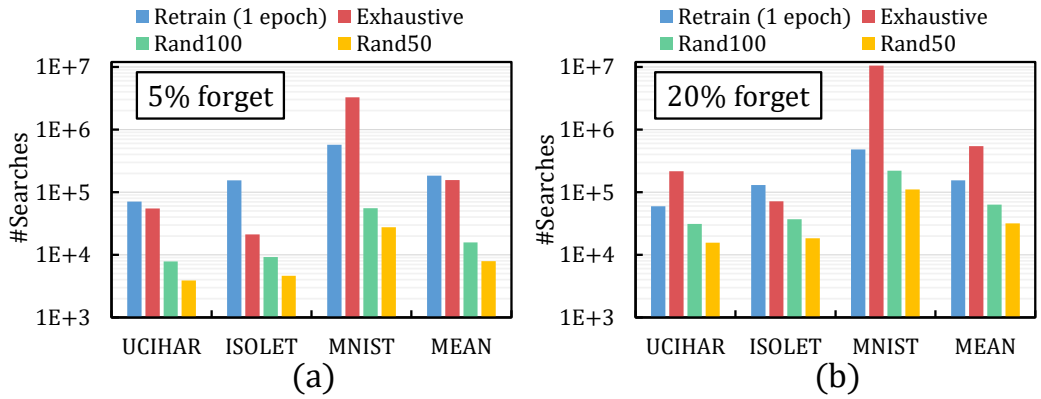


Figure 8. Comparing the number of searches in one epoch of retraining versus `ClearAttack`'s substitute search for forget data ratio of (**a**) 5% and (**b**) 20%.

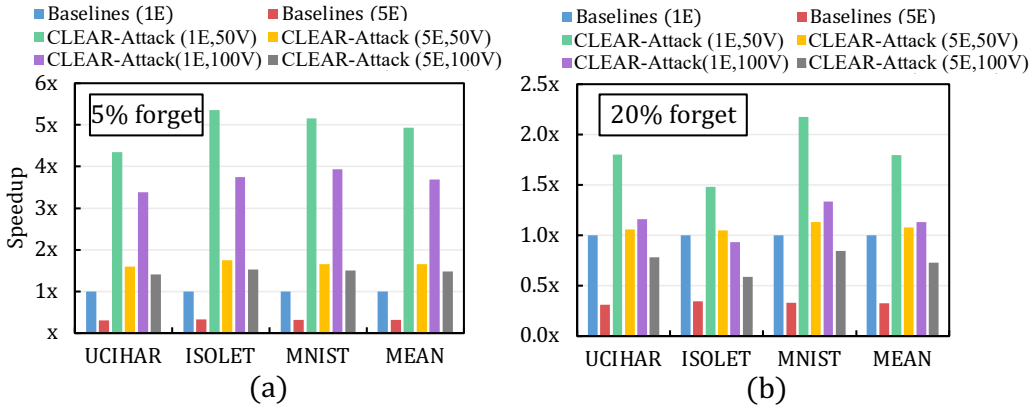


Figure 9. Speedup of `ClearAttack` over baselines. (1 E), (5 E) denote the number of retraining Epochs. (50 V), (100 V) denote the number of vectors in each substitute search in `ClearAttack` . Performance is normalized to baselines (1 E).

4.2. Unlearning as a Defense Against Trojan and Poisoned Data

The proposed `ClearAttack` can identify adversarial data and remove poisoned data influence in HDC. We consider a backdoor attack [39], where samples are relabeled to a target class to embed a trigger. As shown in Table 3, HDC's memorization makes it highly vulnerable: with up to 20% poisoning, trojan success reaches 91–100% while training accuracy remains high, masking the attack. The trojan success rate is the fraction of triggered inputs classified as the target label. Poisoning also reduces generalization, dropping test accuracy to 81–83% (versus 95–97% clean).

Figure 10 illustrates how the proposed unlearning method mitigates this effect. Immediately after unlearning the poisoned samples, the trojan success rate falls to below 3% for UCIHAR (originally 91–97%) and below 1% for ISOLET (originally 100%). Continued tuning on clean retain data further reduces the trojan success rate while gradually restoring performance on both retain and test sets. For example, under the 5% trojan setting, a single epoch of tuning raises test accuracy to 92.41% on UCIHAR and 92.82% on ISOLET while reducing the trojan success rate to 0.32% and 0%, respectively. Notably, these rates are achieved with no retraining at all: a compromised model can be made safe immediately upon detection, with the subsequent tuning epochs serving only to restore retain and

test accuracy. Although Figure 2.4 reports three independent trials and the forget score incorporates 256 oracle and unlearned model runs per forget sample, the accuracy and runtime results are reported as point estimates. Evaluating their variance across repeated runs is left for future work.

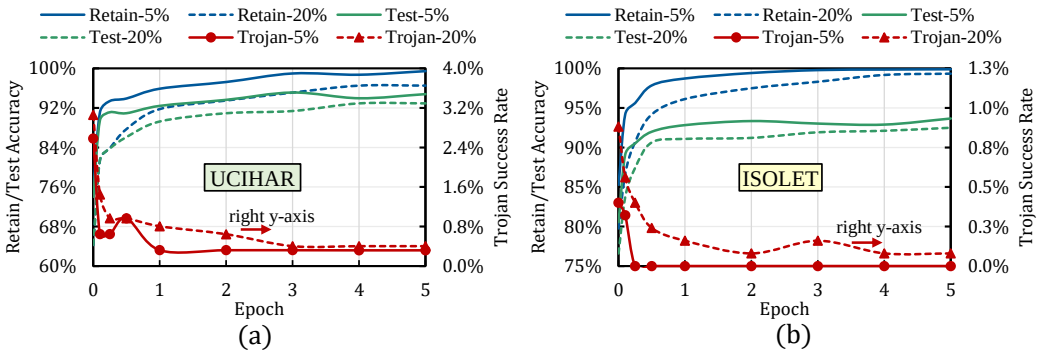


Figure 10. Retain and test data accuracy (left y-axis of each chart) and trojan success rate (right y-axis of each chart) after unlearning the trojan data for UCIHAR and ISOLET datasets respectively in (a) and (b). Here, 5% and 20% labels indicate the poisoning ratio in the training data.

Table 3. Retain/test accuracy and trojan success rate, all before unlearning. Here, 5% and 20% labels indicate the poisoning ratio in the training data.

	Retain-5%	**Retain-20%**	**Test-5%**	**Test-20%**	**Trojan-5%**	**Trojan-20%**
UCIHAR	99.71%	99.50%	94.02%	81.34%	**97.10%**	**90.90%**
ISOLET	100%	100%	92.05%	83.52%	**100%**	**100%**

5. Conclusions

Direct bundling in HDC makes removing specific data influence difficult. We propose `ClearAttack`, which efficiently tracks updates and unlearns data by canceling their effects and substituting samples to avoid catastrophic forgetting. It further preserves accuracy via selective retraining on a small subset. Overall, `ClearAttack` achieves up to 3.6× higher forget scores, 0.19–1.37% accuracy gains, and 5.2× faster performance than baselines. Costing a fraction of one retraining epoch and 6.4% additional memory, `ClearAttack` makes selective forgetting practical on resource-constrained devices, serving both data deletion requests and the removal of poisoned samples without a full retraining pass.

Author Contributions: Conceptualization, A. Asgarinejad, Q. Zhao, F. Ponzina., B. Aksanli and, T. Rosing.; methodology, B. Aksanli, F. Asgarinejad, F. Ponzina, A. Emami,.; software, A. Emami, J. Yang; validation, A. Emami, J. Yang and, Q. Zhao..; formal analysis, A. Emami, F. Asgarinejad; investigation, A. Emami.; resources, T. Rosing, F. Ponzina, F. Asgarinejad; data curation, F. Asgarinejad.; writing—original draft preparation, A. Emami, F. Asgarinejad.; writing—review and editing, T. Rosing, B. Aksanli, F. Asgarinejad., F. Ponzina; visualization, A. Emami and, F. Asgarinejad.; supervision, T. Rosing, B. Aksanli, F. Asgarinejad; project administration, T. Rosing, B. Aksanli, F. Asgarinejad; funding acquisition, T. Rosing, B. Aksanli; All authors have read and agreed to the published version of the manuscript.

Funding: This research received NSF, with award numbers 2112665, 2112167, 2003279, 2120019, 2211386, 2052809, 1911095 and in part by PRISM and CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA.

Data Availability Statement: All datasets are publicly available in the provided links.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chakraborty, A.; Alam, M.; Dey, V.; Chattopadhyay, A.; Mukhopadhyay, D. Adversarial attacks and defences: A survey. *arXiv* **2018**, arXiv:1810.00069.
2. Kanerva, P. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cogn. Comput.* **2009**, *1*, 139–159.
3. Kleyko, D.; Rachkovskij, D.; Osipov, E.; Rahimi, A. A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges. *ACM Comput. Surv.* **2023**, *55*, 175.
4. Thomas, A.; Dasgupta, S.; Rosing, T. A theoretical perspective on hyperdimensional computing. *J. Artif. Intell. Res.* **2021**, *72*, 215–249.
5. Aygun, S.; Moghadam, M.S.; Najafi, M.H.; Imani, M. Learning from hypervectors: A survey on hypervector encoding. *arXiv* **2023**, arXiv:2308.00685.
6. Yuan, X.; He, P.; Zhu, Q.; Li, X. Adversarial examples: Attacks and defenses for deep learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 2805–2824.
7. Asgarinejad, F.; Ponzina, F.; Gungor, O.; Rosing, T.; Aksanli, B. HDXpose: Harnessing Hyperdimensional Computing's Explainability for Adversarial Attacks. In Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design, New York, NY, USA, 27–31 October 2024.
8. Silver, D.L.; Yang, Q.; Li, L. Lifelong machine learning systems: Beyond learning algorithms. In Proceedings of the 2013 AAAI Spring Symposium Series, Palo Alto, CA, USA, 25–27 March 2013.
9. Jain, A.; Patel, H.; Nagalapatti, L.; Gupta, N.; Mehta, S.; Guttula, S.; Mujumdar, S.; Afzal, S.; Sharma Mittal, R.; Munigala, V. Overview and importance of data quality for machine learning tasks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Virtual Event, 6–10 July 2020.
10. Gong, Y.; Liu, G.; Xue, Y.; Li, R.; Meng, L. A survey on dataset quality in machine learning. *Inf. Softw. Technol.* **2023**, *162*, 107268.
11. Liu, K.Z. Machine Unlearning in 2024. Stanford Computer Science. 2024. Available online: https://ai.stanford.edu/~kzliu/blog/unlearning/ accessed on 20 November 2024).
12. Cao, Y.; Yang, J. Towards making systems forget with machine unlearning. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*; IEEE: Piscataway, NJ, USA, 2015.
13. Nguyen, T.T.; Huynh, T.T.; Nguyen, P.L.; Liew, A.W.C.; Yin, H.; Nguyen, Q.V.H. A survey of machine unlearning. *arXiv* **2022** arXiv:2209.02299.
14. Bourtoule, L.; Chandrasekaran, V.; Choquette-Choo, C.A.; Jia, H.; Travers, A.; Zhang, B.; Lie, D.; Papernot, N. Machine unlearning. In *Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP)*; IEEE: Piscataway, NJ, USA, 2021.
15. Cinà, A.E.; Grosse, K.; Demontis, A.; Vascon, S.; Zellinger, W.; Moser, B.A.; Oprea, A.; Biggio, B.; Pelillo, M.; Roli, F. Wild patterns reloaded: A survey of machine learning security against training data poisoning. *ACM Comput. Surv.* **2023**, *55*, 294.
16. Mantelero, A. The EU Proposal for a General Data Protection Regulation and the roots of the 'right to be forgotten'. *Comput. Law Secur. Rev.* **2013**, *29*, 229–235. Corrigendum in *Comput. Law Secur. Rev.* **2013**, *29*, 637.
17. Shastri, S.; Wasserman, M.; Chidambaram, V. The seven sins of {Personal-Data} processing systems under {GDPR}. In Proceedings of the 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19), Renton, WA, USA, 8 July 2019.
18. Kleyko, D.; Rachkovskij, D.A.; Osipov, E.; Rahimi, A. A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations. *ACM Comput. Surv.* **2022**, *55*, 175.
19. Hernández-Cano, A.; Cammarota, R.; Imani, M. Prid: Model inversion privacy attacks in hyperdimensional learning systems. In *Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC)*; IEEE: Piscataway, NJ, USA, 2021.
20. Khaleghi, B.; Imani, M.; Rosing, T. Prive-hd: Privacy-preserved hyperdimensional computing. In *Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC)*; IEEE: Piscataway, NJ, USA, 2020.
21. Hernández-Cano, A.; Matsumoto, N.; Ping, E.; Imani, M. Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system. In *Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*; IEEE: Piscataway, NJ, USA, 2021.
22. Ma, D.; Zhang, S.; Jiao, X. Robust hyperdimensional computing against cyber attacks and hardware errors: A survey. In Proceedings of the 28th Asia and South Pacific Design Automation Conference, Tokyo, Japan, 16–19 January 2023.
23. Zhang, D.; Fan, C.; Li, Z.; Qin, C.; Dan, J. Scale-Spectrum-State Collaborative Fusion Network for Real-Time Extreme Small Object Detection in Intelligent Sports Analytics. *Expert Syst. Appl.* **2027**, *333*, 133831. https://doi.org/10.1016/j.eswa.2026.133831.
24. Liang, L.; Xiang, P.; Huang, H.; Zhang, N.; Qi, P.; Yin, Z.; Zhai, W.; Zhang, D. Sec-GNN-Driven Joint Multidimensional Anti-Eavesdropping Optimization for Secure UAV-Satellite Communications. *IEEE Internet Things J.* **2026**, *13*, 28694–28706. https://doi.org/10.1109/JIOT.2026.3685201.
25. Evaluation for the NeurIPS Machine Unlearning Competition. Available online: https://unlearning-challenge.github.io/assets/data/Machine_Unlearning_Metric.pdf (accessed on 19 November 2025).

26. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **1958**, *65*, 386–408.
27. Liu, J.; Ram, P.; Yao, Y.; Liu, G.; Liu, Y.; Sharma, P.; Liu, S.; Liu, Y.; Ram, P.; Sharma, P.; Yao, Y. Model sparsity can simplify machine unlearning. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 51584–51605.
28. Available online: https://www.kaggle.com/competitions/neurips-2023-machine-unlearning/discussion/459334 (accessed on 20 November 2024).
29. UCI Machine Learning Repository. Available online: https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones (accessed on 24 November 2024).
30. Meng, D.; Chen, H. Magnet: A two-pronged defense against adversarial examples. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017.
31. Zhou, D.; Liu, T.; Han, B.; Wang, N.; Peng, C.; Gao, X. Towards defending against adversarial examples via attack-invariant features. In *Proceedings of the International Conference on Machine Learning*; PMLR: Cambridge, MA, USA, 2021.
32. Gungor, O.; Rosing, T.; Aksanli, B. Roldef: Robust layered defense for intrusion detection against adversarial attacks. In *Proceedings of the 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*; IEEE: NJ, USA, 2024.
33. Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv* **2017**, arXiv:1706.06083.
34. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv* **2017** arXiv:1708.07747.
35. UCI Machine Learning Repository. Available online: http://archive.ics.uci.edu/ml/datasets/ISOLET (accessed on 19 November 2024).
36. Park, H.S.; Jun, C.H. A simple and fast algorithm for K-medoids clustering. *Expert Syst. Appl.* **2009**, *36*, 3336–3341.
37. Salamat, S.; Imani, M.; Khaleghi, B.; Rosing, T. 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, Seaside, CA, USA, 24–26 February 2019.
38. Karunaratne, G.; Le Gallo, M.; Cherubini, G.; Benini, L.; Rahimi, A.; Sebastian, A. In-memory hyperdimensional computing. *Nat. Electron.* **2020**, *3*, 327–337.
39. Gu, T.; Dolan-Gavitt, B.; Garg, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv* **2017**, arXiv:1708.06733.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.