

# Testing and Understanding Deviation Behaviors in FHE-hardened Machine Learning Models

Yiteng Peng<sup>†</sup>, Daoyuan Wu<sup>†\*</sup>, Zhibo Liu<sup>†</sup>, Dongwei Xiao<sup>†</sup>, Zhenlan Ji<sup>†</sup>, Juergen Rahmel<sup>‡</sup> and Shuai Wang<sup>†\*</sup>

<sup>†</sup>The Hong Kong University of Science and Technology, Hong Kong, China

<sup>‡</sup>HSBC, Hong Kong, China

<sup>†</sup>{ypengbp, daoyuan, zliudc, dxiaoad, zjiae, shuaiw}@cse.ust.hk, <sup>‡</sup>juergen.rahmel@hsbc.com.hk

**Abstract**—Fully homomorphic encryption (FHE) is a promising cryptographic primitive that enables secure computation over encrypted data. A primary use of FHE is to support privacy-preserving machine learning (ML) on public cloud infrastructures. Despite the rapid development of FHE-based ML (or HE-ML), the community lacks a systematic understanding of their robustness.

In this paper, we aim to systematically test and understand the deviation behaviors of HE-ML models, where the same input causes *deviant outputs* between FHE-hardened models and their plaintext versions, leading to completely incorrect model predictions. To effectively uncover deviation-triggering inputs under the constraints of expensive FHE computations, we design a novel differential testing tool called HEDIFF, which leverages the margin metric on the plaintext model as guidance to drive targeted testing on FHE models. For the identified deviation inputs, we further analyze them to determine whether they exhibit general noise patterns that are transferable. We evaluate HEDIFF using three popular HE-ML frameworks, covering 12 different combinations of models and datasets. HEDIFF successfully detected hundreds of deviation inputs across almost every tested FHE framework and model. We also quantitatively show that the identified deviation inputs are (visually) meaningful in comparison to regular inputs. Further schematic analysis reveals the root cause of these deviant inputs and allows us to generalize their noise patterns for more directed testing. Our work sheds light on enabling robust HE-ML for real-world usage.

## I. INTRODUCTION

In recent years, there has been rapid growth in fully homomorphic encryption (FHE) algorithms [1–3]. As one of the most exciting cryptographic breakthroughs, FHE enables privacy-preserving machine learning [4–6] that allows normal users to encrypt their data locally and then directly upload the ciphertext data to a remote server for processing. The server performs computation on the encrypted data and returns the encrypted results to the users. Thus, users can protect their confidential data without compromising the convenience of machine learning as a service (MLaaS). Commercial vendors like Intel and IBM are actively developing and promoting their HE-ML frameworks [6, 7], greatly spurring real-world ML applications with privacy considerations [8, 9].

Despite the prosperous development of HE-ML, there is still a lack of systematic understanding of the quality of HE-ML applications. In general, the design of FHE primitives is often subtle and complex [3, 10]. Even worse, FHE protocols are well-known for their slow speed, and we find that modern

HE-ML frameworks often aim to extensively support complex DNN operators, tensor computations, and imperative programming [5–7]. Hence, various optimization and approximation schemes [11, 12] are involved in converting standard floating-point computations into FHE-supported forms. While these efforts substantially reduce the hurdle of employing FHE primitives in MLaaS applications, they may increase the burden on developers to deliver bug-free HE-ML applications. In fact, our preliminary studies in §III have shown that HE-ML models may yield specious outputs under certain inputs.

In this paper, we aim to systematically test and understand the deviation behaviors in FHE-hardened ML models. We target deviations where the same input causes *deviant outputs* between FHE models and their plaintext versions, leading to completely incorrect model predictions. Inspired by the widely used differential testing (DT) [13] technique in the software engineering (SE) community, which aims to find bugs in similar software, a straightforward approach to finding deviation behaviors is to conduct DT directly. This technique will systematically explore the input space and detect inputs that lead to differing outputs between the FHE and plaintext models. However, directly using DT is expensive, as a prediction in FHE model often requires several orders of magnitude more computation than in the plaintext model (e.g., half a minute vs. less than 0.1 millisecond). To deliver an efficient DT specific to FHE models, this paper first conducts a preliminary study to identify a proper metric, which serves as the feedback to drive guided test input selection and mutation. Through empirical analysis, we eventually choose *margin*, defined as the difference between the largest and the second-largest prediction values in the model’s inference result, as our metric. We present details of this preliminary study in §III.

Based on the insights above, we design a novel differential testing tool called HEDIFF, which leverages the margin metric on the plaintext model as guidance to drive targeted testing on FHE-hardened models. Specifically, HEDIFF first conducts margin-based input filtering to identify only those promising ones as candidate seeds for mutation. Then, HEDIFF adds a small amount of noise to each seed and mutates them in ways that could potentially cause deviant behaviors. Inspired by the success of the Projected Gradient Descent (PGD) algorithm [14] in finding adversarial examples (AEs) [15], we adapt PGD to mutate seed inputs in a way that reduces the margin value for our scenario of finding deviation behaviors. Furthermore,

\*Corresponding authors.

with detected deviation inputs on hand, we present a schematic analysis to understand the root cause of deviant behaviors in FHE models, and also generalize the noise patterns of these deviation inputs. The generalized patterns can be treated like weak Universal Adversarial Perturbations (UAPs) [16], enabling further directed identification of deviation inputs. This can be analogous to exploiting the correlations between similar codes to find new defects by utilizing patterns of previously discovered defects in traditional software.

Our evaluation encompasses three mainstream HE-ML frameworks, including TenSEAL [4], Concrete-ML [5], and HElayers [6]. These frameworks are developed and maintained by industrial vendors such as OpenMined, IBM, and Zama. We launch testing towards similar neural network architecture to prominent FHE-friendly ML models [17] and multilayer perceptron (MLP) trained for general and domain-specific tasks with different activation functions. With real-world datasets, including Bank [18], Credits [19], MNIST [20], and DIGITS [21], we simulate scenarios where sensitive data of different types are processed by HE-ML models. HEDIFF generates about 45,500 mutated inputs in total to test HE-ML models. During approximately 54 hours of testing, we detected a total of 2,509 deviation inputs across almost all the tested FHE frameworks and models. We show that these inputs have high (visual) similarity compared with normal inputs, thereby uncovering defects that may cause substantial confusion for users of these FHE frameworks in their daily usage. Our further study depicts the root cause of these defects. Looking ahead, besides using standard test data for accuracy validation, developers can use HEDIFF to test potential deviation behavior in their HE-ML models before releasing them to end users. In sum, we make the following contributions:

- We, for the first time, study flaws introduced in FHE, a highly visible cryptographic primitive that enables secure computation on encrypted data. We reveal flaws within the context of HE-ML models, which can cause confusion or adversarial manipulations during usage.
- We present HEDIFF, an automated testing tool designed to uncover inputs that cause deviated outputs in HE-ML models compared to their plaintext counterparts. HEDIFF employs carefully designed margin-based feedback to guide test input selection and mutation.
- Our large-scale evaluation of mainstream HE-ML frameworks and models exposes a substantial number of flaws. We also discuss the root causes and demonstrate the potential consequences of exploiting these defects.

We maintain HEDIFF to benefit future research at [22].

## II. BACKGROUND AND MOTIVATION

This section introduces the mechanisms and core components that enable efficient FHE computation. We then illustrate how FHE can be employed to enable privacy-preserving ML inference and the significance of HE-ML in practice. Note that we do not specifically distinguish FHE and HE in this paper, and to ease the reading, we refer to FHE-based ML as HE-ML.

### A. Homomorphic Encryption (HE)

HE is a cryptographic primitive that enables computation on encrypted data. A major application scenario for HE is cloud computing, where clients send their data to computationally powerful but untrusted servers for computation. The servers can compute the encrypted data and return encrypted results to the clients, while learning nothing about the plaintext data or results. Denote the encrypted data as  $c$  corresponding to a plaintext message  $\mathbf{m}$ , i.e.,  $Encrypt(\mathbf{m}) = c$ . The equivalence of computation on encrypted data and plaintext data constitutes the fundamental property of HE, as shown as follows:

$$Decrypt(f(c)) = Decrypt(f(Encrypt(\mathbf{m}))) = f(\mathbf{m}) \quad (1)$$

where  $f$  is the function that the clients intended to compute on their data. The function  $f$  is typically composed of addition and/or multiplication operations. Depending on the type and the number of supported operations, HE can be classified into different categories, including partially homomorphic encryption (PHE), somewhat homomorphic encryption (SHE), and fully homomorphic encryption (FHE). PHE supports either addition or multiplication operations, but not both, while SHE supports both but is constrained by the total number of operations. FHE supports an arbitrary number of addition and multiplication operations. Due to its versatility, FHE is generally referred to as *cryptology's holy grail* [2, 23]. As such, this paper focuses on FHE schemes. However, HEDIFF is not limited to FHE and has high generality across different HE algorithms by treating HE-ML frameworks as a black box.

The privacy guarantee of FHE is based on the hardness of certain mathematical problems, particularly the Learning with Errors (LWE) problem [24]. Given an  $n$ -element secret vector  $\mathbf{s} \in \mathbb{F}_q^n$ , where  $\mathbb{F}_q$  denotes a finite field, the client samples a matrix  $\mathbf{A} \in \mathbb{F}_q^{n \times n}$  from a uniform distribution and a noise vector  $\mathbf{e} \in \mathbb{F}_q^n$  from a small-variance Gaussian distribution. Based on the Lattice theory [24], it is nearly impossible for the server to uncover the secret vector  $\mathbf{s}$  from the term  $\mathbf{A}\mathbf{s} + \mathbf{e}$ . Hence, the client can use  $\mathbf{s}$  as a key to encrypt its message  $\mathbf{m}$  by computing  $\mathbf{b} = \mathbf{A}\mathbf{s} + 2\mathbf{e} + \mathbf{m}$ , then sends the ciphertext  $\mathbf{c} = (\mathbf{A}, \mathbf{b})$  to the server. The server can operate on the ciphertext  $\mathbf{c}$  by addition and multiplication operations without learning the plaintext message  $\mathbf{m}$ , and returns the final computed ciphertext  $\mathbf{c}'$  to the client. The client can then decrypt the ciphertext  $\mathbf{c}'$  to obtain the plaintext result  $\mathbf{m}'$ .

**Hurdles of Applying FHE to ML.** Although FHE serves a promising solution to enable privacy-preserving computation, applying FHE to ML is non-trivial because:

**Computation on Real Number.** ML models typically work with real numbers, while FHE can only naturally support integer operations. One viable solution is to convert all real numbers at the *application* level, in which the server rounds the real numbers into integers before applying FHE, with techniques such as model quantization [25]. Such an approach can yield compatible accuracy with the real number counterpart [26]. Another solution is to support real numbers at the *cryptographic* level, in which the FHE scheme natively supports real number

operations. Many FHE schemes have been proposed to support real number operations, the most prominent of which are CKKS [3] and RNS-CKKS [27]. Such schemes are often referred to as *approximate* FHE schemes, as they can only support approximation of real number operations. The accuracy of the approximation is related to the number of bits used to represent the real number. The more bits used, the higher the accuracy and the greater the computational cost.

**Non-linear Functions.** FHE requires the homomorphic function  $f$  to be expressed in terms of addition and multiplication operations or some other operations like bit-shift that the underlying FHE schemes can support. However, in the machine learning domain, many widely-used functions like ReLU and Sigmoid are non-linear, which does not directly map to FHE primitive operations. A plethora of works [28–30] approximate non-linear functions with polynomials of the form  $a_k x^k + \dots + a_1 x + a_0$ , which can be expressed in terms of addition and multiplication operations. Some HE-ML frameworks like Concrete-ML provide built-in support for non-linear function conversion, while others like TenSEAL require users to manually convert the non-linear functions to polynomial form. Besides conversion, the model can also be trained directly with the approximated function in an FHE-friendly format to avoid the noise introduced in the conversion process. Our evaluation covered all these three scenarios.

### B. FHE in MLaaS

To date, FHE is typically employed in MLaaS and commercialized by platforms like Amazon AWS [31] and Google Cloud [32]. Often, the cloud holds an ML model and provides the ML inference service to a client. The client has sensitive data and does not want to reveal it to the cloud, but wants to use the cloud’s model for inference. Meanwhile, the cloud does not want to reveal internal service implementations, particularly the *model architecture* and *parameter weights*, to the client, as designing the model architecture can cost considerable effort, and the model might be trained on proprietary datasets.

To use FHE, the cloud first trains a *plaintext* model using standard ML training techniques. The plaintext model is then transformed into a FHE-compatible format with the help of HE-ML frameworks like TenSEAL [4], finally encrypted and deployed to the cloud. To use the deployed model, the client first encrypts his sensitive data locally, then sends the encrypted data to the cloud. The cloud runs ML inference on the ciphertext from the client, and returns the encrypted result. After that, the client decrypts it to obtain the result in plaintext. During the process, the cloud learns nothing about client data due to the security guarantee of FHE, while the client cannot either learn the model architecture and parameter weights of the cloud.

### C. Research Motivation and Position

**Motivation.** As aforementioned, there has been high interest in commercializing and deploying FHE in practice. Several industrial giants and startups are actively developing and promoting their HE-ML solutions in the market [33–35]. HE-ML has also been adopted in various highly-regulated sectors, including

healthcare [36], finance [37–40], and government [41, 42]. Overall, the market size of FHE has reached 190 million USD in 2022 and is expected to hit 300 million by 2030 [43].

Despite the promising adoption of FHE in practice, the correctness of HE-ML frameworks and corresponding FHE-hardened models has not been systematically studied, and their potential vulnerabilities have not been well understood. We anticipate that the defects in HE-ML frameworks may lead to severe consequences in FHE-hardened models, including missed detections of tax evasion, credit fraud, or even misdiagnosis of patients. Even worse, our tentative explorations show that defects in HE-ML frameworks are common and hard to diagnose, given the algorithmic obscurity of FHE, the complexity of HE-ML frameworks, and the high computational cost of testing FHE. Hence, it is imperative to understand the reliability of HE-ML frameworks and corresponding FHE-hardened models, and to develop effective testing techniques to uncover potential defects.

**Position.** We aim to develop automated testing to uncover defects in the *inference process* of HE-ML models, denoting the most common usage of HE-ML in practice. To our knowledge, few HE-ML frameworks support HE-enabled model training, given the prohibitive computation overheads. The training of ML models is typically performed offline, where the training data is assumed available in plaintext. The trained model is then transformed by HE-ML frameworks into an FHE-compatible format and deployed (e.g., on the cloud) to perform inference. We also assume that the original model is *well trained*, i.e., it has been trained on a large amount of high-quality data and has achieved high accuracy on the validation set. Yet, we assume the predictions of FHE-hardened, well-trained ML models may be incorrect under certain inputs and aim to detect inputs that cause these mispredictions in HE-ML models.

## III. PRELIMINARY STUDY AND OBSERVATION

This section conducts a preliminary study to summarize insights that guide the design of HEDIFF, which will be presented in §IV.

### A. Deviation Inputs and Preliminary Testing

**Definition.** Following the discussion in §II-C, the objective of our preliminary testing is to analyze the characteristics of *deviation-triggering inputs* (or simply called “deviation inputs”), which are inputs  $i$  that can make the FHE-hardened model  $M_f$  to predict the wrong label while its plaintext version  $M_p$  to output the ground-truth label. Specifically, deviation inputs need to satisfy the following constraints:


$$\begin{cases} \text{Argmax}(M_f(i)) \neq \text{Argmax}(M_p(i)) \\ \text{Argmax}(M_p(i)) = \text{label} \end{cases} \quad (2)$$

where  $M_f(i)$  and  $M_p(i)$  denote the logits output of the FHE-hardened and plaintext models, respectively, and *label* is the ground truth of input  $i$ . *Argmax* is used to choose the class with the highest logits, i.e. the prediction class of the model.

**Example.** Table I depicts the logits output (the output without Softmax) of the plaintext and FHE-hardened models using the Concrete-ML framework [5]. The class prediction (i.e., the class with the maximum value of the logits) of the plaintext



TABLE I: Logits output of the same image from the plaintext and FHE-hardened ten-class classification model. The class predictions of the two models are bolded.



Label: 4

Class No.	0	1	2	3	4	5	6	7	8	9
Plain Logits	-2.36	-0.39	-7.57	-6.56	<b>2.95</b>	-1.85	-3.31	2.14	-0.47	2.80
FHE Logits	-2.26	-0.52	-7.53	-6.48	2.75	-1.85	-3.27	2.02	-0.49	<b>2.89</b>

TABLE II: Deviation inputs identified during preliminary testing. The 2-4 columns show the number and ratio of deviation inputs under three mainstream HE-ML frameworks.

Datasets, Models	TenSEAL	Concrete-ML	HElayers
MNIST, Cryptonets-Square	27/5000(0.54%)	1/1000(0.10%)	1/5000(0.02%)
DIGITS, Cryptonets-Sigmoid	10/1437(0.70%)	2/500(0.40%)	4/1437(0.28%)
Credit, MLP-Sigmoid	367/10617(3.46%)	12/1000(1.20%)	367/10617(3.46%)
Bank, MLP-AppReLU	6/6174(0.10%)	11/1000(1.10%)	0/6174(0.00%)

model is 4, meaning the prediction of the fourth position was selected. In contrast, the class prediction of the FHE-hardened model is 9 (i.e., the prediction at the ninth position).

**Preliminary Testing.** We find that deviation inputs are not rare. Rather, we observe such inputs across different encryption schemes, HE-ML frameworks, models, and datasets. Here, we launch a standard differential testing (DT) task — we iterate through the training dataset to count deviation inputs with respect to the objective in Eq. 2. This way, we measure the ratio of deviation inputs residing in the standard training dataset and study their characteristics for further efficient testing.

Table II shows the results of preliminary testing across four datasets, four models, and three HE-ML frameworks, including TenSEAL [4], Concrete-ML [5], and HElayers [6]. Specifically, we iterate through the whole training dataset for DIGITS, Credit, and Bank datasets on TenSEAL and HElayers, and randomly select several samples for testing from the MNIST dataset and the datasets on Concrete-ML due to their extremely time-consuming. Details of models and framework configurations will be further introduced in §V.

**Reflection and Challenge.** While we find deviation inputs across different settings, the ratio of deviation inputs existed in the original training dataset is low. However, this does not mean that this phenomenon is unimportant; on the contrary, we view this problem as critical albeit difficult to detect, especially in FHE-hardened models where the inference computation is time-consuming and “blanket searching” is hardly feasible. Overall, given the objective defined in Eq. 2 and the observations above, one might expect to employ DT to drive automated testing, where we compute and compare the predictions of both the FHE-hardened and plaintext models. However, considering that FHE computation is slow, directly applying DT techniques to this scenario is less feasible. We need a highly targeted approach to minimize computation on FHE-hardened models. Below, in §III-B, we explore leveraging certain metrics to guide the search of deviation inputs.

### B. Explore Efficient Deviation Inputs Searching

To deliver efficient DT and uncover deviation inputs in FHE-hardened models, the key is to minimize the expensive FHE computation. As such, we attempt to identify a particular metric

TABLE III: The number and ratio of deviation inputs that cause the predictions to change to the second most likely class. The deviation inputs are collected from the results in Table II.

Datasets, Models	TenSEAL	Concrete-ML	HElayers
MNIST, Cryptonets-Square	0/27(0.00%)	1/1(100.00%)	0/1(0.00%)
DIGITS, Cryptonets-Sigmoid	7/10(70.00%)	2/2(100%)	3/4(75.00%)
Credit, MLP-Sigmoid	367/367(100.00%)	12/12(100.00%)	367/367(100.00%)
Bank, MLP-AppReLU	6/6(100.00%)	11/11(100.00%)	0/0

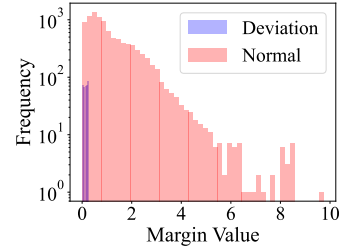


Fig. 1: Statistics of margin value on deviation inputs and normal inputs. The x-axis denotes margin values, and the y-axis denotes the number of normal and deviation inputs.

of deviation inputs on the plaintext model that could be treated as a “mirror.” By calculating and observing this “mirror” metric solely on the plaintext model, we would have confidence that the corresponding input is more likely to trigger deviant outputs in the FHE models. Based on this insight, we now analyze the characteristics of deviation inputs collected in §III-A and explore possible metrics that satisfy the above conditions.

**Observation.** Through a detailed analysis of the identified deviation inputs, we find that nearly all of them trigger the model to predict the second most likely class in the plaintext model. We thus measure and report the ratio of deviation inputs that fall into this category. As shown in Table III, the ratio is high across nearly all datasets and models. Moreover, when comparing the logits of the plaintext model and the FHE-hardened model for each normal/deviation input, the absolute difference in logits for the prediction classes with the two highest probabilities is almost the same. For instance, consider the example in Table I, where the fourth and ninth classes have the highest probabilities in the plaintext and FHE-hardened model. The difference in the logits of the fourth and ninth classes is about 0.20 and 0.09, respectively. In other words, if the difference between the logits of the most likely and the second most likely classes is marginal, deviation inputs may be more likely to exist when the logits of these classes change by approximately the same absolute values.

**Margin.** Through the above observation and analysis, it is natural to consider the *margin* as our metric, given margin is defined as the difference between the largest prediction value and the second-largest prediction value in the model’s inference result. Formally, given a model  $M$  and an input  $x$ , we have  $Margin(x) = pred_i - pred_j$ , where  $i = Argmax(M(x))$ ,  $pred_i = M(x)_i$  and  $j = Argmax(M(x)_{k \neq i})$ ,  $pred_j = M(x)_j$ . In deep learning theory, margin is deemed the minimum distance of an input to the decision boundary [44]. While deviation behavior is due to the difference in the decision boundaries between FHE-hardened and plaintext models, data

that is close to the decision boundary of the plaintext model is more likely to cause deviation, as a slight difference in the decision boundary of FHE-hardened model near this data may induce an altered prediction, i.e., deviation behavior. Thus, margin, as a metric for the distance to the decision boundary, should be a good choice to identify deviation inputs.

**Empirical Validation.** We conduct an experiment to validate the effectiveness of the margin metric. Specifically, we compare the margin of each deviation input and normal input from a statistical perspective. We chose the deviation inputs found from Credit dataset, as shown in Table II. We then count the number of normal and deviation inputs within different margin value ranges. As in Fig. 1, we observe a clear trend that most deviation inputs (denoted with the blue bar) have small margin values. We interpret that inputs with smaller margin values are closer to the decision boundary, and therefore, are more likely to reflect the boundary differences between the plaintext and FHE-hardened models through subtle mutations.

**Formalization.** To further illustrate the effectiveness of the margin metric, we formalize the relationship between margin and deviation inputs. Let margin be  $\phi(i) = \text{top1}(\text{logits}_p(i)) - \text{top2}(\text{logits}_p(i))$ , where  $\text{top1}$  and  $\text{top2}$  are used to get the max and second-max value in  $\text{logits}_p(i)$ , which means the logits value of plaintext model given input  $i$ . Consider the max error  $\delta(i)$  introduced in FHE-hardened model’s prediction: for each value of class  $j$  in  $\text{logits}_f(i)$  we have:  $|\text{logits}_f(i)_j - \text{logits}_p(i)_j| \leq \delta(i)$ , i.e.

$$\text{logits}_p(i)_j - \delta(i) \leq \text{logits}_f(i)_j \leq \text{logits}_p(i)_j + \delta(i) \quad (3)$$

For the difference of two classes  $j_1$  and  $j_2$ , we have:

$$\begin{aligned} \text{logits}_p(i)_{j_1} - \text{logits}_p(i)_{j_2} - 2\delta(i) &\leq \\ \text{logits}_f(i)_{j_1} - \text{logits}_f(i)_{j_2} & \\ \leq \text{logits}_p(i)_{j_1} - \text{logits}_p(i)_{j_2} + 2\delta(i) & \end{aligned} \quad (4)$$

Suppose  $j_1$  and  $j_2$  correspond to the max and second max value in  $\text{logits}_p(i)$ , respectively, we have:

$$\phi(i) - 2\delta(i) \leq \text{logits}_f(i)_{j_1} - \text{logits}_f(i)_{j_2} \leq \phi(i) + 2\delta(i) \quad (5)$$

As  $\phi(i) \geq 0$  (by the definition of margin) and error  $\delta(i)$  is necessarily small (since large error would compromise both the usability of FHE computations and the FHE-hardened model’s accuracy), the smaller the  $\phi(i)$ , the higher probability that  $\phi(i) - 2\delta(i) < 0$  and  $\text{logits}_f(i)_{j_1} - \text{logits}_f(i)_{j_2} < 0$ , which will make the prediction label in FHE-hardened model change. In short, we consider margin to be a straightforward yet highly effective metric to guide the search for deviation inputs.

#### IV. DESIGN OF HEDIFF

Margin’s effectiveness for exploring deviant outputs in an FHE-hardened ML model (as illustrated in §III) motivates us to design a novel margin-guided tool named HEDIFF to test and understand deviation behaviors in FHE-hardened models.

**Overview.** Fig. 2 shows a high-level workflow of HEDIFF, which consists of three main steps. ① Given a set of original data inputs  $\mathcal{O}$  from the given dataset, HEDIFF performs margin-based seed filtering (§IV-A) to select inputs  $\mathcal{S}$  with the lowest margin values on the plaintext model as seeds. The insight is that these seeds are more likely to trigger deviant outputs in the

FHE-hardened model. ② HEDIFF then conducts margin-guided differential testing (§IV-B) aimed at mutating the inputs towards lower margin values on the plaintext model and employs DT as an oracle to identify seeds  $\mathcal{D}$  that are deviation inputs. ③ HEDIFF does not stop at identifying individual deviation inputs but also attempts to understand them by generalizing the noise patterns of these deviation inputs (§IV-C). The generalized patterns can be treated like weak UAPs and enable further identification of deviant inputs. For example, for the  $\mathcal{S} - \mathcal{D}$  seed inputs that originally do not produce deviant outputs in the second step, we can further mutate them based on the noise patterns generalized from the  $\mathcal{D}$  deviant outputs.

##### A. Margin-based Input Filtering for Seeds

Not all data inputs from the given dataset are suitable to be evolved into deviation inputs. We need to identify only those promising ones as candidate seeds for further mutation in §IV-B. Specifically, we found that inputs with lower margin values of the prediction outputs on the plaintext model are more likely to evolve into deviation inputs than those with higher margins, making them more suitable as *seeds*. For example, our experiment in §VI-B shows that if we use the top 200 inputs in the dataset with the lowest margin values as seeds, the effectiveness of using them to find deviation inputs is greater than that of using the randomly selected 200 inputs.

With this observation, HEDIFF sorts the training data inputs  $\mathcal{O}$  by their margin values on the plaintext model and selects the samples  $\mathcal{S}$  with the smallest margin values as seeds, which have a higher potential to evolve into deviation inputs. We will describe the concrete configuration of  $\mathcal{O}$  and  $\mathcal{S}$  in §VI.

##### B. Margin-oriented Differential Testing

With the selected seed inputs  $\mathcal{S}$ , we need to mutate them in a way that they can eventually cause deviant behaviors. To achieve this, we add a small amount of noise to each seed and observe whether such a mutated input evolves toward potential deviation. According to insights obtained from §III, such a mutation should evolve towards a decreased margin value of the generated prediction output on the plaintext model, so that the mutated inputs have a higher chance of causing deviant behaviors in FHE-hardened models.

Toward this objective, a straightforward approach is to use random mutation. For example, for each seed input, we could randomly mutate it ten times and keep only those with decreased margin values on the plaintext model. However, it is hard for such a random approach to always identify suitable mutated inputs. Moreover, it requires more computation due to its blind testing nature. Therefore, we seek a more targeted mutation method. Given that the PGD algorithm [14] has demonstrated its effectiveness in finding AEs for subjects like images, we explore how to customize PGD in our scenario to target the mutation of seed inputs towards the direction of lowering the margin value. To clarify, although we apply the idea of well-known PGD in our work, we are not looking for AEs, but rather deviation inputs that cause incorrect predictions in FHE-hardened models. More discussions from conceptual,

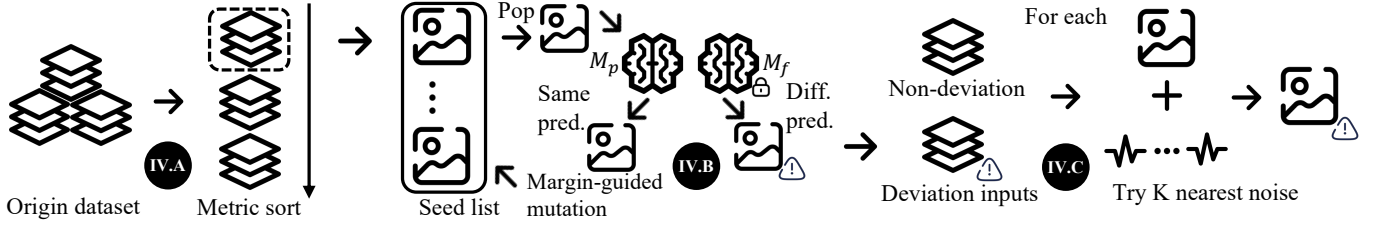


Fig. 2: A high-level workflow of HEDIFF.

---

**Algorithm 1** Margin-Oriented Differential Testing.

---

```

1: function MarginPGD( $M_p, i, \text{Iteration noise bound } \zeta$ )
2:    $i' \leftarrow i, \text{stepSize} \leftarrow \zeta/4$ 
3:    $\text{lossFunc}(x) = -\text{MARGIN}(\text{PREDICTVECTOR}(M_p(x)))$ 
4:   for 1...steps do
5:      $i' \leftarrow i' + \text{CLAMP}(\text{stepSize} * \text{Sgn}(\nabla \text{lossFunc}(i')), \zeta)$ 
6:      $\text{stepSize} \leftarrow \text{stepSize}/2$ 
7:    $\text{noise} \leftarrow i' - i$   $\triangleright$  Obtain added noise from the mutation
8:   return noise
9: function MarginDT( $M_p, M_f, \text{Corpus of seed inputs } \mathcal{S}, \text{Max mutation}$ 
    $\text{times } N_{max}, \text{Iteration noise bound } \zeta, \text{Total noise bound } \epsilon$ )
10:   $\mathcal{Q} \leftarrow \mathcal{S}, \mathcal{D} \leftarrow \emptyset$ 
11:   $\text{cnt}_{mu} \leftarrow 0$ 
12:  while  $\text{cnt}_{mu} < N_{max}$  and  $\mathcal{Q} \neq \emptyset$  do
13:     $i \leftarrow \mathcal{Q}.\text{POP}()$ 
14:     $\text{pred}_p, \text{label}_p \leftarrow \text{PREDICTVECTOR}(M_p(i))$ 
15:     $\text{pred}_f, \text{label}_f \leftarrow \text{PREDICTVECTOR}(M_f(i))$ 
16:    if  $\text{label}_p \neq \text{label}_f$  and  $\text{label}_p = i.\text{label}$  then
17:      Add  $i$  in  $\mathcal{D}$   $\triangleright$  Collect deviation inputs
18:    else
19:       $i' \leftarrow i + \text{MarginPGD}(M_p, i, \zeta)$   $\triangleright$  Margin-oriented mutation
20:       $i' \leftarrow \text{CLAMPNOISE}(i', \epsilon)$   $\triangleright$  Control cumulative noise scale
21:      Add  $i'$  in  $\mathcal{Q}$ 
22:     $\text{cnt}_{mu} \leftarrow \text{cnt}_{mu} + 1$ 
23:  return  $\mathcal{D}$ 

```

---

technical, and empirical perspectives on the difference between AEs and deviation inputs are provided in §VII.

The standard form of PGD seeks to improve the loss function value of specific data on the target model by adding multiple steps of noise along the gradient. Similarly, we treat the loss function as a negative indicator of the margin value and quickly reduce the margin value according to the PGD algorithm with random start. However, if we only modify the loss function, the PGD algorithm will easily fall into localized perturbations, thus failing to reduce the margin value, making this algorithm even less effective than using random mutation. This issue does not occur with the original PGD algorithm because the original loss function does not have a bound and can be continuously increased. However, the minimum value of the margin is 0, and thus it is highly susceptible to falling into perturbations around 0 due to larger step sizes. Therefore, we use an adaptive step size in place of the fixed step in the PGD algorithm, so that we can continuously and rapidly decrease the margin value.

With the support of the margin-oriented PGD algorithm, we now present the complete DT algorithm in Alg. 1. It consists of a building block function *MarginPGD* to mutate the seed inputs towards lower margin values on the plaintext model, and a main function *MarginDT* that employs DT as an oracle to eventually identify deviation inputs  $\mathcal{D}$  from the given seeds  $\mathcal{S}$ .

Specifically, in lines 1-8, *MarginPGD* takes the plaintext model  $M_p$ , a normal input  $i$  and an iteration bound  $\zeta$  as inputs, and returns a *noise* that makes  $i + \text{noise}$  has a lower margin value on the plaintext model. To achieve this, we define our loss function as the negative margin in line 3; when we increase the loss function with PGD, we decrease the margin value of the corresponding data with respect to  $M_p$ . In lines 4-5, we iteratively add noise to  $i'$  in the direction of the gradient of decreasing margin value. CLAMP constrains the noise in the range of  $-\zeta$  to  $\zeta$ ; if a value in the noise is less than  $-\zeta$  (or greater than  $\zeta$ ), it is set to  $-\zeta$  (or  $\zeta$ ). Then, we use an adaptive step size to avoid falling into localized perturbations in line 6.

In *MarginDT*, we first initialize the seed pool  $\mathcal{Q}$  with the filtered seed inputs  $\mathcal{S}$  and the deviation inputs pool  $\mathcal{D}$  as empty in line 10. We then iteratively mutate the seed inputs in  $\mathcal{Q}$  until the number of mutations  $\text{cnt}_{mu}$  reaches a predefined threshold  $N_{max}$  or all seeds in  $\mathcal{Q}$  are mutated to deviation inputs in line 12. For each iteration, we pop a least recently used seed input  $i$  from queue  $\mathcal{Q}$  in line 13, and predict the output vectors of the plaintext model  $M_p$  and FHE-hardened model  $M_f$  in lines 14-15. If the prediction labels of  $M_p$  and  $M_f$  are different and the prediction label of  $M_p$  is the same as the ground truth label of  $i$ , we add  $i$  to the deviation inputs pool  $\mathcal{D}$  in line 16. Otherwise, we use *MarginPGD* to generate a mutated input  $i'$  from  $i$  in line 19. At line 20, CLAMPNOISE constrains the cumulative noise of  $i'$  within the range of  $-\epsilon$  to  $\epsilon$ , where  $\epsilon$  is the total noise bound (note  $\zeta$  is the per iteration noise bound).  $i'$  will be added to the seed pool  $\mathcal{Q}$  in line 21. Finally, we return the deviation inputs pool  $\mathcal{D}$  in line 23.

### C. Generalizing and Leveraging Noise Patterns

As mentioned earlier, we not only identify individual deviation inputs but also try to understand these inputs, specifically whether they exhibit general noise patterns. The generalized patterns can be treated like weak UAPs, enabling further identification of deviation inputs. To this end, we first conduct a theoretical analysis on how the noise pattern of a deviation input could be generalized to other inputs. We then use this understanding to further drive HEDIFF to identify more deviation inputs from the seeds that originally do not produce deviant outputs.

**Theoretical Analysis.** Fig. 3 shows the schematic view of our theoretical analysis. Given an input  $x_2$  that cannot be transformed into a deviation input during differential testing, we can find its nearest input  $x_1$  that can be transformed into a



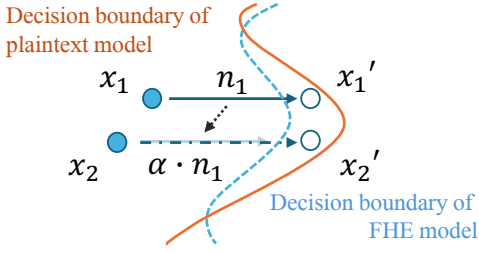


Fig. 3: Illustrating how the noise pattern of a deviation input  $x_1$  can be generalized and used to transform another input  $x_2$  that originally does not produce deviant outputs during DT.

---

**Algorithm 2** Pattern-Based Mutation.

---

```

1: function PatternMutation( $M_p, M_f$ , Corpus of non-deviant seed inputs
    $S_{non}$ , Corpus of deviation inputs  $\mathcal{P}$ , Total noise bound  $\epsilon$ )
2:    $\mathcal{D}_p \leftarrow \emptyset$ 
3:   for  $s_i \in S_{non}$  do
4:      $n_1, n_2, \dots, n_K \leftarrow \text{FINDNEARESTNOISE}(s_i, \mathcal{P})$ 
5:     for  $n_i \in \{n_1, n_2, \dots, n_K\}$  do
6:        $\alpha_i \leftarrow \text{FINDSCALEFACTOR}(M_p, s_i, n_i)$ 
7:       if  $\alpha_i \neq \text{None}$  then
8:          $s_i' \leftarrow s_i + \text{CLAMP}(\alpha_i \cdot n_i, \epsilon)$ 
9:          $pred_p, label_p \leftarrow \text{PREDICTVECTOR}(M_p(s_i'))$ 
10:         $pred_f, label_f \leftarrow \text{PREDICTVECTOR}(M_f(s_i'))$ 
11:        if  $label_p \neq label_f$  and  $label_p = s_i.label$  then
12:          Add  $s_i'$  in  $\mathcal{D}_p$ 
13:        BREAK
14:   return  $\mathcal{D}_p$ 

```

---

deviation input with noise  $n_1$ . As these two points,  $x_1$  and  $x_2$ , are close to each other, perturbed inputs with noise  $n_1$ ,  $x_1 + n_1$  and  $x_2 + n_1$ , should also be similar. As we know  $x_1 + n_1$  is the deviation input, i.e.,  $x_1 + n_1$  is in the gap between the decision boundary of the plaintext model and the FHE-hardened model, as illustrated in Fig. 3. Therefore,  $x_2 + n_1$ , being close to  $x_1 + n_1$ , has a high probability of also being in this gap.

However, if we directly use  $x_2 + n_1$ , this may not result in a deviation input because the decision boundaries of the plaintext and FHE models are non-linear. Therefore, instead of directly applying  $n_1$ , we consider it as a meaningful noise direction, and we need to scale it to make it more effective. With the observation in §III, we can still use the margin value of  $x_2 + \alpha \cdot n_1$  on the plaintext model as a guide for finding an appropriate scale factor, i.e., we can find the  $\alpha$  that makes the margin value of  $x_2 + \alpha \cdot n_1$  on the plaintext model as small as possible while keeping the prediction unchanged.

**Using the Noise Patterns.** As an application of the generalized noise patterns, we further drive HEDIFF to identify more deviation inputs from the seeds  $S_{non} := S - \mathcal{D}$  that originally do not produce deviant outputs during DT in §IV-B. We present our pattern-based mutation algorithm in Alg. 2.

Specifically, for each non-deviant seed  $s_i \in S_{non}$ , we can find the  $K$  nearest deviation inputs with the function FINDNEARESTNOISE in line 4, which computes the L2 distance between  $s_i$  and each deviation input  $x_i$  without noise, and return  $K$  deviation noises  $n_1, n_2, \dots, n_K$  such that  $Sim_{L2}(s_i, x_j)_{j=1 \dots K}$  are the  $K$  lowest. Then, in lines 5-6, for each noise  $n_i$ , we find the corresponding scale factor  $\alpha_i$

using the function FINDSCALEFACTOR. This function tries to find an  $\alpha_i$  that makes the margin value of  $s_i + \alpha_i \cdot N_i$  on the plaintext model low enough while the prediction still equals  $s_i.label$  with binary search. In line 8, after finding a proper  $\alpha_i$ , we constrain each value of noise  $\alpha_i \cdot n_i$  with the function CLAMP to avoid any noise value exceeding the noise bound  $\epsilon$  (like what we did in Alg. 1). Finally, we compare the result of  $s_i + \alpha_i \cdot n_i$  on the FHE-hardened model with  $s_i.label$  to determine whether  $s_i + \alpha_i \cdot n_i$  is a deviation input in lines 9-13. If so, we add it to the deviation inputs pool  $\mathcal{D}_p$  and then break the loop.

## V. IMPLEMENTATION AND SETUP

HEDIFF is written primarily in Python with about 2,000 lines of code. To facilitate the reproducibility of results, we provide the source code and the datasets used in the evaluation at [22]. We will maintain HEDIFF and update it with new features and more documents to benefit future research. HEDIFF does not require any modifications to the models or the HE-ML frameworks, which makes it easy to launch on different settings. There is no special configuration for seed filtering and margin-guided mutation. As for noise patterns, when collecting them, we only use the noise pattern of the nearest data to the input data and do not consider if its label is the same as the input data, for the seek of balancing time consumption and scalability. Details will be discussed further in §VI-C. Below, we present the evaluation setup in line with the statistics in Table IV.

**HE-ML Frameworks and FHE Schemes Tested.** HEDIFF can test various HE-ML models implemented using different HE-ML frameworks. As mentioned earlier in §III, we target three mainstream HE-ML frameworks: TenSEAL, Concrete-ML, and HELayers.

TenSEAL [4] is a library for tensor homomorphic encryption operations based on Microsoft SEAL [45]. Based on the encryption library SEAL, TenSEAL provides vector operation support for BFV [46] and CKKS schemes. CKKS, which supports real-number operations, is commonly considered the most suitable FHE scheme for ML applications. Therefore, we also use this scheme to encrypt the tested ML models. Additionally, we select parameters to support the full depth of multiplications to ensure that models' errors do not arise from decryption failures. To handle the non-linear function Sigmoid, we follow TenSEAL's recommendation [4, 47] to approximate the non-linear function with a polynomial function.

Concrete-ML [5], provided by Zama, a cryptography company focusing on FHE, is built on their FHE compiler Concrete. Concrete-ML supports the TFHE scheme [10]. TFHE is a scheme for computations on integers and provides the Programmable Bootstrapping (PBS) mechanism [48], which reduces noise with Bootstrapping and can compute arbitrary functions using a lookup table. This way, we can avoid manually approximating non-linear activation functions after quantizing the model into an integer model. Instead, we can directly utilize Concrete-ML's functionality to implement activation functions.

HELayers [6], developed by IBM, bridges ML frameworks (like PyTorch [49] and TensorFlow [50]) and basic FHE

TABLE IV: The evaluation setup of HEDIFF and the overall statistics.

Framework	Model	Datasets	Plaintext Accuracy	Encrypted Accuracy	Avg. Inference Time Per Input	#Initial Seeds	#Total Mutations	#Deviation Input After Filtering	#Total Deviation Inputs
TenSEAL	Cryptonets-Square	MNIST	98.37%	97.93%	2.30s	1000	5000	11	277
	Cryptonets-Sigmoid	DIGITS	90.56%	90.56%	1.99s	500	2500	13	90
	MLP-Sigmoid	Credit	71.86%	73.22%	0.32s	1000	5000	287	288
	MLP-AppRELU	Bank	86.20%	85.88%	0.43s	1000	5000	3	454
Concrete-ML	Cryptonets-Square	MNIST	98.50%	98.40%	32.02s	500	2500	55	338
	Cryptonets-Sigmoid	DIGITS	90.56%	90.56%	28.38s	250	1250	3	52
	MLP-Sigmoid	Credit	71.90%	71.90%	19.41s	500	2500	109	337
	MLP-AppRELU	Bank	85.70%	85.80%	17.70s	500	2500	66	298
HElayers	Cryptonets-Square	MNIST	98.66%	98.62%	0.83s	1000	5000	0	2
	Cryptonets-Sigmoid	DIGITS	91.11%	90.28%	0.58s	500	2500	4	78
	MLP-Sigmoid	Credit	71.86%	73.22%	0.13s	1000	5000	287	287
	MLP-AppRELU	Bank	86.20%	86.20%	0.12s	1000	5000	0	6

libraries (like HEaaN [3] and SEAL). Similar to the previous two frameworks, HElayers depends on third-party libraries to provide the FHE functionalities. However, unlike the previous two, HElayers offers a more flexible way to customize the low-level FHE library rather than just using the default one. In the evaluation, we also choose SEAL as our underlying library and test its CKKS scheme. As HElayers does not support automatic conversion for Sigmoid, we use the same method as in TenSEAL to approximate the Sigmoid function.

**Datasets and Models.** We assess HEDIFF across various data types, such as images and tabular data, and choose widely recognized ML tasks, including image classification, credit score prediction, and deposit subscription prediction. Specifically, we chose representative datasets: MNIST [20], DIGITS [21], Credit [19], and Bank [18]. The MNIST dataset is widely used for handwritten digits and consists of 28x28 pixel grayscale images labeled from 0 to 9. The DIGITS dataset is similar to MNIST but with smaller 8x8 pixel images. For these two image datasets, we employ the neural network architecture similar to a popular FHE-friendly model, Cryptonets [17], yet with different activation functions. In particular, for MNIST, we use the popular square activation function in the HE-ML field, and for DIGITS, we use the activation function Sigmoid to investigate the impact of non-linear activation functions.

The Credit dataset is a tabular dataset commonly used for credit score prediction. It comprises various features related to individuals’ credit profiles, such as age, income, and credit history. The task is to predict whether a given individual is likely to have good or bad credit based on these features. The Bank dataset is another tabular dataset used for predicting deposit subscriptions, containing information about bank clients, including their demographic, economic, and banking features. We use similar models for these two tabular datasets. Specifically, we use undersampling to balance these two datasets, and then use a two-layer MLP with Sigmoid activation for the Credit dataset and a two-layer MLP with an approximation of the ReLU activation [51] for the Bank dataset to investigate the impact of training with polynomially approximated non-linear activation functions.

**Clarification on Datasets.** One may wonder about the choice of datasets in our evaluation, as contemporary DNN testing works often use more complex datasets like ImageNet [52] or traffic scenes [53]. We clarify that in practical FHE scenarios, computation over sensitive tabular data is the mainstream. In

contrast, conducting FHE over complex media data like high-resolution images or videos is not practical. Nevertheless, we still use two image datasets (MNIST and DIGITS) to evaluate HEDIFF and show its generality over different data types.

## VI. EVALUATION

Our evaluation follows the setup noted in §V. All experiments are conducted on a server with an AMD Ryzen 3970X CPU and 256GB RAM. We aim to answer the following RQs. **RQ1:** Can HEDIFF effectively find deviation-triggering inputs for HE-ML models? **RQ2:** Does the margin-guided approach outperform the random approach in seed filtering and mutation? **RQ3:** Can noise patterns of the nearest data be effectively used to find new deviation-triggering inputs?

### A. RQ1: Effectiveness of HEDIFF

To answer RQ1, we present the evaluation results of HEDIFF in Table IV with the configuration described above. In the evaluation on TenSEAL and HElayers, for datasets like MNIST, Credit, and Bank, which have more than 3K data points in training dataset, we set the seed number at 1K and the maximum mutation number at 5K. For the small dataset DIGITS, which has less than 2K data points, we set the seed number at 500 and the maximum mutation number at 2.5K. For the extremely time-consuming Concrete-ML, we halve the seed and maximum mutation numbers. For comparison between plaintext and encrypted models, we use the entire test dataset for models on TenSEAL and HElayers, and sample 1K test data for datasets except DIGITS on Concrete-ML.

The subtle differences between the prediction accuracy of plaintext and encrypted models demonstrate the proper configurations of evaluated FHE frameworks, indicating that HE-ML models do not change the functionality of the plaintext model and the deviation inputs found by HEDIFF are not caused by incorrect security parameter configurations. This is also illustrated by Table II, which shows the low ratio of deviation inputs in the original training data.

As shown in Table IV, results suggest that although plaintext and encrypted models have similar prediction accuracy, this similarity does not equate to the robustness of the encrypted models. HEDIFF can still filter and generate a large number of deviation inputs from the seed data for most settings. Cryptonets-Square and MLP-AppRELU are two linear models that do not involve non-linear activation function conversion. This way, the main root causes of deviations are imprecise





Fig. 4: Examples of added noise.

TABLE V: Average L2 and maximum distance of noise. We show the average L2 and maximum distance per pixel for image datasets, and for tabular datasets, we show the average L2 and maximum distance per column. A dash in the table means that no deviation was found during the mutation process.

HE-ML Framework	MNIST		DIGITS		Credit		Bank	
	L2	Max	L2	Max	L2	Max	L2	Max
TenSEAL	0.0008	0.0443	0.0036	0.0465	0.0037	0.0288	0.0025	0.0226
Concrete-ML	0.0004	0.0246	0.0033	0.0472	0.0006	0.0064	0.0011	0.0084
HElayers	0.0002	0.0100	0.0035	0.0468	-	-	0.0022	0.0209

weights in the encrypted models introduced by encoding floats to fixed-point real numbers in CKKS and model quantization in TFHE, along with accumulated noise during encrypted computation. As shown in the results of the table, HElayers performs best, suggesting that HElayers may introduce the least noise in encoding and computation for these two models.

Besides deviations localized by seed filtering, we also observe that HEDIFF can generate many deviation inputs based on non-deviation seeds. An outlier case is the MLP-Sigmoid model on TenSEAL and HElayers, i.e., with CKKS. Although we can localize a high ratio of deviation inputs during seed filtering, nearly no deviation inputs are found in the mutation process. This is likely because, with a rather conservative noise threshold, it is difficult to move these data to the gaps between the decision boundaries of the plaintext and encrypted models.

Although we found hundreds of deviation inputs with HEDIFF, it is worth emphasizing that the noise introduced to deviation inputs is generally mild and does not affect the “visual” content of those deviation-triggering inputs. This is made possible by our cumulative noise control introduced in §IV-B. Such stealthily-tweaked deviation inputs can likely cause high confusion of HE-ML frameworks and applications in practice. Below, we present both quantitative and case studies.

Fig. 4 exhibits two deviation inputs with their original seeds and added noise for MNIST with TenSEAL and DIGITS with HElayers. As can be seen, the deviation inputs manifest high visual similarity with the original inputs, yet these deviation inputs result in different outputs of TenSEAL/HElayers-hardened models. Due to limited space, we present more examples online in our repository [22]. Moreover, we quantify the scale of noise generated by HEDIFF in Table V. Each column of data has two values, representing the average L2 distance between the original seed and generated deviation inputs, and the average maximum noise value, respectively. Overall, we interpret the average L2 and maximum distances are being reasonably low, indicating that the produced deviation inputs are close to normal inputs across different models and data types.

**Answer to RQ1:** Although FHE-hardened models show satisfactory performance with a negligible accuracy downgrade and within a reasonable testing time, HEDIFF effectively finds plenty of deviation inputs that may mislead and confuse users across nearly all scenarios.

### B. RQ2: Effectiveness of Margin’s Guidance

In §IV, we proposed two margin-guided approaches for seed filtering and mutation, and a method to generalize the noise pattern of deviation inputs. In this section, to answer RQ2, we investigate the effectiveness of two margin-guided approaches across different datasets and HE-ML frameworks. We will discuss the generalization of noise patterns in §VI-C. Specifically, we compare the margin-guided approaches with the random approaches in seed filtering and mutation. In the mutation process, we further consider the standard form of the PGD algorithm implemented by Torchattacks [54] as another stronger baseline. We set the seed number at 200 (100) and the mutation number at 1000 (300) for datasets on TenSEAL and HElayers (Concrete-ML), with results reported in Table VI.

The results in Table VI show that margin-guided filtering can localize many more deviation inputs in seeds in comparison to random filtering. From another perspective, margin-guided mutation can find more deviation inputs than random mutation under either margin-guided or random filtering strategies. While PGD mutation finds more deviation inputs than random mutation, margin-guided mutation outperforms PGD in most cases, especially when synergized with margin-guided seed filtering. We also report that while random mutations are slightly faster than the other two mutation strategies, PGD mutation and margin-guided mutation take about the same time to find deviation inputs. To clarify, the computation of these mutation strategies only differ in plaintext models, while the main time consumption comes from the encrypted inference of the FHE-hardened models. In other words, the choice of different mutation strategies has only a negligible impact on the speed of our testing. Overall, we recommend using margin-guided seed filtering and mutation together, which can presumably yield the most effective testing results in a satisfactory time frame.

**Answer to RQ2:** Compared with the random approach and PGD mutation, HEDIFF’s margin-guided design significantly contributes to its effectiveness and can substantially help users uncover more potential deviation inputs in FHE-hardened models.

### C. RQ3: Characteristics of Noise Patterns

In this section, we answer RQ3 by starting with validating the assumption that close data will also have similarity in the direction of the noise pattern, as illustrated in Fig. 3. Specifically, we choose the TenSEAL framework as our testbed and compare the approach of choosing  $K$  nearest data, i.e., the approach used in HEDIFF, with other three approaches: selecting  $K$  random data and selecting  $K$  nearest/random data with the same label. For each approach, denoted as  $A$ ,

TABLE VI: Filtering and Mutation Strategy Effectiveness. Random Filter (Margin Filter) refers to random (margin-guided) filtering, and Random (PGD, Margin) refers to random (PGD, margin-guided) mutation. Numbers in the O. columns denote #deviation inputs found during seed filtering, and numbers in the T. columns denote #deviation inputs found after mutation. We mark the largest #deviation inputs using **blue** and **yellow** under random filter and margin filter, respectively.

HE-ML Framework	Datasets	Random Filter						Margin Filter					
		Random		PGD		Margin		Random		PGD		Margin	
		O.	T.	O.	T.	O.	T.	O.	T.	O.	T.	O.	T.
TenSEAL	MNIST	0	7	1	6	1	3	4	6	3	16	5	88
	DIGITS	1	7	2	7	0	15	6	9	5	39	6	66
	Credit	8	22	2	16	8	41	69	70	69	69	69	70
	Bank	1	3	0	0	0	31	3	12	3	3	3	135
Concrete-ML	MNIST	1	1	0	2	0	1	27	56	30	39	31	79
	DIGITS	0	0	0	0	1	3	3	3	2	7	3	29
	Credit	0	2	0	18	2	15	40	61	40	40	41	69
	Bank	3	5	3	7	0	5	42	55	41	42	41	68
HElayers	MNIST	0	0	0	0	0	1	1	1	0	0	0	0
	DIGITS	1	1	0	6	0	12	4	11	4	35	4	70
	Credit	9	21	9	19	7	52	69	69	69	69	69	69
	Bank	0	0	0	0	0	0	0	0	0	0	0	0

TABLE VII: Noise Pattern Similarity.  $TopK(Random)_{w(w/o)}$  means the similarity of K nearest(random) data with(without) the same label consideration.

Metric	$K_{max}$	MNIST	DIGITS	Bank
$TopK_w$	1	0.6731	0.8861	0.9831
	3	0.6260	0.8377	0.9730
	5	0.5941	0.8090	0.9664
$TopK_{w/o}$	1	<b>0.6976</b>	<b>0.9022</b>	<b>0.9836</b>
	3	0.6604	0.8779	0.9741
	5	0.6371	0.8594	0.9680
$Random_w$	1	0.4260	0.7527	0.7655
	3	0.4346	0.7410	0.7662
	5	0.4360	0.7497	0.7647
$Random_{w/o}$	1	0.3588	0.6993	0.7651
	3	0.3599	0.6791	0.7620
	5	0.3739	0.6847	0.7620

we measure its noise pattern direction similarity performance  $Sim_A$  using the method described below.

Given a set of deviation inputs  $\mathcal{D}$ , for each deviation inputs  $x_d + n_d \in \mathcal{D}$ , we choose  $K$  deviation inputs  $x_1 + n_1, \dots, x_K + n_K \in \mathcal{D} \setminus \{x_d + n_d\}$  using approach A, and calculate the average cosine similarity between  $n_1 \sim n_K$  and  $n_d$  as  $Sim_d$ . Then,  $Sim_A$  is defined as the average of  $Sim_d$  for all deviation inputs. This can be formulated as:

$$Sim_A = \frac{1}{|\mathcal{D}|} \sum_{x_d + n_d \in \mathcal{D}} \frac{1}{K} \sum_{i=1}^K \frac{n_d \cdot n_i}{\|n_d\| \cdot \|n_i\|} \quad (6)$$

We compare the performance of  $K$  nearest data with and without same label and random  $K$  data with and without same label, and the results are shown in Table VII. We choose  $K = \{1, 3, 5\}$  in the experiment.

Table VII shows that noise patterns of closer data are more likely to be similar. This is consistent with our intuition in Fig. 3. Also, we find that the similarity when  $K = 1$  is higher than when  $K = 3$  or 5, meaning that the noise pattern of the nearest data is also the most similar among the noise patterns

TABLE VIII: Noise Pattern Effectiveness. Numbers in the #Dev. and #Q. columns mean #deviation inputs found and #total queries to FHE-hardened models, respectively.

Metric	$K_{max}$	MNIST		DIGITS		Bank	
		#Dev.	#Q.	#Dev.	#Q.	#Dev.	#Q.
$TopK_w$	1	13	62	6	28	81	522
	3	13	100	6	50	83	1393
	5	14	114	6	62	85	1979
$TopK_{w/o}$	1	11	57	4	26	81	513
	3	11	96	6	51	85	1212
	5	11	112	6	58	87	1571
$Random_w$	1	7	45	4	24	58	504
	3	7	80	5	44	66	1374
	5	8	89	5	55	67	2032
$Random_{w/o}$	1	3	35	4	15	56	471
	3	3	65	4	30	60	1124
	5	3	75	4	35	61	1451

of  $K$  nearest data. Moreover, when comparing data with and without the same label ( $TopK_w$  and  $TopK_{w/o}$ ), we find that the similarity of noise patterns is not strongly affected by the label of the data, suggesting that the noise pattern is more likely to be determined by the data itself rather than the label.

We further compare the effectiveness of finding new noise patterns with the approach described in §IV-C using noise patterns from  $K$  nearest/random data with/without the same label. The results are shown in Table VIII. Both  $TopK_{w/o}$  and  $TopK_w$  perform similarly well in finding new deviation inputs compared to random approaches, which is consistent with the results in Table VII. Considering that  $TopK_{w/o}$  has the best similarity performance when  $K = 1$  and good performance in finding new deviation inputs when  $K$  is higher, we choose  $TopK_{w/o}$  with  $K = 1$  as the noise pattern leveraging strategy to make a trade-off between query times and scalability. In other words, we use a small  $K$  to save on query times in the default setting and use  $TopK_{w/o}$  to keep the potential of finding more deviation inputs with fewer queries as  $K$  increases.

**Answer to RQ3:** We show that normal input data with a close distance are very likely to be transformed into deviation inputs with similar noise patterns. This further allows HEDIFF to mutate inputs in a directed manner, improving its effectiveness in finding more deviation inputs.

## VII. DISCUSSION

**Deviation Inputs vs. AEs.** Deviation inputs can mislead HE-ML models from making correct predictions. Although similar to adversarial examples (AEs), we clarify that they are *distinct* from ML AEs found in prior works [14, 15].

Conceptually, conventional AEs specifically alter the predictions of plaintext DNN models only. In contrast, according to our definition in § III-A, deviation inputs cause the FHE-hardened model and its plaintext version to produce different predictions—typically, the plaintext version’s prediction is correct while the FHE-hardened version’s prediction is incorrect. In addition, conventional AEs are pervasive in DNNs and are believed to stem from inadequate training and unsmooth classification boundaries [55]. Differently, HEDIFF’s findings are specific to FHE frameworks, and detected deviation inputs are

primarily caused by the subtle differences between classification boundaries of FHE-hardened and plaintext models.

Technically, HEDIFF uncovers inputs that trigger distinct predictions in HE-ML models compared to plaintext models. The deviation inputs retain the *same prediction labels* in the plaintext models but alter the predictions in the HE-ML models. Due to the extreme time-consuming of FHE computations, HEDIFF considers plaintext models as “mirrors” and uses information provided by plaintext models to guide the search for deviation inputs. In contrast, AEs typically change plaintext model predictions and are detected mainly with the information from the plaintext model itself.

Empirically, while the community is alert to the risks from AEs, we lack proper attention to potential deviations caused by FHE. The gap between regular and FHE-hardened models can mislead developers about their robustness, since a model that performs well with plaintext may not maintain the same level of robustness when adapted for FHE. Such a misunderstanding can potentially introduce security risks in financial, medical, and other sensitive areas.

**Bug Localization and Mitigation.** We envision the potential feasibility of localizing certain root causes of the detected deviations. As analyzed in §II-A, the HE-ML models may yield deviant outputs due to approximations in the non-linear functions and the conversion from floating-point to fixed-point numbers or integers. This way, it may become possible to localize the root neurons or layers in HE-ML models. Moreover, with those localized root neurons/layers, we can fine-tune them to increase accuracy and robustness. We deem this interesting yet challenging for future work, primarily due to 1) the contemporary HE-ML frameworks not offering fine-grained control (e.g., neuron-level) over the FHE approximations, and 2) as a tradeoff, increasing accuracy and robustness may likely lead to a decrease in the inference speed. We advocate for the SE and HE-ML community to explore this direction based on the insights from HEDIFF.

**Alternative Testing Feedback.** In this paper, HEDIFF gradually finds inputs to decrease the logits margin until the deviation is large enough to flip model predictions. This design of testing feedback abstracts the HE-ML model internals (e.g., orthogonal to the types of approximations employed for the non-linear functions), making the testing pipeline of HEDIFF more general and applicable to various HE-ML settings. We also notice prior DNN testing works [56, 57] that leverage the model internals (layer-wise or neuron-wise outputs) to guide input mutation. While this may offer finer-grained feedback (e.g., focusing on the most critical neurons), speed is the primary concern when applying such methods to test FHE-hardened models. Hooking into model internals (e.g., every neuron) incurs unacceptable overhead, particularly in the HE-ML setting, where substantial computation resources are used for encrypted computations. We believe that the feedback currently provided on model outputs is *sufficient* and consider it an interesting area for future work to explore finer-grained testing feedback.

## VIII. RELATED WORK

**Privacy-preserving machine learning.** Besides HE-ML, many other approaches have emerged to achieve privacy-preserving machine learning. Secure multiparty computation (MPC) [58, 59] is one of the promising solutions in which multiple non-colluding parties privately complete computation without revealing individual components. Early works have explored MPC-based linear regression, logistic regression, and neural networks [60–62]. Other existing works leverage federated learning techniques to safely aggregate gradients from multiple users [63–65]. Furthermore, recent research has also studied using trusted hardware environments like ARM TrustZone and Intel SGX to protect ML models [66, 67].

**Testing Privacy-enhancing Technologies.** Given the prosperous development of privacy-enhancing technologies, the task of checking the correctness of these privacy-related algorithms and protocols is demanding. Recent works propose the use of symbolic execution, constraint solving, and fuzzing to test and verify differential privacy (DP) programs [68–71]. Besides DP, MPC compilers and their machine learning applications have also been tested by the community [72, 73]. Zero-knowledge Proof (ZKP) compilers and applications, due to their industry adoption in second-level blockchains [74], are receiving extensive attention for testing and verifying their correctness [75–78]. In contrast, this work initializes the first effort to test and understand defects in HE-ML.

**Testing Machine Learning Systems.** Deep learning has become the crux of many real-world applications, and the reliability of deep learning systems has attracted increasing attention. Some existing works have explored testing deep learning models with the idea of coverage from traditional software testing [56, 79–81]. Moreover, some works have studied the correctness of deep learning compilers and libraries [82–85]. However, the testing of HE-ML has not been well studied. This work fills this gap by proposing a novel testing approach to efficiently detect defects in HE-ML systems.

## IX. CONCLUSION

We present HEDIFF, a margin-guided DT framework for HE-ML models. HEDIFF successfully uncovered thousands of inputs that cause deviant outputs in commonly used HE-ML models. We show that the uncovered defects can cause great confusion in the daily usage of HE-ML models, and the noise patterns generated by margin-oriented mutation have the potential to be generalized to find more deviations. We conclude by discussing the applications of HEDIFF and possible mitigation strategies for the uncovered defects.

## ACKNOWLEDGEMENT

We thank the anonymous reviewers for their valuable feedback. We also thank the developers of the HE-ML projects we used in our evaluation. The HKUST authors were supported in part by a RGC GRF grant under the contract 16214723, RGC CRF grant under the contract C6015-23G, and research fund provided by HSBC.



## REFERENCES

- [1] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.
- [2] A. Silverberg, "Fully homomorphic encryption for mathematicians," *Women in Numbers 2: Research Directions in Number Theory*, vol. 606, p. 111, 2013.
- [3] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology—ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3–7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.
- [4] A. Benaïssa, B. Retiat, B. Cebere, and A. E. Belfedhal, "TenSEAL: A library for encrypted tensor operations using homomorphic encryption," 2021.
- [5] Zama, "Concrete ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists," 2022, <https://github.com/zama-ai/concrete-ml>.
- [6] E. Aharoni, A. Adir, M. Baruch, N. Drucker, G. Ezov, A. Farkash, L. Greenberg, R. Masalha, G. Moshkovich, D. Murik, H. Shaul, and O. Soceanu, "HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data," *Privacy Enhancing Technology Symposium (PETs) 2023*, 2023.
- [7] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "ngraph-he2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of the 7th ACM workshop on encrypted computing & applied homomorphic cryptography*, 2019, pp. 45–56.
- [8] W. Ao and V. N. Boddeti, "AutoFHE: Automated adaption of CNNs for efficient evaluation over FHE," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 2173–2190.
- [9] R. Ran, X. Luo, W. Wang, T. Liu, G. Quan, X. Xu, C. Ding, and W. Wen, "Spencnn: orchestrating encoding and sparsity for fast homomorphically encrypted neural network inference," in *International Conference on Machine Learning*. PMLR, 2023, pp. 28 718–28 728.
- [10] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.
- [11] L. Bergerat, A. Boudi, Q. Bourgerie, I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, "Parameter optimization and larger precision for (t) fhe," *Journal of Cryptology*, vol. 36, no. 3, p. 28, 2023.
- [12] E. Aharoni, N. Drucker, G. Ezov, H. Shaul, and O. Soceanu, "Complex encoded tile tensors: Accelerating encrypted analytics," *IEEE Security & Privacy*, vol. 20, no. 5, pp. 35–43, 2022.
- [13] W. M. McKeeman, "Differential testing for software," *Digital Technical Journal*, vol. 10, no. 1, pp. 100–107, 1998.
- [14] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.
- [15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.
- [16] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765–1773.
- [17] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International conference on machine learning*. PMLR, 2016, pp. 201–210.
- [18] S. Moro, P. Cortez, and P. Rita, "A data-driven approach to predict the success of bank telemarketing," *Decision Support Systems*, vol. 62, pp. 22–31, 2014.
- [19] I.-C. Yeh and C.-h. Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert systems with applications*, vol. 36, no. 2, pp. 2473–2480, 2009.
- [20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [21] A. Asuncion, "Uci machine learning repository, university of california, irvine, school of information and computer sciences," <http://www.ics.uci.edu/~mllearn/MLRepository.html>, 2007.
- [22] "Research artifact," <https://sites.google.com/view/hediff>.
- [23] D. J. Wu, "Fully homomorphic encryption: Cryptography's holy grail," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 21, no. 3, pp. 24–29, 2015.
- [24] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.
- [25] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.
- [26] A. Stoian, J. Frery, R. Bredehoft, L. Montero, C. Kherfallah, and B. Chevallier-Mames, "Deep neural networks for encrypted inference with tfhe," in *Cyber Security, Cryptology, and Machine Learning*, S. Dolev, E. Gudes, and P. Paillier, Eds. Cham: Springer Nature Switzerland, 2023, pp. 493–500.
- [27] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full rms variant of approximate homomorphic encryption," in *Selected Areas in Cryptography—SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*. Springer, 2019, pp. 347–368.
- [28] A. Al Badawi, C. Jin, J. Lin, C. F. Mun, S. J. Jie, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, "Towards the alexnet moment for homomorphic encryption: Hcnn, the first homomorphic cnn on encrypted data with gpus," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1330–1343, 2020.
- [29] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *International Conference on Machine Learning*. PMLR, 2019, pp. 812–821.
- [30] W. Z. Srinivasan, P. Akshayaram, and P. R. Ada, "Delphi: A cryptographic inference service for neural networks," in *Proc. 29th USENIX Secur. Symp.*, 2019, pp. 2505–2522.
- [31] AWS, "Cryptographic computing - amazon web services (aws)," <https://aws.amazon.com/security/cryptographic-computing/>, 2024.
- [32] M. Guevara, "Expanding our fully homomorphic encryption offering - google developers blog," <https://developers.googleblog.com/en/expanding-our-fully-homomorphic-encryption-offering/>, 2023.
- [33] Duality, "Duality technologies - secure data collaboration products," <https://dualitytech.com/>, 2023.
- [34] H. N. Security, "Google leverages open-source fully homomorphic encryption library - help net security," <https://www.helpnetsecurity.com/2022/09/15/duality-technologies-google-fhe-transpiler/>, 2022.
- [35] Duality, "Our partnership with aws brings together the power of our innovative duality platform and the global reach and scalability of aws, providing our customers with seamless access to our solutions and services," <https://dualitytech.com/partners/aws/>, 2023.
- [36] C. Byrne, "New privacy-enhancing technology revolutionizing healthcare - digital cxo," <https://digitalcxo.com/article/new-privacy-enhancing-technology-revolutionizing-healthcare/>, 2023.
- [37] C. D. Magazine, "Why tackling financial crime calls for a privacy-first approach - cyber defense magazine," <https://www.cyberdefensemagazine.com/why-tackling-financial-crime-calls-for-a-privacy-first-approach/>, 2023.
- [38] T. W. S. Journal, "Banks start using information-sharing tools to detect financial crime," <https://www.wsj.com/articles/banks-start-using-information-sharing-tools-to-detect-financial-crime-11658741402>, 2022.
- [39] A.-T. Insight, "Protecting data privacy in the fight against financial crime - a team," <https://a-teaminsight.com/blog/protecting-data-privacy-in-the-fight-against-financial-crime/?brand=ati>, 2022.
- [40] G. T. Review, "Exclusive: Duality technologies enters trade finance fraud prevention," <https://www.gtreview.com/news/fintech/exclusive-duality-technologies-enters-trade-finance-fraud-prevention/>, 2022.
- [41] A. Authority, "Darpa awards duality technologies multimillion dollar contract to accelerate machine learning on encrypted data," <https://authority.com/machine-learning/darpa-awards-duality-technologies-multimillion-dollar-contract-to-accelerate-machine-learning-on-encrypted-data/>, 2023.
- [42] W. Examiner, "Defense officials eye anti-quantum encryption to shore up protection of classified material - washington examiner," <https://www.washingtonexaminer.com/news/909771/defense-officials-eye-anti-quantum-encryption-to-shore-up-protection-of-classified-material/>, 2023.
- [43] D. Gaikwad, "Homomorphic encryption market is predicted to witness over 9% cagr by 2030," <https://www.linkedin.com/pulse/homomorphic-encryption-market-predicted-witness-over-9-gaikwad>, 2023.

- [44] G. Elsayed, D. Krishnan, H. Mobahi, K. Regan, and S. Bengio, "Large margin deep networks for classification," *Advances in neural information processing systems*, vol. 31, 2018.
- [45] "Microsoft SEAL (release 4.1)," <https://github.com/Microsoft/SEAL>, Jan. 2023, microsoft Research, Redmond, WA.
- [46] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.
- [47] H. Chen, R. Gilad-Bachrach, K. Han, Z. Huang, A. Jalali, K. Laine, and K. Lauter, "Logistic regression over encrypted data from fully homomorphic encryption," *BMC medical genomics*, vol. 11, pp. 3–12, 2018.
- [48] I. Chillotti, M. Joye, and P. Paillier, "Programmable bootstrapping enables efficient homomorphic inference of deep neural networks," in *Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5*. Springer, 2021, pp. 1–19.
- [49] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.
- [50] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., "TensorFlow: a system for Large-Scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.
- [51] R. E. Ali, J. So, and A. S. Avestimehr, "On polynomial approximations for privacy-preserving and verifiable relu networks," *arXiv preprint arXiv:2011.05530*, 2020.
- [52] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.
- [53] D. Seita, "Bdd100k: A large-scale diverse driving video database," *The Berkeley Artificial Intelligence Research Blog. Version*, vol. 511, p. 41, 2018.
- [54] H. Kim, "Torchattacks: A pytorch repository for adversarial attacks," *arXiv preprint arXiv:2010.01950*, 2020.
- [55] K. Leino, Z. Wang, and M. Fredrikson, "Globally-robust neural networks," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6212–6222.
- [56] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 1–18.
- [57] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 146–157.
- [58] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with honest majority," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 307–328.
- [59] A. Wigderson, M. Or, and S. Goldwasser, "Completeness theorems for noncryptographic fault-tolerant distributed computations," in *Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC'88)*, 1988, pp. 1–10.
- [60] Q. Jia, L. Guo, Z. Jin, and Y. Fang, "Preserving model privacy for machine learning in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 8, pp. 1808–1822, 2018.
- [61] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 19–38.
- [62] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *2013 IEEE symposium on security and privacy*. IEEE, 2013, pp. 334–348.
- [63] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.
- [64] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *arXiv preprint arXiv:1711.10677*, 2017.
- [65] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.
- [66] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi, "Darknetz: towards model privacy at the edge using trusted execution environments," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 161–174.
- [67] L. Hanzlik, Y. Zhang, K. Grosse, A. Salem, M. Augustin, M. Backes, and M. Fritz, "Mlcapsule: Guarded offline deployment of machine learning as a service," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3300–3309.
- [68] Z. Ding, Y. Wang, G. Wang, D. Zhang, and D. Kifer, "Detecting violations of differential privacy," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 475–489.
- [69] Y. Wang, Z. Ding, D. Kifer, and D. Zhang, "Checkdp: An automated and integrated approach for proving differential privacy or finding precise counterexamples," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 919–938.
- [70] H. Zhang, E. Roth, A. Haeberlen, B. C. Pierce, and A. Roth, "Testing differential privacy with dual interpreters," *arXiv preprint arXiv:2010.04126*, 2020.
- [71] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1895–1912.
- [72] Q. Pang, Y. Yuan, and S. Wang, "Mpcdiff: Testing and repairing mpc-hardened deep learning models." NDSS, 2024.
- [73] Y. Li, D. Xiao, Z. Liu, Q. Pang, and S. Wang, "Metamorphic testing of secure multi-party computation (mpc) compilers." ESEC/FSE, 2024.
- [74] R. Yarlykov, "Solving scalability with zkSync: A blockchain review," <https://metalamp.io/magazine/article/how-zkp-and-zk-rollups-help-solve-the-scalability-problem-a-review-of-the-zkSync-blockchain>, 2024.
- [75] S. Pailoor, Y. Chen, F. Wang, C. Rodríguez, J. Van Geffen, J. Morton, M. Chu, B. Gu, Y. Feng, and I. Dillig, "Automated detection of under-constrained circuits in zero-knowledge proofs," *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 1510–1532, 2023.
- [76] H. Wen, J. Stephens, Y. Chen, K. Ferles, S. Pailoor, K. Charbonnet, I. Dillig, and Y. Feng, "Practical security analysis of Zero-Knowledge proof circuits," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 1471–1487.
- [77] J. Liu, I. Kretz, H. Liu, B. Tan, J. Wang, Y. Sun, L. Pearson, A. Miltner, I. Dillig, and Y. Feng, "Certifying zero-knowledge circuits with refinement types," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 1741–1759.
- [78] D. Xiao, Z. Liu, Y. Peng, and S. Wang, "Mtzk: Testing and exploring bugs in zero-knowledge (zk) compilers," in NDSS, 2025.
- [79] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1039–1049.
- [80] Z. Ji, P. Ma, Y. Yuan, and S. Wang, "Cc: Causality-aware coverage criterion for deep neural networks," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1788–1800.
- [81] Y. Yuan, Q. Pang, and S. Wang, "Revisiting neuron coverage for dnn testing: A layer-wise and distribution-aware criterion," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1200–1212.
- [82] D. Xiao, Z. Liu, Y. Yuan, Q. Pang, and S. Wang, "Metamorphic testing of deep learning compilers," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 1, pp. 1–28, 2022.
- [83] Q. Shen, H. Ma, J. Chen, Y. Tian, S.-C. Cheung, and X. Chen, "A comprehensive study of deep learning compiler bugs," in *Proceedings of the 29th ACM Joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2021, pp. 968–980.
- [84] H. V. Pham, T. Lutellier, W. Qi, and L. Tan, "Cradle: cross-backend validation to detect and localize bugs in deep learning libraries," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1027–1038.
- [85] H. Ma, Q. Shen, Y. Tian, J. Chen, and S.-C. Cheung, "Fuzzing deep learning compilers with hirgen," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 248–260.