AdvHunter: Detecting Adversarial Perturbations in Black-Box Neural Networks through Hardware Performance Counters

Manaar Alam
New York University Abu Dhabi
United Arab Emirates
alam.manaar@nyu.edu

Michail Maniatakos
New York University Abu Dhabi
United Arab Emirates
michail.maniatakos@nyu.edu

ABSTRACT

The paper introduces AdvHunter, a novel strategy to detect *adversarial examples* (AEs) in Deep Neural Networks (DNNs). AdvHunter operates effectively in practical *black-box* scenarios, where only *hard-label* query access is available, a situation often encountered with proprietary DNNs. This differentiates it from existing defenses, which usually rely on white-box access or need to be integrated during the training phase - requirements often not feasible with proprietary DNNs. AdvHunter functions by monitoring data flow dynamics within the computational environment during the inference phase of DNNs. It utilizes *Hardware Performance Counters* to monitor microarchitectural activities and employs principles of *Gaussian Mixture Models* to detect AEs. Extensive evaluation across various datasets, DNN architectures, and adversarial perturbations demonstrate the effectiveness of AdvHunter.

1 INTRODUCTION

Due to the widespread usage of Deep Neural Networks (DNNs) and their critical role in various domains, security analysis of DNNs has become imperative. This paper focuses on a specific security vulnerability in DNNs known as *adversarial examples* (AEs) [5]: an inference-time attack where an adversary carefully introduces subtle perturbations to input data, leading DNNs to produce incorrect predictions. These perturbations, often imperceptible to the human eye, can significantly degrade the performance of DNNs, leading to misclassifications and potential security breaches. The growing threats of AEs have led to the development of numerous defenses. While techniques like *adversarial training* can strengthen DNNs against specific types of AEs [15], they often leave DNNs vulnerable to unknown attacks [6]. Some effective defenses in *white-box* scenarios involve monitoring intermediate features of DNNs [17], but these are impractical for proprietary DNNs where model internals are unavailable. Defenses in *soft-label* black-box settings, which examine prediction confidence scores [2], face risks such as *model-stealing attacks* [14]. `EMShepherd` stands out as the sole solution for detecting AEs in *hard-label* black-box settings by using electromagnetic emissions during DNN inference [7], but it is limited to physically accessible DNN deployments and requires specialized equipment. To overcome these limitations, we introduce AdvHunter, a novel strategy to detect AEs by analyzing data flow dynamics during DNN inference in a hard-label black-box setting.

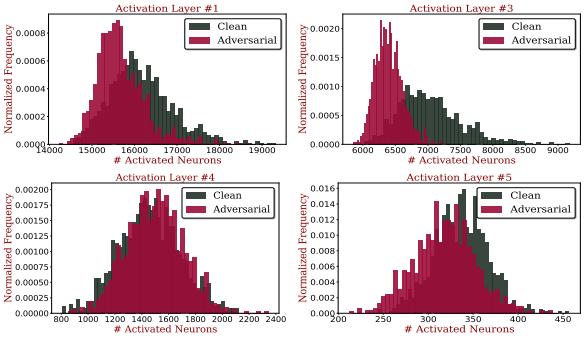

Figure 1: Distributions of *Activated Neurons* at different activation layers for clean and adversarially perturbed inputs.

***Adversarial Impact on Neuron Activations*:** AEs manipulate DNNs to produce incorrect outputs by steering neuron activations toward a *specific pattern* uncharacteristic of the predicted category. This behavior deviates from the processing of clean inputs, where neuron activations follow a *more generalized pattern* for a particular category. We present a case study using a Convolutional Neural Network (CNN) with four convolutional layers and two fully connected layers, each paired with ReLU activation except the last layer. The CNN is trained on the CIFAR-10 dataset. We analyze normalized frequency distributions of *activated neurons* in the CNN across two batches of 1000 randomly selected inputs, as depicted in Figure 1. Without loss of generality, we consider that the first batch contains clean inputs from the 'bird' category. In contrast, the second batch contains inputs from other categories, adversarially perturbed using the *Fast Gradient Sign Method* (FGSM) [9] with an attack strength of 0.1 to misclassify them as 'bird'. For conciseness, Figure 1 focuses on distributions for the first and final three activation layers. It can be noticed that in Activation Layer #3, there are clear differences in the distribution of activated neurons, though this distinction is less pronounced in other activation layers. It is evident that adversarially perturbed inputs causing misclassifications result in significantly different neuron activations compared to clean inputs for the same prediction category.

***Motivation*:** In a black-box scenario, a defender lacks visibility into internal neuron activations. However, when a DNN processes an input with adversarial perturbation, distinctive patterns of neuron activations lead to a unique data flow sequence during computations compared to clean inputs. *In this paper, we utilize the fact and approach AEs detection from a novel perspective, focusing on the effect of data flow dynamics in a computational environment during the inference phase of a DNN.* We observe that this unique data

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655682

flow sequence induces discernible *microarchitectural activities* (e.g., cache operations, branching instructions, etc.) for AEs compared to clean inputs. In this paper, we aim to monitor this side-channel information from microarchitectural activities to detect when a DNN processes inputs embedded with adversarial perturbations. We use *Hardware Performance Counters* (HPCs), present in most modern processors, to monitor these microarchitectural activities. We use HPC data and principles of the *Gaussian Mixture Model* to detect AEs. HPCs have previously been used to reverse-engineer a DNN [1]. However, to the best of our knowledge, HPCs have never been used as a defense to detect AEs in DNNs. Moreover, the widespread availability of HPCs across computing infrastructures, ranging from IoT devices to cloud environments, amplifies the usability of the method across a broad spectrum of use cases.

***Contributions*:** The contributions of the paper are as follows:

- We introduce AdvHunter, a novel strategy to identify AEs in DNNs operating in a hard-label black-box setting, which requires no prior knowledge of the type of adversarial perturbations.
- We utilize HPCs to analyze variations in microarchitectural activities caused by data flow dynamics during DNN inference to identify potential AEs.
- Unlike `EMShepherd` [7], AdvHunter does not require physical access to DNNs or sophisticated equipment. The easy accessibility of HPCs in various applications, including cloud-based remote environments, makes it a more flexible and practical choice.
- All implementations of AdvHunter are open-sourced online at: https://github.com/momalab/AdvHunter. We evaluate the efficacy of AdvHunter using both targeted and untargeted variants of three distinct attack strategies: FGSM [9], PGD [8], and DeepFool [13]; three benchmark image classification datasets: FashionMNIST, CIFAR-10, and GTSRB; and three standard CNN architectures: EfficientNet, ResNet18, and GoogleNet.

2 RELATED WORK

The increasing threats of AEs have led to the development of numerous defenses. Adversarial training, for instance, enhances DNN robustness against AEs [10]. However, this often focuses on specific types of AEs, leaving vulnerabilities to other, unknown attacks [6]. Another line of defense involves detecting AEs during model execution by monitoring intermediate execution features of DNNs [17]. These methods are particularly effective in *white-box* scenarios, where the defender has full access to DNN internals. This assumption is often impractical, especially for proprietary DNNs, where model parameters are considered intellectual property. In *black-box* scenarios, where DNN internals are inaccessible, existing methods typically analyze confidence scores from inference queries to identify AEs [2]. While effective, a primary limitation of these approaches is their reliance on *soft-label* black-box access to a DNN, where each prediction is associated with a confidence score. However, several studies have highlighted that such soft-label access makes a DNN an easy target for *model-stealing attacks* [14], a significant concern for vendors providing proprietary DNN models. To prevent this, vendors often obfuscate prediction confidence or provide only hard-label access [11], where predictions are given without revealing any confidence, making existing black-box defenses that rely on prediction confidence ineffective. Defenses based on analyzing input statistical properties to identify AEs [18] have been demonstrated to be ineffective in real-world settings due to challenges in generalizing to new attack types [16]. Additionally, ensemble-based approaches for identifying AEs face practical deployment challenges due to their high computational demands [4].

***Related Work in a Hard-Label Black-Box Scenario*:** To the best of our knowledge, `EMShepherd` [7] stands out as the only available solution for detecting AEs where DNNs operate in a hard-label black-box setting. It utilizes electromagnetic (EM) side-channel emissions during the inference phase, making it applicable when internal access to a DNN is restricted. However, it is important to highlight that `EMShepherd` is predominantly suited to scenarios where DNNs are deployed on devices that are physically accessible to defenders. Moreover, defenders require sophisticated equipment to measure EM emissions precisely. In addition, the framework does not extend its capabilities to scenarios where DNNs are remotely accessed, such as in cloud-based machine learning services. These limitations restrict the usability of `EMShepherd` in a wide range of practical applications.

3 PRELIMINARIES

***Hardware Performance Counters (HPCs)*:** HPCs are special-purpose registers present in most modern processors [3]. They are designed to monitor specific microarchitectural events, ranging from cache misses and branch mispredictions to retired instructions. HPCs are crucial to understanding hardware utilization, facilitating code optimization, and process tuning operations. Most modern processors support thousands of HPC events, but due to a limited number of built-in HPC registers, only a select few can be observed simultaneously. On Linux kernels with version 2.6.31 and above, the `perf` tool provides access to these counters with high granularity.

***Gaussian Mixture Model (GMM)*:** GMM is a probabilistic model that assumes data points are generated from a mixture of several Gaussian distributions with unknown parameters [12]. It is a method to represent complex, multimodal data distributions. Given a set of observations $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$, where $x_i \in \mathbb{R}^d$, the objective of GMM is to fit $\mathcal{X}$ as a mixture of $\mathcal{K}$ Gaussian distributions. The likelihood of data under this model is given by:

$$p(x_i) = \sum_{k=1}^{\mathcal{K}} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)$$

where π_k is mixing coefficient for the k^{th} Gaussian, with $\sum_{k=1}^{\mathcal{K}} \pi_k = 1$ and $\mathcal{N}(x_i|\mu_k, \Sigma_k)$ is the multivariate Gaussian distribution with mean vector μ_k and covariance matrix Σ_k.

4 THREAT MODEL

We adopt a threat model similar to `EMShepherd` [7], an existing solution for detecting AEs in a black-box setting. The assumptions for adversary and defender are discussed as follows:

***Adversary*:** We assume the strongest adversary operating under a white-box setting, which means they have full knowledge of the DNN they target. This attack model allows the adversary to create the most effective AEs with minimal perturbations tailored to mislead the target DNN.

***Defender*:** Contrary to the adversary, we assume a defender with the most limited capability operating under a hard-label black-box setting. The defender does not have access to the target DNN's inputs, parameters, intermediate outputs, or prediction probabilities.

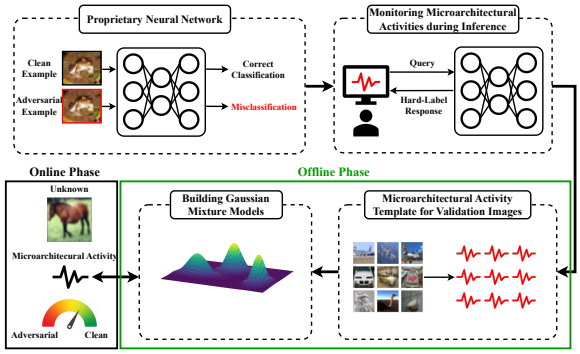

Figure 2: AdvHunter Overview: A two-phase AEs detection method utilizing microarchitectural activities. The offline phase constructs a benign template from clean validation images and trains GMMs for each output category to capture typical computational behavior. The online phase uses the GMMs for anomaly detection on unknown inputs to identify potential AEs.

The only information available to the defender is the DNN's prediction category, along with the ability to monitor HPC activities during inference. Given a limited set of clean validation images (assumption in line with `EMShepherd` [7]), the defender aims to distinguish whether an unknown input is clean or perturbed with adversarial noise using HPC data gathered during its inference.

5 METHODOLOGY

5.1 Brief Overview

When a defender obtains a proprietary DNN model from a vendor, they are limited to accessing its hard-label predictions. However, they have the capability to observe microarchitectural activities during the processing of any input. AdvHunter comprises two phases: offline and online. Figure 2 provides an overview of AdvHunter.

- **Offline Phase:** The defender utilizes a set of clean validation images to construct a template that reflects microarchitectural activities occurring during the inference phase, characterizing typical (non-malicious) activities. The benign template is subsequently used to build individual GMMs for each output category, capturing the computational behavior of DNN when processing typical clean inputs during inference operation.
- **Online Phase:** When presented with an unknown input, the defender monitors microarchitectural activities during its inference operation. Utilizing GMMs built in the offline phase, the defender employs an anomaly detection strategy to determine whether the unknown input is perturbed with adversarial noise.

5.2 Monitoring Microarchitectural Activities

Let us consider AdvHunter can monitor N events simultaneously. In order to minimize the impact of noise on HPC measurements due to other process execution in the background, the HPC measurements are repeated R times. We represent the distribution of a specific event n after R measurements as: $[\mathscr{E}_n]_{1\leq n\leq N} = \{e_n^{(1)}, e_n^{(2)}, \cdots, e_n^{(R)}\}$, where $e_n^{(r)}$ indicates the value of the n^{th} HPC event during the r^{th} iteration. Let us assume that there are C distinct output categories and the defender has access to M clean validation images for each of these categories, denoted by $\{x_1^{(c)}, x_2^{(c)}, \cdots, x_M^{(c)}\}_{1\leq c\leq C}$, where $x_m^{(c)}$ signifies the m^{th} image belonging to category c. Thus, for each $x_m^{(c)}$, the defender will have a distribution of N events, denoted as: $\{\overline{\mathscr{E}}_1^{(m_c)}, \overline{\mathscr{E}}_2^{(m_c)}, \cdots, \overline{\mathscr{E}}_N^{(m_c)}\}$, with $\overline{\mathscr{E}}_n^{(m_c)}$ being the mean of the distribution $\mathscr{E}_n$ for $x_m^{(c)}$. Hence, in the offline phase, the defender will construct a dataset of event statistics for M clean validation images across each of the C categories and N events, denoted as:

$$[\mathscr{D}_c]_{1\leq c\leq C} = \begin{pmatrix} \overline{\mathscr{E}}_1^{(1_c)} & \overline{\mathscr{E}}_2^{(1_c)} & \cdots & \overline{\mathscr{E}}_N^{(1_c)} \\ \overline{\mathscr{E}}_1^{(2_c)} & \overline{\mathscr{E}}_2^{(2_c)} & \cdots & \overline{\mathscr{E}}_N^{(2_c)} \\ \vdots & \vdots & \ddots & \vdots \\ \overline{\mathscr{E}}_1^{(M_c)} & \overline{\mathscr{E}}_2^{(M_c)} & \cdots & \overline{\mathscr{E}}_N^{(M_c)} \end{pmatrix}$$

The dataset $\mathscr{D}_c$ is a template that characterizes non-malicious microarchitectural activities of DNN during inference for the output category c. In summary, $\overline{\mathscr{E}}_N^{(M_c)}$, for instance, denotes the value obtained by selecting the M^{th} validation image from category c, observing the N^{th} HPC event for R times, and then calculating the mean of these R observations.

5.3 Building GMMs

We observe that the values corresponding to each HPC event are distributed according to a mixture of Gaussian distributions across all clean validation images. We provide experimental evidence for the same in Section 6. Based on this observation, we employ probabilistic GMMs to model the template for each HPC event separately. Consequently, we build N distinct GMMs for each of the N events across each of the C output categories. In other words, we construct individual GMMs for the dataset $[\mathscr{D}_c^n]_{1\leq n\leq N, 1\leq c\leq C} = \{\overline{\mathscr{E}}_n^{(1_c)}, \overline{\mathscr{E}}_n^{(2_c)}, \cdots, \overline{\mathscr{E}}_n^{(M_c)}\}$ (i.e., n^{th} column of $\mathscr{D}_c$). A GMM aims to estimate parameters $\Theta = \{\pi_1, \cdots, \pi_{\mathcal{K}}, \mu_1, \cdots, \mu_{\mathcal{K}}, \Sigma_1, \cdots, \Sigma_{\mathcal{K}}\}$ that maximize the likelihood of data belonging to a dataset, assuming that the dataset has a mixture of $\mathcal{K}$ Gaussians. Details of these parameters are discussed in Section 3. The parameter estimation is done using the Expectation-Maximization algorithm. We briefly discuss the algorithm for dataset $\mathscr{D}_c^n$ in Algorithm 1. Determining the optimal number of Gaussian mixtures in a GMM is a common

Algorithm 1: Expectation-Maximization for Gaussian Mixture Models (GMM)

Data: Data points $\mathscr{D}_c^n$, Number of Gaussians $\mathcal{K}$
Result: Optimal parameters $\Theta^* = \{\pi_k^*, \mu_k^*, \Sigma_k^*\}$ for each Gaussian k

Initialization:
Choose initial values for the parameters $\Theta = \{\pi_k, \mu_k, \Sigma_k\}$ for each Gaussian k
while *change in log-likelihood $\mathcal{L}$ is not below a threshold or maximum number of iterations is not reached* **do**
  for *each data point $d_i \in \mathscr{D}_c^n$* **do**
    Compute γ_{ik} using: $\gamma_{ik} = \frac{\pi_k \mathcal{N}(d_i|\mu_k,\Sigma_k)}{\sum_k \pi_k \mathcal{N}(d_i|\mu_k,\Sigma_k)}$
  for *each Gaussian k* **do**
    Update μ_k^{new} using: $\mu_k^{new} = \frac{\sum_i \gamma_{ik} d_i}{\sum_i \gamma_{ik}}$
    Update Σ_k^{new} using: $\Sigma_k^{new} = \frac{\sum_i \gamma_{ik}(d_i-\mu_k^{new})(d_i-\mu_k^{new})^T}{\sum_i \gamma_{ik}}$
    Update π_k^{new} using: $\pi_k^{new} = \frac{\sum_i \gamma_{ik}}{|\mathscr{D}_c^n|}$
  Compute log-likelihood $\mathcal{L}$ using: $\mathcal{L} = \sum_i \log\left(\sum_k \pi_k \cdot \mathcal{N}(d_i|\mu_k, \Sigma_k)\right)$

challenge. We use the Bayesian Information Criterion (BIC) [12] to identify the optimal number of Gaussian mixtures. Given several models, each with varying numbers of Gaussians, the model with the lowest BIC value is typically selected as the best model, signifying an optimal value of $\mathcal{K}$.

Once the GMMs have converged, best-fit parameters in Θ^* are used to characterize typical, non-malicious activities. These parameters help in specifying a threshold, which is subsequently used during the real-time online phase for anomaly detection. Let us consider the optimal GMM parameters for the n^{th} event and c^{th} output category is denoted as $\Theta^*(c,n) = \{\pi_k^*(c,n), \mu_k^*(c,n), \Sigma_k^*(c,n)\}_{1\leq k\leq \mathcal{K}}$. The associated threshold using dataset $\mathscr{D}_c^n$ is computed as: Let $\mathscr{L}_c^n$ be the distribution of negative log-likelihood for all $d_i \in \mathscr{D}_c^n$. Hence,

$$\mathscr{L}_c^n = \left\{ -\log\left(\sum_k \pi_k^*(c,n)\cdot \mathcal{N}(d_i|\mu_k^*(c,n),\Sigma_k^*(c,n))\right)\right\}_{1\leq i\leq M}$$

Adhering to the common heuristic of *three-sigma rule*, the threshold for the n^{th} event and c^{th} output category is computed as $\Delta_c^n = \mu_{\mathscr{L}_c^n} + 3\times\sigma_{\mathscr{L}_c^n}$, where $\mu_{\mathscr{L}_c^n}$ and $\sigma_{\mathscr{L}_c^n}$ are the mean and standard deviation of $\mathscr{L}_c^n$, respectively.

5.4 Detecting Anomaly

In the real-time online phase, given an unknown image, denoted as x_u, the first step is to obtain distributions for N events represented by $\{\overline{\mathscr{E}}_1^{(u)}, \overline{\mathscr{E}}_2^{(u)}, \ldots, \overline{\mathscr{E}}_N^{(u)}\}$. The process of obtaining distributions is similar to how it was performed during offline phase. Let the predicted category for x_u is denoted by $\hat{c}$. After obtaining distributions, we compute negative log-likelihood for the n^{th} event using $\Theta^*(\hat{c},n)$ as:

$$l_n^u = -\log\left(\sum_k \pi_k^*(\hat{c},n)\cdot\mathcal{N}(\overline{\mathscr{E}}_n^{(u)}|\mu_k^*(\hat{c},n),\Sigma_k^*(\hat{c},n))\right)$$

We conclude that image x_u contains an adversarial perturbation if the computed value l_n^u exceeds threshold $\Delta_{\hat{c}}^n$ (i.e., $l_n^u > \Delta_{\hat{c}}^n$), specifically when considering the event n.

6 EXPERIMENTAL RESULTS

Setup: In our evaluation, we examine three distinct scenarios. The dataset, CNN architectures, and clean accuracy for each of these scenarios are outlined in Table 1. We evaluate three distinct attack strategies: FGSM [9], PGD [8], and DeepFool [13], to demonstrate that the effectiveness of AdvHunter is independent of perturbation methods. We consider the L_∞-norm attack for FGSM and PGD and the L_2-norm attack for DeepFool to illustrate AdvHunter's potency across different attack norms. Additionally, we explore two variants of these attacks: untargeted and targeted, to exhibit the generalizability of AdvHunter against both variants. Without loss of generality, we randomly select 'shirt' from FashionMNIST, 'frog' from CIFAR10, and 'speed limit (30km/h)' from GTSRB as target classes for targeted attacks. The method is equally applicable to other target classes as well. We use the popular Python-based deep learning library PyTorch (version 2.0.1+cpu) for all implementations. We performed all experiments on a system powered by an Intel i7-9700 processor with a frequency of 3.00 GHz. Each HPC measurement is repeated 10 times (i.e., $R = 10$) to minimize the impact of noise due to execution of other processes in the background.

Table 1: Evaluation Scenarios along with Clean Accuracies

Scenario	Dataset	CNN Architecture	Clean Accuracy
S1	FashionMNIST	EfficientNet	92.34%
S2	CIFAR10	ResNet18	88.59%
S3	GTSRB	DenseNet201	96.67%

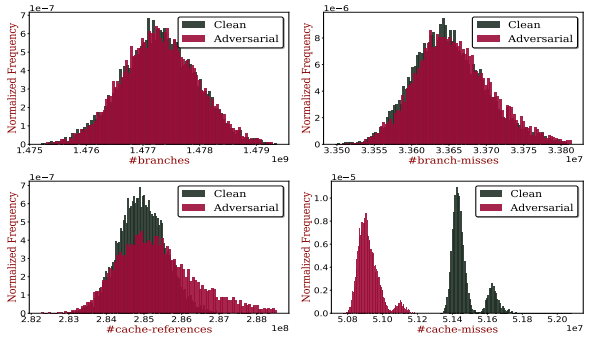

Figure 3: Distributions of different HPC events for clean and corresponding AEs in scenario S2, employing a targeted variant of FGSM.

Evaluation: We begin by studying five core HPC events: *instructions*, *branches*, *branch-misses*, *cache-references*, and *cache-misses* (i.e., $N = 5$). Functionalities of these events are self-explanatory from their names. For brevity, we present distributions of *branches*, *branch-misses*, *cache-references*, and *cache-misses* in scenario **S2**, employing a targeted variant of FGSM with an attack strength of $\epsilon = 0.5$. Under this attack setting, the targeted adversarial accuracy of the CNN is 94.04%, signifying a successful attack. Figure 3 demonstrates the behavior of the aforementioned events for clean inputs and corresponding AEs. We can observe that distributions for *branches* and *branch-misses* substantially overlap for both input types, implying a high degree of similarity. We excluded the plot for *instructions* as it follows an almost identical trend. The overlap is marginally reduced for *cache-references*. However, *cache-misses* presents a significant distinction between the two input types. This figure demonstrates that processing different input types during inference does not impact all events equally. This figure also substantiates the assertion made in Section 5.3 that HPC events are distributed following a mixture of Gaussian distributions.

The performance of AdvHunter to distinguish between clean and AEs considering the previous attack setting across all five previously mentioned events is shown in Table 2. The first column lists categories from the CIFAR10 dataset, excluding the target class. The performance for each category against the target class is described in terms of accuracy and the F_1-score for each event. To illustrate how to interpret the table, consider the following example: when CNN processes clean images of the 'frog' category and AEs originally of the 'airplane' category but misclassified to 'frog', AdvHunter can accurately identify clean versus AEs with a 99.25% accuracy when developed for *cache-misses*. We can observe that for *instructions*, *branches*, *branch-misses*, and *cache-references*, AdvHunter produces a significantly low accuracy and F_1-score, indicating that they are less reliable in differentiating between clean and AEs, which is also evident from the overlapping nature of distributions shown in Figure 3 for these events[1]. In contrast,

[1]Variations in accuracy across different categories shown in Table 2 are attributed to the varying effectiveness of AEs for each class, primarily due to the intricacies of

Table 2: Performance of AdvHunter in identifying clean and AEs for different HPC events in scenario S2, employing a targeted FGSM variant.

		instructions		branches		branch-misses		cache-references		cache-misses	
category	**target class**	accuracy	F_1-score	accuracy	F_1-score	accuracy	F_1-score	accuracy	F_1-score	accuracy	F_1-score
airplane	frog	49.55	0.0251	49.52	0.0307	50.38	0.0480	51.43	0.0967	99.25	0.9945
automobile		49.62	0.0437	49.62	0.0382	50.75	0.0737	51.63	0.1040	98.84	0.9876
bird		49.80	0.0420	50.15	0.0533	50.15	0.0588	51.26	0.0932	98.95	0.9886
cat		50.80	0.0683	50.38	0.0499	50.63	0.0737	50.88	0.0736	98.74	0.9866
deer		50.40	0.0624	50.18	0.0587	49.80	0.0402	50.50	0.0590	99.04	0.9896
dog		50.13	0.0550	49.82	0.0439	49.95	0.0478	66.63	0.5184	98.85	0.9876
horse		50.80	0.0719	50.38	0.0554	50.25	0.0462	67.72	0.5391	98.94	0.9886
ship		49.60	0.0327	49.55	0.0289	50.00	0.0533	51.35	0.0966	98.94	0.9886
truck		50.53	0.0627	50.10	0.0423	50.68	0.0737	53.76	0.1719	99.25	0.9915
overall		**50.14**	**0.0515**	**49.97**	**0.0446**	**50.29**	**0.0572**	**55.02**	**0.1947**	**98.98**	**0.9892**

cache-misses consistently displays a considerably higher accuracy (ranging from 98.74% to 99.25%) and F_1-score (ranging from 0.9866 to 0.9945). The last row of the table, labeled overall, provides average performance across all categories. It provides a consolidated view, highlighting that the best performance is observed for *cache-misses*, with an average accuracy of 98.98% and an F_1-score of 0.9892. *A high F_1-score signifies that AdvHunter is not only efficient in identifying clean and AEs but also produces low false positives and false negatives.* As discussed previously, when AEs are processed by a DNN, they activate a set of neurons not typically activated by clean inputs associated with the produced output category. This unusual neuron activation leads to a distinctive memory access footprint, as AEs cause different data to be loaded into the processor cache compared to clean inputs, owing to the varied neuron activations. This results in a different pattern of *cache misses* despite the DNN performing the same operations. As the DNN performs the same operations for all inputs, no noticeable variation is observed for other core HPC events like *instructions*, *branches*, or *branch-misses*, as also evident from Figure 3 and low detection capability for these events in Table 2. The results of AdvHunter's performance across various attack configurations and scenarios outlined in Table 1 using *cache-misses* are presented in Figure 4. The figure demonstrates that as the attack strength increases, there is a predictable drop in the accuracy of the model for various untargeted attacks and a rise in targeted accuracy for various targeted attacks. This trend is highlighted by examining three increasing levels of attack strength for both FGSM and PGD[2]. For DeepFool, the attack strength is set as default as its original paper. Notably, AdvHunter demonstrates its high capability against each attack type, as evidenced by the high F_1-scores in Figure 4.

***Ablation Study*:** In order to gain a more comprehensive understanding, we study four different HPC events related to cache-misses: *L1-dcache-load-misses*, *L1-icache-load-misses*, *LLC-load-misses*, and *LLC-store-misses* (i.e., $N = 4$ in this case). The functionalities of these events are self-explanatory from their names. For brevity, distributions of these events in scenario **S2**, employing an untargeted variant of FGSM with an attack strength of $\epsilon = 0.01$ are shown in Figure 5. When examining distributions for both input types, we observe that *L1-icache-load-misses* shows substantial overlap.

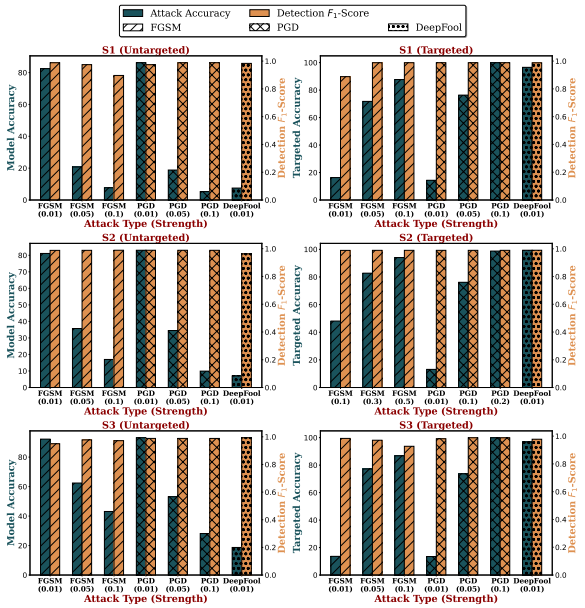

Figure 4: Effectiveness of various attacks (in terms of accuracy) and the performance of AdvHunter for corresponding attacks (measured by F_1-score) using *cache-misses* event across different scenarios.

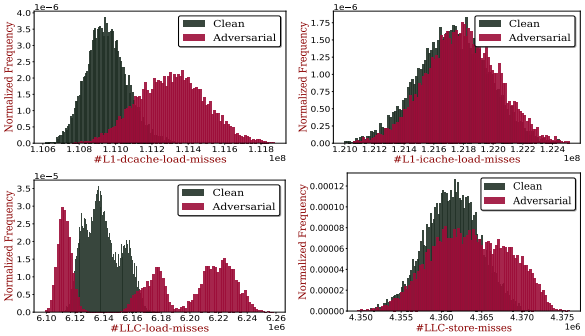

Figure 5: Distributions of different HPC events related to cache-misses for clean and corresponding AEs in scenario S2, employing an untargeted variant of FGSM.

In contrast, *LLC-store-misses* displays more distinctive patterns. However, *LLC-load-misses* and *L1-dcache-load-misses* events exhibit a significant difference between the two input types. This observation suggests that not all cache events are impacted uniformly when processing varying input types during inference. The overall

decision boundaries. Consequently, activation patterns differ across classes, influencing the HPC events differently.

[2]Targeted attacks, being stricter regarding misclassifications, require higher attack strength than untargeted attacks for some scenarios.

Table 3: Performance of AdvHunter (measured by F_1-score) in identifying clean and AEs for different HPC events related to cache-misses considering S2, employing an untargeted variant of FGSM.

	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$
L1-dcache-load-misses	0.7696	0.7258	0.6748
L1-icache-load-misses	0.0547	0.0622	0.0564
LLC-load-misses	0.9394	0.7938	0.3595
LLC-store-misses	0.3214	0.3347	0.2113

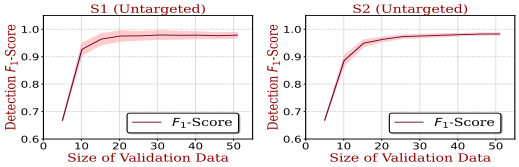

Figure 6: Performance of AdvHunter (measured by F_1-score) in identifying clean and AEs using *cache-misses* across varying validation data sizes in S1 and S2, employing an untargeted variant of FGSM. The shaded regions around the plots represent standard deviation.

performance of AdvHunter in distinguishing between clean and AEs, assuming the previous attack setting with different attack strengths across four cache-based HPC events under consideration (*L1-dcache-load-misses*, *L1-icache-load-misses*, *LLC-load-misses*, and *LLC-store-misses*), is shown in Table 3. We can observe that *L1-icache-load-misses* produces the lowest F_1-scores across all attack strengths, indicating its poor detection ability. In contrast, events associated with data cache misses (*L1-dcache-load-misses*, *LLC-load-misses*, and *LLC-store-misses*) perform better. This disparity stems from the fact that DNN inference for clean inputs and AEs have similar program flows, which poses a challenge for the instruction cache in differentiating between them. However, the data cache, especially *L1-dcache-load-misses*, is more effective in distinguishing the two because of the differences in data-flow dynamics, as discussed previously. Although monitoring *L1-dcache-load-misses* and other data cache events alone may not be as effective as monitoring comprehensive *cache-misses*, the main point of this discussion is to highlight that AdvHunter can leverage the data-flow dynamics in DNN inference to effectively differentiate between clean inputs and AEs despite their similar program flows.

In Figure 6, the performance of AdvHunter is analyzed for different sizes of validation datasets (i.e., M), focusing on *cache-misses* in scenarios **S1** and **S2** for a concise overview. This evaluation uses the untargeted variant of FGSM with an attack strength of $\epsilon = 0.01$ for brevity. The values in the x-axis represent the number of validation samples in each category. The y-axis displays the mean F_1-score of AdvHunter's performance and its standard deviation, calculated from 30 unique randomly selected validation datasets for each considered size. For **S1**, the F_1-score saturates once the validation data reaches a size of approximately $M = 30$ samples for each category. In the case of **S2**, the F_1-score saturates once the validation data reaches a size of approximately $M = 40$ samples per category. In contrast, **S3** (not shown in the figure for brevity as it follows a similar trend), which uses the more complex GTSRB dataset featuring 43 classes (i.e., $C = 43$) (as opposed to the 10 classes (i.e., $C = 10$) in CIFAR10 and FashionMNIST), requires a larger validation set for effective modeling of typical, non-malicious behavior. For **S3**, AdvHunter requires approximately $M = 60$ samples per category, after which the F_1-score saturates. In the `EMShepherd` framework [7], the authors did not clearly specify the number of samples used to train the anomaly detector. However, it is logical to assume that training a combined model consisting of an EM Classifier and a Variational Auto Encoder within `EMShepherd` would require more samples than those needed for modeling GMMs in the AdvHunter framework. These findings suggest that AdvHunter is effective even with relatively small validation datasets.

7 CONCLUSION

This paper introduces a novel strategy, called AdvHunter, for detecting AEs in DNNs under a realistic hard-label black-box setting. AdvHunter analyzes variations in microarchitectural activities during DNN inference to identify AEs without needing advanced equipment or physical access to DNNs. Extensive evaluation demonstrates its efficiency, emphasizing the potential of computational data flow dynamics in enhancing DNN security.

Acknowledgements: This work has been supported by the NYUAD Center for Cyber Security under RRC Grant No. G1104.

REFERENCES

[1] Manaar Alam and Debdeep Mukhopadhyay. 2019. How Secure are Deep Learning Algorithms from Side-Channel based Reverse Engineering?. In *56th Annual Design Automation Conference, DAC 2019, Las Vegas, NV, USA.*
[2] Ahmed Aldahdooh et al. 2023. Revisiting model's uncertainty and confidences for adversarial example detection. *Applied Intelligence* 53, 1 (2023), 509–531.
[3] Reza Azimi et al. 2005. Online performance analysis by statistical sampling of microprocessor performance counters. In *19th Annual International Conference on Supercomputing, ICS 2005, Cambridge, Massachusetts, USA.*
[4] Yi Cai et al. 2023. Ensemble-in-One: Ensemble Learning within Random Gated Networks for Enhanced Adversarial Robustness. In *37th AAAI Conference on Artificial Intelligence, AAAI 2023, Washington, DC, USA.*
[5] Anirban Chakraborty et al. 2021. A survey on adversarial attacks and defences. *CAAI Transactions on Intelligence Technology* 6, 1 (2021), 25–45.
[6] Jacob Clarysse et al. 2023. Why adversarial training can hurt robust accuracy. In *11th International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda.*
[7] Ruyi Ding et al. 2023. EMShepherd: Detecting Adversarial Samples via Side-channel Leakage. In *ACM Asia Conference on Computer and Communications Security, ASIA CCS 2023, Melbourne, VIC, Australia.*
[8] Yinpeng Dong et al. 2018. Boosting Adversarial Attacks With Momentum. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA.*
[9] Ian J. Goodfellow et al. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA.*
[10] Jaewon Jung et al. 2023. Fast Adversarial Training with Dynamic Batch-level Attack Control. In *60th ACM/IEEE Design Automation Conference, DAC 2023, San Francisco, CA, USA.*
[11] Sanjay Kariyappa and Moinuddin K. Qureshi. 2020. Defending Against Model Stealing Attacks With Adaptive Misinformation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA.*
[12] Geoffrey J. McLachlan and David Peel. 2000. *Finite Mixture Models.* Wiley.
[13] Seyed-Mohsen Moosavi-Dezfooli et al. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA.*
[14] Daryna Oliynyk et al. 2023. I Know What You Trained Last Summer: A Survey on Stealing Machine Learning Models and Defences. *Comput. Surveys* 55, 14 (2023), 324:1—-324:41.
[15] Zhuang Qian et al. 2022. A survey of robust adversarial training in pattern recognition: Fundamental, theory, and methodologies. *Pattern Recognition* 131 (2022), 108889.
[16] Florian Tramèr. 2022. Detecting Adversarial Examples Is (Nearly) As Hard As Classifying Them. In *International Conference on Machine Learning, ICML 2022, Baltimore, Maryland, USA.*
[17] Yuhang Wu et al. 2021. Beating Attackers At Their Own Games: Adversarial Example Detection Using Adversarial Gradient Directions. In *35th AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Event.*
[18] Yijun Yang et al. 2022. What You See is Not What the Network Infers: Detecting Adversarial Examples Based on Semantic Contradiction. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, CA, USA.*