# Detecting Backdoor Attacks in Black-Box Neural Networks through Hardware Performance Counters

Manaar Alam, Yue Wang, and Michail Maniatakos
Center for Cyber Security, New York University Abu Dhabi, Abu Dhabi, United Arab Emirates
{alam.manaar, yw3576, michail.maniatakos}@nyu.edu

***Abstract*—Deep Neural Networks (DNNs) have made significant strides, but their susceptibility to *backdoor attacks* still remains a concern. Most defenses typically assume access to *white-box* models or *poisoned data*, requirements that are often not feasible in practice, especially for proprietary DNNs. Existing defenses in a *black-box* setting usually rely on confidence scores of DNN's predictions. However, this exposes DNNs to the risk of *model stealing attacks*, a significant concern for proprietary DNNs. In this paper, we introduce a novel strategy for detecting backdoors, focusing on a more realistic black-box scenario where only *hard-label* (i.e., without any prediction confidence) query access is available. Our strategy utilizes data flow dynamics in a computational environment during DNN inference to identify potential backdoor inputs and is agnostic of *trigger* types or their locations in the input. We observe that a clean image and its corresponding backdoor counterpart with a trigger induce distinct patterns across various *microarchitectural activities* during the inference phase. We exploit these variations captured by *Hardware Performance Counters* (HPCs) and use principles of the *Gaussian Mixture Model* to detect backdoor inputs. *To the best of our knowledge, this is the first work that utilizes HPCs for detecting backdoors in DNNs*. Extensive evaluation considering a range of benchmark datasets, DNN architectures, and trigger patterns shows the efficacy of the proposed method in distinguishing between clean and backdoor inputs using HPCs.**

***Index Terms*—Neural Networks, Backdoor Attacks, Hardware Performance Counters, Side-Channel Information**

## I. 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. In this paper, we focus on addressing a specific security vulnerability in DNNs known as *backdoor attacks* [1], [2]: a training-time attack in which an adversary poisons training data with a stealthy *trigger*, ensuring the trained DNN functions normally for clean inputs, but produces adversary-chosen misclassifications when the specific trigger is present in the input. The growing threats of backdoor attacks have led to the development of numerous defenses [3]–[7]. However, most existing methods require access to poisoned training data or information on triggers [4], [7], which is commonly unavailable in practice, as vendors typically keep training data for their machine learning services private due to privacy concerns. Other methods, although not requiring access to poisoned data, still need *white-box* access [3], [5]. This is often impractical, especially for proprietary DNNs, where model parameters are considered intellectual property.

**Related Work in a Black-Box Scenario:** Backdoor detection in a *black-box* scenario remains a relatively unexplored area. Chen *et al.* first introduced a method to reverse engineer triggers by analyzing confidence scores from inference queries and using anomaly detection to identify backdoor inputs [8]. The method further requires model fine-tuning to patch the backdoor. Dong *et al.* proposed a method utilizing clean validation images and a gradient-free optimization strategy that uses prediction confidence to reverse engineer triggers and detect backdoor inputs [9]. Liu *et al.*, on the other hand, proposed a method that skips trigger reverse engineering; instead, they analyze prediction confidence across a set of clean validation images to identify backdoor inputs [10]. Fu *et al.* proposed a method by applying five new metrics to multiple synthetic images generated by incorporating parts of the input into clean validation images to detect backdoor inputs [11].

TABLE I: Comparison with Existing Defenses in a Black-Box Setting

| | works without prediction confidence? | needs small validation samples? | works without trigger reverse engineering? | works without model fine-tuning? |
|---|---|---|---|---|
| Chen *et al.* [8] | ✗ | Not Required | ✗ | ✗ |
| Dong *et al.* [9] | ✗ | ✗ | ✗ | ✓ |
| Liu *et al.* [10] | ✗ | ✗ | ✓ | ✓ |
| Fu *et al.* [11] | ✓ | ✓ | ✓* | ✓ |
| **Ours** | ✓ | ✓ | ✓ | ✓ |

∗: Effectiveness can be influenced by trigger's size and location.

Although these methods effectively detect backdoors in a black-box scenario, they come with challenges. A primary limitation is their dependence on *soft-label* black-box access to a DNN [8]–[10], where each prediction of DNN 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 [12], a significant concern for vendors providing proprietary DNN models. To protect against model stealing, vendors often obfuscate prediction confidence. Alternatively, only *hard-label* access is provided [13], where predictions are given without revealing any confidence, making existing black-box defenses that rely on prediction confidence ineffective. Moreover, some defenses involve a complex process of trigger reverse engineering and model fine-tuning [8], [9]. The size and location of the trigger on input can also influence the effectiveness of certain defenses [11]. Furthermore, some of these defenses require a large amount of clean validation images [9], [10], which may not always be affordable to defenders. Hence, in this paper, we introduce a method designed for a hard-label black-box setting, which is agnostic to the position or nature of triggers and does not need any trigger reverse engineering or model fine-tuning. Our method requires only a minimal set of clean validation images. We present a comparative overview of our approach with existing black-box defenses in Table I.

**Motivation:** The backdoor trigger represents a specific feature an adversary intentionally trains a DNN to recognize. When an

979-8-3503-4859-0/DATE24/© 2024 EDAA

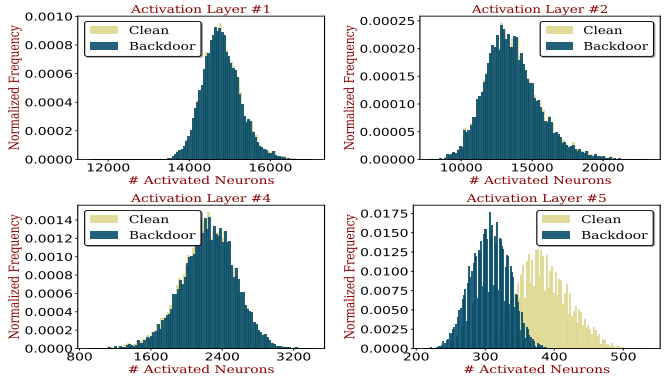


Fig. 1: Distributions of *Activated Neurons* at different activation units for both clean and corresponding backdoor inputs with a trigger.

adversary embeds the trigger on an input, it forces the DNN to produce a particular response. As a result, the presence of a trigger activates a very *specific set of neurons* in the DNN, creating a unique pathway of activated neurons to respond to the trigger. This behavior is distinct from how the DNN reacts to clean inputs, where it is designed to be sensitive to a broader range of features and thus activates a *more generalized set of neurons*. We present a case study using a primitive Convolutional Neural Network (CNN) to highlight the specific phenomenon. The CNN comprises five convolutional layers, each followed by a ReLU activation. It is trained on the well-known CIFAR-10 dataset where a 'single-pixel white dot' is embedded as a backdoor trigger at the bottom right corner for a small set of inputs. For conciseness, Fig. 1 provides a comparative view of frequency distributions of activated neurons for clean and corresponding backdoor inputs with a trigger for the first two (top row) and last two (bottom row) activation units. While neuron activations in initial activation units appear indistinguishable for both types of inputs, there is a marked difference for the final activation unit. As we delve deeper into CNN, it is evident that neuron activations must be significantly different to force misclassifications [14].

In a black-box scenario, a defender lacks visibility into internal neuron activations. However, when a backdoored DNN processes an input with a trigger, distinctive patterns of neuron activations lead to a unique sequence of computations compared to clean inputs. *In this paper, we utilize the fact and approach backdoor 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 computation sequence induces discernible *microarchitectural activities* (e.g., cache operations, branching instructions, etc.) for backdoor inputs compared to clean inputs. In this paper, we aim to monitor this side-channel information from microarchitectural activities to detect when a DNN processes input embedded with a trigger. 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 backdoor inputs. HPCs have previously been used to reverse-engineer a DNN [15]. However, to the best of our knowledge, HPCs have never been used as a defense to detect backdoors in DNNs.

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

- We propose a strategy to detect backdoor inputs in DNNs under a hard-label black-box scenario that does not assume any knowledge about triggers, does not require access to a poisoned dataset, and relies on only a minimal set of clean validation samples.
- We utilize HPCs to analyze variations in microarchitectural activities caused by data flow dynamics during DNN inference to identify potential backdoors.
- We evaluated the efficacy of the proposed method using three distinct trigger patterns, benchmark image classification datasets: MNIST, CIFAR-10, and GTSRB, and three standard CNN architectures: LeNet5, VGG11, and ResNet18.
- The implementations are open-sourced at: https://github.com/momalab/dnn-backdoor-detection-DATE24.git

## II. Preliminaries

***Hardware Performance Counters*:** Hardware Performance Counters (HPCs) are special-purpose registers present in most modern processors [16]. 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 2.6.35 and above, the `perf` tool provides access to these counters with high granularity.

***Gaussian Mixture Model*:** A Gaussian Mixture Model (GMM) is a probabilistic model that assumes data points are generated from a mixture of several Gaussian distributions with unknown parameters [17]. 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 $\pi_k$ is the 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 $\mu_k$ and covariance matrix $\Sigma_k$.

## III. Threat Model

**Adversary:** The adversary's threat model aligns with those discussed in earlier studies [1], [3]–[5]. As a vendor of machine learning services, the adversary can embed a backdoor in a DNN model during training. Let us consider a training dataset $\mathcal{D} = \{(x_i, y_i)\}$, where $x_i$ is a $d$-dimensional image and $y_i$ is the corresponding ground truth label. The adversary can insert a specific trigger (e.g., a patch) into a clean image $x_i$ as $x_i' = \mathcal{A}(x_i, m, p)$, where $\mathcal{A}$ is a function to apply the trigger, $m \in \{0,1\}^d$ is a binary mask to decide the position of the trigger, and $p$ is the trigger pattern. The adversary selects a subset $\mathcal{D}' \subset \mathcal{D}$ containing small percentage of training samples and creates poisoned data such as:

$$\mathcal{D}'_p = \{(x'_i, y'_i) | x'_i = \mathcal{A}(x_i, m, p), y'_i = y_t, (x_i, y_i) \in \mathcal{D}'\}$$

where $y_t$ is the adversary-chosen target class. A classification model $\mathcal{M}$ is trained on the poisoned training dataset $(\mathcal{D} \setminus \mathcal{D}') \cup \mathcal{D}'_p$. The attack is considered successful if $\mathcal{M}$ can classify any image with trigger $p$ as $y_t$ with a high success rate, while its accuracy for clean images is on par with the normal model.

**Defender:** We consider a *hard-label* black-box scenario for the defender. In this scenario, the defender does not have access to the poisoned training dataset, nor can they have white-box access to $\mathcal{M}$. The defender can only query $\mathcal{M}$ to obtain its predictions but cannot acquire any prediction probabilities. However, the defender has the ability to monitor HPC activities during DNN inference. Given a limited set of clean validation images (the assumption of having access to a set of clean validation images aligns with existing defense strategies [9]–[11]), the defender's goal is to distinguish whether an input is clean or backdoor using only HPC data during inference.

## IV. Methodology

### A. Brief Overview

Machine learning vendors can potentially embed backdoor triggers into DNN models. When a defender receives a model from a vendor, they can only access its hard-label predictions, but they can monitor microarchitectural activities when any input is processed. Our proposed backdoor detection method comprises two phases: offline and online. Fig. 2 presents a visual overview of the technique.

- **Offline Phase:** The defender utilizes a set of clean validation images and constructs a template of microarchitectural activities during the inference phase that characterizes typical (non-malicious) activities. The benign template is then used to build a GMM, capturing the computational behavior of DNN on typical clean inputs during inference operation.
- **Online Phase:** When presented with an unknown input, the defender monitors microarchitectural activities during prediction operation. Utilizing the GMM built in offline phase, the defender employs an anomaly detection strategy to determine whether the input contains any embedded trigger.

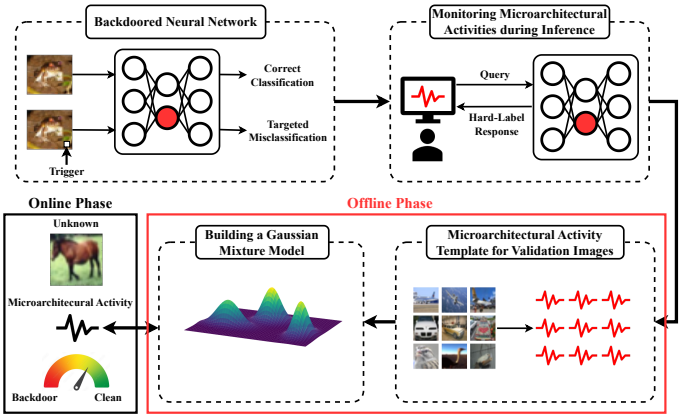


Fig. 2: **Overview:** A two-phase backdoor detection method utilizing microarchitectural activities. The offline phase constructs a benign template from clean validation images and trains a GMM to capture typical computational behavior. The online phase uses the GMM for anomaly detection on unknown inputs to identify potential backdoors.

### B. Monitoring Microarchitectural Activities

We assume that the proposed method monitors $N$ events simultaneously. In order to minimize the impact of noise on HPC measurements due to other process execution in the background, we repeat HPC measurements $R$ times. We represent the distribution of the event $n$ after $R$ measurements as: $[\mathscr{E}_n]_{1 \le n \le N} = \{e_n^{(1)}, e_n^{(2)}, \cdots, e_n^{(R)}\}$, where $e_n^{(r)}$ indicates the value of the $n^{th}$ HPC event during the $r^{th}$ iteration. Let us also assume that the defender has $M$ clean validation images denoted by $\{x_1, x_2, \cdots, x_M\}$, where $x_m$ is a $d$-dimensional image. Thus, for each $x_m$, the defender will have a distribution of $N$ events denoted as: $\{\overline{\mathscr{E}}_1^{(m)}, \overline{\mathscr{E}}_2^{(m)}, \cdots, \overline{\mathscr{E}}_N^{(m)}\}$, where $\overline{\mathscr{E}}_n^{(m)}$ is the mean of the distribution $\mathscr{E}_n$ for $x_m$. Hence, in the offline phase, the defender will have a dataset of event statistics for $M$ images and $N$ events denoted as:

$$\mathscr{D} = \begin{pmatrix} \overline{\mathscr{E}}_1^{(1)} & \overline{\mathscr{E}}_2^{(1)} & \cdots & \overline{\mathscr{E}}_N^{(1)} \\ \overline{\mathscr{E}}_1^{(2)} & \overline{\mathscr{E}}_2^{(2)} & \cdots & \overline{\mathscr{E}}_N^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \overline{\mathscr{E}}_1^{(M)} & \overline{\mathscr{E}}_2^{(M)} & \cdots & \overline{\mathscr{E}}_N^{(M)} \end{pmatrix}$$

The dataset $\mathscr{D}$ is a template that characterizes non-malicious microarchitectural activities of DNN during inference.

### C. Building a GMM

We observe that each event in template $\mathscr{D}$ follows a mixture of Gaussian distributions across all clean validation images. We provide experimental evidence for the same in Section V. Therefore, in this approach, we use probabilistic GMMs to capture the template for each HPC event separately. In other words, we intend to construct $N$ number of GMMs for each of the $N$ events having the dataset $[\mathscr{D}_n]_{1 \le n \le N} = \{\overline{\mathscr{E}}_n^{(1)}, \overline{\mathscr{E}}_n^{(2)}, \cdots, \overline{\mathscr{E}}_n^{(M)}\}$. 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 II. The parameter estimation is done using the Expectation-Maximization (EM) algorithm. We briefly discuss the algorithm for dataset $\mathscr{D}_n$ in Algorithm 1. Determining the optimal number of components $\mathcal{K}$ in a GMM is a common challenge. We use the Bayesian Information Criterion (BIC) [17] to identify the optimal number of Gaussian mixtures. Given several models with different numbers of Gaussians, the model with the lowest BIC value is typically selected as the best model with an optimal value of $\mathcal{K}$.

Once the GMM has converged, best-fit parameters $(\Theta^* = \{\pi_1^*, \cdots, \pi_\mathcal{K}^*, \mu_1^*, \cdots, \mu_\mathcal{K}^*, \Sigma_1^*, \cdots, \Sigma_\mathcal{K}^*\})$ are used to model normal, non-malicious activities. These parameters help in specifying a threshold, which is subsequently used during the real-time online phase for anomaly detection. The threshold for the $n^{th}$ event using dataset $\mathscr{D}_n$ and corresponding optimal GMM parameters $\Theta^*$ is computed as follows: Let $\mathscr{L}_n$ be the distribution of negative log-likelihood for all $d_i \in \mathscr{D}_n$. Hence,

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

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

**Data:** Data points $\mathscr{D}_n$, Number of Gaussians $\mathcal{K}$
**Result:** Parameters $\Theta = \{\pi_k, \mu_k, \Sigma_k\}$ for each Gaussian $k$

**Initialization:**
Choose initial values for the parameters $\Theta$
**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* $d_i \in \mathscr{D}_n$ **do**
    Compute $\gamma_{ik}$ 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 $\mu_k^{new}$ using: $\mu_k^{new} = \frac{\sum_i \gamma_{ik} d_i}{\sum_i \gamma_{ik}}$
    Update $\Sigma_k^{new}$ 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 $\pi_k^{new}$ using: $\pi_k^{new} = \frac{\sum_i \gamma_{ik}}{|\mathscr{D}_n|}$
  Compute log-likelihood $\mathcal{L}$ using: $\mathcal{L} = \sum_i \log\left(\sum_k \pi_k \cdot \mathcal{N}(d_i|\mu_k,\Sigma_k)\right)$

Following common rule of thumb, the threshold for the $n^{th}$ event is computed as $\Delta_n = \mathscr{L}_n^{\mu} + 3 \times \mathscr{L}_n^{\sigma}$, where $\mathscr{L}_n^{\mu}$ and $\mathscr{L}_n^{\sigma}$ are the mean and standard deviation of $\mathscr{L}_n$, respectively.

### D. 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 the offline phase. After obtaining distributions, we compute negative log-likelihood for the $n^{th}$ event as:

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

We conclude that image $x_u$ contains an embedded backdoor trigger if the computed value $l_n^u$ exceeds threshold $\Delta_n$ (i.e., $l_n^u > \Delta_n$), specifically when considering the event $n$.

## V. Experimental Results

**Setup:** For our evaluation, we examine three different scenarios for backdoor insertion.

- **S1:** Using the MNIST dataset, we train a LeNet5 model. We use a 'single-pixel white dot' at the bottom-right corner of an image as a trigger [1].
- **S2:** Using the CIFAR10 dataset, we train a ResNet18 model. We use a '3x3 black and white checkerboard' pattern at the top-left corner of an image as a trigger [1].
- **S3:** Using the GTSRB dataset, we train a VGG11 model. We use the 'invisible triggers' [18].

We use various trigger types to demonstrate that the effectiveness of the proposed method is independent of trigger types and their locations in input. Without loss of generality, we randomly select '0' in MNIST, 'airplane' in CIFAR10, and 'speed limit (20km/h)' in GTSRB as the target class for backdoor 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 to minimize the impact of noise due to the execution of other processes in the background.

**Evaluation:** We begin by studying five fundamental HPC events: *instructions*, *branches*, *branch-misses*, *cache-references*, and *cache-misses*. The functionalities of these events are self-explanatory from their names. For brevity, distributions of *branches*, *branch-misses*, *cache-references*, and *cache-misses* considering **S2** are shown in Fig. 3. The figure demonstrates the behavior of different events for clean inputs and corresponding backdoor inputs with triggers. We can observe that distributions for the event *branch-misses* for both input types have a high degree of overlap. An almost similar trend is observed for *branches* and *cache-references* events. However, *cache-misses* event presents a significant distinction between two input types. This demonstrates that processing different input types during inference does not impact all events equally.

The performance of the proposed method to distinguish between clean and backdoor inputs considering **S2** across all five previously mentioned events is shown in Table II. The first column lists categories from the CIFAR10 dataset, excluding the backdoor 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 DNN processes clean images of the 'automobile' category and backdoor images originally of the same category but misclassified to 'airplane', the proposed method can accurately identify clean versus backdoor inputs with a 93.22% accuracy when developed for *cache-misses* event. We can observe that for the events *instructions*, *branches*, *branch-misses*, and *cache-references*, the proposed method produces a significantly low accuracy and $F_1$-score, indicating that they are less reliable in differentiating between clean and backdoor inputs, which is also evident from the overlapping nature of distributions shown in Fig. 3 for these events. In contrast, *cache-misses* event consistently

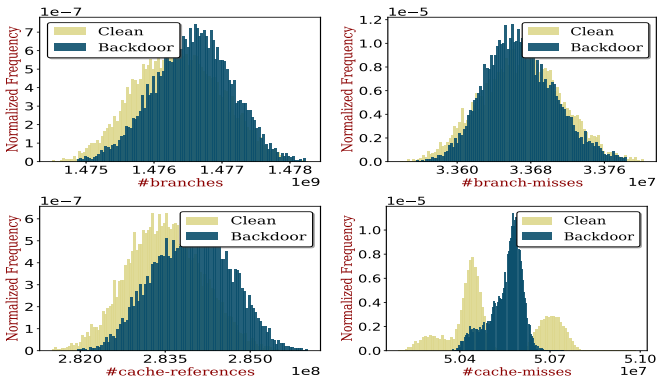


Fig. 3: Distributions of different HPC events for both clean and corresponding backdoor inputs with a trigger considering **S2**.

TABLE II: Performance of the proposed method in identifying clean and backdoor inputs for different HPC events considering **S2**.

| | | **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 |
| automobile | airplane | 48.65 | 0.02 | 49.35 | 0.03 | 48.74 | 0.01 | 50.15 | 0.00 | 93.22 | 0.93 |
| bird | | 48.35 | 0.01 | 48.60 | 0.01 | 48.64 | 0.01 | 50.18 | 0.01 | 92.31 | 0.92 |
| cat | | 49.70 | 0.00 | 49.70 | 0.01 | 49.67 | 0.01 | 49.62 | 0.00 | 82.03 | 0.79 |
| deer | | 50.08 | 0.00 | 50.00 | 0.01 | 50.60 | 0.04 | 50.25 | 0.01 | 82.16 | 0.78 |
| dog | | 49.62 | 0.01 | 49.82 | 0.01 | 49.77 | 0.04 | 48.74 | 0.07 | 86.29 | 0.84 |
| frog | | 50.08 | 0.01 | 50.48 | 0.03 | 49.30 | 0.02 | 55.34 | 0.21 | 87.01 | 0.85 |
| horse | | 49.22 | 0.03 | 49.72 | 0.05 | 49.22 | 0.02 | 60.67 | 0.37 | 93.78 | 0.93 |
| ship | | 49.12 | 0.03 | 49.72 | 0.06 | 49.40 | 0.02 | 51.53 | 0.16 | 67.92 | 0.53 |
| truck | | 49.50 | 0.06 | 49.35 | 0.07 | 49.75 | 0.02 | 57.39 | 0.31 | 74.45 | 0.66 |
| **overall** | | **49.3689** | **0.0189** | **49.6378** | **0.0311** | **49.4544** | **0.0211** | **52.6522** | **0.1267** | **84.3522** | **0.8033** |

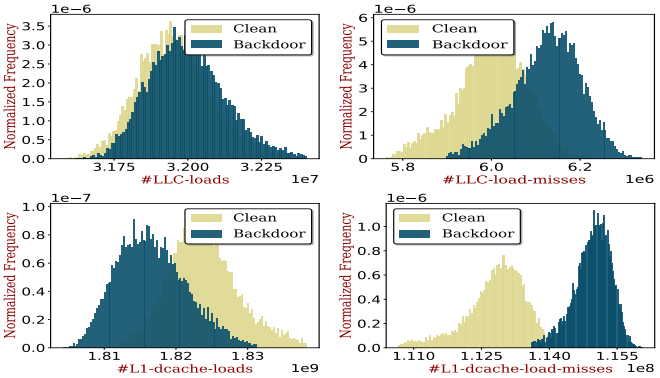


Fig. 4: Distributions of different cache events for both clean and corresponding backdoor inputs with a trigger considering **S2**.

displays a considerably higher accuracy (ranging from 67.92% to 93.78%)[1] and $F_1$-score (ranging from 0.53 to 0.93). 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* event, with an average accuracy of 84.3522% and an $F_1$-score of 0.8033. A high $F_1$-score signifies that the method is not only efficient in identifying clean and backdoor inputs but also produces low false positives and false negatives. As previously mentioned, backdoor input, when processed by a DNN, activates a unique set of neurons not typically activated by benign inputs, creating distinctive activation patterns. Consequently, these unique activations cause noticeable differences in memory access patterns, leading to a discernible variation in *cache-misses*, as also evident in Fig. 3.

In order to gain a more comprehensive understanding, we study seven different cache-based HPC events: *LLC-loads*, *LLC-load-misses*, *LLC-stores*, *LLC-store-misses*, *L1-dcache-loads*, *L1-dcache-stores*, and *L1-dcache-load-misses*. The functionalities of these events are self-explanatory from their names. For brevity, distributions of *LLC-loads*, *LLC-load-misses*, *L1-dcache-loads*, and *L1-dcache-load-misses* considering **S2** are shown in Fig. 4. When examining distributions for both input types, we observe that *LLC-loads* event shows substantial overlap. In contrast, events such as *LLC-load-misses* and *L1-dcache-loads* display more distinctive patterns. However, *L1-dcache-load-misses* event exhibits 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 performance of the proposed method to distinguish between clean and backdoor inputs considering **S2** across five cache-based HPC events (*LLC-load-misses*, *LLC-store-misses*, *L1-dcache-loads*, *L1-dcache-stores*, and *L1-dcache-load-misses*) is shown in Table III. The overall accuracy for the events *LLC-loads* is 51.4467% with an $F_1$-score of 0.0722, while for *LLC-stores*, the accuracy is 51.2689% with an $F_1$-score of 0.0744 (which are not shown in table for brevity). We observe that *L1-dcache-load-misses* event consistently displays a significantly higher accuracy (ranging from 96.57% to 99.75%) and $F_1$-score (ranging from 0.96 to 1.00) compared to other cache-based events. Hence, if we consider *L1-dcache-load-misses* event for detecting backdoors, the overall accuracy and $F_1$-score of the method improve by 17.31% and 22.83%, respectively, as compared to only considering *cache-misses* event. The L1 cache, being smaller than the LLC, offers finer monitoring granularity. This allows it to detect subtle computation differences more accurately, like those from backdoor trigger activations. The larger LLC masks these detailed memory access patterns, so *L1-dcache-load-misses* is more precise in identifying backdoor inputs.

The performance of the proposed method to distinguish between clean and backdoor inputs considering all three scenarios for *LLC-load-misses* and *L1-dcache-load-misses* is shown in Table IV. We selected these events to demonstrate the effectiveness of the proposed method for L1 cache over LLC. For **S1**, we observe that the accuracy and $F_1$-score are not as high as in other scenarios. This is primarily attributed to the use of the LeNet5 architecture in **S1**, which is relatively shallow with five layers. In contrast, models used in other scenarios are deeper and more complex. Deeper networks accumulate distinctions in cache behaviors across their multiple layers. Even with the shallow network in **S1**, the proposed method manages to secure an accuracy of 75.5601% and an $F_1$-score of 0.7156 when considering *L1-dcache-load-misses*. For the same reason, the performance of the method for **S2** is better than **S3**, as ResNet18 in **S2** is deeper than VGG11 in **S3**. Additionally, when considering *L1-dcache-load-misses*, performance metrics for all scenarios are consistently better than those observed with *LLC-load-misses*. The average improvement in accuracy and $F_1$-score is by 40.56% and 81.28%, respectively. This further highlights the effectiveness of the lower-level L1 cache

[1]Variation in accuracy across different categories arises because backdoor attacks have varying effectiveness for each class, primarily due to the intricacies of decision boundaries. Consequently, activation patterns differ across classes.

TABLE III: Performance of the proposed method in identifying clean and backdoor inputs for different cache events considering **S2**.

| | | **LLC-load-misses** | | **LLC-store-misses** | | **L1-dcache-loads** | | **L1-dcache-stores** | | **L1-dcache-load-misses** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **category** | **target class** | accuracy | $F_1$-score | accuracy | $F_1$-score | accuracy | $F_1$-score | accuracy | $F_1$-score | accuracy | $F_1$-score |
| automobile | airplane | 52.96 | 0.11 | 50.65 | 0.08 | 50.58 | 0.02 | 52.75 | 0.1 | 96.57 | 0.96 |
| bird | | 63.77 | 0.43 | 51.71 | 0.10 | 68.54 | 0.54 | 62.24 | 0.39 | 97.54 | 0.97 |
| cat | | 80.22 | 0.76 | 51.29 | 0.10 | 87.12 | 0.86 | 75.58 | 0.68 | 99.39 | 0.99 |
| deer | | 72.73 | 0.64 | 51.38 | 0.13 | 80.95 | 0.77 | 71.49 | 0.61 | 99.25 | 0.99 |
| dog | | 68.36 | 0.54 | 52.16 | 0.12 | 72.17 | 0.62 | 68.41 | 0.54 | 99.60 | 0.99 |
| frog | | 67.30 | 0.51 | 53.25 | 0.15 | 64.94 | 0.46 | 67.02 | 0.51 | 99.75 | 1.00 |
| horse | | 69.78 | 0.57 | 52.97 | 0.13 | 63.12 | 0.41 | 68.70 | 0.55 | 99.65 | 1.00 |
| ship | | 61.42 | 0.38 | 51.13 | 0.09 | 56.01 | 0.21 | 61.95 | 0.39 | 99.45 | 0.99 |
| truck | | 63.15 | 0.41 | 51.76 | 0.11 | 55.77 | 0.20 | 60.16 | 0.34 | 99.34 | 0.99 |
| **overall** | | **66.6322** | **0.4833** | **51.8111** | **0.1122** | **66.5778** | **0.4544** | **65.3667** | **0.4567** | **98.9489** | **0.9867** |

TABLE IV: Performance of the proposed method in identifying clean and backdoor inputs for different scenarios considering *LLC-load-misses* and *L1-dcache-load-misses*.

| | **LLC-load-misses** | | **L1-dcache-load-misses** | |
|---|---|---|---|---|
| | accuracy | $F_1$-score | accuracy | $F_1$-score |
| **S1** | 59.0733 | 0.4267 | 75.5601 | 0.7156 |
| **S2** | 66.6322 | 0.4833 | 98.9489 | 0.9867 |
| **S3** | 60.6975 | 0.4935 | 88.1733 | 0.8487 |

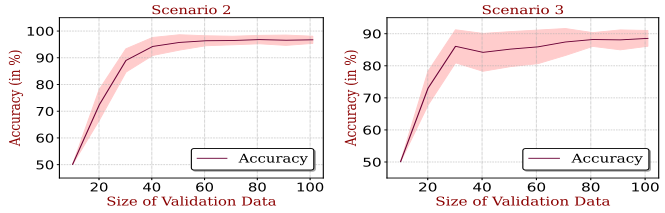


Fig. 5: Accuracy of the proposed method in distinguishing between clean and backdoor images using *L1-dcache-load-misses* across varying validation data sizes in **S2** and **S3**. The shaded regions around the plots represent standard deviation.

compared to LLC in the proposed detection strategy.

Fig. 5 demonstrates the accuracy of the proposed method across various sizes of validation data, focusing on **S2** and **S3** for conciseness. The plot represents the average accuracy and standard deviation derived from 30 distinct sets of randomly chosen validation data for each size. For **S2**, the accuracy saturates once the validation data reaches a size of approximately 60 samples. Similarly, for **S3**, the accuracy saturates once the validation data reaches a size of approximately 100 samples. **S3** requires more validation data as GTSRB in **S3** has more categories (43 classes) than CIFAR10 in **S2** (10 classes). Hence, the proposed method requires more validation data to model typical non-malicious behavior. For **S1**, the method requires approximately 80 samples, after which the accuracy saturates. Based on these observations, we can conclude that the proposed method performs efficiently with a small validation set.

## VI. Conclusion

This paper introduces a novel strategy for detecting backdoors in DNNs under a realistic hard-label black-box setting. Without knowledge of triggers or access to poisoned datasets, the proposed method analyzes variations in microarchitectural activities during DNN inference to successfully identify backdoor inputs using only a minimal set of clean validation samples. 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 and the NYUAD Global PhD Fellowship.

## References

[1] T. Gu *et al.*, "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks," *IEEE Access*, vol. 7, pp. 47230–47244, 2019.

[2] Y. Wang *et al.*, "Stop-and-Go: Exploring Backdoor Attacks on Deep Reinforcement Learning-Based Traffic Congestion Control Systems," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4772–4787, 2021.

[3] K. Liu *et al.*, "Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks," in *Research in Attacks, Intrusions, and Defenses - 21st International Symposium, Greece, September 10-12*, 2018.

[4] B. Tran *et al.*, "Spectral Signatures in Backdoor Attacks," in *Annual Conference on Neural Information Processing Systems, December 3-8, Canada*, 2018.

[5] B. Wang *et al.*, "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks," in *IEEE Symposium on Security and Privacy, USA, May 19-23*, 2019.

[6] E. Sarkar *et al.*, "Backdoor Suppression in Neural Networks using Input Fuzzing and Majority Voting," *IEEE Design & Test*, vol. 37, no. 2, pp. 103–110, 2020.

[7] Y. Wang *et al.*, "A Trigger Exploration Method for Backdoor Attacks on Deep Learning-Based Traffic Control Systems," in *60th IEEE Conference on Decision and Control, USA, December 14-17*, 2021.

[8] H. Chen *et al.*, "DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks," in *28th International Joint Conference on Artificial Intelligence, China, August 10-16*, 2019.

[9] Y. Dong *et al.*, "Black-box Detection of Backdoor Attacks with Limited Information and Data," in *IEEE/CVF International Conference on Computer Vision, Canada, October 10-17*, 2021.

[10] G. Liu *et al.*, "An Adaptive Black-Box Defense Against Trojan Attacks (TrojDef)," *IEEE Transactions on Neural Networks and Learning Systems (Early Access)*, 2022.

[11] H. Fu *et al.*, "Differential Analysis of Triggers and Benign Features for Black-Box DNN Backdoor Detection," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 4668–4680, 2023.

[12] J. Beetham *et al.*, "Dual Student Networks for Data-Free Model Stealing," in *International Conference on Learning Representations, Rwanda, May 1-5*, 2023.

[13] S. Kariyappa and M. K. Qureshi, "Defending Against Model Stealing Attacks With Adaptive Misinformation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, USA, June 13-19*, 2020.

[14] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *13th European Conference on Computer Vision, Switzerland, September 6-12*, 2014.

[15] M. Alam and D. Mukhopadhyay, "How Secure are Deep Learning Algorithms from Side-Channel based Reverse Engineering?," in *56th Annual Design Automation Conference, USA, June 02-06*, 2019.

[16] R. Azimi *et al.*, "Online performance analysis by statistical sampling of microprocessor performance counters," in *19th Annual International Conference on Supercomputing, USA, June 20-22*, 2005.

[17] G. J. McLachlan and D. Peel, *Finite Mixture Models*. Wiley Series in Probability and Statistics, Wiley, 2000.

[18] Y. Li *et al.*, "Invisible Backdoor Attack with Sample-Specific Triggers," in *IEEE/CVF International Conference on Computer Vision, Canada, October 10-17*, 2021.