BACKDOORS WITHOUT TRIGGERS: HARDWARE-CONDITIONAL MISCLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Machine learning models are routinely deployed on a wide range of computing hardware. Although such hardware is typically expected to produce identical results, differences in its design can lead to small numerical variations during inference. In this work, we show that these variations can be exploited to create backdoors in machine learning models. The core idea is to shape the model's decision function such that it yields different predictions for the same input when executed on different hardware. This effect is achieved by locally moving the decision boundary close to a target input and then refining numerical deviations to flip the prediction on selected hardware. We empirically demonstrate that these hardware-triggered backdoors can be created reliably across common GPU accelerators. Our findings reveal a novel attack vector affecting the use of third-party models, and we investigate different defenses to counter this threat.

1 INTRODUCTION

Hardware acceleration is a cornerstone of machine learning inference. Depending on the application, learning models are routinely deployed on a wide range of computing hardware, from inexpensive consumer GPUs to high-performance accelerators. While these devices differ significantly in efficiency and energy consumption, a key assumption is that they compute identical results, enabling seamless deployment. Interestingly, this assumption does not fully hold in practice. Differences in hardware design give rise to small numerical variations during operation, which impact reproducibility but are generally considered harmless (Schlögl et al., 2024; Yuan et al., 2025).

In this paper, we assume a security perspective and ask: Could numerical deviations across hardware be exploited to bind misclassifications to specific hardware? To answer this question, we build on an important insight: surprisingly, a learning model does not induce a single decision function in practice; rather, it induces a family of very similar functions, depending on the hardware on which it is executed. While these functions remain numerically close to one another in benign settings, an adversary may manipulate a model so that their gap widens and certain decisions occur only on selected hardware. We refer to this novel attack type as a *hardware-triggered backdoor*.

To explore the feasibility of this attack, we introduce a method for manipulating a model so that it yields conflicting predictions for a selected input when executed on different hardware. We achieve this effect by carefully pushing the model's decision boundary close to the selected input and then amplifying numerical deviations to flip the prediction on selected hardware only. Unlike traditional backdoors, this attack does not rely on a trigger in the input. Instead, the numerical behavior of the hardware acts as a latent trigger that changes the prediction. As a result, the model behaves as expected on all inputs when run on non-targeted hardware, making the attack highly stealthy. For example, a downloaded model might show no sign of manipulation during testing on an Nvidia A100, yet misclassify selected inputs once moved to production on an H100 GPU.

We empirically find that this approach is effective and independent of accidental numerical instability. Instead, hardware-dependent deviations can be systematically induced for common model architectures, enabling us to identify a malicious model parameterization with a success rate of over 90%, with no impact on model performance. The resulting backdoors can be targeted to particular hardware accelerators and can withstand variations in inference, such as input perturbations, batching, and mixed-precision inference.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We conclude that these hardware-triggered backdoors pose a threat whenever third-party models are deployed across device setups. As a countermeasure, we investigate different defenses and evaluate their effectiveness, with positive results. Our findings highlight that the security of machine learning must be considered holistically from trained models down to the underlying hardware, as numerical deviations, even if seemingly small, may have adversarial effects.

In summary, we make the following major contributions:

1. **Hardware-triggered backdoors.** We introduce a new type of backdoor in which malicious behavior is activated by specific hardware rather than an input trigger.
2. **Causal analysis of differences.** We perform a layer-wise causal analysis to identify where hardware-induced differences arise during inference.
3. **Evaluation of efficacy.** We demonstrate the efficacy of hardware-triggered backdoors over different model architectures, hardware devices, and small input perturbations.

2 Numerical Deviations

In theory, inference in machine learning models is a well-defined process in which an input is passed through a decision function using learned parameters. From this perspective, operations such as matrix multiplication or convolution are precisely specified, leaving no room for deviations. In practice, however, these operations are performed using floating-point numbers of limited precision. While this precision can be adapted, it remains finite, rendering arithmetic inherently imprecise (IEEE, 2019).

Non-associativity. The primary source of this imprecision is the non-associativity of floating-point addition $a + (b + c) \neq (a + b) + c$ (Schlögl et al., 2024), which implies that the result of a sum depends on the order of the additions. Many operations, including matrix multiplication, convolution, aggregations, and attention, rely on a series of additions. Their order is shaped by the underlying hardware resources, for instance through choices of block size and warp scheduling.

We can illustrate this effect with a simple matrix $M \in \mathbb{R}^{100\times100}$ whose entries are all equal to 0.01. When computing the squared Frobenius norm of M on two GPUs using the source code given in Listing 1 in the appendix, we observe slightly deviating results,

$$\begin{aligned} ||M||_F^2 &= \mathrm{tr}(M^T M) = 1 \\ &\approx 0.9999999403953552 \quad \text{(Nvidia A100)} \\ &\approx 0.9999990463256836 \quad \text{(Nvidia H100)}. \end{aligned}$$

Notation. We consider a learning model θ that induces a theoretical decision function $f_\theta \colon \mathbb{R}^n \to \mathbb{R}^c$, which maps an n-dimensional input to c class logits. In practice, the realized behavior of this function depends on the hardware $h \in \mathbb{H}$ on which the model is executed, where $\mathbb{H}$ denotes a set of functionally equivalent hardware platforms, such as different GPU accelerators. Accordingly, we model the effective decision function as

$$f_\theta \colon \mathbb{R}^n \times \mathbb{H} \longrightarrow \mathbb{R}^c, \tag{1}$$

thereby making explicit its joint dependence on the input and the hardware. In line with prior work, we consider this function to behave deterministically (Zhang et al., 2025; Möller et al., 2025).

During inference, this hardware dependence can induce unexpected discrepancies from the correct result. When predictions are obtained by selecting the class with the largest logit, that is,

$$F_\theta(x; h) = \arg\max_i f_\theta(x; h)_i, \tag{2}$$

where $f(\cdot)_i$ denotes the i-th logit, it may occur that

$$F_\theta(x; h_1) \neq F_\theta(x; h_2), \tag{3}$$

for an input x and two devices h_1 and h_2. In such cases, identical inputs processed by the same model are assigned different class labels solely due to the underlying hardware.



(a) Step 1: Shaping of decision boundary

(b) Step 2: Refinement of deviation

Figure 1: Construction of hardware-triggered backdoors: (a) The decision boundary is moved close to the target $\hat{x}$; (b) Hardware deviations are amplified between h_1 and h_2.

3 Hardware-Triggered Backdoors

Numerical errors arise naturally in floating-point-based inference, but, due to their small magnitude, rarely influence class predictions. Hence, they are generally regarded as harmless. We challenge this view by exploring whether hardware-based deviations can be deliberately exploited to create misclassifications in learning models.

3.1 Threat Model

We consider a supply-chain attack scenario in which the attacker acts as a malicious provider, that distributes a model containing a hardware-triggered backdoor to one or more victims. The attacker seeks to find a parameterization such that the model behaves as expected on the attacker's hardware, allowing for auditing of the model, while misclassifying pre-chosen inputs on the victim's hardware.

In the following, we focus on the image domain for its diverse set of model architectures and widespread availability. However, due to the attack operating in the representation space, the method is generally domain-agnostic—even in discrete input domains. This broad applicability is particularly relevant for security-critical applications, such as face recognition, malware detection, and fraud detection, where a single successful misclassification can constitute a security incident. In practice, such models may be distributed as components of larger software systems, where the attacker can have prior knowledge of the victim's execution environment.

3.2 Attack Strategy

Let us begin by considering a target input $\hat{x}$ that the adversary aims to misclassify on one hardware device h_1 but not on another device h_2. The attacker's objective is to induce a deviation such that

$$F_\theta(\hat{x}; h_1) \neq F_\theta(\hat{x}; h_2), \tag{4}$$

while ensuring that, for all other inputs, the behavior of the modified model θ remains indistinguishable from that of the original one. Note that in this case $\hat{x}$ is deliberately selected, whereas in common errors caused by numerical deviations the affected inputs arise at random.

At a first glance, implementing this attack appears straightforward: one could formulate a loss that enforces conflicting predictions across hardware and optimize it using gradient-based methods, similar to existing backdoor attacks (Liu et al., 2018; Shafahi et al., 2018). Unfortunately, this approach is infeasible. First, numerical deviations are hardware-triggered, and so are the corresponding gradients. Second, the deviations are non-differentiable, ruling out the tools commonly used in backdoor attacks.

To overcome these challenges, we build on a key observation: in practice, a learning model does not induce a single decision function, but rather a family of closely related functions that depend on the underlying hardware. While these functions are numerically close, their differences can become consequential in regions where the model is sensitive to small perturbations. In particular, when an input lies close to the model's decision boundary, even minor deviations may suffice to change the predicted label.

Based on this observation, we propose a two-step attack strategy, as illustrated in Figure 1. First, we locally move the decision boundary into the vicinity of a target input on one hardware device (Figure 1a). Second, we adjust the numerical deviations between two devices, such that the respective decision functions disagree on the prediction of the input (Figure 1b).

3.3 Shaping the Decision Boundary

For the first step, we treat the model as a single decision function and operate on it using standard gradient-based optimization. To this end, we optimize a proxy loss $\mathcal{L}$ that encourages proximity to the decision boundary while constraining deviations from the original model behavior. All computations are performed on a single hardware device h_1. The resulting optimization problem is

$$\arg\min_{\theta} \mathcal{L}(\theta, y) \quad \text{with } y = f_\theta(\hat{x}; h_1) \tag{5}$$

where the proxy loss $\mathcal{L}$ consists of three components

$$\begin{aligned} \arg\min_{\theta} \quad & \alpha\Big(\max_i y_i - \max_{j \neq \arg\max_i y_i} y_j\Big) && \text{(boundary proximity loss)} \\ & + \beta\Big(\max\big(\max_{i \neq t} y_i - y_t, 0\big)\Big) && \text{(source class consistency loss)} \\ & + \gamma\Big(\|\theta - \bar{\theta}\|^2\Big) && \text{(regularization loss)} \end{aligned} \tag{6}$$

with α, β, and γ controlling the relative influence of the individual terms during optimization and t is the original class of the input $\hat{x}$. The first loss term encourages proximity of $\hat{x}$ to the decision boundary by minimizing the difference between the two largest logits, effectively creating a tie. The second term penalizes deviations from the original source class t of $\hat{x}$. This ensures stealthiness on non-target devices as the input remains at its original label. Finally, to keep the manipulation localized, we regularize deviations from the original unmodified model $\bar{\theta}$.

3.4 Refining Deviations

After reshaping the decision boundary, the target input $\hat{x}$ lies in its immediate vicinity, yet it is still likely to receive the same prediction across all hardware platforms. The goal of the second step is to exploit this fragile configuration and amplify hardware-dependent divergence. As these deviations are non-differentiable, however, we must resort to heuristic strategies to manipulate them across hardware devices. In particular, we consider two types of manipulations.

Topological permutation. We introduce a *topological permutation*, which preserves mathematical equivalence under exact arithmetic but alters the order of floating-point operations, resulting in platform-dependent deviations. Specifically, we use a permutation matrix P_i and its inverse P_i^{-1} to permute weights of the model. Given a matrix multiplication $W_1 W_2$ inside the model, we construct:

$$W_1 W_2 = \underbrace{(P_1 W_1)}_{\hat{W}_1} \underbrace{(P_1^{-1} W_2)}_{\hat{W}_2} . \tag{7}$$

This construction applies to models with at least two consecutive linear layers, a pattern present in different architectures, including transformers.

Parameter perturbation. We consider small perturbations of the model parameters themselves. Concretely, we select a set of k bits in the parameters and flip their values, thereby introducing limited numerical changes into the computation. Although this mechanism is more powerful, it affects the computation even under exact arithmetic, which may decrease model utility.

3.5 Alternating Optimization

Finally, we combine both steps in an alternating optimization procedure. In each iteration, the decision boundary is first locally shifted toward the target input. Subsequently, both strategies are applied to search for a split decision across the selected hardware. To address the heuristic nature of the manipulation strategies, we construct m candidate models in each iteration. We terminate once a model exhibits a functional backdoor. Moreover, during optimization we discard candidates that fail to preserve a selected level ρ of the original model's performance.

Table 1: Success rates of finding a model parameterization for a pre-chosen input (float32).

GPU	ViT	ResNet	EfficientNet	SigLIP
H100	$94\,\% \pm 6\,\%$	$100\,\% \pm 0\,\%$	$100\,\% \pm 0\,\%$	$94\,\% \pm 6\,\%$
A100	$98\,\% \pm 3\,\%$	$75\,\% \pm 43\,\%$	$100\,\% \pm 0\,\%$	$98\,\% \pm 3\,\%$
A100-MIG40	$98\,\% \pm 3\,\%$	$75\,\% \pm 43\,\%$	$100\,\% \pm 0\,\%$	$97\,\% \pm 3\,\%$
A40	$94\,\% \pm 6\,\%$	$100\,\% \pm 0\,\%$	$100\,\% \pm 0\,\%$	$96\,\% \pm 3\,\%$
RTX6000	$94\,\% \pm 6\,\%$	$100\,\% \pm 0\,\%$	$100\,\% \pm 0\,\%$	$95\,\% \pm 6\,\%$

4 Evaluation

Equipped with an approach for exploiting numerical deviations, we are ready to empirically investigate hardware-triggered backdoors. First, we assess their efficacy for a single target input and a pair of hardware devices. We then generalize the setup to multiple target inputs and groups of devices. Lastly, we examine whether the attack vector is exploitable on non-GPU devices. We further include an ablation study of the components in Appendix F. To foster reproducibility, we release the source code to reproduce our experiments at *https://blinded-for-review*.

Evaluation setup. We consider five common Nvidia GPUs, spanning four architectural generations (Appendix B). These devices differ in microarchitectural details and supported numerical formats, making them well suited for studying hardware-dependent behavior. Moreover, we consider float32, float16, and bfloat16 as widely used numerical formats on these devices. We employ four vision models: *ResNet-18* (He et al., 2016) and *EfficientNetV2-S* (Tan & Le, 2021), which primarily rely on convolutional layers, as well as a *Vision Transformer (ViT)* with a 32×32 input patch size (Dosovitskiy et al., 2021). Lastly, we measure attack efficacy on *SigLIP* (Zhai et al., 2023), a zero-shot classification vision language model. All models are initialized from public pretrained weights, and all experiments are conducted on *ImageNet* (Deng et al., 2009). Target inputs are sampled uniformly at random from the training set so as not to affect clean performance. Full details can be found in Appendix C.

Attack setup. We follow the two-step attack procedure described in Section 3. After a preliminary study, we fix $\beta = 0.1$ and $\gamma = 10\,000$, and use an adaptive schedule for α to gradually adjust the influence of the decision-boundary term with 500 gradient descent steps per iteration. Moreover, we set the number of bit flips to $k = 5$ and the number of candidate models to $m = 256$, with 128 candidates refined with permutations and 128 with bit flips. We define $\rho = 95\%$ as the minimum performance that must be retained. In this configuration, the attack already yields satisfactory results after six iterations, and we therefore use it throughout all experiments. Finally, all experiments use a batch size of one to avoid numerical deviations induced by batching.

4.1 Attack Efficacy

As a first experiment, we investigate the efficacy of our backdoor across pairwise combinations of hardware platforms. In particular, for each pair of devices, we conduct the attack 100 times using independently sampled target inputs. Each run starts from a fresh copy of the pretrained model and applies up to six iterations of the two-stage approach described in Section 3. We repeat this experiment for float32, float16, and bfloat16 as model data types. To accommodate the large size of SigLIP, we limit the number of modifiable parameters as described in Appendix E.

Results. The results for float32 are summarized in Table 1, while the corresponding measurements for float16 and bfloat16 are reported in Appendix D. We find that hardware-triggered backdoors can be reliably embedded into almost all evaluated models, GPUs and data types. The sole exception is the A100 and A100-MIG40 pair on ResNet. In this case, the two devices exhibit bit-identical behavior, as the virtualization on the A100-MIG40 does not affect model behavior. For all other pairs, we find parameterizations for above 94% of cases. Furthermore, all backdoored models retain an accuracy close to original model performance. Interestingly, the backdoors created by our approach are even highly effective under full-precision data types (float32), despite prior work suggesting increased numerical precision as a potential mitigation against hardware-induced effects (Yuan et al., 2025).

Table 2: Success rates of finding a parameterization with increasing *numbers of target inputs*.

$\lvert\hat{\mathbf{X}}\rvert$	ViT	ResNet	EfficientNet
1	$99\% \pm 1\%$	$100\% \pm 0\%$	$100\% \pm 0\%$
2	$93\% \pm 1\%$	$98\% \pm 1\%$	$98\% \pm 1\%$
3	$79\% \pm 1\%$	$81\% \pm 2\%$	$91\% \pm 1\%$
4	$54\% \pm 2\%$	$55\% \pm 4\%$	$69\% \pm 2\%$
5	$28\% \pm 1\%$	$29\% \pm 4\%$	$41\% \pm 2\%$

Table 3: Success rate for finding parameterizations in the *one-vs-rest* case.

GPU	ViT	ResNet	EfficientNet
H100	64 %	99 %	94 %
A100	90 %	99 %	98 %
A100-MIG40	93 %	—	94 %
A40	69 %	90 %	97 %
RTX6000	66 %	96 %	99 %

4.2 Multiple Target Inputs

Thus far, we have focused on creating backdoors for a single target input. In practice, however, an attacker may wish to implant multiple backdoors into the same model, for instance to increase the likelihood of activation or to target several inputs simultaneously. To this end, we generalize the optimization objective in Equation (5) from a single input to a set of targets $\hat{x} \in \hat{X}$ optimizing over the sum of the inputs' losses. We repeat the previous experiment using this objective with $|\hat{X}| \in 2, 3, 4, 5$. As before, each configuration is evaluated over 100 independent runs, with $\hat{X}$ sampled uniformly at random from the training set. Note that in this setting the target images are likely unrelated, and the attack therefore needs to induce independent local manipulations of the decision boundary.

Results. Table 2 shows the success rates when increasing the amount of target inputs. We observe that, as the number of target increases, the success rate decreases across all models. While backdoors can be realized with success rates above 50% for up to four inputs, larger target sets render the attack ineffective. This behavior is intuitive: jointly positioning multiple inputs near their respective decision boundaries while preserving overall model utility requires balancing potentially conflicting objectives.

4.3 One-vs-Rest Trigger

Similar to the multiple-target setting, an attacker may also seek more selective control over the target hardware, for instance by activating malicious behavior on exactly one device while all others continue to exhibit benign behavior. We denote this setting as "one-vs-rest triggers".

For a selected platform h_1, the attacker aims to induce a misclassification, while all remaining platforms $h_{>1}$ must retain the correct prediction. Compared to the pairwise case, this setting is strictly more challenging, as the backdoor must remain dormant across multiple non-target platforms simultaneously. As before, each experiment is repeated 100 times with independently sampled target inputs. We treat each GPU architecture once as the target platform and group all remaining platforms as non-targets. Since the A100-MIG40 performs bit-identical to the A100 on ResNet, we exclude it for this model.

Results. The results of this experiment are shown in Table 3. We find that one-vs-rest triggers can be embedded across many devices and models, with success rates exceeding 90% for most configurations. The backdoors are less effective when targeting ViT on the H100, A40, or RTX6000 GPUs, but still achieve success rates above 60%. Overall, these results demonstrate that hardware-triggered backdoors can be made selective, such that it is only activated on a specific device.

4.4 Cross-Platform Generalization

Using the same experimental setup as in Section 4.1, we examine whether hardware-triggered backdoors generalize across computing environments. Specifically, we evaluate the same model across GPU- and CPU-based platforms. For GPU-based platforms, we consider NVIDIA H100 and A100 GPUs, while for CPU-based computing, we consider the Intel MKL (Intel, 2025) and OpenBLAS (Xianyi et al., 2012) libraries. We evaluate all pairwise combinations of GPU and CPU platforms to assess cross-platform generalization.

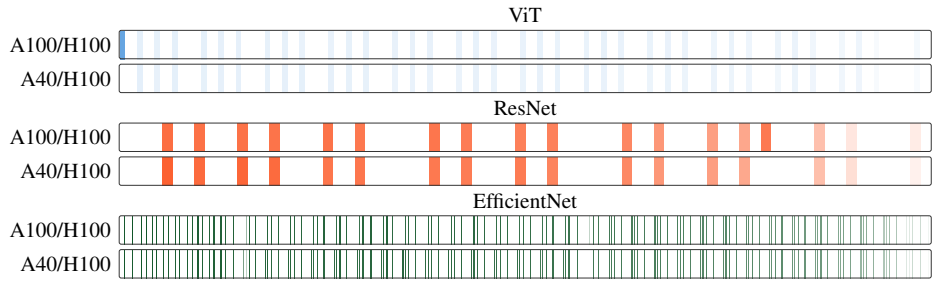

Figure 2: Aggregated logit differences $\Delta(h_1 \xrightarrow{i} h_2)$ over multiple backdoored models of the same architecture when patching activations from an A100 to an H100. Darker colors indicate stronger impact of the layer.

Results. For both ResNet and EfficientNet the success rate of 100% transfers to the CPU-GPU cross-platform experiments, as well as for the combinations including A100 on ViT. The only exceptions are the pairs H100-MKL and H100-OpenBLAS on ViT with success rates of 96% and 95% respectively. These high success rates between GPU and CPU are in line with prior work by Möller et al. (2025): Experiments on devices by a single manufacturer differ in hardware features, whereas cross-platform attacks induce numerical deviations from differences in software libraries.

5 Causal Localization

Building on the demonstrated efficacy of hardware-triggered backdoors, we next examine *where* hardware-dependent behavior arises within the backdoored model. That is, we aim to localize the latent trigger of the backdoors. For this analysis, we build on *activation patching* (Vig et al., 2020; Meng et al., 2022), which determines causal influence by replacing internal activations. In particular, we adapt this idea to a cross-hardware setting by replacing the execution of individual layers across devices. This allows us to isolate how layer-specific deviations affect the prediction.

Specifically, for a given backdoored model and a pair of hardware platforms h_1 and h_2, we execute the target input $\hat{x}$ on h_1 up to layer i. The resulting activation is then injected as input to layer $i+1$ on h_2, where execution continues to the output. We denote this mixed execution as $h_1 \xrightarrow{i} h_2$. To measure how far intermediate predictions are from one class to the other, we can compute their logit difference,

$$\delta(\hat{x}; h_1 \xrightarrow{i} h_2) = f(\hat{x}; h_1 \xrightarrow{i} h_2)_a - f(\hat{x}; h_1 \xrightarrow{i} h_2)_b \tag{8}$$

where a is the class predicted on h_1 and b is the class predicted on h_2, with $a \neq b$. By definition, we then have $\delta(\hat{x}; h_1 \xrightarrow{0} h_2) < 0$ and $\delta(\hat{x}; h_1 \xrightarrow{L} h_2) > 0$ for a model with L layers. That is, class a attains a larger logit on h_1 than on h_2, and vice versa for b when no patching occurs.

With the help of the logit differences, we can create a trace over all layers of a model, indicating how hardware-induced deviations evolve during inference. An example of traces is shown in Appendix G. We observe that the traces behave significantly differently across the three model architectures. For ViT, the shift in differences is dominated by the *first* layer. This layer corresponds to the convolutional patch embedding, which is the only convolution in the model and introduces the largest deviations. For EfficientNet and ResNet, the flipped prediction emerges as a cumulative effect *across* layers. In these architectures, the logit differences resemble a random walk that occasionally crosses the decision boundary.

To capture systematic effects rather than behavior of individual models, we aggregate these traces across multiple backdoored models of the same architecture and quantify the average contribution of each layer to the prediction difference:

$$\Delta(h_1 \xrightarrow{i} h_2) = \sum_{\hat{x}} |\delta(\hat{x}; h_1 \xrightarrow{i} h_2) - \delta(\hat{x}; h_1 \xrightarrow{i-1} h_2)|. \tag{9}$$

Figure 2 shows the resulting average layer differences for two hardware pairings and confirms the trends observed in the individual traces. Different hardware combinations exhibit distinct profiles. For

instance, the convolutional layer in ViT shows no measurable implementation differences between the H100 and A100/A40, even though the same operation induces deviations in EfficientNet and ResNet. Notably, even when only small deviations from linear and attention layers remain for ViT on these platforms, the success rate does not decrease.

Takeaway. Our analysis reveals two key insights. First, hardware-dependent deviations arise through different patterns in the models. They may originate in early layers as well as emerge from the accumulation of many small deviations across layers. Second, regardless of where these differences originate, even very small hardware-dependent effects can be sufficient to flip the prediction once the model operates in a sensitive regime. Motivated by this observation, we further examine in Appendix E whether the attack can be restricted to modifications of individual layers.

6 COUNTERMEASURES

We proceed to study countermeasures against hardware-triggered backdoors and evaluate their effectiveness. We refer to Appendix H for more details and formal descriptions of the evaluated factors.

Input perturbation. As a first line of defense, we investigate how perturbing the inputs with additive noise of increasing magnitude, measured in ULPs (units in the last place), affects backdoor success. We find that the backdoor is robust to small input perturbations and does not rely on an exact bit pattern. Even with the defense the Vision Transformer remains susceptible at low perturbation rates, while the defense is slightly more effective for ResNet and EfficientNet. Around 10^3 ULPs, all models converge to a remaining success rate near 30%. Only at larger perturbation levels does backdoor effectiveness decrease substantially, dropping below 14% at 10^4 ULPs and to approximately 1% or less at 10^5 ULPs across architectures. While sufficiently large perturbations can suppress the attack, they also introduce substantial distortion to the input, making this approach impractical.

Varying batch size. Thus far, we have focused on inference with batch size 1. In practice, however, models are often evaluated in batches, which can alter execution order and numerical behavior. To assess the robustness of hardware-triggered backdoors under this variation, we duplicate the target input $\hat{x}$ into batches of size $k \in \{1, \ldots, 256\}$ and measure success rates for different values of k, unknown to the adversary. Across model architectures and hardware pairs, we observe four regimes: success remains near 100%, drops to around 50%, drops further to 20%, or collapses near 0%. Consequently, batching does not provide a reliable mitigation, as it may suppress backdoors in some deployments while leaving others largely unaffected.

Replacing data types. While the attack can reliably embed backdoors across numerical formats, deployments may downcast high-precision models at inference time. When executing backdoored float32 models using mixed-precision inference, we find that success rates drop substantially but remain non-zero (approximately 25% for ViT, 20% for EfficientNet, and 10% for ResNet). This indicates that mixed-precision inference weakens hardware-triggered backdoors, though it does not reliably eliminate them in practice.

Additional fine-tuning. Finally, we consider an active modification of the model as a defense: fine-tuning on clean data. As shown in Table 4, a single gradient step removes most backdoors across model, with success diminishing further over additional steps. These results suggest that hardware-triggered backdoors can be effectively erased through continued training, but only if the defender actively modifies the model prior to deployment.

Table 4: Backdoor success after fine-tuning.

# Steps	ViT	ResNet	EfficientNet
1	5.98 %	0.34 %	0.00 %
2	2.24 %	0.11 %	0.20 %
3	0.62 %	0.22 %	0.00 %

Overall, these results show that hardware-triggered backdoors remain effective under moderate deployment variability and are not reliably disrupted by incidental changes, such as batching or mixed precision. In contrast, active intervention that modifies the model reduces backdoor success rates substantially and is our recommended approach for practical mitigation of the attack.

7 Related Work

Numerical imprecision. A growing body of work has studied how floating-point arithmetics introduce numerical variation during inference (Schlögl et al., 2024; Yuan et al., 2025). In adversarial settings, such variability has been leveraged to create inconsistencies or undermine theoretical guarantees. For example, Jia & Rinard (2021) exploit numerical differences to evade neural network verification, Möller et al. (2025) craft inputs that yield inconsistent predictions across software backends, and Zhang et al. (2025) fingerprint inference pipelines based on floating-point behavior. Unlike these approaches, our work examines how numerical imprecision can be exploited to create backdoors within the learning models themselves.

Hardware attacks. Another branch of research has studied hardware-based attacks against machine learning systems. These include bit-flip attacks (Rakin et al., 2019; Yao et al., 2020; Li et al., 2024), which manipulate a model's parameters by inducing memory faults, and side-channel attacks (Batina et al., 2019; Hua et al., 2018), which leak information about model parameters through hardware components. Both classes of attacks operate at inference time and require the adversary to compromise the victim's system. Our attack departs from this paradigm: the malicious behavior is embedded in the model, and triggering it requires no system-level access.

Backdoor attacks. Classic backdoor attacks implant malicious behavior during training so that a model behaves normally unless a trigger appears in the input (Liu et al., 2018; Gu et al., 2019; Tang et al., 2020). Later work has developed more stealthy variants, including data and loss manipulation (Shumailov et al., 2021; Bagdasaryan & Shmatikov, 2021), payload and compression-based mechanisms (Li et al., 2021; Tian et al., 2022), and attacks that make use of software or hardware manipulations (Clifford et al., 2024; Li et al., 2025).

Most closely to our work are approaches that exploit numerical effects in a supply-chain setting. For example, Chen et al. (2025) demonstrate that benign compiler transformations can be abused to introduce malicious behavior, while other works introduce backdoors induced solely through quantization effects (Hong et al., 2021; Ma et al., 2023). In contrast, we show that hardware-dependent deviations themselves can act as a latent trigger: the backdoor is neither encoded in the input nor tied to a specific compiler or quantization, but instead emerges from the interaction between a backdoored model and its execution hardware.

8 Limitations

Although we show that floating-point deviations can be reliably exploited to embed malicious behavior into a model, our results rely on specific assumptions about the adversary and deployment setting. In particular, our threat model assumes that an adversary can modify model parameters prior to deployment and has partial knowledge of a stable execution environment. While we find CUDA versions 12.6, 12.8, and 13.0 to be bit identical in outputs, changes in the software stack might decrease the effectiveness of the attack. Furthermore, our analysis of countermeasures indicates that, while the resulting backdoors exhibit a degree of robustness to incidental perturbations, they remain vulnerable to active interference by a defender. In particular, interventions that introduce additional numerical variability or modify the execution pipeline can substantially reduce attack effectiveness.

9 Conclusion

Hardware acceleration is an integral component of machine learning systems, yet its numerical behavior is often treated as a negligible detail. We show that this view is misleading: minor numerical differences can be used as a per-device footprint to activate malicious behavior. Our findings indicate that the security of machine learning must be viewed in a broader context. Security risks extend beyond models and algorithms to the full computing stack on which they operate. Developing methods to secure this end-to-end stack will be critical for the safe deployment of future machine learning systems. On a higher level, our findings reveal a covert mechanism for embedding hardware-specific behavior into systems.

AI USE STATEMENT

In this work, we used generative AI tools to polish our writing and find grammar errors. Every suggested text change was implemented by the authors and every section of the final text was manually reviewed. LLMs were further used to provide code for simple and/or standard methods. No important, original, or non-standard component was created by an LLM. The implementation was tested for correctness by multiple authors. Critical sections of the code were manually audited by multiple authors. We have not used generative AI tools for any other tasks with required disclosure. We take responsibility for the final content of this work, including text, claims or artifacts.

ETHICS STATEMENT

A potential risk of this work is that an attacker could misuse the proposed technique to manipulate real-world models. While such misuse cannot be ruled out, we reduce this risk by also introducing defenses, some of which are readily applicable in practice. In addition, we raise awareness of a hidden attack surface that arises from the interplay between hardware and machine learning. We hope that our work encourages practitioners to consider this attack surface and, where appropriate, apply suitable countermeasures, including the proposed defenses. We believe that identifying vulnerabilities and failure modes is a necessary step toward building safer and more reliable machine learning infrastructure.

REPRODUCIBILITY STATEMENT

We describe all major components of our method and publish our implementation as part of the supplementary materials (including the documentation to set up the experiments). We further describe all models, datasets, and experimental setups in the main text as well as the appendix of this work.

REFERENCES

Eugene Bagdasaryan and Vitaly Shmatikov. Blind backdoors in deep learning models. In *USENIX Security Symposium*, pp. 1505–1521, 2021.

Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. {CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*, pp. 515–532, 2019.

Simin Chen, Jinjun Peng, Yixin He, Junfeng Yang, and Baishakhi Ray. Your compiler is backdooring your model: Understanding and exploiting compilation inconsistency vulnerabilities in deep learning compilers. *arXiv preprint arXiv:2509.11173*, 2025.

Eleanor Clifford, Ilia Shumailov, Yiren Zhao, Ross Anderson, and Robert Mullins. Impnet: Imperceptible and blackbox-undetectable backdoors in compiled neural networks. In *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pp. 344–357. IEEE, 2024.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255. IEEE, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Sanghyun Hong, Michael-Andrei Panaitescu-Liess, Yigitcan Kaya, and Tudor Dumitras. Qu-antization: Exploiting quantization artifacts for achieving adversarial outcomes. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:9303–9316, 2021.

Weizhe Hua, Zhiru Zhang, and G Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

IEEE. Standard for Floating-Point Arithmetic. *IEEE Std 754-2019*, 2019.

Intel. Math kernel liberary. *https://software.intel.com/en-us/intel-mkl/*, 2025.

Kai Jia and Martin Rinard. Exploiting verified neural networks via floating point numerical error. In *International Symposium on Static Analysis (SAS)*, pp. 191–205. Springer, 2021.

Shaofeng Li, Xinyu Wang, Minhui Xue, Haojin Zhu, Zhi Zhang, Yansong Gao, Wen Wu, and Xuemin Sherman Shen. Yes,{One-Bit-Flip} matters! universal {DNN} model inference depletion with runtime code fault injection. In *USENIX Security Symposium*, pp. 1315–1330, 2024.

Xiang Li, Ying Meng, Junming Chen, Lannan Luo, and Qiang Zeng. Rowhammer-based trojan injection: One bit flip is sufficient for backdooring dnns. In *USENIX Security Symposium*, pp. 6319–6337, 2025.

Yuanchun Li, Jiayi Hua, Haoyu Wang, Chunyang Chen, and Yunxin Liu. Deeppayload: Black-box backdoor attack on deep learning models through neural payload injection. In *IEEE/ACM International Conference on Software Engineering (ICSE)*, pp. 263–274. IEEE, 2021.

Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *Network And Distributed System Security Symposium (NDSS)*. Internet Soc, 2018.

Hua Ma, Huming Qiu, Yansong Gao, Zhi Zhang, Alsharif Abuadbba, Minhui Xue, Anmin Fu, Jiliang Zhang, Said F Al-Sarawi, and Derek Abbott. Quantization backdoors to deep learning commercial frameworks. *IEEE Transactions on Dependable and Secure Computing*, 21(3):1155–1172, 2023.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:17359–17372, 2022.

Jonas Möller, Lukas Pirch, Felix Weissberg, Sebastian Baunsgaard, Thorsten Eisenhofer, and Konrad Rieck. Adversarial inputs for linear algebra backends. In *International Conference on Machine Learning (ICML)*, 2025.

Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1211–1220, 2019.

Alex Schlögl, Nora Hofer, and Rainer Böhme. Causes and effects of unanticipated numerical deviations in neural network inference frameworks. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2024.

Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6106–6116, 2018.

Ilia Shumailov, Zakhar Shumaylov, Dmitry Kazhdan, Yiren Zhao, Nicolas Papernot, Murat A Erdogdu, and Ross J Anderson. Manipulating sgd with data ordering attacks. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:18021–18032, 2021.

Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning (ICML)*, pp. 10096–10106. PMLR, 2021.

Ruixiang Tang, Mengnan Du, Ninghao Liu, Fan Yang, and Xia Hu. An embarrassingly simple approach for trojan attack in deep neural networks. In *ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 218–228, 2020.

Yulong Tian, Fnu Suya, Fengyuan Xu, and David Evans. Stealthy backdoors as compression artifacts. *IEEE Transactions on Information Forensics and Security*, 17:1372–1387, 2022.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:12388–12401, 2020.

Zhang Xianyi, Wang Qian, and Zhang Yunquan. Model-driven level 3 blas performance optimization on loongson 3a processor. In *IEEE 18th international conference on parallel and distributed systems*, pp. 684–691, 2012.

Fan Yao, Adnan Siraj Rakin, and Deliang Fan. {DeepHammer}: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1463–1480, 2020.

Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. Understanding and mitigating numerical sources of nondeterminism in llm inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11975–11986, 2023.

Cheng Zhang, Hanna Foerster, Robert D. Mullins, Yiren Zhao, and Ilia Shumailov. Hardware and software platform inference. In *International Conference on Machine Learning (ICML)*, 2025.

A Example of Deviations

The following Python code provides a simple example of a matrix multiplication within the computation of the squared Frobenius norm. When executed on different Nvidia devices, the large number of additions leads to slight deviations, as discussed in Section 2.

Listing 1: Computation of the squared Frobenius norm.

```python
import torch

M = torch.full(
    (100, 100),
    0.01,
    dtype=torch.float32,
    device="cuda"
)

print(torch.trace(torch.matmul(M, M)).item())
```

B Hardware Platforms

Table 5: Overview of considered GPU platforms.

GPU	Architecture	Chip
Nvidia H100	Hopper	GH100
Nvidia A100	Ampere	GA100
Nvidia A100 (MIG-40GB)	Ampere	GA100
Nvidia A40	Ampere	GA102
Nvidia Quadro RTX 6000	Turing	TU102

We consider the five GPUs spanning three architectural generations with four different chipsets listed in Table 5. We focus on same-vendor (Nvidia) models due to their widespread use in both academia and industry.

A particularly challenging case in our setup is the A100 (MIG-40GB). While it is identical in hardware to the standard A100, the use of Multi-Instance GPU (MIG) introduces a virtualization layer that slightly alters the execution environment. As we show in Section 4.1, this difference can be sufficient to trigger backdoors in some models. Other models, however, remain unaffected and exhibit bit-identical behavior across these two devices.

C Models and Data

Table 6: Models used in our experiments

Model	Parameters	Top-1 Accuracy	Weights / Revision
ViT-B/32	∼88.2M	75.9 %	IMAGENET1K_V1
ResNet-18	∼11.7M	69.8 %	IMAGENET1K_V1
EfficientNetV2-S	∼21.5M	81.3 %	IMAGENET1K_V1
SigLIP So400M Patch14-384	∼877M	79.2 %	9fdffc5

We employ four different vision models in our experiments. The first three, Vision Transformer (ViT), ResNet, and EfficientNet, are standard classification models for ImageNet. ResNet and EfficientNet primarily rely on convolutions, where as ViT uses attention mechanisms for image classification. All models are initialized using the default pre-training configurations from PyTorch / TorchVision for the ImageNet-1K dataset.

In contrast to the other models, SigLIP is a vision-language model trained using a contrastive objective. It does not produce class logits directly. Instead, we perform zero-shot classification by constructing textual prompts of the form `"a photo of a {IMAGENET LABEL}"` and computing cosine similarity between image and text embeddings. These similarity scores are treated as surrogate logits and passed through a softmax for evaluation and backpropagation. The model is initialized with the weights from Hugging Face.

Model Links. The models along with their default pre-trained weights can be found under:

1. *https://docs.pytorch.org/vision/main/models/generated/torchvision.models.vit_b_32.html*
2. *https://docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html*
3. *https://docs.pytorch.org/vision/main/models/generated/torchvision.models.efficientnet_v2_s.html*
4. *https://huggingface.co/google/siglip-so400m-patch14-384*

Model Licenses. The employed SigLIP model is distributed via Hugging Face under Apache license 2.0. The remaining models are bundled with PyTorch / TorchVision, which both use a BSD-3-Clause.

1. SigLIP: *https://huggingface.co/google/siglip-so400m-patch14-384*
2. PyTorch: *https://github.com/pytorch/pytorch/blob/main/LICENSE*
3. Torchvision: *https://github.com/UiPath/torchvision/blob/master/LICENSE*

Dataset License. All ImageNet-pretrained models rely on the ImageNet dataset, which is distributed under a custom, non-commercial research license: *https://image-net.org/download.php*

D Attack Performance across Data Types

Our attack methodology is agnostic to the hardware platform, model architecture, and the employed floating-point data type. For brevity, we report results only for float32 in Section 4.1. Results for float16 and bfloat16 are provided in Table 7 and Table 8, respectively.

Table 7: Success rates of finding a model parameterization for a pre-chosen input (float16).

GPU	ViT	ResNet	EfficientNet
H100	99.75 %	100.00 %	100.00 %
A100	100.00 %	75.75 %	100.00 %
A100-MIG40	100.00 %	75.75 %	100.00 %
A40	100.00 %	100.00 %	100.00 %
RTX6000	99.75 %	100.00 %	100.00 %

Table 8: Success rates of finding a model parameterization for a pre-chosen input (bfloat16).

GPU	ViT	ResNet	EfficientNet
H100	100.00 %	100.00 %	100.00 %
A100	100.00 %	75.50 %	100.00 %
A100-MIG40	100.00 %	75.50 %	100.00 %
A40	99.75 %	100.00 %	100.00 %
RTX6000	99.75 %	100.00 %	100.00 %

E Single-Layer Attack

Our causal analysis in Section 5 reveals how the embedded backdoors affect different layers of the considered learning models. However, it does not address whether modifications to these layers are strictly *necessary* for the attack. In principle, changes to a layer could still cause a backdoor, even if that layer produces identical results across hardware platforms. The changes can shift the activations such that existing hardware-dependent deviations in other layers align with the attacker goal.

To test this hypothesis, we repeat the attack on ViT from Section 5 while now restricting all parameter modifications to the initial convolution. Since the first layer does not support permutation-based modifications, we perform the attack in a reduced configuration using only bit flips. Note that the initial layer for ViT has numerical deviations for some hardware combinations while being identical for others as shown in Figure 2.

We find that restricting the attack to a single layer does not reduce its effectiveness. The success rate changes only marginally, from $96\,\% \pm 5\,\%$ to $95\,\% \pm 5\,\%$. This result indicates that neither the presence nor the magnitude of hardware-triggered differences in the modified layer is a prerequisite for a successful attack. When the initial convolution of the ViT is identical, attention and linear layers are the only causes of slight numerical deviations. Therefore, we conclude that hardware-triggered backdoors can be implemented in any model that exhibits deviations on the target hardware platform, even when these are minor.

We use this insight to improve our implementation when evaluating SigLIP by only targeting the heavy-impact convolution layer of the model (`vision_model.embeddings.patch_embedding`). Since, each inference must be done on a separate machine (with the respective hardware), synchronizing model states after changes is a non-trivial overhead. By limiting changes to single layers, we severely reduce the communication overhead.

F Ablation Study

We conduct an ablation study that investigates the two steps of our attack in detail. To this end, we evaluate four attack variants: a *base* variant that applies only the first step; a *permutation* variant and a *perturbation* variant that apply either mechanism as second step on top; and a *full* variant that combines both mechanisms. For all variants, we focus on ViT, as it is the only architecture for which both mechanisms are applicable due to its consecutive linear layers.

Results. Our ablation study demonstrates the interplay of our attack's components. The base variant alone reaches a success rate of 56%, while its combination with either permutation or perturbation increases the success rate to 90% and 94%, respectively. The full variant finally achieves the highest

success rate, 96%, indicating that implicit and explicit modifications act in a complementary manner when refining numerical deviations.

G Single-Layer Analysis

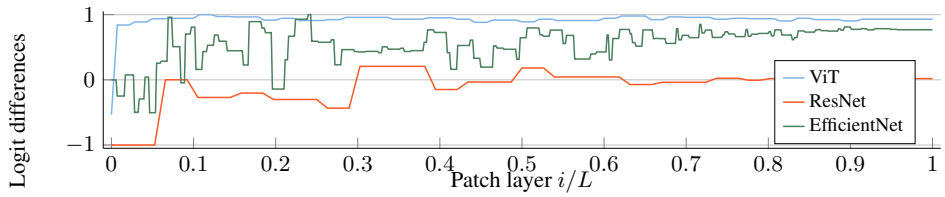

Figure 3: Layer-wise logit differences for backdoored models. The differences $\delta(\hat{x}; h_1 \xrightarrow{i} h_2)$ move from the original class to the target class as activations are patched from an A100 to an H100.

Figure 3 shows traces for backdoored models targeting an A100 and an H100, where the differences are normalized to $[-1, +1]$ for visualization. The backdoor sample moves from one class, when the logit differences is negative, to another class, when the logit difference is positive. For ViT, the first layer dominates the differences and subsequent layers only marginally influence the logit value; i.e., the first layer already flips the prediction. Both EfficientNet and ResNet have the largest jumps in logit differences due to their convolutions. Since, EfficientNet has a deeper architecture than the other models, the larger jumps appear more frequent.

H Details of Defense Evaluation

In Section 6, we evaluate our backdoor against different defenses. Specifically, we examine how many of the backdoors $\hat{x} \in \hat{\mathcal{X}}$ uncovered in the experiment in Section 4 remain effective after modifying a property of the inference environment. Formally, for each experiment we define a success metric that accounts for the modified property and report the average success rate over all backdoors $\hat{x} \in \hat{\mathcal{X}}$.

Input perturbation. For this defense, we apply a perturbation δ drawn from a uniform distribution to each target input $\hat{x}$. The magnitude of δ is defined in units of the last place (ULPs), with $\|\delta\|_\infty = d$. For a target input $\hat{x}$ and a perturbation δ, we measure success simply as

$$\mathbb{1}\big[F_\theta(\hat{x} + \delta; h_1) \neq F_\theta(\hat{x} + \delta; h_2)\big]. \tag{10}$$

Batch size defense. For the defense based on batch size, we need to consider two possible reasons why a backdoor may fail. First, the backdoor may fail due to changes in numerical deviations induced by the chosen *batch size*. Second, the backdoor may also fail because the target input appears at a different *batch index* than anticipated by the adversary. To account for both effects, we average the success rate over all pairs of batch indices for a given batch size k. Formally, this is computed as

$$\frac{1}{k^2} \sum_{i=1}^{k} \sum_{j=1}^{k} \mathbb{1}\big[F_\theta(\hat{x}_i; h_1) \neq F_\theta(\hat{x}_j; h_2)\big], \tag{11}$$

where $\hat{x}_i$ and $\hat{x}_j$ denote the target input at batch index i and j, respectively.

Data type defense. For the defense based on floating-point data types, we define a function $\downarrow$ that represents a dynamic downcasting of the high-precision float32 model to float16 or bfloat16 using PyTorch's automatic mixed precision feature. We then measure the success rate of backdoors on the downcasted model parameters as

$$\mathbb{1}\big[F_{\downarrow\theta}(\hat{x}; h_1) \neq F_{\downarrow\theta}(\hat{x}; h_2)\big]. \tag{12}$$

Fine-tuning defense. In the final experiment, the user actively modifies the model weights after the attacker has installed the backdoor. To this end, we perform fine-tuning for n steps using stochastic gradient descent with a learning rate of 10^{-4} and a momentum of 0.9, on randomly sampled batches of size 256 from the ImageNet training set. We denote the resulting fine-tuned model by $\dot{\theta}$ and measure backdoor success as follows

$$\mathbb{1}\left[F_{\dot{\theta}}(\hat{x}; h_1) \neq F_{\dot{\theta}}(\hat{x}; h_2)\right]. \tag{13}$$