When Side-Channel Attacks Break the Black-Box Property of Embedded Artificial Intelligence

Benoît Coqueret, Mathieu Carbone, Olivier Sentieys, Gabriel Zaid

When Side-Channel Attacks Break the Black-Box Property of Embedded Artificial Intelligence

Benoît Coqueret
Thales ITSEF
Toulouse, France
University of Rennes, INRIA, IRISA
Rennes, France
benoit.coqueret@thalesgroup.com

Mathieu Carbone
Thales ITSEF
Toulouse, France
mathieu.carbone@thalesgroup.com

Olivier Sentieys
University of Rennes, INRIA, IRISA
Rennes, France
olivier.sentieys@inria.fr

Gabriel Zaid
Thales ITSEF
Toulouse, France
gabriel.zaid@thalesgroup.com

ABSTRACT

Artificial intelligence, and specifically deep neural networks (DNNs), has rapidly emerged in the past decade as the standard for several tasks from specific advertising to object detection. The performance offered has led DNN algorithms to become a part of critical embedded systems, requiring both efficiency and reliability. In particular, DNNs are subject to malicious examples designed in a way to fool the network while being undetectable to the human observer: the adversarial examples. While previous studies propose frameworks to implement such attacks in black box settings, those often rely on the hypothesis that the attacker has access to the logits of the neural network, breaking the assumption of the traditional black box. In this paper, we investigate a real black box scenario where the attacker has no access to the logits. In particular, we propose an architecture-agnostic attack which solve this constraint by extracting the logits. Our method combines hardware and software attacks, by performing a side-channel attack that exploits electromagnetic leakages to extract the logits for a given input, allowing an attacker to estimate the gradients and produce state-of-the-art adversarial examples to fool the targeted neural network. Through this example of adversarial attack, we demonstrate the effectiveness of logits extraction using side-channel as a first step for more general attack frameworks requiring either the logits or the confidence scores.

CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures**.

AISec '23, November 30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0260-0/23/11…$15.00
https://doi.org/10.1145/3605764.3623903

KEYWORDS

Deep learning, Embedded systems, Black box attack, Side-channel attack, Adversarial examples

ACM Reference Format:
Benoît Coqueret, Mathieu Carbone, Olivier Sentieys, and Gabriel Zaid. 2023. When Side-Channel Attacks Break the Black-Box Property of Embedded Artificial Intelligence. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec '23), November 30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3605764.3623903

1 INTRODUCTION

For the past ten years, Deep Neural Networks (DNNs) have improved the automation level in many tasks from image generation [23] to object detection for autonomous driving [49], and Deep Learning (DL) tools are more and more considered by developers. With the parallel efforts provided by dedicated hardware designers to make DNNs affordable resource-wise [52], it is not surprising that DL is now thriving in the embedded systems world. Such systems are often used in critical environments [3, 5], making it vital to be able to assert proof of security and reliability. However, DNNs bring both new targets and new threats, thus challenging these requirements. For instance, a typical DNN model necessitates a large amount of data and computational time to train and fine tune the model, making it a valuable intellectual property (IP) for the owner and a potential target for an attacker. Researches in the security of DNNs have brought to light vulnerabilities both in the physical implementation of these models (hardware level) and in the design of the algorithm (software level).

Those vulnerabilities can be mainly decomposed into three scenarios. First, model extraction attacks aim at stealing the trainable parameters (*e.g.*, weights) or the hyperparameters of the model (*e.g.*, neural network architecture) in order to create a clone, compromising the IP. Tramèr *et al.* [54] demonstrated the potential of this attack method by taking advantage of prediction APIs in order to copy a small DNN. In [6], Batina *et al.* proposed a hardware attack scenario based on side-channel attacks, in order to steal the IP and the inputs of a targeted embedded model. The practicability of the latter scenario has nevertheless been questioned when an attacker has to deal with real-world and complex DNN implementations

[34]. A second key consideration for the IP comes from the privacy of the input data during training and inference time. Model inversion attacks are designed to predict the inputs of the model, breaking the potential confidentiality of the input data. In their study, Fredriskon *et al.* [21] proved that one could learn sensitive genomic information from the patients, necessitating only the confidence scores of the model. Finally, security flaw in DNNs enables an opponent to disrupt the model during inference. This can be achieved by poisoning the data during training to misclassify or create backdoor during inference, as Yue *et al.* have shown [57]. This kind of weakness has also been exploited following a hardware attack strategy, namely, fault injection attack. In [39], Liu *et al.* inject faults *via* laser emission to successfully change the output of the embedded network causing a misclassification. But, this strategy can also be done by perturbing the inference data in an imperceptible way to the human observer but such as it fools the network. This scenario is known as evasion or adversarial attacks.

The generation of adversarial examples has been an active research field since its formulation for DL framework in 2013 by Szegedy *et al.* [53]. Since then, several methods, separated by the threat model of the attacker, have emerged. On one end of the spectrum, the attacker is supposed to have full knowledge of the DNN model and its implementation, this is the *white box assumption*. In this scenario, several attacks based on optimization problems perform well and allow an attacker to fool the model. Some of the most impactful methods are FGSM [24], JSMA [46] and C&W [14] attack. On the other end of the spectrum, the attacker is supposed to have no previous knowledge of the DNN model, this is the *black box assumption*. In this case, an attacker, referred as Eve in the rest of this work, can only interact with the DNN model in order to capture the predictions it provides. Therefore, she cannot generate adversarial examples by optimization, as the parameters of the model are unknown. A popular solution is to train, using the targeted model as an oracle, a substitute model [29] with the goal of copying the original. This DNN will allow the attacker to work in a white box setting and use the previously mentioned attack frameworks. We argue that this method is far less practical, as it offers lower transfer rate, necessitates higher distortion rate, or both [16]. A lower transfer rate will defeat the purpose of the substitute network by making it useless, and higher distortion will make the examples more noticeable contradicting with their definition and purpose. The performance might also be impacted by hyperparameters, such as the architecture, of the substitute model, increasing the complexity of the overall attack. Those issues have led researchers to improve the attack by adding assumption to the black box scenario. In particular, having access to the logits or the probability vector is one strong hypothesis that is commonly made in several frameworks [16, 55]. While generation of more powerful adversarial examples is made possible with this assumption, it is far from a practical one. Indeed, one developer can only return the predicted value without providing any access to the logits or the probability distribution. If so, the attacker can thus no longer use these methods.

Contributions: To solve the black box constraint, we propose a new attack framework, targeting embedded DL systems and combining software and hardware attacks. Our method exploits physical side-channel leakages to perform template attacks [15] and deep learning-based side-channel attacks (DLSCA) [12, 41] to extract the value of the logits vector by targeting the operations during the computation of the softmax layer. Since this layer is used to transform, independently from the previous ones, the logits into probabilities, our attack is free from any restricting assumption on the architecture. Added to the fact that the softmax function is commonly accepted as a standard for the majority of multiclass classifier, it is an ideal target. The key idea of our contribution is to add another step to state-of-the-art evasion attacks [16, 55] to produce a generic methodology that allows an attacker to generate adversarial examples in a black box setting, *i.e.*, without prior knowledge on the DNN parameters, including the logits or the probability vector. Our main contributions are:

- We introduce a new generic attack scenario which extracts the logits from an embedded DL system. Based on this strategy, an attacker takes advantage of side-channel attacks to improve the practicability of state-of-the-art black box adversarial attacks.
- We demonstrate the success of our work and its complementarity with state-of-the-art attacks by designing an end-to-end methodology to build powerful adversarial examples by solving the optimization problem proposed by Carlini and Wagner [14], via logits extraction and zeroth order optimization.
- To prove the practicability of our contribution, we implemented a real case scenario on a microcontroller and validated our strategy on a 8-bit quantized DNN using NNOM [40]. This experimental setup illustrates the benefits of side-channel attacks to mitigate a black-box scenario.

Although this paper focus on the generation of adversarial examples using the ZOO framework, we would like to emphasize that the logit extraction presented here is specific neither to this task (evasion attack), nor to this framework (ZOO). For example, both model extraction methods proposed in [50] and in [33] need access to the logit vectors to efficiently steal the model's parameters. With our methodology, an attacker could conduct side-channel attacks to extract the logits in a black-box settings before the application of the methods described in [50] and [33]. In other words, the main purpose of this contribution, is to demonstrate the suitability of hardware attacks in making software attacks more practicable under the black-box context by introducing a method to remove a previous assumption, *i.e.*, the access to the logits.

This paper is organized as follows. Section 2 presents the notations and the background needed through this study. We then describe the principle of our attack (*i.e.*, extraction of the logits) and our end-to-end methodology in Section 3. Finally, we introduce our experimental setup and evaluate our methodology in Section 4 before to conclude in Section 5.

2 BACKGROUND

In this section we introduce the necessary background for the comprehension of this paper. We first briefly present deep neural network and quantization. We then describe the two approaches that we will use in our framework: adversarial examples and side-channel attacks.

2.1 Notation

Let calligraphic letters $\mathcal{X}$ denote sets, the corresponding capital letters X (resp. bold capital letters) denote random variables (resp. random vectors $\mathbf{T}$) and the lowercases x (resp. $\mathbf{t}$) denote their realizations. We will use the following notation $\mathbf{T}[i]$ to describe the i-th elements of a vector $\mathbf{T}$. Throughout this paper, the function modeled by a DNN is denoted as $f : \mathcal{X} \rightarrow \mathcal{Y}$ and it characterizes its ability to classify a data $X \in \mathcal{X}$ (*e.g.*, an image) over a set of $|\mathcal{Y}|$ classes. The logits, also known as confidence scores, used to perform this classification will be defined by Z. To ease the identification of adversarial examples (resp. quantized data), the notation X^* (resp. $\tilde{X}$) will be considered. Finally, the probability of observing an event X is denoted by $\texttt{Pr}[X]$, such that a conditional probability of observing an event X knowing an event Y is denoted $\texttt{Pr}[X|Y]$.

2.2 Deep Neural Networks

Machine Learning uses statistical tool and optimization algorithms [22] to approximate a function which performs a *classification* or a *regression* task. In this paper, a particular focus is brought to the classification task. In this setting, a deep neural network is designed to approximate a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ which assigns a unique label $y \in \mathcal{Y}$ to a data $x \in \mathcal{X}$. To approximate a function f which correctly classifies each input, a set of labeled data $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \in (\mathcal{X} \times \mathcal{Y})^n$ is provided in order to monitor its configuration via a loss function and backpropagation. This process is known as *supervised learning*, the interested readers may refer to [22] to get further details on the training process of a DNN. To design the function f, a parametric approach which approximates linear and non-linear function is used. One solution consists in using a neural network, which is defined as the repetition of a base unit, the *neuron*, stacked by *layers*, forming a combination of simple functions. An example of the most simple architecture is provided in Figure 1. It is characterized by a set of fully-connected layers, which corresponds to the multi-layer perceptron (MLP). In particular, a neural network is composed by an *input layer* (*i.e.*, the identity over the input data), an *output layer* (*i.e.*, the function which computes the probability vector), and a set of *hidden layers* (*i.e.*, the functions which map the input data to a given class in $\mathcal{Y}$).

Once the function f is correctly designed, the purpose is to select among the possible output classes the most likely class y among the possible outputs included in $\mathcal{Y}$. To do so, the last layer is represented by a vector of confidence scores associated with each of the possible class. One way to normalize the confidence scores is to perform a softmax function in order to approximate a probability distribution $\texttt{Pr}[Y|X = x]$. Therefore, the decision-making of the DNN does not only provide the most likely candidate to solve the classification problem, but also the *a posteriori* probability of the remaining $|\mathcal{Y}| - 1$ other classes. This softmax function can be computed such that $s(\mathbf{Z}[i]) = \frac{e^{\mathbf{Z}[i]}}{\sum_{j=1}^{|\mathcal{Y}|} e^{\mathbf{Z}[j]}}$ where $Z[i]$ denotes the score related to the i-th class.

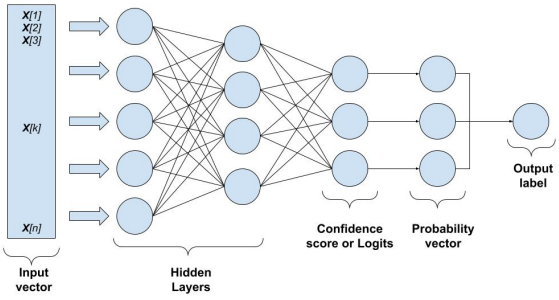


Figure 1: Example of an MLP neural network.

2.3 Quantization

As mentioned in the previous section, DNN models require to keep in memory large amount of data, from weight to intermediate values computed during the inference stage, such as activation value, which can be challenging in resource-constrained environments. One of the most common solution is quantization, which consists in using low precision arithmetic for the model's data. Quantization reduces the number of accesses to the off-chip memory and, as such, the overall power consumption is reduced [25]. Two main methodologies for quantization have emerged: Quantization Aware Training (QAT) and Post Training Quantization (PTQ). The first performs the training of the network while taking into account the quantization of the parameters and the latter quantizes the weights once the network is trained. While numerous complementary adaptation to DL algorithms have been proposed recently [19, 27, 28], in this paper, we focus on the PTQ solution.

2.4 Adversarial Examples

Evasion attack is a type of attack, where a malicious user tries to craft specific inputs such as they are not recognized correctly by the model while appearing normal to a human observer. These modified inputs are referred to as adversarial examples. Their generation is made by adding imperceptible perturbations (*e.g.*, noise) to the original data in order to force a prediction to a (specific) wrong class.

Definition 1 (Adversarial examples [9]). *Given a neural network defined by a function $f : \mathcal{X} \rightarrow \mathcal{Y}$, let $X \in \mathcal{X}$ be the original input, $(y, y') \in \{1, \ldots, |\mathcal{Y}|\}^2$ the expected label and the targeted adversarial class, then an adversarial example is defined as $\boldsymbol{X}^* = \boldsymbol{X} + \epsilon$ with*

$$\epsilon = \underset{\epsilon}{\text{argmin}} ||\epsilon||_p,$$
$$s.t f(\boldsymbol{X}^*) = y' \text{ (targeted attack)},$$
$$s.t f(\boldsymbol{X}^*) \neq y \text{ (untargeted attack)},$$

where $||\epsilon||_p$ is the l_p norm.

The generation of adversarial examples is often limited by the level of previous knowledge on the neural network, and in the case where a malicious user has full knowledge over the target, it can be seen as an optimization problem [14, 53].

Carlini and Wagner l2 attack (C&Wl2). In [14], the authors tested several objective functions and selected the following formulation for the optimisation problem. Given an objective function g_{obj}, an input $\mathbf{X}$ of dimension n and $\mathbf{X}^*$ the targeted adversarial example and y' the adversarial class:

$$\begin{aligned} \text{minimize } & \|\mathbf{X}^* - \mathbf{X}\|_2^2 + c \times g_{obj}(\mathbf{X}^*, y'), \\ \text{s.t } & \mathbf{X}^* \in [0,1]^n, \end{aligned}$$

Given an hyperparameter k set to 0 in our tests, as suggested by Carlini and Wagner [16], the best objective function make use of the logit vector $\mathbf{Z} \in \mathbb{R}^{|\mathcal{Y}-1|}$ and is given by

$$g_{obj}(\mathbf{X}, y') = \max(\max_{i \neq y'}(\mathbf{Z}(\mathbf{X})[i]) - (\mathbf{Z}(\mathbf{X})[y']), -k).$$

Their framework is still considered to produce state-of-the-art results in a white box setting [16], but as such requires full knowledge over the target which is not always a practical solution. On the other hand, in a complete black box setting where the attacker has no knowledge of the DNN parameters, the generated examples are weaker, meaning that they require bigger (and thus more noticeable) transformation of the inputs to produce the same perturbation in the output [16]. Studies have therefore tried to reduce the gap between the white box generated adversarial examples and the black box ones by using their ability to transfer from one model to another. In this scenario, the malicious user would perform the generation on a substitute model in the white box paradigm, and rely on the possibility of a transfer. Another strategy has been to increase the knowledge of the attacker, by allowing her to have access to the logit or the probability vector of the model, an estimation of the gradient can be made, and it is then possible to generate a white box-like adversarial example.

Zeroth Order Optimization (ZOO). Chen *et al.* [16] adapted the loss function from the *C&Wl2* attack to use the probability vector $\mathbf{F}$ instead of the logits. In their method, the objective function to minimize becomes

$$g_{obj}(\mathbf{X}, y') = \max(\max_{i \neq y'}(\log \mathbf{F}(\mathbf{X}))[i] - \log(\mathbf{F}(\mathbf{X}))[y'], -k).$$

They also used finite difference as a proxy for the gradients obtained via backpropagation in a white box settings. Given a scalar $h \in \mathbb{R}$ and a unit vector $\mathbf{E} \in \mathbb{R}^n$:

$$\frac{\partial f}{\partial \mathbf{X}_i} \approx \frac{f(\mathbf{X} + h\mathbf{E}[i]) - f(\mathbf{X} - h\mathbf{E}[i])}{2h}$$

Another similar framework is the SPSA attack [55], which makes direct use of the logits to approximate the gradient also *via* finite difference. Both methods were designed on full-precision network. However, in the context of embedded systems, recent researches have investigated the possibility to use them on quantized neural networks [9]. Quantization, especially quantization of the activation values, can deteriorate the generation of adversarial examples by rounding the original and the crafted inputs into the same quantization bucket. In their study, the authors investigate several frameworks to test their capacity to perform following different metrics [9]. Their results show that quantization is not by itself a defence against adversarial examples and that all generation frameworks do not react in the same way to quantization.

2.5 Side-Channel Attacks

Historically, side-channel analysis (SCA) is a class of cryptographic attack in which an attacker tries to exploit the vulnerabilities of the implementation of a real-word crypto-system for key recovery by analyzing its physical characteristics *via* side-channel traces, like power consumption or electromagnetic emissions. During the execution of an algorithm into a crypto-system, side-channel traces record the intermediate variable being processed. In [6], Batina *et al.* were among the first to transpose this attack to the DNN paradigm by targeting the architecture and the parameters of the model with a Simple Power Attack (SPA) and a Correlation Electromagnetic Attack (CEMA). During these, the opponent targets a specific critical variable (*e.g.*, the weights of the DNN) and use a statistical distinguisher to extract it.

Proposition 1 (Optimal distinguisher [26]). *The optimal distinguisher $\mathcal{D}_{opt}$ in SCA context is defined by $\mathcal{D}_{opt}(\boldsymbol{X}, Y) = \Pr[\boldsymbol{X}|Y]$.*

In other words, the conditional probability mass function (*pmf*) is the best model we can build in the SCA context. However, such probability remains unknown for the attacker. Therefore, an approximation of such unknown conditional *pmf* is required. To conduct such approximation, the best solution is to perform a *profiled attack*. In this scenario, it is assumed that the attacker has an access to a copy of the victim's device with a full knowledge to the targeted component.

Definition 2 (Targeted Device). *In the context of profiled attack, the targeted device $\mathcal{D}^*$ corresponds to a system implementing an insecure function on a specific hardware for which the attacker has a model allowing her to infer information on sensitive variables.*

Definition 3 (Open Device). *Given a target device $\mathcal{D}^*$, an open device $\mathcal{D}$ refers to a copy of the targeted device where the attacker has access to both the physical traces $\mathcal{X}_p = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N_p}\}$ and the corresponding labels $\mathcal{Y}_p = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{N_p}\}$ to build a sub-optimal solution to Proposition 1.*

This kind of attacks is performed in two phases, *i.e.*, profiling and exploitation phases:

(1) In the profiling phase, the attacker gathers leakage information from an open copy $\mathcal{D}$ of the target device $\mathcal{D}^*$, embedding the AI, to extract a sensitive intermediate variable y. For this purpose, a set of N_p profiling physical traces $\mathcal{X}_p = \{\mathbf{x}_1, \ldots, \mathbf{x}_{N_p}\}$ are collected encompassing time samples related to the manipulation of Y. Then, an estimation of multivariate class-conditional probability distributions of $\mathbb{X}$ is computed ($\Pr[\mathbb{X}|Y = y]$)) over all classes $y \in \mathcal{Y}$.
(2) In the exploitation phase, the attacker gathers leakage information from the target device $\mathcal{D}^*$, embedding the AI, focusing on the identical, but now unknown, class y. Considering the adversary has the set of probability distribution estimations at disposal (precomputed during profiling phase), the attack is mounted against a set of N_a exploitation

traces $\mathcal{X}_a = \{\mathbf{x}_1, \ldots, \mathbf{x}_{N_a}\}$. This attack is based on *maximum a posteriori* (MAP) rule by estimating likelihood over all classes in $\mathcal{Y}$. Then, by looking at the most likely candidate via maximization, *i.e.*, $\underset{y \in \mathcal{Y}}{\texttt{argmax}} \sum_{i=1}^{N_a} \log(\texttt{Pr}[\mathbf{X} = \mathbf{x}_i | Y = y])$, an estimation is made on the correct class y. For computation stability, one usually computes *log-likehood*.

To find a sub-optimal solution to Proposition 1, two approaches can be considered namely, template attacks [15] or DLSCA [12, 41]. Whereas the template attacks approximate the unknown conditional *pmf* $\texttt{Pr}[\mathbf{X}|Y]$, the DLSCA uses supervised learning to estimate the unknown conditional *pmf* $\texttt{Pr}[Y|\mathbf{X}]$. Even if both solutions are equivalent up to a constant in SCA[1], some benefits and limitations can be expressed for each method.

Template attacks. Therefore, to approximate the unknown conditional *pmf* $\texttt{Pr}[\mathbf{X}|Y]$, the profiling phase consists in estimating the mean vector $\boldsymbol{\mu}_y$ and the covariance matrix Σ_y for each class $y \in \mathcal{Y}$ from a set $\mathcal{X}_p$ of profiling physical traces. Then, the sub-optimal solution of Proposition 1 is defined by a Gaussian probability density function (*pdf*) with parameters $\boldsymbol{\mu}_y$ and Σ_y. However, the huge dimensionality[2] of $\mathbf{X}$ can sometimes make the estimation of Proposition 1 a complex task. Therefore, a popular solution consists in selecting a small portion of the physical traces based on some statistical tests [18] (*e.g.* SNR [42], difference of means [15], T-Test [7]) or dimensionality reduction techniques (*e.g.* PCA [4], LDA [17], KDA [13]). In this paper, only the SNR metric will be considered for selecting a small portion of the physical traces. Another limitation of the template attacks relies on the Gaussian assumption. Indeed, when the underlying Gaussian assumption does not hold, estimating $\texttt{Pr}[\mathbf{X}|Y]$ is known to be hard in practice [11]. This has led the side-channel community to find alternatives based on Machine Learning techniques.

Deep learning-based side-channel attacks (DLSCA). Over the past few years, deep learning approaches have been investigated in the SCA context to mitigate the impact of some countermeasures, namely desychronization [12, 43] and masking [8, 41]. Similarly to the template attacks, two phases are required. First, the profiling phase is applied to approximate the true unknown conditional probability $\texttt{Pr}[Y|\mathbf{X}]$ by following the strategy detailed in Section 2.2. Indeed, a DNN is designed by the attacker to select a function f which maps a physical trace to a sensitive variable $y \in \mathcal{Y}$. The structure-*agnostic* property of the MLP is one of its main advantages. Indeed, to consider it, no particular assumption has to be made on the data structure. In a side-channel context, this is beneficial to automatically find a function which is not limited to the Gaussian assumption.

[1]In the SCA context, if Y is uniformly distributed over $\mathcal{Y}$, $\texttt{Pr}[\mathbf{X}|Y] = \frac{\texttt{Pr}[Y|\mathbf{X}] \cdot \texttt{Pr}[\mathbf{X}]}{\texttt{Pr}[Y]} = \epsilon \cdot \texttt{Pr}[Y|\mathbf{X}]$ where ϵ does not depend on Y.

[2]In real-world use-case, the dimensionality of physical traces can easily exceed 10^7.

3 SIDE-CHANNEL ATTACKS MAKE ADVERSARIAL EXAMPLES MORE PRACTICABLE

This section introduces a new black-box attack scenario against embedded neural networks. It takes advantage of hardware side-channel attacks to demonstrate the ability of an attacker to extract sensitive information (*i.e.*, logits) which are usually considered as known. As this assumption is usually not practicable, a new contribution must be introduced to extract those logits. This section demonstrates the synergy between hardware and software attacks to defeat the security of an embedded neural network. Section 3.1 describes the attack strategy by introducing our contribution, as well as which threat models can be considered in such scenario. Finally, Section 3.2 details our contribution through the identification of the attack path, and the practical recommendations that are needed to perform it.

3.1 Methodology

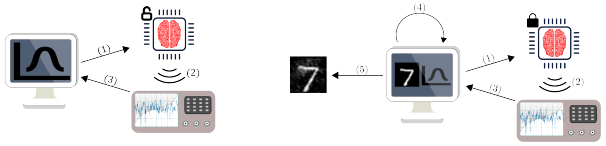
Our contribution provides an independent method to break free from assumption traditionally restricting the state-of-the-art frameworks for black box generation of adversarial examples. For example, two widely adopted solutions, namely ZOO [16] and SPSA [55] attacks, require access to the logits or the probability vector, while being under the assumption of the black box. Following the black box definition of this paper, such strategy could not be conducted by any attacker as she would only have access to the predicted label included in $\mathcal{Y}$. Therefore, the requirements of ZOO [16] and SPSA [55] are not fulfilled. In this context, our contribution takes advantage of hardware attacks to complete state-of-the-art adversarial example generation, and provides an access to the logits and the probability vector in a black box context. This forms an overall assumption-free attack (*e.g.*, no previous knowledge on the DNN's hyparemters or parameters is required, nor access to the logits, confidence scores or even output label).

The first step of our attack is to design a sub-optimal solution of Proposition 1; this is a crucial step that needs an access to the copy of the targeted device. As explained, in Definition 3, $\mathcal{D}$ enables the malicious user to create a statistical distinguisher, which will be used to attack the softmax function of the targeted device $\mathcal{D}^*$ running the neural network. This corresponds to the profiling phase.

The process, illustrated in Figure 2a, can be detailed as follows:

- (1) A set of random input data, for which the logits are known, are sent to the embedded neural network for prediction;
- (2) During prediction, the attacker captures the related side-channel traces which contain information on the manipulated logits;
- (3) The attacker creates a sub-optimal solution of Proposition 1 (*e.g.*, template, DLSCA) to characterize the leakage model related to the logits' manipulation.

These steps require a strong assumption that we will detail in Section 4.1. Once this step is performed, our global attack methodology can be illustrated as in Figure 2. It follows the same general steps as a classical ZOO or SPSA attack by querying the network with an input and transforming progressively the image until an adversarial



(a) Profiling phase (b) Attack phase

Figure 2: Schematic description of the profiling and attack phases of our attack.

example is created as described in Section 2.4. This is an iterative process with one iteration being composed of the following steps:

- (1) One data is sent n times to the embedded neural network for prediction;
- (2) Meanwhile, the attacker captures the n corresponding side-channel traces containing information related to the manipulated logits;
- (3) The attacker performs the logit extraction attack using a sub-optimal solution to Proposition 1, *e.g.*, templates or DNN models (see Section 2.5), and the newly captured attack traces. Then, the extracted logits are provided to the adversarial example methods;
- (4) Using a gradient free method (*e.g.*, ZOO, SPSA), the attacker estimates the perturbation to add to the inputs in order to create a successful adversarial example that will fool the embedded neural network.

Since a single iteration is usually not sufficient to construct a successful adversarial example, steps (1) to (4) are repeated until the perturbation causes a misclassification from the network (*i.e.*, step (5) in Figure 2) or until the generation framework stops. Therefore, if we denote m_{adv} as the maximum number of iterations to create an adversarial example and n the maximum number of traces necessary to successfully extract the logits, the global complexity C of the attack can be written as

$$C = O(n \cdot m_{\text{adv}}). \tag{1}$$

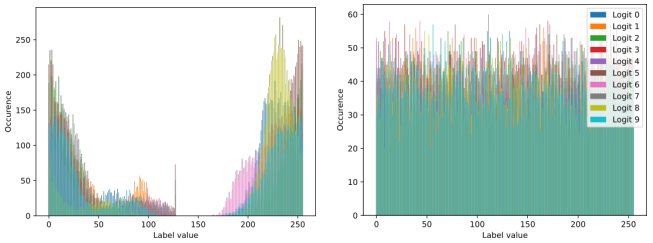
In comparison with gradient free optimization methods, the overall attack complexity is multiplied by the number of attack traces that are needed to retrieve the logits. But it is an overall improvement, as it allows attackers to perform an end-to-end black box attack. However, as shown in Equation 1, the practicability of the attack scenario highly depends on the ability of the side-channel attack (*i.e.*, template attacks, DLSCA) to extract the logits.

3.2 General Observation

Logits extraction. As mentioned in the previous section, the main idea in our attack consists in retrieving the logits manipulated by the embedded neural network through the use of side-channel attacks. In particular, the targeted operations induced in a layer, or in an activation function, (as illustrated in Section 2.2) should probably leak information on the logits. Since the softmax function is computed to approximate the probability distribution $\Pr[Y|X = x]$ from the logits, it is a natural target from a side-channel perspective to extract information related to these sensitive variables. In addition, as side-channel attacks are still new to DL framework, no countermeasures are implemented to protect the device against these threats. As our attack is independent from the architecture, since we target a natural and frequently used function, namely softmax, the proposed methodology can be applied to all neural network architectures which solve a classification task. Indeed, extracting the logits from the softmax is beneficial to perform gradient-free generation frameworks proposed in the state of the art.

A natural distribution difficulty. As stated in Section 2.5, template attack and DLSCA can be considered to extract the corresponding logits in a profiled scenario. However, for an unbiased SCA, the profiling phase must be conducted with the logits uniformly distributed over $\mathcal{Y}$. The under-representation of certain logits will lead to poor estimation of the distribution parameters (*i.e.*, the pair $(\boldsymbol{\mu}_y, \Sigma_y)$ for template attack, and the weights for DLSCA) leading to a deterioration of the attack performance. This issue is especially true when targeting the logits of a DNN. Indeed, the purpose of the embedded neural network is to select an output among the possible classes in $\mathcal{Y}$, as such the DNN is trained to maximize one value in the logits vector and keep the other ones as low as possible. It is then likely that, for a given model, the distribution of the logits will be over-represented for the lower values of the output range and under-represented in the higher part. An example of class distributions is provided in Figure 3a. The logits, in this example, are represented using 8-bit signed integers, as such the values over 127 correspond to negative logits. We can then observe that the distribution follows the pattern of over and under-representation of classes that we previously described. If the adversary targets an open-source DL framework (*e.g.*, PyTorch [47], Tensorflow [1], FINN [56], NNOM [40]), the source code can be modified in order to force the uniform distribution of the logits during the profiling phase. Because most of the embedded neural networks are generated from open-source DL frameworks, the attacker can naturally have an access to the source code. Then, the attacker can modify this code on the open device $\mathcal{D}$ to mitigate this issue, as illustrated in Figure 3b. Added to the fact that our attack is architecture-agnostic, this allows the malicious user to build an unbiased SCA which can be applied on all the targeted embedded devices $\mathcal{D}^*$ using the same open source DL framework as in $\mathcal{D}$.

This strategy has been considered in this paper and the following section proposes to experimentally validate this new attack strategy.



(a) Non-uniform distribution (b) Uniform distribution

Figure 3: Example of the distribution of logits on a real use-case over a set of 10000 samples.

4 EXPERIMENTAL RESULTS

In this section, we apply our methodology on an neural network implemented in a microcontroller. First, we define the threat model that we consider. Then, Section 4.2 describes the experimental setup, while Section 4.3 details the security flaw exploited in this paper. Finally, Section 4.4 (resp. Section 4.5) evaluates the practicability of the logits extraction (resp. the end-to-end attack scenario).

4.1 Threat Model

Scenario. In this study we contextualize the threat model we consider in this paper. Following Section 3, the primary goal of the attackers is to extract the logits from the embedded neural network by performing side-channel attacks. In this paper, our strategy is based on profiling attack scenario, namely template attacks and DLSCA. As mentioned in Section 2.5, such scenario suggests that the attacker has a physical access to a copy of the targeted device. This threat considers attackers who obtain the same hardware reference $\mathcal{D}$ as the targeted device $\mathcal{D}^*$ on the market. As it is assumed that the DL framework used to embed the DNN model on the targeted device is known, the attacker can build its own model on $\mathcal{D}$ in order to extract the logits from $\mathcal{D}^*$. Indeed, as the attack scenario introduced in Section 3 is architecture-independent (*i.e.*, the softmax function is targeted), the knowledge of the DNN structure $\mathcal{D}^*$ is not required. Consequently, the attacker can easily construct its own copy on $\mathcal{D}$ of the targeted function embedded in $\mathcal{D}^*$ to satisfy the requirements of the profiling attack scenario. Once this profiling phase has been performed on $\mathcal{D}$, the attacker can extract the logits from the targeted device $\mathcal{D}^*$. Therefore, this threat model is considered at inference stage. In other words, the targeted embedded neural network has already been trained and deployed. For example, this threat considers attackers who come from the DNN accelerator design team, or as insiders in the companies hosting DNN infrastructure so that they are capable to get a physical access to the device.

Capability. To conduct such scenario, it is assumed that the attacker has knowledge on the DL framework embedded into the targeted system. Knowing the reference of the targeted hardware device, the adversary can modify the source code of the DL framework to mitigate the issues introduced in Section 3.2. This assumption is in accordance with the scenario previously introduced. Secondly, attackers can acquire power consumption or electromagnetic emanation traces of the targeted embedded neural network in high resolution through the use of an oscilloscope. This assumption is in accordance with the scenario previously introduced. As the attack strategy targets a function of the DL framework that is independent of the underlying neural network architecture (*e.g.*, softmax function), no prerequisite on the DNN structure is needed.

4.2 Experimental Setup

Hardware implementation. We validate our contribution by targeting an ARM Cortex-M7 microcontroler unit (MCU) on an STM32-F767 board running at a frequency of 216 MHz. This board incorporates 2 Mbytes of flash memory and 512 Kbytes of SRAM, which is enough to mount quantized models on the board. We choose to target a classifier based on the denseNet [30] architecture, including a softmax as the last layer. The model was trained on the MNIST dataset with the Adam optimizer, and using Tensorflow [2] for development. MNIST is a dataset for image recognition on handwritten digits composed of $60,000$ training data and $10,000$ test images [38]. This dataset considers 10 classes. As a consequence, 10 logits must be extracted to validate the methodology introduced in Section 3. In the context of embedded system, we choose to use quantization to reduce the impact on the resources. The inputs, weights, and activation values of the model are quantized to 8 bits, using deterministic PTQ. NNOM, an open-source framework, was used to facilitate the incorporation on the MCU [40]. NNOM translates the model from a Tensorflow object to a C project, which can then be directly build and run on the board. The generated project was compiled without optimisation from the compiler. We decided to only test on the NNOM framework but we expect our attack methodology to translate to other frameworks, as any unprotected manipulation of the logits in the assembly code will induce side channel leaks. We leave evaluation of other frameworks and hardware implementations for future work.

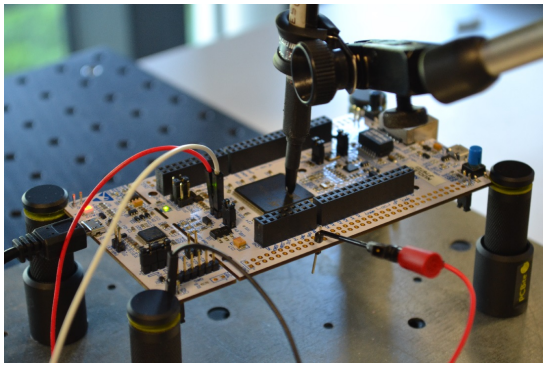


Figure 4: Experimental setup.

Practical setup. A probe from Langer (EMV-Technik RF-U 2,5 with a frequency range going from 30MHz up to 3GHz) is connected to an amplifier (ZLF-2000G+) to capture the electromagnetic (EM) side-channel leakages. To acquire the EM signal, a Lecroy oscilloscope (2.5 GHz WaveRunner 625Zi) is used in our setup. To ease the leakage exploitation, a trigger surrounding the targeted function (*i.e.*, `local_softmax_q7()` from NNOM) has been added. As the softmax computation is distinguishable from the other patterns, the configuration of the trigger is not considered difficult. A quick overview of our experimental setup is illustrated in Figure 4.

To get a better insight on our contribution, 10 independent attacks are performed and the success rate metric is provided to identify the number of attack traces that are required to successfully retrieve each logit. As a reminder, the success rate metric defines the probability that an attack succeeds in recovering the true logit amongst all hypotheses in $\mathcal{Y}$.

Logits distribution. Following the recommendations provided in Section 3.2, we evaluated the distribution of the logits on the trained embedded neural network. As expected, a non-uniform distribution is observed (see Figure 3a). To solve this issue, we

	Logit 0	Logit 1	Logit 2	Logit 3	Logit 4	Logit 5	Logit 6	Logit 7	Logit 8	Logit 9
Maximum SNR value	$2.3 \cdot 10^{-1}$	$2.2 \cdot 10^{-1}$	$5.0 \cdot 10^{-2}$	$3.5 \cdot 10^{-2}$	$2.0 \cdot 10^{-2}$	$2.0 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$1.6 \cdot 10^{-2}$	$1.0 \cdot 10^{-1}$
SNR threshold	$5.0 \cdot 10^{-3}$	$5.0 \cdot 10^{-3}$	$8.0 \cdot 10^{-3}$	$8.0 \cdot 10^{-3}$	$8.0 \cdot 10^{-3}$	$5.0 \cdot 10^{-3}$	$5.0 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$
Dimension of reduced traces	6,514	8,840	1,654	1,302	774	2,866	2,223	5,813	6,186	6,299

Table 1: Preprocessing on the EM traces depending on the obtained SNR value.

Listing (1) Part of the `local_softmax_q7` implementation in NNOM.

```
1  /* Base is
       initialized
       to (int16_t)
       -257 */
2    /* We first
       search for
       the maximum
       */
3    for (i = 0; i
       < dim_vec; i
       ++)
4    {
5        if (vec_in
       [i] > base)
6        {
7            base =
        vec_in[i];
8        }
9    }
```

Listing (2) Assembly code of line 5 from Listing (1).

```
1  ; if (vec_in[i] > base)
2  ldr r3, [r7, #32] ; Loading
       the address of the index
       "i"
3  ldr r2, [r7, #12] ; Loading
       the address of z
4  add r3, r2        ; Set the
       pointer to the i-th
       element of confidence
       score z
5  ldrsb.w r3, [r3]  ; Loading of
        the i-th element of z
6  sxth  r3, r3      ; Extension
       the i-th element of z to
       a 32-bit
7  ldrsh.w r2, [r7, #30] ; Base
       value loading
8  cmp r2, r3        ; Comparison
       between base and the i-th
        element of z
```

Figure 5: C and Assembly codes from the NNOM softmax function.

modified the source code of the NNOM softmax function to force its input (*i.e.*, the logits) to follow a uniform distribution. The results we obtained are highlighted in Figure 3b. This requirement is highly recommended to mitigate the bias in the suboptimal solution of Proposition 1 (*i.e.*, template attacks, DLSCA), but not mandatory as stated in Section 3.2.

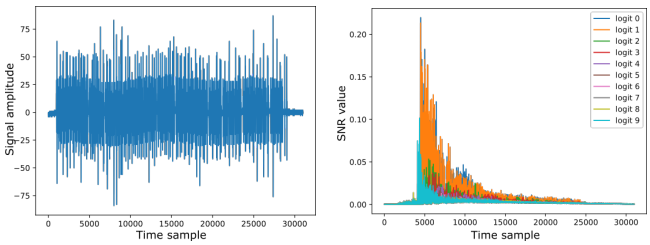
4.3 Security Flaw

To validate the methodology introduced in Section 3.1, the NNOM softmax function is targeted. We choose this implementation as it is a common solution in the literature [20, 35, 51, 58]. Whereas traditional softmax function is defined by a normalization which maps the logits (or the confidence scores) to a probability distribution following $s(\mathbf{Z}[i]) = \frac{e^{\mathbf{Z}[i]}}{\sum_{j=1}^{|\mathcal{Y}|} e^{\mathbf{Z}[j]}}$ where $\mathbf{Z}[i]$ denotes the logits related to the i^{th} class, the NNOM framework proposes an alternative to this natural approach. In particular, instead of computing the exponentiation of each logit as suggested in the classical softmax function, NNOM captures the maximum value $\texttt{max}_z$ from the vector of logits $\mathbf{Z}$. Then, all logits lower than $(\texttt{max}_z - 8)$ are ignored, while others are normalized following the function `local_softmax_q7()`. In particular, Listing 1 is an extract of the C code source of the softmax computation from NNOM. This portion searches the maximum value from the `vec_in` vector which characterizes the vector of logits $\mathbf{Z}$.

For each element included in `vec_in`, the base value is updated if its value is lower than the i^{th} element of `vec_in` (*i.e.*, $\mathbf{Z}[i]$). Listing 2 defines the assembly code which characterizes the "if"-condition involved in line 5 of Listing 1. The logit $\mathbf{Z}[i]$ stored into the Flash memory is loaded into register `r3` (see line 5 in Listing 2). During the load instruction, a side-channel attack can be performed to extract information related to the targeted logit. In addition, the `base` value is updated if the "if"-condition is respected. Therefore, when the `base` value is loaded into register `r2` (line 7 in Listing 2), leakages could be observed on the current maximum value in `vec_in`. This load instruction suggests the manipulation of some z_i values multiple times which can make the leakage exploitation more practicable. An noteworthy remark is that the side-channel attack considered in this paper can be easily transferred from a DNN architecture to another as long as `local_softmax_q7` of NNOM is used as well as the Nucleo-144.

We wanted to confirm our analysis of this security flaw by targeting directly for a signal associated with the value of logits. Figure 6a represents the electromagnetic trace associated to the processing of the softmax function for different logits. Figure 6b displays the SNR for the first logit computed. As expected from our analysis of Listing 1, the ratio between the signal associated to this variable and the noise change during the execution time. This displays the manipulation of the targeted variable in the assembly code. Remark: readers have to notify that the scenario proposed in this paper can be conducted on any type of DL implementation since the manipulation of the logits occurs in the assembly code.

To validate our attack methodology, for each logit, a set of 1,900,000 EM traces of 31,000 time samples is captured for the profiling set, while 1,000 traces are acquired for the attack phase. To reduce the needs of computation, the EM traces are reduced following the SNR peaks we obtained on each logit. In that purpose, a threshold is configured for each logit such that all the time samples associated with a higher SNR peak are kept. The threshold is defined in order to find the best trade-off between computational



(a) Electromagnetic trace of the softmax execution.

(b) SNR associated with each logit.

Figure 6: Preprocessing of an EM trace.

time and leakage information. The results we obtained are defined in Table 1. As mentioned in Section 2.5, other dimensionality reduction techniques such as PCA [4], LDA [17] or KDA [13] could be used as suitable alternatives.

4.4 Evaluation of the Logit Extraction

To evaluate the logit extraction process, three attacks have been conducted. All attacks have the same number of profiling traces on the open sample $\mathcal{D}$ to build their sub-optimal solution to Proposition 1. On the contrary, the number of attack traces used on the targeted sample $\mathcal{D}^*$ to achieve a similar average success rate, is different from one attack to another. This metric will be used to compare the efficiency of the three methods. First, template attacks [15] are performed in order to assess the ability of Gaussian distribution to capture the dependencies between the targeted logits and the EM traces. As such assumption can not hold in practice, we evaluated a more generic solution which considers a broad family of models: *multinomial logistic regression*. As suggested by Masure *et al.* [44], this class of models defines the set of all polynomial transformations such that a clear connection can be proposed with template attacks. Finally, to get a full overview of the methodology, MLPs are constructed in order to perform a deep learning-based side-channel attack and recover the logits of the embedded neural network. While the latter solution is beneficial from a performance perspective, the lack of interpretability makes its configuration a difficult task.

Template attack results. The template attacks are the naive solution when profiled attacks are conducted. The obtained results are displayed in Figure 7a where each colour of a curve corresponds to the success rate for a given logit (out of 10) as a function of the number of traces used for the attack. Accordingly, the purple curve corresponds to the success rate for 5^{th} logit out of the 10 possible classes averaged on 10 independent experiments. For this specific logit, 8 attack traces are required to successfully extract (*i.e.*, a success rate of 100%) the targeted logit value. From Figure 7a, it can be also observed that after only 11 attack traces, the average attack success rate is around 66%. In addition, while some logits are recovered with a very small amount of traces (*e.g.*, 8 traces for the 5^{th} logit), others are not successfully retrieved (*e.g.*, no successful attack is obtained when the 10^{th} logit is targeted). This observation suggests that additional investigations can be conducted to improve even more the performance of template attacks. An attacker could increase the number of collected traces on the open sample $\mathcal{D}$ in order to improve the profiling phase and build a better solution to Proposition 1. Other setup configurations (*e.g.*, dimensionality reduction technique, SNR threshold) could significantly improve the performance gain. The results we obtained illustrate the practical limitation of the template attacks. To mitigate this issue, an attacker can use another technique based on multinomial logistic regression in order to consider a less restrictive class of model.

Multinomial logistic regression results. To investigate the benefits of this class of models, a simple MLP without hidden layers, nor activation functions, is configured. Only an output softmax function is used to solve the multiclass-classification problem. During the profiling phase we used the traces collected on $\mathcal{D}$ and the associated labels (*e.g.* the logits) to train the MLP via supervised learning using the Adam optimizer [36]. This profiling process is conducted over 20 epochs such that the batch size (resp. the learning rate) is set to 512 (resp. 10^{-5}).

In comparison with the template attack result, Figure 7b shows a performance improvement when the multinomial logistic regression is considered. In particular, after 10 attack traces, the average success rate is about 80%, and with this method all the logits are extracted at least once, which was not the case with the template attack. However, the attack on the 5^{th} logit performs poorly in comparison with other logits. Consequently, if an attacker wants to follow the methodology introduced in Section 3, the wrong label estimation will impact the generation of adversarial examples. Even if some state-of-the-art results suggest that the knowledge of the top-k logits is sufficient to generate adversarial examples [31, 32], this scenario is not considered in the scope of this paper.

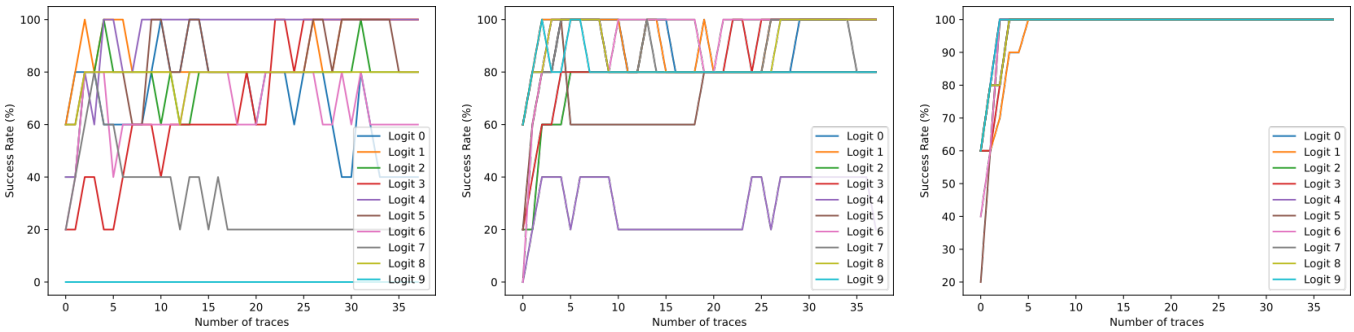
DLSCA results. Finally, the most generic solution consists in designing a DNN model to approximate the best possible solution to Proposition 1. We target the embedded system developed to solve the classification problem on the MNIST dataset running on the targeted device $\mathcal{D}^*$, with another network trained on the EM traces of the logit's processing on the open device $\mathcal{D}$. The MLP trained on the traces of $\mathcal{D}^*$ is composed of 3 hidden layers of $1,000$, $1,000$ and 100 neurons respectively and, in order to approximate non-linear functions, each neuron uses the ReLU activation function. The same configuration as for the multinomial logistic regression is used (*i.e.*, 20 epochs and a batch size of 512), whereas the learning rate is adapted to 10^{-4}. The results obtained with this setting are observed in Figure 7c. Interestingly, when the DL technique is used to extract the logits from the embedded neural network, all the sensitive variables are retrieved within 5 EM traces. As expected, this method offers the best performance compared to the others.

Through this process, it can be confirmed that an attacker can perform side-channel attacks using a sub-optimal solution to Proposition 1 to extract the logits from the embedded neural network. Regarding the global complexity of the attack (C in Equation 1), the attacker needs $5 \times m_{\text{adv}}$ attack traces to create a given adversarial example. In our experiment, we set m_{adv}, the maximum number of iterations to create an adversarial example, to $10,000$. Consequently, if the generation of adversarial examples is possible, the total number of attack traces required to conduct an end-to-end attack is $50,000$ in the worst case[3]. However, it should be mentioned that, sometimes, $10,000$ iterations are not sufficient to generate such adversarial example using the ZOO framework. Consequently, the total number of attack traces highly depends on the performance of the adversarial example tool. The following section investigates an end-to-end scenario of our proposition through the use of the ZOO framework.

4.5 Evaluation of the Complete Attack

Once the extraction is performed using one of these methods, an attacker can perform the generation of adversarial examples in a black box setting using gradient-free frameworks. To validate this

[3]In practice, to defeat a protected cryptographic system, a side channel attack may require millions of traces.



(a) Template attacks **(b) Multinomial logistic regression** **(c) Deep-learning based side-channel attacks**

Figure 7: Success rate results on targeting NNOM softmax function. Each colour of a curve corresponds to the success rate of a given logit (out of 10) as a function of the number of traces used for the attack.

statement, we use the open-source implementation of the ZOO attack[4] provided by [16] to attack our embedded model. Since this framework uses the probability vector in the objective function, as described in Section 2.4, we compute it with the extracted logits and solve the optimization problem accordingly. We confront the results of our attack with those obtained after white-box generation using the Basic Iterative Method [37] from the Cleverhans framework [45] on a substitute network. The substitute network is based on the same architecture as the targeted network and we use the same methodology as [48] and trained it using 10% of the training data labeled by the embedded model. We evaluate the transfer rate, the average distortion, and the average number of request, to generate an adversarial example. Our results are included in Table 2.

The number of request necessary to produce an adversarial example is much higher when using our framework. This is caused by the profiling attack which needs requests to the system to acquire the physical traces and build the model. The number of traces for this phase is highly dependent on the implementation and the surrounding noise. Since we are using optimization method in our profiling phase, this number is highly experimental and could be much lower for another hardware. However, the main advantage of our framework is its genericity. Once the profiling phase is finished, an attacker will not have to fully retrain it while targeting any models implemented using NNOM. On the other hand, to train the substitute network, the malicious user will need to have access to training data specific for each task and will potentially have to retrain a substitute for each new targeted DNN. As expected, even in an optimal case with a substitute based on the same architecture, not all the white-box-generated adversarial examples are transferable with an average rate of 56%. Since ZOO directly uses the targeted network, the generated examples are by definition directly adversarial for this network. The high value in the distortion for both frameworks seems to be due to the 8-bit quantization of the input, which forces the perturbation to be larger to cause an impact.

With this design, we demonstrated that an attacker can extend existing state-of-the-art frameworks to generate powerful adversarial examples in a black box scenario using our end-to-end methodology. We only evaluated our framework with the ZOO attack and against only a substitute network, as they are no other method, to the best of our knowledge, that proposes a generation process that does not require access to any internal or output values (logits, confidence score). In that context, the purpose of this paper is to introduce a method to reduce the necessary assumptions. The case of boundary attacks [10], which only use the output labels to generate adversarial example, can even be considered for situations where these labels are not directly known by the attacker. We leave the comparison with boundary attacks, when access to the output labels is possible, for future work, as well as evaluations on more complex datasets.

	Transfer rate (%)	Average Distortion (L2)	Number of request
Logits extraction and ZOO	100	7.54	$1.9 \cdot 10^6$
Substitute network	56.61	5.80	6000

Table 2: Comparison between our framework and generation on a substitute network.

5 CONCLUSION

This study has for purpose to prove that it is worth considering side-channel and, more generally, physical attacks against DL implementation, in addition to software attacks. To do so, we demonstrate the ability of an attacker to generate powerful adversarial examples by breaking the black box assumption through the exploitation of side-channel attacks against an embedded DNN. In particular, we contribute to the state of the art by adding a side-channel extraction of the logits to gradient-free adversarial example frameworks using these values.

To validate our methodology, a practical scenario has been conducted on a microcontroller implementing a denseNet with the NNOM framework. Based on a code analysis, security flaws have been identified and exploited in the NNOM softmax function and the logit vector has been extracted, in the best case, within 5 attack traces. We finalize our attack by combining our physical attack with the ZOO framework for adversarial example generation, confirming the complementarity between software and hardware attacks.

[4] https://github.com/IBM/ZOO-Attack

To further improve our contribution, some future works can be suggested. Vulnerability in the softmax function can be studied in some widely used open-source DL frameworks (*e.g.*, PyTorch, Tensorflow, FINN) or on more optimized hardware implementations (*e.g.*, on FPGA). Further investigation on the attack can be focused on the impact of a wrong logit estimation on the final attack and on adapting it to perform non-profiled side-channel attacks, which reduce the attacker's capability. Finally, while the security of embedded AI system is still a new research direction, finding countermeasures against our contribution is crucial to make the AI solutions more secure.

ACKNOWLEDGMENTS

We thank all the anonymous reviewer for their comments, which helped improve the quality of our work. The authors would also like to thank Rémy Baillet and Enzo Rosarini for fruitful discussions about the hardware implementation which improved the quality of our work.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 265–283.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Z. Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. 2016. TensorFlow: A system for large-scale machine learning. *ArXiv* abs/1605.08695 (2016).

[3] Tarek Abdelzaher, Nora Ayanian, Tamer Basar, Suhas Diggavi, Jana Diesner, Deepak Ganesan, Ramesh Govindan, Susmit Jha, Tancrede Lepoint, Benjamin Marlin, Klara Nahrstedt, David Nicol, Raj Rajkumar, Stephen Russell, Sanjit Seshia, Fei Sha, Prashant Shenoy, Mani Srivastava, Gaurav Sukhatme, Ananthram Swami, Paulo Tabuada, Don Towsley, Nitin Vaidya, and Venu Veeravalli. 2018. Toward an Internet of Battlefield Things: A Resilience Perspective. *Computer* 51, 11 (2018), 24–36. https://doi.org/10.1109/MC.2018.2876048

[4] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. 2006. Template Attacks in Principal Subspaces. In *Cryptographic Hardware and Embedded Systems - CHES 2006*, Louis Goubin and Mitsuru Matsui (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–14.

[5] Pallavi Sunil Bangare and Kishor P. Patil. 2022. Security Issues and Challenges in Internet of Things (IOT) System. In *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. 91–94. https://doi.org/10.1109/ICACITE53722.2022.9823709

[6] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel. In *USENIX Security Symposium*.

[7] George Becker, Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Tim Kouzminov, Andrew Leiserson, Mark Marson, Pankaj Rohatgi, and Sami Saab. 2013. Test Vector Leakage Assessment (TVLA) methodology in practice (Extended Abstract).

[8] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. 2020. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering* 10, 2 (2020), 163–188. https://doi.org/10.1007/s13389-019-00220-8

[9] Rémi Bernhard, Pierre-Alain Moëllic, and Jean-Max Dutertre. 2019. Impact of Low-Bitwidth Quantization on the Adversarial Robustness for Embedded Neural Networks. *2019 International Conference on Cyberworlds (CW)* (2019), 308–315.

[10] Wieland Brendel, Jonas Rauber, and Matthias Bethge. 2017. Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. *ArXiv* abs/1712.04248 (2017).

[11] Nicolas Bruneau, Sylvain Guilley, Annelie Heuser, Damien Marion, and Olivier Rioul. 2017. Optimal side-channel attacks for multivariate leakages and multiple models. *Journal of Cryptographic Engineering* 7 (2017), 331–341.

[12] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. 2017. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10529)*, Wieland Fischer and Naofumi Homma (Eds.). Springer, 45–68. https://doi.org/10.1007/978-3-319-66787-4_3

[13] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. 2017. Kernel Discriminant Analysis for Information Extraction in the Presence of Masking. In *Smart Card Research and Advanced Applications*, Kerstin Lemke-Rust and Michael Tunstall (Eds.). Springer International Publishing, Cham, 1–22.

[14] Nicholas Carlini and David A. Wagner. 2016. Towards Evaluating the Robustness of Neural Networks. *2017 IEEE Symposium on Security and Privacy (SP)* (2016), 39–57.

[15] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. 2002. Template Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers (Lecture Notes in Computer Science, Vol. 2523)*. Springer, 13–28. https://doi.org/10.1007/3-540-36400-5_3

[16] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017. ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models. *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security* (2017).

[17] Marios Choudary and Markus Kuhn. 2015. Efficient Stochastic Methods: Profiled Attacks Beyond 8 Bits. In *Smart Card Research and Advanced Applications*, Marc Joye and Amir Moradi (Eds.). Springer International Publishing, Cham, 85–103.

[18] Omar Choudary and Markus G. Kuhn. 2014. Efficient Template Attacks. In *Smart Card Research and Advanced Applications*, Aurélien Francillon and Pankaj Rohatgi (Eds.). Springer International Publishing, Cham, 253–270.

[19] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proc. IEEE* 108, 4 (2020), 485–532. https://doi.org/10.1109/JPROC.2020.2976475

[20] Mathieu Dumont, Kevin Hector, Pierre-Alain Moellic, Jean-Max Dutertre, and Simon Pontié. 2023. Evaluation of Parameter-based Attacks against Embedded Neural Networks with Laser Injection. arXiv:2304.12876 [cs.CR]

[21] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015).

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org/

[23] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[24] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. *CoRR* abs/1412.6572 (2014).

[25] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *arXiv: Computer Vision and Pattern Recognition* (2015).

[26] Annelie Heuser, Olivier Rioul, and Sylvain Guilley. 2014. Good Is Not Good Enough. In *Cryptographic Hardware and Embedded Systems – CHES 2014*, Lejla Batina and Matthew Robshaw (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 55–74.

[27] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *ArXiv* abs/1503.02531 (2015).

[28] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv* abs/1704.04861 (2017).

[29] Weiwei Hu and Ying Tan. 2017. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *ArXiv* abs/1702.05983 (2017).

[30] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2016. Densely Connected Convolutional Networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 2261–2269.

[31] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. 2018. Black-box Adversarial Attacks with Limited Queries and Information. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 2137–2146. https://proceedings.mlr.press/v80/ilyas18a.html

[32] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. 2019. Prior Convictions: Black-box Adversarial Attacks with Bandits and Priors. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BkMiWhR5K7

[33] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alexey Kurakin, and Nicolas Papernot. 2019. High Accuracy and High Fidelity Extraction of Neural Networks. In *USENIX Security Symposium*.

[34] Raphaël Joud, Pierre-Alain Moëllic, Simon Pontié, and Jean-Baptiste Rigaud. 2023. A Practical Introduction to Side-Channel Extraction of Deep Neural Network Parameters. In *Smart Card Research and Advanced Applications*, Ileana Buhan and Tobias Schneider (Eds.). Springer International Publishing, Cham, 45–65.

[35] Raphaël Joud, Pierre-Alain Moëllic, Simon Pontié, and Jean-Baptiste Rigaud. 2023. A Practical Introduction to Side-Channel Extraction of Deep Neural Network Parameters. In *Smart Card Research and Advanced Applications*, Ileana Buhan and Tobias Schneider (Eds.). Springer International Publishing, Cham, 45–65.

[36] Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[37] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *ArXiv* abs/1607.02533 (2016).

[38] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791

[39] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. 2017. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 131–138. https://doi.org/10.1109/ICCAD.2017.8203770

[40] Jianjia Ma. 2020. *A higher-level Neural Network library on Microcontrollers (NNoM)*. https://doi.org/10.5281/zenodo.4158710

[41] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. 2016. Breaking Cryptographic Implementations Using Deep Learning Techniques. In *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 10076)*, Claude Carlet, Anwar Hasan, and Vishal Saraswat (Eds.). Springer, 3–26. https://doi.org/10.1007/978-3-319-49445-6_1

[42] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2007. *Power analysis attacks - revealing the secrets of smart cards*. Springer.

[43] Loïc Masure, Nicolas Belleville, Eleonora Cagli, Marie-Angela Cornélie, Damien Couroussé, Cécile Dumas, and Laurent Maingault. 2020. Deep Learning Side-Channel Analysis on Large-Scale Traces. In *Computer Security – ESORICS 2020*, Liqun Chen, Ninghui Li, Kaitai Liang, and Steve Schneider (Eds.). Springer International Publishing, Cham, 440–460.

[44] Loïc Masure, Gaëtan Cassiers, Julien Hendrickx, and François-Xavier Standaert. 2022. Information Bounds and Convergence Rates for Side-Channel Security Evaluators. Cryptology ePrint Archive, Paper 2022/490. https://eprint.iacr.org/2022/490 https://eprint.iacr.org/2022/490.

[45] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. 2018. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. *arXiv preprint arXiv:1610.00768* (2018).

[46] Nicolas Papernot, Patrick Mcdaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2015. The Limitations of Deep Learning in Adversarial Settings. *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (2015), 372–387.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates Inc., Red Hook, NY, USA.

[48] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. 2021. DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories. *2022 IEEE Symposium on Security and Privacy (SP)* (2021), 1157–1174.

[49] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. 2015. You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), 779–788.

[50] David Rolnick and Konrad Paul Kording. 2019. Reverse-engineering deep ReLU networks. In *International Conference on Machine Learning*.

[51] Manuele Rusci, Marco Fariselli, Martin Croome, Francesco Paci, and Eric Flamand. 2023. Accelerating RNN-Based Speech Enhancement on a Multi-core MCU with Mixed FP16-INT8 Post-training Quantization. In *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, Irena Koprinska, Paolo Mignone, Riccardo Guidotti, Szymon Jaroszewicz, Holger Fröning, Francesco Gullo, Pedro M. Ferreira, Damian Roqueiro, Gaia Ceddia, Slawomir Nowaczyk, João Gama, Rita Ribeiro, Ricard Gavaldà, Elio Masciari, Zbigniew Ras, Ettore Ritacco, Francesca Naretto, Andreas Theissler, Przemyslaw Biecek, Wouter Verbeke, Gregor Schiele, Franz Pernkopf, Michaela Blott, Ilaria Bordino, Ivan Luciano Danesi, Giovanni Ponti, Lorenzo Severini, Annalisa Appice, Giuseppina Andresini, Ibéria Medeiros, Guilherme Graça, Lee Cooper, Naghmeh Ghazaleh, Jonas Richiardi, Diego Saldana, Konstantinos Sechidis, Arif Canakoglu, Sara Pido, Pietro Pinoli, Albert Bifet, and Sepideh Pashami (Eds.). Springer Nature Switzerland, Cham, 606–617.

[52] Ahmad Shawahna, Sadiq M. Sait, and Aiman H. El-Maleh. 2019. FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. *IEEE Access* 7 (2019), 7823–7859.

[53] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, D. Erhan, Ian J. Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *CoRR* abs/1312.6199 (2013).

[54] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs. In *USENIX Security Symposium*.

[55] Jonathan Uesato, Brendan O'Donoghue, Aäron van den Oord, and Pushmeet Kohli. 2018. Adversarial Risk and the Dangers of Evaluating Against Weak Attacks. *ArXiv* abs/1802.05666 (2018).

[56] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '17)*. Association for Computing Machinery, New York, NY, USA, 65–74. https://doi.org/10.1145/3020078.3021744

[57] Chang Yue, Peizhuo Lv, Ruigang Liang, and Kai Chen. 2022. Invisible Backdoor Attacks Using Data Poisoning in the Frequency Domain. *ArXiv* abs/2207.04209 (2022).

[58] Raphael Zingg and Matthias Rosenthal. 2020. Artificial intelligence on microcontrollers. https://digitalcollection.zhaw.ch/handle/11475/20055 Embedded World Conference 2020, Nürnberg, 25.-27. Februar 2020.