Fault Injection on Embedded Neural Networks: Impact of a Single Instruction Skip

Clément Gaine, Pierre-Alain Moellic, Olivier Potin, Jean-Max Dutertre

Fault Injection on Embedded Neural Networks: Impact of a Single Instruction Skip

Clément Gaine*, Pierre-Alain Moëllic†‡, Olivier Potin*, Jean-Max Dutertre*
†CEA Tech, Centre CMP, Equipe Commune CEA Tech - Mines Saint-Etienne, F-13541 Gardanne, France ‡Univ. Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France
pierre-alain.moellic@cea.fr
*Mines Saint-Etienne, CEA, Leti, Centre CMP, F-13541 Gardanne, France
c.gaine@emse.fr, olivier.potin@emse.fr, dutertre@emse.fr

***Abstract*—With the large-scale integration and use of neural network models, especially in critical embedded systems, their security assessment to guarantee their reliability is becoming an urgent need. More particularly, models deployed in embedded platforms, such as 32-bit microcontrollers, are physically accessible by adversaries and therefore vulnerable to hardware disturbances. We present the first set of experiments on the use of two fault injection means, electromagnetic and laser injections, applied on neural networks models embedded on a Cortex M4 32-bit microcontroller platform. Contrary to most of state-of-the-art works dedicated to the alteration of the internal parameters or input values, our goal is to simulate and experimentally demonstrate the impact of a specific fault model that is *instruction skip*. For that purpose, we assessed several modification attacks on the control flow of a neural network inference. We reveal integrity threats by targeting several steps in the inference program of typical convolutional neural network models, which may be exploited by an attacker to alter the predictions of the target models with different adversarial goals.**

I. Introduction

Security of Machine Learning (ML) models is one of the most important challenge of modern Artificial Intelligence, amplified by the massive deployment of models (more particularly neural networks) in a large variety of hardware platforms. Those platforms include devices with strong constraints in terms of memory, computing ability, latency or energy (e.g., for IoT-oriented applications). The *adversarial* and *privacy-preserving* ML communities have already demonstrated an impressive set of threats that target the integrity, confidentiality and availability of models [18]. However, most of these attacks can be referred as *theoretical* or *algorithmic* since they consider a model as an *abstraction* and do not rely on the specific features of their software or hardware implementations. Most recently, the attack surface has been significantly widened with such *implementation*-based attacks that leverage software or hardware characteristics as well as theoretical backgrounds highlighted by previous attacks. It is the case for *weight-based adversarial attacks* such as the Bit-Flip Attack (BFA) [20] that directly disturbs the internal parameters of a deep neural network model stored in memory (typically, DRAM or Flash). Interestingly, in the BFA, the selection of the most sensitive parameters follows a gradient-based approach similar to classical white-box *adversarial examples* crafting methods. This leads to only a few bit-flips to drop the accuracy of a state-of-the-art convolutional neural network to a random-guess level. Another example, is the use of side-channel analysis [12] or fault injection attacks (as rowhammer in [19]), to totally or partially recover the values of parameters so that it could significantly increase the efficiency of a *model extraction* attack that aims at stealing a black-box protected model.

Except for passive side-channel analysis, most of these new implementation-based threats are data-oriented fault injection attacks targeting the stored parameters. In this work, we highlight another important attack vectors caused by fault injection that target the instruction flow, more particularly with *instruction skips*. To the best of our knowledge, this work is the first to demonstrate instruction skip with laser and electromagnetic fault injection in the inference of neural network models deployed in a Cortex-M 32-bit microcontroller.

Our contributions are the followings:

- We used two injection means on a Cortex-M4 platform, electromagnetic and laser injections, and target the inference of a standard convolutional neural network performing an image classification task.
- We demonstrate and analyze the impact of a single instruction skip at different critical paths of the inference: convolutional layers, bias additions, activation functions.
- We highlight two potential adversarial exploitation: memory effect and forced prediction.

II. Background & Related works

A. Background

Neural network models. A supervised neural network model $M_\Theta(x)$ is a parametric model trained to optimally map an input space $\mathcal{X} = \mathbb{R}^n$ (e.g., images) to an output space $\mathcal{Y}$. For a classification task, $\mathcal{Y}$ is a finite set of labels $\{1, ..., C\}$. The neural network model $M_\Theta : \mathcal{X} \rightarrow \mathcal{Y}$, with parameters Θ (also referred as *weights*), classifies an input $x \in \mathcal{X}$ to a set of raw or normalized scores in $\mathbb{R}^C$ so that the predicted label is $\hat{y} = \arg\max(M_\Theta(x))$. M_Θ is trained by minimizing a loss function $\mathcal{L}\big(M_\Theta(x), y\big)$ that quantifies the error between the prediction $M_\Theta(x)$ and the *groundtruth* y. The training process aims at finding the best parameters that minimize the loss on the training dataset.

A Perceptron (also called *neuron*) is the basic functional element of a neural network. It first processes a weighted sum of the input $x = (x_0, ..., x_j, x_{n-1}) \in \mathbb{R}^n$ with its trainable parameters θ and b (called *bias*), then it non-linearly maps the output thanks to an *activation function* σ: $a(x) = \sigma(\theta_0 x_0 + ... + \theta_{n-1} x_{n-1} + b)$, where a is the perceptron output. A classical activation function is the rectified linear unit (ReLU) defined as $\sigma(x) = max(0, x)$.

MultiLayer Perceptron (MLP) are deep neural networks composed of several vertically stacked neurons called *layers*. These layers are called *fully-connected* or *dense* or even *linear*. For a MLP, a neuron from layer l gets information from all neurons belonging to the previous layer $l-1$, therefore the output of a neuron is defined as in 1:

$$a_j^l(x) = \sigma\Big(\sum_{i \in (l-1)} \theta_{i,j} a_i^{l-1} + b_j \Big) \tag{1}$$

where $\theta_{i,j}$ is the weight that connects the j^{th} neuron of the l^{th} layer and the i^{th} neuron of the previous layer $(l-1)$, b_j is the bias of neuron j of layer l and a_i^{l-1} and a_j^l are the outputs of neuron i of layer $(l-1)$ and neuron j of layer l, respectively.

Convolutional Neural Network (CNN) is another type of neural network models that used convolutions with a set of *kernels* (also called *filters*). The trainable weights are the parameters of the kernels and are shared among the input. The kernels are usually square with low dimensions, typically 3x3 for image classification. Therefore, for a *convolutional layer* composed of K kernels of size Z applied on an input tensor of size $H \times W \times C$, the weights tensor Θ will have the shape $[K, Z, Z, C]$ (i.e., KCZ^2 parameters without bias, $(K+1)CZ^2$ otherwise). A naive implementation of the convolution of an input tensor X and a set of K kernels is detailed in algorithm 1.

Algorithm 1 Convolution layer (K kernels)

Input: Tensor X of size $H \times H \times C$, parameters tensor Θ of size $Z \times Z \times C \times K$, bias tensor of size K
Output: Tensor Y of size $H \times H \times K$

```
for k in [1, K] do
    for x in [1, H] do
        for y in [1, H] do
            Y_{x,y,c} = B_k
            for m in [1, Z] do
                for n in [1, Z] do
                    for c in [1, C] do
                        Y_{i,j,c} += θ_{m,n,k,c} · X_{x+m,y+n,k}
return Y
```

The output is also mapped with an activation function such as ReLU. Then, a third operation is applied with a *pooling* process that aims at reducing the dimensions of the output tensor by locally summing it up with some statistics. A classical approach is to apply a *Max pooling* or an *Average pooling* with a 2x2 kernel over the output tensor Y of size $H \times H \times K$ so that the resulting tensor is half the size $(H/2) \times (H/2) \times K$. Pooling also provides interesting (small) translation invariance property.

Embedded models. For a typical 32-bit microcontroller, the model parameters are stored in the Flash memory and internal computations (i.e., mainly multiply-accumulations and non-linear activation) are processed in SRAM. To embed complex ML models and fit the memory and latency requirements, classical compression techniques encompass model pruning [26] and quantization [25], [5]. For 32-bit MCU, 8-bit quantization of the weights is a standard performed as a post-processing step (after training) or at training step with training-aware quantization methods. Post-training 8-bit quantization is proposed as the default configuration in many deployment tools (e.g., TF-Lite, CubeMX.AI, NNoM, MCUNet) and may be applied for both weigths and activation outputs.

We used the NNoM (Neural Network on Microcontroller)[1] deployment framework, an open-source library with a full access to the source code (C) that enables 8-bit quantization for the weights, biases, activation values and output scores. The quantization is performed in the same way as in ARM CMSIS-NN [14] and relies on a uniform symmetric powers-of-two scheme (Eq. 2) that avoid division operation with only integer additions, multiplications and bit shifting.

$$x_i = \lfloor x_f \cdot 2^{7-dec} \rceil, \; dec = \lceil log_2\big(max\big(|X_f|\big)\big) \rceil \tag{2}$$

where X_f is a 32-bit floating point tensor, x_f a value of X_f, x_i its 8-bit counterpart and 2^{7-dec} the quantization scale.

Fault injection attacks (FIA) are active hardware threats [2] that usually require a physical access to the victim device [4]. Fault injection techniques gather global approaches such as voltage or clock glitching and moderate/high-cost methods such as laser (LFI) or electromagnetic (EMFI) that reach high temporal and spatial accuracy [1]. EMFI involves generating a magnetic field that causes voltage variations in the circuit, leading to alter propagation times of the signals through logic gates. This can result in faults where assembly instruction are modified or skipped. For LFI, a laser diode emits photons that create a photocurrent when they reach the sensitive points of the targeted microcontroller, resulting in voltage variations. This can change bit values, leading to instruction opcode modifications, which can transform one instruction into another or the `nop` instruction. In this case, we obtain an *instruction skip* similar to those obtained with EMFI. Interestingly, even though these two methods use different physical mechanisms, the results can be similar. Importantly, because of their precision and effectiveness, EMFI and LFI are standard fault injection means used in hardware security testing laboratories for security assessment or certification purposes [23].

B. Related works and positioning

Fault injection against deployed neural network models mainly focus on the alteration of the internal parameters stored in memory or the instructions flow. A reference for parameter-based attack is the Bit-Flip Attack (BFA) [20] that has been practically demonstrated with rowhammer on a DRAM platform [22]. Typically, BFA successes in dropping the average

[1] https://majianjia.github.io/nnom/

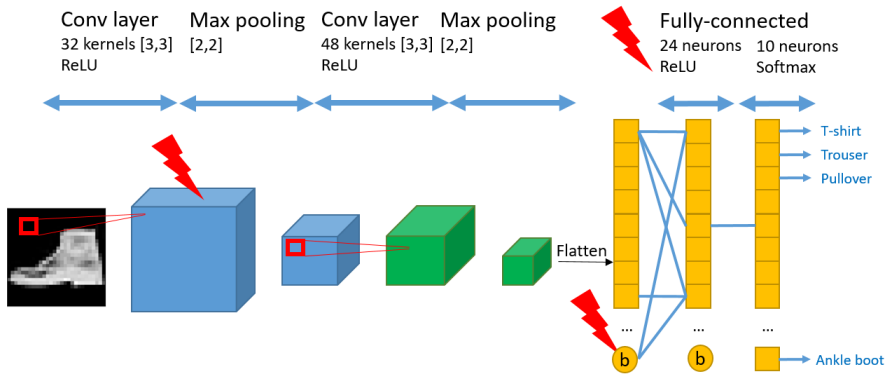


Fig. 1: Illustration of the CNN model. Red lightnings indicate the three investigated attack paths with instruction skip.

accuracy of a state-of-the-art CNN model to a random-guess level with only a few tens of bit-flips. Rowhammer and bit-flips on the parameters have also been exploited for a model extraction scenario [19] where the adversary knows the model architecture but has only access to less than 10% of the training dataset. The attack first uses rowhammer (as in the rambleed attack [13]) to guess the value of the most significant bit of almost 90% of the parameters of a victim model. Then, the adversary trains a *substitute* model by constraining the weigths values with information previously extracted.

In the context of embedded neural network models, laser fault injection has been demonstrated on a 8-bit microcontroler (ATmega328P) by Breier et.al. [4] by targeting the instructions of some activation functions (ReLU, sigmoid, tanh implemented in C). Then, simulations on a 4-layer MLP trained on MNIST showed that it is necessary to perform a lot of faults (more than 50% neurons faulted) on the last hidden layer to reach a reasonable attack success rate ($> 50\%$). Other simulation works by Jap et.al. [11] showed that a single bit modification on the Softmax activation function at the end of a neural network can lead to a misclassification. Liu et.al. [16] achieved misclassification using clock glitch on a FPGA-based deep learning accelerator. Changing other physical parameters such as supply voltage can decrease accuracy. Salami et.al. [21] demonstrated on FPGA-based CNN accelerators that in order to decrease the accuracy, it was necessary to reduce the supply voltage by at least 25%.

To the best of our knowledge, our work is the first to demonstrate the impact of a single fault disrupting the instruction flow on the performance of a CNN model deployed in a Cortex-M platform. Contrary to [4] we consider a full inference program embedded with state-of-the-art deployment tool and analyze different attack paths. Additionally, with this scope, our experiments are the first to demonstrate both electromagnetic and laser injections for a complete inference program (CNN trained on the standard Fashion MNIST dataset) on a 32-bit ARM Cortex-M4 platform.

III. Experimental Setup

A. Device under test, model and dataset

We used a 32-bit ARM Cortex-M4 microcontroller as target which can operate at a frequency of up to 100 MHz. The device has a 512KB Flash memory and a RAM of 128KB.

We focused our experiments on a typical convolutional neural network model trained for a supervised image classification task. We used the standard Fashion-MNIST dataset, which consists of 70,000 (60K for training and 10K for testing) 28x28 grayscale images divided into 10 cloth categories. Our model is composed of two convolutional layers with respectively 32 and 48 kernels of size $[3, 3]$ with ReLU as activation. Each layer is followed by a Max pooling layer of size $[2, 2]$. The end of the model is composed of two fully-connected layer with respectively 24 and 10 neurons. The activation function is ReLU except for the last layer which typically used Softmax to provide normalized outputs. The model (illustrated in Fig 1) has a total of 70,914 parameters and reaches an accuracy of 91% on the complete test set.

The trained model (with TensorFlow v2) is deployed on our target device with the NNoM library [17] that offers a 8-bit model quantization of the parameters, activation and output prediction scores and a complete white-box access to the inference code. The accuracy of the deployed 8-bit model is evaluated directly on the development board over limited random sets of 100 inputs. We observed that the implementation of the quantization scheme in NNoM raises integer overflow that may impact the accuracy depending on the test sets. Over different test sets the model has an accuracy from 77% to 88% (i.e., close to the accuracy of the full-precision model over the 10K test set). However, we noticed that our results are similar from one test sets to another (even by fixing the overflow issue). Therefore, for our fault injection experiments, we keep the same 100-input test set that reaches the lowest precision (77%).

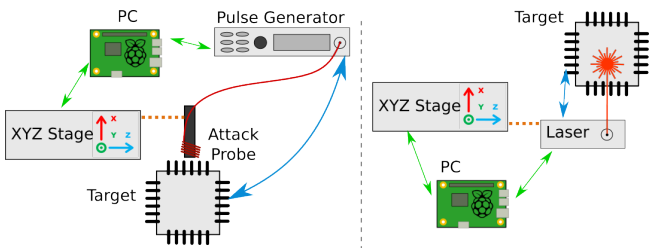


Fig. 2: EMFI (left) and LFI (right) setups.

B. Fault injection setup

For EMFI, we used a voltage pulse generator that can deliver 200 V pulses, connected to an injection probe as illustrated in Figure 2. The control voltage of the pulse generator is 200 V with a rise time of 2 ns for a pulse width of 10 ns.

For LFI, we use an 1,064 nm infrared laser beam with pulse energy of 0.1W, a pulse duration of 50 ns and a spot size of 5 µm. The laser attacks are performed on the rear face of the silicon, which requires to decapsulate the component. For the laser experiments, the operating frequency of the card is reduced from 100 MHz to 50 MHz.

The triggering of the laser and electromagnetic shot is synchronized by a signal generated by the target device and the delay between its rising edge and the triggering of the shot is adjustable. This enables to control the magnetic field and laser beam to target a specific instruction.

C. Test code

First, we need to map the sensitive areas of our device, that is to say the locations where exploitable faults are obtained. We used a simple assembly test code (Fig. 3) to identify the locations where fault injections can be successfully performed. This code performs simple register manipulations and is long enough to not require precise time synchronization. By comparing the readback values of the registers after execution of the test code with and without fault injection, we aimed at identifying a location where a *sub* instruction is not executed. We successfully obtained an *instruction skip* type fault.

```
movs r3, #1
movs r4, #55
movs r5, #55
nop
...
nop
sub R4, R4, R3
...
sub R4, R4, R3
sub R5, R5, R3
...
sub R5, R5, R3
//readback of registers
```

Fig. 3: Test code manipulating registers

With EMFI, the result of this mapping procedure is a 200 µm by 100 µm sensibility area within a chip size of 4 mm by 4 mm. It was necessary to use a very precise electromagnetic probe as described in [9]. The positions of the probe for successful injections are indicated in blue on Figure 4. The positions used for LFI are distinct from those for EMFI and depicted in red in Figure 4. These results are consistent compared to other reference works such as [7].

IV. Instruction skip on a CNN inference

After characterizing the sensitive areas according to our fault model (instruction skip) and injection means (EM and laser pulse), we detail, in this section, three experiments on the inference program of our convolutional neural network model trained and tested on FashionMNIST. We targeted three attack paths that correspond to critical elements of a CNN model:

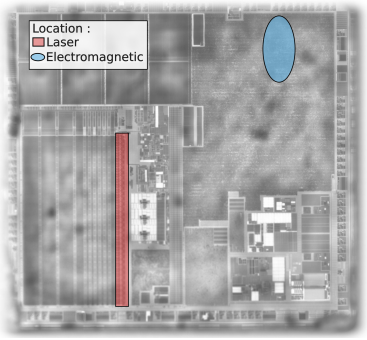


Fig. 4: Sensitive areas according to the different injection means.

```
//for (i=0, shift_idx=0; i<ch_im_out;
      i++, shift_idx+=shift_steps)
ldr   r3, [r7, #100]
adds  r3, #1
str   r3, [r7, #100]
ldr   r2, [r7, #72]
ldr   r3, [r7, #68]
add   r3, r2
str   r3, [r7, #72]
ldrh.w   r3, [r7, #132]
ldr   r2, [r7, #100]
cmp   r2, r3
blt.w   80037fa
```

Fig. 5: Assembly code of the loop over convolution filters.

- the first convolutional layer that extract low-level *features* from the input;
- the bias addition and the ReLU activation function for the first fully-connected layer.

An illustration of these attack paths on our CNN model is presented in Fig. 1.

A. Targeting the first convolution layer

Experiments. Parameter-based attacks such as the BFA [20] have highlighted the sensitivity of the first convolutional layers of CNN models against adversarial perturbations [10]: an alteration of the initial features map grows and propagates through the network leading to a misprediction. However, this is achieved by targeting some specific kernels since others are resistant against the perturbation of their parameters. With our experiments, with only one instruction skip on the main convolution loop, we aimed at analyzing the sensitivity of this critical part of the inference.

During a convolution operation, a loop over the filters is executed to carry out the convolution computations (as in Algorithm 1). The assembly code of this loop is given in Fig. 5. With an instruction skip, our objective is to prematurely interrupt the loop over the filters, thereby halting the execution of the convolutions. The impact of the instruction skip strongly depends of the implementation. In our case, such a fault completely breaks the convolution process: if a fault is injected for the kernel j then the convolutions with kernels $[j+1; K]$ are not processed. We discuss that point in Section V.

Through simulation, we observed that a valid attack path is to jump the branch instruction (highlighted in red). Fig. 6(a)

reports the impact on the model accuracy of simulated fault injections (the skip of the jump instruction, hence ending the computation loop) as a function of the last processed filter index (the remaining filters were skipped). We used five different random test sets of 100 images each. Despite minor variations, the results on each dataset are comparable. Thus, we can therefore consider that the 100 images of the first test set are representative of the general behavior (even if the accuracy is slightly below the average).

Logically, the accuracy of the model decreases as the filters loop was exited earlier. However, we observe that the last kernels do not have a significant impact: exiting the loop from the 17^{th} kernel – i.e. not performing almost half of the kernels – has a limited effect on the accuracy (from 75% to 82%). A possible explanation is that most of the deep neural network models are over-parametrized which can be observed when applying compression techniques such as *model pruning* that typically remove, for standard CNN, a large part of kernels without any significant drop of performance. Here, we can make the hypothesis that most of the useful features are *captured* by the first kernels.

To demonstrate this attack in practice, we conducted EMFI on the 100 images of the dataset 1. The experimental and simulation results, presented in Fig. 6 (b), are almost identical. However, since the repeatability of the EMFI is not perfect, it happens that the fault injection is not successful, which explains some results slightly above the simulation curve (accuracy is higher by a few percents). Other times, the EMFI will cause the board to crash, which occurs at filters 14 and 19 explaining the significant accuracy drops at these indexes. For LFI, it appeared that triggering the position and the injection delay at different filters within the loop was very challenging. Therefore, we only processed one laser fault injection on the first filter to see if the LFI result was similar with our simulation and EMFI. We logically reached a random-guess level (10%) since the forward pass is completely altered.

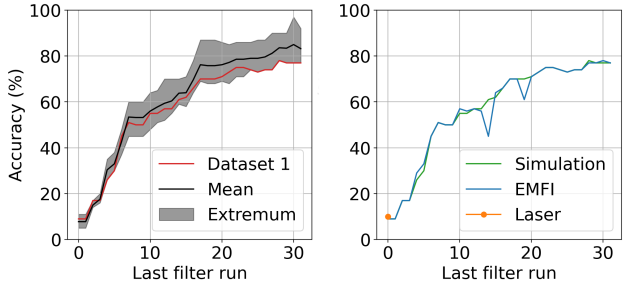


Fig. 6: **a.** Simulations over 5 different test sets (dataset used in **a** is refered as *Dataset 1*) **b.** Accuracy with an instruction skip on the first convolution layer (simulation and EMFI)

Exploitation. Interestingly, we observed that exiting the loop prematurely at the first filters causes the inference results to match with the last correctly executed inference, as exemplified in Figure 7. This *memory effect* can be exploited in critical applications, such as the authentication of an unauthorized person after one with the appropriate permissions. As this type of attack leaves no trace or fault in the circuit and will be overwritten by correct data after the next prediction, it can be difficult to detect, except for monitoring the process time (e.g. by using an instruction counter). This attack is possible because the result of the convolution layer calculations is stored in RAM. Therefore, to prevent such attacks, it is compulsory to perform a RAM memory reset between two inferences to clear the stored results.

Fig. 7: Illustration of the *memory effect* when skipping the first convolutional layer.

B. Targeting the bias values

Experiments. The first dense layers in the model contains biases that can be modified to alter the inference results. By modifying the `store` instruction that initializes the biases, we observed significant corruption of bias values resulting in mispredictions in our simulations. Specifically, we wrote an address value to the register instead of the bias so that the bias takes a significantly higher value. Fault injections were performed using laser injection and the value is similar to the simulation. Although the induced fault differs with EMFI, it results in bias values that are different from the initial values. This has significant effects on the inference results as shown in Table I. The accuracy is only detailed for the first 4 biases but they are comparable for all 24 neurons.

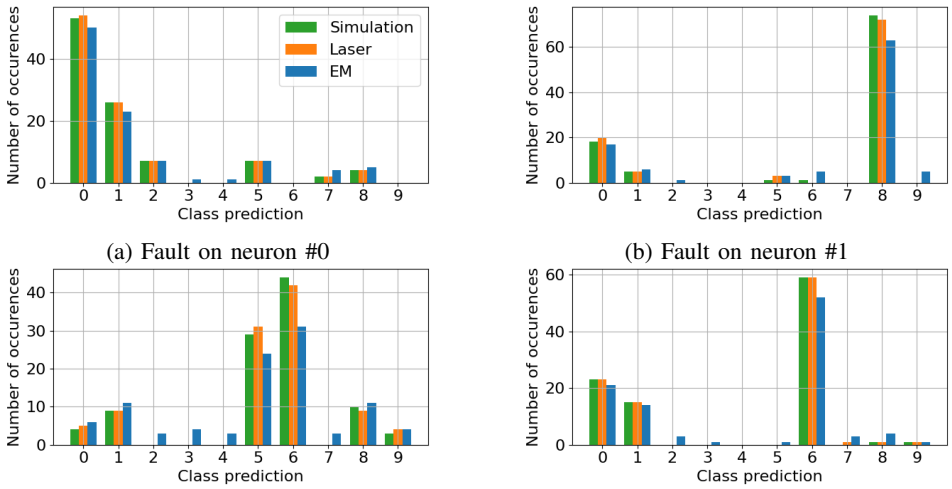
TABLE I: Accuracy when faulting the bias of one of the 4 first neurons

Neuron tested	Accuracy Simulation	Accuracy Laser	Accuracy EM
0	38%	37%	40%
1	26%	26%	37%
2	40%	41%	56%
3	28%	28%	35%
no fault	77%	77%	77%

Exploitation. The accuracy of the model was found to be highly sensitive to a single injection, with modifications to the bias of a neuron favoring certain predictions over others, as presented in Figure 8. For example, modifying the bias of the first neuron resulted in the model mostly predicting *T-shirt* (label 0) about 50 times out of 100 tests, regardless whether the simulation, electromagnetic or laser injection we used. The results for the other bias calculations are similar for the accuracy and majority inferences. These results demonstrate that it is possible to significantly bias the model predictions.

An attacker can then easily force a prediction by choosing to fault the calculation of a single bias;

To protect against this type of attack, the bias value of the dense layer can be reset if it exceeds a certain bound. In simulation on the first 4 bias calculations with bounds between -2048 and 2048 to the output value of the neuron, we obtained the same results to those without fault injection. We recovered



(a) Fault on neuron #0

(b) Fault on neuron #1

(c) Fault on neuron #2

(d) Fault on neuron #3

Fig. 8: Majority output inference when faulting bias on the 4 first neurons

the initial accuracy which had been reduced between 26 and 40% with the fault injection.

C. Targeting the activation function

Experiments. We targeted the activation function of the first *dense* layer composed of 24 neurons (with instruction skip faults). For this layer, the activation function is ReLU ($\sigma(x) = \max(0; x)$) which is generally used in most state-of-the-art deep convolutional network models. Fig. 9 (a) presents the assembly code of ReLU used in our NNoM implementation.

We first simulated the impact of instruction skip. It appears that it is possible to alter ReLU in two different ways:

- force a reset (target the blue instructions in Fig. 9 (a)): output is zero while the input is positive. Therefore, the activation is constantly null: $\sigma(x) = 0$.
- skip the reset (target the red instructions in Fig. 9 (a)): the activation is turned into the identity function and is not set to zero if the input is negative: $\sigma(x) = x$.

Interestingly, our simulations shown that skipping the reset causes more mispredictions than forcing a reset. Consequently, we focused on this kind of attack for our experiments.

Exploitation. Skipping the reset of the ReLU activation function has minor impact on the accuracy of the model, as depicted in the Figure 9 (b). Therefore, when targeting the first fully-connected layer, our experiments show that it is less effective to target the ReLu activation function than the biases if we seek to reduce the accuracy. This observation is coherent with the first experiments from Breier et.al. [4] that demonstrated the need of performing a significant number of faults on ReLU to alter the overall accuracy of a 5-layer MLP model (faults were injected on the penultimate layer). We observe that the results of simulation experiments are similar to those obtained with LFI, with an accuracy decreasing to 74%. The accuracy was reduced to 61% with EMFI where we observed a certain number of outputs equal to zero (less than those without faults) that indicates few forced resets (i.e. forcing a reset while the input is positive). This behavior did not appear in simulation or with LFI. That highlights the fact that, even though simulations can predict the majority of behaviors, experimental studies are compulsory to accurately account for the effect of a complex fault model.

V. Discussions

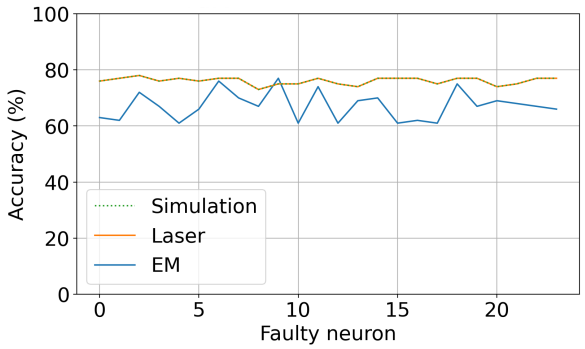
A. Comparison of injection techniques and limitations

LFI is known as a very effective injection means because of a high temporal and spatial accuracy [23]. However, contrary to EMFI, silicon has to be visible to perform LFI, therefore it is necessary to decapsulate the components, a delicate step that complicates the implementation of the attack. Another practical point between LFI and EMI is the cost of the characterization environment, significantly higher for LFI with approximately 100k € compared to 30k € for the EMFI bench. Experimentally, the search for the sensitive zone was more tedious in EMFI as we encountered many freezes and restarts of the target device, which was less common with the laser pulses. Moreover, our fault model being a single instruction skip, it was straightforward to simulate the impact of such faults on the inference process and compare the match between simulation outputs and observed ones with real injections. We observed a higher similarity between simulations and LFI than with EMFI (typically for the experiments on ReLU in IV-C). This difference is explained by a lower repeatability of EMFI due to its lower spatial accuracy compared to laser.

A classical limitation of our security characterization is related to the synchronization of the injections: dedicated instructions inserted in our test programs activated an output of the target that was used to trigger our fault injectors after a programmable delay (i.e. our experiments were performed in a white-box setting). However, since the implementation code includes many loops that generate an electromagnetic leak with a particular and detectable signature, one can consider the use of a device to temporally trigger the injections, such as presented in [8]. We keep the use of such techniques for further experiments.

```
// if (data[i] < 0) {
ldr   r2, [r7, #4]
ldr   r3, [r7, #12]
add  r3, r2
ldrsb.w r3, [r3]
cmp  r3, #0
bge.n  80042ee
//         data[i] = 0;
ldr r2, [r7, #4]
ldr  r3, [r7, #12]
add  r3, r2
movs r2, #0
strb r2, [r3, #0]
//     }
```

(a) ReLU assembly code.



(b) Impact on accuracy (Simulation, LFI, EMI)

Fig. 9: Target ReLU activation function. (Left) Assembly code. (Right) Impact on accuracy (simulation, LFI, EMI, laser and simulation curves overlap) when skipping the zeroing of the ReLU activation function.

B. Neural network implementation

Our experiments are based on a classical CNN model deployed thanks to the NNoM platform, which has the advantage of being open source, covering classical types of layer and reaching interesting inference performance. Further works would investigate how different implementations (e.g., MCUNet [15]) behave when exposed to instruction skips even for challenging black-box tools (e.g., STMCubeMX.AI).

Target a single convolutional kernel. According to what we observed with the NNoM implementation, we can highlight some interesting outcome that pave the way to further analysis of different types of implementations for the most critical functional structures of the inference (and then lead to more robust neural network inference implementations). For example, we simulated an implementation that allows for the non-execution of a single filter in the convolution layer, rather than producing a premature exit as presented in section IV-A. This attack is also possible on the NNoM implementation by replaying the instruction that increments the loop counter. As a result, an adversary could skip only one filter and then exclude one channel from the resulting features map. Our results show that the accuracy can be reduced by up to 20% depending on the targeted filter. This significant drop of accuracy is consistent with what observed with only few bit-flips with the BFA [20] that usually result in *turning off* an initially important kernel.

Using CMSIS-NN. With NNoM, it is possible to use as backend the CMSIS-NN [14] library from ARM[2] and, in that case, the implementations may differ. Although our results were obtained without using the CMSIS-NN library, the bias loading code remains the same, making the attack transferable. For the convolution operation, CMSIS-NN uses the `im2col` algorithm [14] that transforms the input image and the set of filters in a new matrix representation so that the convolution is processed through efficient matrix multiplications. We performed a first set of simulation tests on an implementation using the CMSIS-NN library that shown that a single instruction skip in the loop over the dimension of the output tensor (i.e. the number of kernels) of the first convolutional layer causes a forced output: we obtained the label 0 (T-shirt) for 98 inputs over 100. This behavior is also highly critical and may be exploited by an adversary to impact the integrity of the model. Thus, further experiments are necessary to analyze potential vulnerabilities to the `im2col` implementation.

[2]also used in STMCubeMX.AI from STMicroelectronics

C. Protections and Exploitation for confidentiality concern

As a first step, we focused our experiments on the direct impact of faults on the model performance (here, the standard accuracy), i.e. we mainly focus our work on task-integrity purpose. However, recent milestone works such as [19] demonstrated how fault injection techniques can be exploited to leak critical information about a model that may help an adversary for a model extraction attack. An interesting future work is to analyze what kind of information about the parameters can be revealed by one or several instruction skips. A first insight is that skipping filters will give important information about the importance of these filters for the prediction. This is what we observed with our experiments on the first convolutional layer (section IV-A) since we highlighted the fact that exiting the loop from the 17^{th} kernel has a limited effect on the accuracy, meaning that most of the most important kernels are the first ones. Therefore, an adversary may put his effort only on recovering a small part of the parameters which can significantly facilitate the attack.

Many software countermeasures have been proposed by the hardware security community to protect critical algorithms (e.g., cryptographic modules) [3] and we mentioned some obvious implementation advice to reduce the impact of the faults we performed in section IV. However, the main challenge in terms of defense, is the length of the inference code that contains many loops. Therefore, many protections based on local verification (including ones relying on redundancy) can lead to prohibitive additional costs and significantly reduce the performance of the system. Promising defense schemes encompass the protections based on CFI (Control Flow Integrity) that aims at checking a program execution flow and detecting potential alteration [24], [6].

VI. Conclusion

We investigated the effect of single instruction skip fault attacks on a neural network model embedded in an ARM

Cortex-M4 microcontroller. We used two standard powerful injection means, laser and electromagnetic injection, as well as simulations. We identified several vulnerabilities at different positions of the model architecture. More particularly, it is possible to prematurely exit the loop over the convolutional filters, leading to incorrect predictions, and even to a so-called memory effect if the whole convolution loop is skipped (if so, the faulted inference outputs the prediction of the previous one). Additionally, we demonstrated that instruction skips can alter the bias computation in fully-connected layer that may force the output prediction towards a chosen label. In a context of critical security concerns related to the large-scale deployment of AI systems, with upcoming regulation and certification actions, these results (the first with such an experimental scope) highlight the urgent need to properly evaluate the intrinsic robustness of embedded models and pave the way to further analysis to cover more models types, devices as well as assess the relevance of state-of-the-art protections for embedded machine learning inference programs.

ACKNOWLEDGMENT

This work is supported by (CEA-Leti) the European project InSecTT (ECSEL JU 876038) and by the French ANR in the *Investissements d'avenir* program (ANR-10-AIRT-05, irtnanoelec); and (MSE) by the ANR PICTURE program. This work benefited from the French Jean Zay supercomputer with the AI dynamic access program.

REFERENCES

[1] Agoyan, M., Dutertre, J.M., Mirbaha, A.P., Naccache, D., Ribotta, A.L., Tria, A.: How to flip a bit? pp. 235–239 (2010)

[2] Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. Proceedings of the IEEE **100**(11), 3056–3076 (2012)

[3] Barenghi, A., Breveglieri, L., Koren, I., Pelosi, G., Regazzoni, F.: Countermeasures against fault attacks on software implemented aes: effectiveness and cost. In: Proceedings of the 5th Workshop on Embedded Systems Security. pp. 1–10 (2010)

[4] Breier, J., Hou, X., Jap, D., Ma, L., Bhasin, S., Liu, Y.: Practical fault attack on deep neural networks. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 2204–2206 (2018)

[5] Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. Advances in neural information processing systems **28** (2015)

[6] De Clercq, R., Götzfried, J., Übler, D., Maene, P., Verbauwhede, I.: Sofia: software and control flow integrity architecture. Computers & Security **68**, 16–35 (2017)

[7] Dutertre, J.M., Menu, A., Potin, O., Rigaud, J.B., Danger, J.L.: Experimental analysis of the electromagnetic instruction skip fault model and consequences for software countermeasures. Microelectronics Reliability **121**, 114133 (2021)

[8] Fanjas, C., Gaine, C., Aboulkassimi, D., Pontié, S., Potin, O.: Combined fault injection and real-time side-channel analysis for android secure-boot bypassing. In: Buhan, I., Schneider, T. (eds.) Smart Card Research and Advanced Applications. pp. 25–44. Springer, Cham (2023)

[9] Gaine, C., Aboulkassimi, D., Pontie, S., Nikolovski, J.P., Dutertre, J.M.: Electromagnetic Fault Injection as a New Forensic Approach for SoCs. In: 2020 IEEE International Workshop on Information Forensics and Security (WIFS). pp. 1–6. IEEE, New York, United States (Dec 2020)

[10] Hector, K., Moëllic, P.A., Dumont, M., Dutertre, J.M.: A closer look at evaluating the bit-flip attack against deep neural networks. In: 2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS). pp. 1–5. IEEE (2022)

[11] Jap, D., Won, Y.S., Bhasin, S.: Fault injection attacks on softmax function in deep neural networks. p. 238–240. CF '21, Association for Computing Machinery, New York, NY, USA (2021)

[12] Joud, R., Moëllic, P.A., Pontié, S., Rigaud, J.B.: A practical introduction to side-channel extraction of deep neural network parameters. In: Smart Card Research and Advanced Applications: 21st International Conference, CARDIS 2022. pp. 45–65. Springer (2023)

[13] Kwong, A., Genkin, D., Gruss, D., Yarom, Y.: Rambleed: Reading bits in memory without accessing them. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 695–711. IEEE (2020)

[14] Lai, L., Suda, N., Chandra, V.: Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. arXiv preprint arXiv:1801.06601 (2018)

[15] Lin, J., Chen, W.M., Lin, Y., Gan, C., Han, S., et al.: Mcunet: Tiny deep learning on iot devices. Advances in Neural Information Processing Systems **33**, 11711–11722 (2020)

[16] Liu, W., Chang, C.H., Zhang, F., Lou, X.: Imperceptible misclassification attack on deep learning accelerator by glitch injection. In: 2020 57th ACM/IEEE Design Automation Conference (DAC). pp. 1–6 (2020)

[17] Ma, J.: A higher-level Neural Network library on Microcontrollers (NNoM). Zenodo (Oct 2020)

[18] Papernot, N., McDaniel, P., Sinha, A., Wellman, M.P.: Sok: Security and privacy in machine learning. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 399–414. IEEE (2018)

[19] Rakin, A.S., Chowdhuryy, M.H.I., Yao, F., Fan, D.: Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 1157–1174 (2022)

[20] Rakin, A.S., He, Z., Fan, D.: 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)

[21] Salami, B., Onural, E.B., Yuksel, I.E., Koc, F., Ergin, O., Kestelman, A.C., Unsal, O.S., Sarbazi-Azad, H., Mutlu, O.: An experimental study of reduced-voltage operation in modern fpgas for neural network acceleration (2020)

[22] Yao, F., Rakin, A.S., Fan, D.: {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)

[23] Yuce, B., Schaumont, P., Witteman, M.: Fault attacks on secure embedded software: Threats, design, and evaluation. Journal of Hardware and Systems Security **2**, 111–130 (2018)

[24] Zgheib, A., Potin, O., Rigaud, J.B., Dutertre, J.M.: A cfi verification system based on the risc-v instruction trace encoder. In: 2022 25th Euromicro Conference on Digital System Design (DSD). pp. 456–463. IEEE (2022)

[25] Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)

[26] Zhu, M., Gupta, S.: To prune, or not to prune: Exploring the efficacy of pruning for model compression. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings. OpenReview.net (2018)