SCI-FI: a Smart, aCcurate and unIntrusive Fault-Injector for Deep Neural Networks

G. Gavarini, A. Ruospo, E. Sanchez
Politecnico di Torino, DAUIN, Torino, Italy

2023 IEEE European Test Symposium (ETS) | 979-8-3503-3634-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ETS56758.2023.10173957

***Abstract*—In recent years, the reliability of Deep Neural Networks (DNN) has become the focus of an increasing number of research activities. In particular, researchers have focused on understanding how a DNN behaves when the underlying hardware is affected by a fault. This is a challenging task: slight changes in a network architecture can significantly impact how the network reacts to faults. There are several approaches to simulate the behaviour of a faulty network: the most accurate one is to perform low-level fault simulations. Nonetheless, this task is very time-consuming and costly to be implemented. Even though the injection time can be reduced by injecting faults at the application level, for sufficiently large networks, this time is still very high, requiring weeks to complete a single simulation. This work aims at providing a fast and accurate solution for injecting software-level faults in a DNN that is independent of its architecture and does not require any modification to its structure. For this reason, this paper introduces SCI-FI, a Smart, aCcurate and unIntrusive Fault-Injector. SCI-FI smartly reduces the fault injection time required for a complete fault simulation of the network by taking advantage of two fundamental mechanisms: Fault Dropping and Delayed Start. Experimental results from various ResNet, DenseNet and EfficientNet architectures targeting the CIFAR-10 and ImageNet datasets show that combining these techniques drastically reduces the simulation time, which can last up to 70% less.**

***Index Terms*—Deep Neural Network, Reliability, Speed-Up, Fault Injection, Fault Injector**

I. Introduction

Applications based on Deep Neural Networks (DNNs) have started during the last years, to pervade our life. Actually, predictive models based on DNNs have acquired more space on very different applications, ranging from object identification, facial recognition, healthcare applications, handwriting analysis, and many others. The advent of DNNs is not new, actually, the first perceptron was proposed in the second half of the last century; though, their massive use has been possible only in our days thanks to the improvements in their architecture and the hardware running the application.

The extensive use of DNNs also in safety-critical applications brought new issues related to the assessment of the reliability of the DNN-based applications. One of the most relevant mechanisms used to achieve the DNN assessment is based on performing a set of fault injection (FI) campaigns where single or multiple faults are injected during the DNN inference and the collected results are analyzed to determine how the systems behaves in the case of a fault. Clearly, these campaigns can be executed at different abstraction levels, starting from a very low one based on injecting faults in a hardware description of the real device running the DNN, up to higher abstraction levels exploiting, for example, the application level to perform the injections. The lower the abstraction level, the higher the accuracy of the fault simulation; however, using a higher abstraction level drastically reduces the time to fault simulate the targeted system.

Fault simulation techniques at the hardware level were intensively promoted during the 90's, in [1], for example, a very effective and optimized algorithm was proposed, and this kind of algorithms are adopted in very popular commercial fault simulation and ATPG tools. A different approach presented in [2] proposed to improve the fault injection campaigns by including different optimization mechanisms to reduce the time required to simulate each fault. The basic idea was to fully simulate the fault-free design, storing the golden run, and then, simulating every fault by loading the state of the design just before the fault is activated. At this time, the circuit is modified injecting the fault and the circuit continues its execution until the end of the run where the fault effects are collected.

In this paper, we propose a solution at software level that is similar to the one introduced in [2] at hardware level. The main idea behind our approach is to only run the faulty DNN during the time when the fault may create a difference compared against the golden run, and then, thanks to a set of very fast comparisons, understanding if the fault effect needs to be propagated or not until the end of the inference. The proposed fault injector is called SCI-FI, a Smart, aCcurate and unIntrusive Fault-Injector. SCI-FI basically exploits the advantage of two main mechanisms: Fault Dropping and Delayed Start. The experimental results have been collected on three different DNN architectures (i.e., ResNet, DenseNet and EfficientNet) targeting the CIFAR-10 and ImageNet datasets, showing that the proposed techniques are able to drastically reduce the fault simulation time, reaching up to a 70% reduction time.

The rest of the paper is organized as follows: in section II a brief panoramic of high-level FI is provided. Section III introduces the proposed approach, while Section IV defines the case study. Finally, Section V discusses the experimental results, and Section VI draws the conclusions.

II. Background and Related Works

In the last few years, there has been a significant amount of research conducted in the literature to offer methods to make DNN reliability analysis easier. Several of them rely on specific frameworks that support the execution of FI campaigns. They can be performed by considering only the DNN model or the entire system, including the underlying hardware. According to this and to other parameters, it is possible to classify the different FI methodologies for DNNs in three main categories: simulation-based, platform-based, and radiation-based. A detailed study as well as the advantages and disadvantages are presented in [3].

It is important to mention that the majority of FI tools are built on the basis of two popular frameworks: TensorFlow [4] and PyTorch [5], open-source machine learning and deep learning frameworks by Google and the Linux Foundation, respectively.

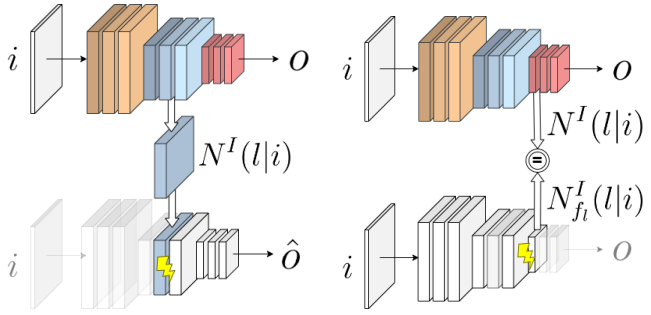
Chen *et al.* in [6] presented TensorFI, a high-level FI framework for TensorFlow-based applications. It is a flexible tool that can be used to inject faults at the TensorFlow graph level, particularly at the output of the TensorFlow operators. Since TensorFlow does not expose the operators and most of the execution occurs "behind the scenes", TensorFI is created by duplicating the original TensorFlow graph. Then, at inference time, it is possible to choose between the golden operators and the faulty ones. Only transient faults are injected in one of the following formats: by flipping a single bit in one (or all) the data item(s) of the target operator; by shuffling one (or all) the data item(s) of the target operator into random values; by changing the output of the target operator into all zeros.

Mahmoud *et al.* proposed a runtime perturbation tool for DNNs based on the PyTorch framework named PyTorchFI [7]. It allows performing fault injections in weights and/or neurons in convolutional operations. Perturbations in weights are performed offline by modifying the weight tensor, while neuron values are modified during the forward pass of a computational model by exploiting the `hook` functionality. Even though PyTorchFI operates at the application level of DNNs, it can also model lower level faults (such as register-level) by mapping them to single or multiple bit-flips in single or multiple neurons. A further FI tool for quantifying the resilience of DNNs was proposed in [8]. It enables the DNNs execution directly on the GPUs and targets permanent faults occurring in the memory, which is the unit hosting the neural network parameters. At the application level, errors are injected in the weights, the activations and the hidden states through bit-flips. Particularly, they are injected at construction time (static) and evaluation time (dynamic). The former are injected off-line, before the inference is executed. The latter are injected during the inference phase, introducing a minimum performance overhead.

III. Proposed Solution

SCI-FI is a fault injector that aims to speed up the software fault injection process on GPUs while correctly simulating the behaviour of the faulty network. One of the strengths of the FI is that it does not require modifying the network under exam. SCI-FI's workflow can be described as follows:

1) *Network Instrumentation*: this step is used to gather information about the network. The retrieved information is used both for generating the fault list and to speed up the injection process;
2) *Fault list Generation*: during this step, a fault list, based on the information collected from the instrumentation phase and the fault model selected, is generated. Faults can affect either the weights of the network (i.e., its parameters), or the elements of the output feature map of each layer (i.e., the neurons). A fault list can be statistical, where a reduced number of faults is obtained by imposing an error margin and a confidence level for the observed metric, random or exhaustive;
3) *Fault Injection Campaign*: in this step, faults are injected *batch-wise*. Typically, a DNN does not receive input individually, but receives multiple inputs in a single *batch*. A batch-wise injection means that, for every batch, all the faults from the fault list are injected one at a time.



(a) Delayed Start (b) Fault Dropping

Fig. 1: Graphic representation of the speed-up techniques. In both figures, the input features map of the golden network (on top) are used during the FI on the faulty network (on bottom).

The proposed SCI-FI combines various techniques that allow to speed up fault injection campaigns without any loss in the accuracy of the measurements.

In Section III-A we discuss how to avoid computing certain operations of the network that are unaffected by faults with *Delayed Start*. In Section III-B instead, we propose a *Fault Dropping* technique to drop faults that do not propagate the error. The solution of our FI to the drawbacks of Delayed Start and Fault Dropping are discussed in Section III-C. Finally, in section III-D we introduce the fault models implemented in SCI-FI. All the consideration discussed in the following sections are valid under the single-fault assumption.

A. Delayed Start

Fault simulating a neural network imposes some useful constraints on the environment we need to reproduce:

- The parameters of the network are always the same, with the only exception of those that get faulty;
- The inputs of the DNN are always the same, and constitute the injection test set I.

At the most basic level, a DNN is a collection of layers executed sequentially. While there are some exceptions (e.g., Skip Connections), usually, the input of a layer depends uniquely on the output of the previous layer. As a matter of facts, the implemented fault models in this fault injector have been assumed to having an impact only starting from a specific layer. As a consequence, for a fixed image, that layer input is always the same. In other words, given a network N, a layer $l \in N$, an image $i \in I$ and the set of all the possible faults F_l affecting the layer l, it holds that:

$$N^I(l|i) = N^I_{f_l}(l|i) \;\; \forall \, f_l \in F_l \tag{1}$$

Where $N^I(l|i)$ is the input feature map of the golden network's layer l for input i and $N^I_{f_l}(l|i)$ the input feature map of the same layer when the network is affected by any of the possible faults $f_l \in F_l$. This consideration can be extended to the set of faults F_{l^+} happening in all the layers L^+ after layer l.

Consequently, since the number of all the possible images $i \in I$ is limited, it is possible to store, for every layer l, its golden input

Network	Dataset	Dataset Size	Accuracy [%]	Injectable Weights [M]	Injected Weights	Injectable Neurons [M]	Injected Neurons
ResNet-20	CIFAR10	10,000	91.25	0.27	16,540	0.19	16,497
ResNet-56	CIFAR10	10,000	93.03	0.85	16,609	0.53	16,589
DenseNet-121	ImageNet	50,000	74.43	6.87	16,637	6.89	16,637
EfficientNet	ImageNet	50,000	77.69	5.29	16,636	6.76	16,637

TABLE I: Details of the CNNs under test. The accuracy is referred to CIFAR-10 test set and ImageNet validation set.

feature map $N^I(l|i)$. This means that, when a fault is injected in layer l *it is not necessary to simulate the behaviour of the whole network*, but it is possible to start the execution of the network from l using as input the stored golden value $N^I(l|i)$. This is shown in Fig. 1a. We refer to this technique as *Delayed Start*.

B. Fault Dropping

Recent works show that most of the faults do not affect the network output [9]–[11]. This is mostly true for 32-bit floating point representation, and very critical to other data representations [11]. However, there are other factors that can mask a fault, such as the Rectified Linear Unit (ReLU) activation function [12]. Indeed, it can prevent the propagation of faults on negative weights. When analysing DNNs used for image classification tasks, we can observe that faults can have different impacts [11]: they can (i) change the class predicted by the network (critical faults or Silent Data Corruption, i.e., SDC-1), (ii) change the vector score or (iii) have no impact at all. Similar classifications can be made for other kind of networks, such as networks used for image segmentation [12]. If the fault has no impact on the output, it is possible that its effect is masked early on.

As explained in Section III-A, the inputs and the parameters are always the same during a fault injection campaign. For a given input $i \in I$, it is possible to compare the output feature map $N^O_{f_l}(l|i)$ of a layer $l \in N$ affected by a fault $f_l \in F_l$ and compare it to its golden value $N^O(l|i)$. *If the input of a layer depends uniquely on the output of the previous layer*, it holds that *a fault with no impact on the output feature map of a layer, has no impact on the output of the network*, that is:

$$N^O(l|i) = N^O_{f_l}(l|i) \implies N(i) = N_{f_l}(i)\ ,\ \forall f_l \in F_l \qquad (2)$$

Where $N(i)$ is the output of the network N for input i. This consideration can be extended to all the layers L^+ consequent to the layer l where a fault is injected. As a result of Eq. 2, since we can store all the possibles golden output feature maps, *we can prematurely end the inference* whenever a fault $f_l \in F_l$ does not modify the output feature map of layer l. We refer to this technique as *Fault Dropping*. The reader should note that, since a fault is dropped only if it has no impact in the affected output feature map, this technique does not introduce any inaccuracy.

A further improvement to Fault Dropping is to consider the input feature map of the layers that follow the one where the fault is injected instead of its output feature maps. This has two main benefits:

1) If Fault Dropping is used in conjunction with Delayed Start, we only need to store the input feature maps, effectively halving the memory required;
2) Take advantage of the fault masking property of the ReLU, when it is applied to the output feature map.

With this improvement, given the set of layers L^+ after layer l, Eq. 2 becomes:

$$N^I(l^+|i) = N^I_{f_l}(l^+|i) \implies N(i) = N_{f_l}(i)$$
$$\forall\ l^+ \in L^+; f_l \in F_l \qquad (3)$$

In Fig. 1b it is shown how, after injecting a fault, the input feature map of the following layer is equal to the corresponding golden input feature map, therefore the faulty output is the same as the golden output.

C. Working with Skip Connections

The main drawback of the proposed Delayed Start and Fault Dropping mechanisms is that they work only if the input of a layer depends uniquely on the output of the previous layer. In fact, many DNN architectures implement a solution that invalidates this hypothesis: the *skip-connection*. A skip connection feeds the output of a layer forward to one of its non-immediate successors. These constructs have substantial differences in how they are implemented in different architectures.

For example, DenseNet-121 [13] and its successors, EfficientNet [14], are structured into consecutive sets of layers called *Dense Blocks*: skip connections are present *inside* a block but not *between* blocks, as shown in Fig. 2. This means that Eq. 1 and Eq. 3 still holds at block level, where there are no skip connections. In order to apply the proposed techniques even in presence of skip connections, SCI-FI works according to this logic: the user can specify which layers or blocks should be used for the application of Fault Dropping and Delayed Start.

D. Fault Models

SCI-FI includes four different fault models, two targeting the network parameters and two the artificial neurons. In particular, it is possible to inject:

1) Bit-Flip faults on the parameters (i.e. the network weights);
2) Stuck-at faults on the parameters;
3) Byzantine neurons [15]: one or more elements of the output feature map are set to a random value in the range of the other element of the feature map;
4) Crashed neurons [15]: one or more elements of the output feature map are set to zero.

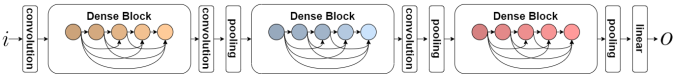


Fig. 2: DenseNet usage of skip connections.

Injecting faults in the DNN weights is straightforward: once the network weights are loaded and the fault has been selected, replace the golden value of the target weight with its faulty value. The faulty value is computed when the fault list is generated to avoid computing it every time a new batch is loaded.

Fault in the output feature maps are injected with the help of a mask. To increase the speed at which injection can be carried out, a fault in the output feature map is seen as an operation between tensors. In particular, let's call O the tensor corresponding to the output feature map of an injectable layer, F is the fault tensor, containing the value of the faulty neuron and M is the mask tensor, containing information about the position of each fault in the output feature map. The faulty version $\hat{O}$ of the output feature map can be computed as the sum of the element-wise product of O and F with their corresponding masks:

$$\hat{O} = O \cdot (1 - M) + F \cdot M \tag{4}$$

IV. Case Study

SCI-FI has been developed as a fault injector tool based on the well-known PyTorch framework. As cases of study, this work examines three different networks families: ResNet, DenseNet and EfficientNet. Details about the networks are shown in Table I. In particular, this work performs fault injections on pre-trained ResNet-20 and ResNet-56 [16] using CIFAR10 as dataset. The test-set accuracy reached by the networks is 91.25% and 93.03% respectively. Further experiments are performed on pre-trained DenseNet-121 [13] and EfficientNet [14] on the ImageNet dataset. These networks reach a validation-set accuracy of 74.43% and 77.69% respectively. To deal with the skip connections present in these networks, the proposed techniques have been applied at block level. For the CIFAR10 based networks, the fault injection campaigns are carried out over the test-set that contains 10,000 images, 1,000 for each of the 10 classes. On the other hand, the campaign targeting the ImageNet based networks, are performed over the validation-set containing 50,000 images, 50 for each of the 1,000 classes.

Exhaustively injecting all the faults for all the images is prohibitively expensive: in the simplest case of bit-flips on the network parameters, assuming a 32-bit representation of the weights, $270,000 \cdot 64 = 17,280,000$ fault injections are required to completely fault simulate ResNet-20, the smallest network. For this reason, in the following section, SCI-FI performances are analysed when performing statistical fault injections as described in [17]. In particular, we are interested in observing the critical rate (i.e., numbers of faults that change the prediction of the network) with an error margin of 1% and a confidence level of 99%. Details on the number of injected faults are reported in Table I.

Network Name	Batch Size	Speed-up [%]	Avg. Memory Occupation [%]
ResNet-20	32	17.58	26.19
ResNet-20	256	26.67	62.86
ResNet-20	1024	34.39	64.84
ResNet-56	32	20.64	137.25
ResNet-56	256	30.71	212.67
ResNet-56	1024	38.91	230.26
DenseNet-121	16	30.89	99.81
DenseNet-121	64	66.97	207.41
DenseNet-121	128	**71.27**	78.29
EfficientNet	16	31.30	204.67
EfficientNet	32	40.66	215.30
EfficientNet	64	44.87	208.46

TABLE II: Performances of SCI-FI when injecting on the parameters compared with the baseline FI.

Network Name	Batch Size	Speed-up [%]	Memory Occupation [%]
ResNet-20	32	18.48	26.09
ResNet-20	256	20.45	41.48
ResNet-20	1024	24.62	39.01
ResNet-56	32	2.44	133.90
ResNet-56	256	16.04	179.75
ResNet-56	1024	23.11	191.95
DenseNet-121	16	9.56	69.02
DenseNet-121	64	36.28	56.15
DenseNet-121	128	**37.38**	69.70
EfficientNet	16	11.95	165.13
EfficientNet	32	25.69	185.39
EfficientNet	64	27.54	168.17

TABLE III: Performances of SCI-FI when injecting on the neurons compared with the baseline FI.

All the FI campaigns have been performed using an Intel(R) Xeon(R) Gold 6238R CPU @2.20GHz paired with a GPU NVIDIA GeForce RTX 3060 Ti with 8 GB of Memory.

V. Experimental Results

In this section, the performances of SCI-FI are discussed in terms of fault injection time and memory overhead. In particular, in Section V-A we discuss the results obtained when injecting faults in the parameters of the network. Then, in Section V-B the performances of SCI-FI are provided when dealing with injecting faults in the neurons of the network.

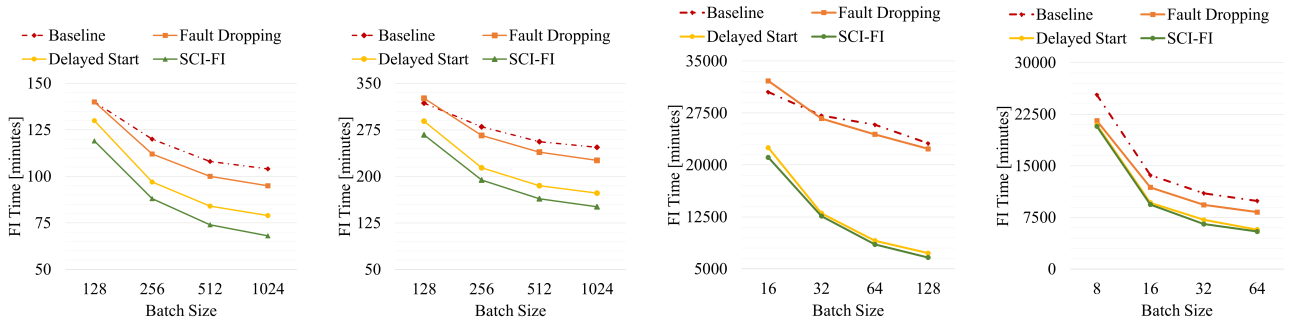
The baseline FI used in all the experiments is based on PytorchFI. In particular, the baseline FI modifies how PyTorch injects faults in the feature maps, taking advantage of the mask mechanism described in Section III-D. All the modifications discussed in this, and in the following sections (besides Delayed Start and Fault Dropping) have been also applied to the baseline FI.

The main drawback of SCI-FI is the additional video memory required for performing a FI campaign, since it is necessary to save the golden outputs of the DNN according the injected fault. To reduce the memory footprint, the following measures have been taken:

- The FI campaign is carried out batch-wise and not fault-wise. All faults are injected in a single batch, then the following batch is loaded, and the same process is repeated. The main advantage of this is that the input feature map of a batch is loaded only once in memory. This also increases the speed-up, as less time is spent moving data from disk to memory.
- Only the input feature map of the injected layer and of the following one are loaded to memory. This slightly decreases the speed-up as some time is spent moving the input feature map from RAM to video memory; however, it drastically reduces the video memory occupation.

A. Injecting on the parameters

First, we analyse the performances of the networks when faults are injected as stuck-at faults in the parameters. Fig. 3 shows the



(a) ResNet-20 (b) ResNet-56 (c) DenseNet-121 (d) EfficientNet

Fig. 3: Statistical parameter fault injection time for selected networks.

time (in minutes) required for a complete fault injection campaign when using Delayed Start and Fault Dropping. Moreover, Table II reports the speed-up and the additional memory occupation of SCI-FI compared to the baseline FI results. In particular, we can see an improvement of up to 71% for DenseNet-121, the largest networks. Moreover, it emerges that larger networks benefit more from the speed-up techniques than smaller networks: when comparing ResNet-20 with its scaled-up version ResNet-56, we can observe an improvement of up to 5% for a given batch-size. Similarly, even if they are different architectures, we can observe how DenseNet-121 has a speed-up that is 37% better than ResNet-20, a network that has 30 times fewer parameters.

Furthermore, while increasing the batch size improves marginally the baseline FI execution time, it has a more pronounced impact on SCI-FI performances. For example, in Fig. 3a we can observe that the performances of the baseline FI with a batch size of 1,024 are still worse than the performances of the SCI-FI with a batch size of 128, only one fourth of its size. Similar results can be observed for all the other networks.

Finally, Table II also reports the additional memory occupation. This value is the average, across all the injections, of the maximum memory occupied, intend as active memory plus reserved memory (i.e., cached). It should be noted that, even in the more extreme cases, where the memory occupation is significant, SCI-FI speed-up is bigger than the one obtained by increasing the batch size to match the same memory occupation. In the case of EfficientNet, while a batch of 32 requires roughly 108 hours with SCI-FI, a batch of 128 requires 152 hours with the baseline FI. The memory occupation increase caused by a larger batch size is 300%, while the memory occupation increase of the speed-up is only 215.30%.

B. Injecting on the neurons

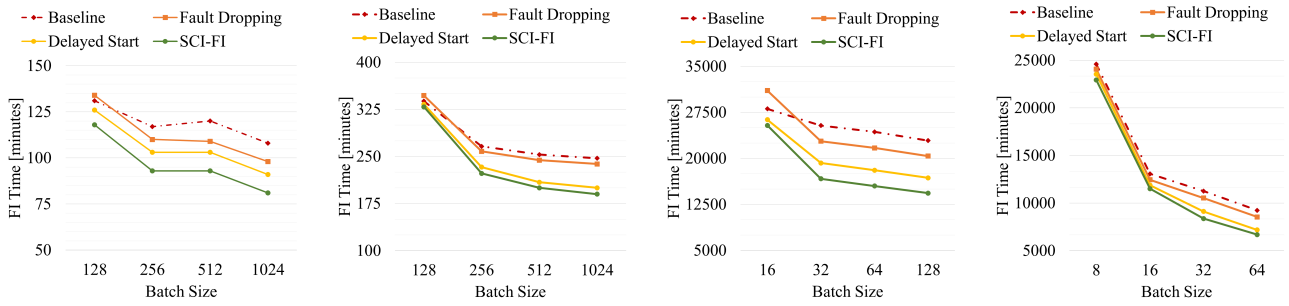
In this second section, we perform an analysis of the SCI-FI performances when injecting a single byzantine neuron fault. To implement the masking described in Section III-D using the PyTorch framework, there are mainly two possible choices: (i) the fault is injected using `hooks` or (ii) the injectable layers is replaced by a copy of the one that contains also the mask application. The usage of hooks, although widely used by other injectors [5], is often very slow, as it requires an additional function call and is not easily optimizable on the GPU. In fact, these construct were mainly built to debug the behaviour of complexes networks and should be avoided if possible. The alternative is to replace the injectable layer with an identical copy that contains a modified forward function. The scope of this forward is to compute the output of the golden layer and then apply, on top of that, the faulty mask as described in Eq. 4. In these experiments, only one fault is injected per layer.

Table III shows the performances of SCI-FI against the baseline. It must be noted that, in this case, the baseline also implements the mask strategy: the performances of this basic injector are possibly even better than what offered widely available in tools such as *PyTorchFI*. Fig. 4 instead compares the performances of the baseline against fault injectors employing the speed-up techniques proposed in Section III. The speed-up offered by SCI-FI is considerable even with a different fault model, reaching a peak of 37% in DenseNet-121. In particular, we observe how the speed-up increases with a larger batch size. When compared with the parameter fault injection, we observe how the faulty inferences require roughly the same amount of time.

Table IV compares the average time required by SCI-FI to perform parameter and neuron FIs with the average time required to perform a golden inference (i.e., without any fault injection) in PyTorch. We use this as a baseline for the proposed method. In particular, we observe that SCI-FI requires considerably less time: a possible explanation is that faults are injected batch-wise. Typically, a fault is injected in a batch, an inference is performed, a new batch is loaded, and the same fault is injected. This fault-wise injection, however, requires loading multiple times the same batch for different faults. With a batch-wise injection, instead, a fault is

Network Name	Batch Size	Golden Time [ms]	Param FI Time [ms]	Neuron FI Time [ms]
ResNet-20	32	20	5	5
ResNet-20	256	105	8	9
ResNet-20	1024	398	26	31
ResNet-56	32	36	14	13
ResNet-56	256	118	18	21
ResNet-56	1024	415	57	71
DenseNet-121	16	98	25	30
DenseNet-121	64	364	41	74
DenseNet-121	128	690	64	138
EfficientNet	16	69	11	14
EfficientNet	32	116	16	20
EfficientNet	64	228	26	32

TABLE IV: Time required to execute a single batch for a golden PyTorch inference and faulty SCI-FI inferences.



(a) ResNet-20 (b) ResNet-56 (c) DenseNet-121 (d) EfficientNet

Fig. 4: Statistical neuron fault injection time for selected networks

injected in a batch, an inference is carried out, then a new fault is injected using the same input batch. This allows SCI-FI to load a batch only once, taking advantage of cached data. Therefore, on average, SCI-FI faulty inferences require as much as 10 times less time than PyTorch golden inferences. As a final note, we want to highlight that SCI-FI neuron injection is achieved with a tensor multiplication (Eq. 4): increasing the number of faults injected in a layer does not impact the cost of the FI process.

For what concerns the memory footprint, we can observe how the memory occupation is, for a given network, independent of the batch size. Furthermore, values differ from the ones observed with the parameter fault injection, despite the fact that a different fault model does not change the proposed methodology. The variation can be explained by the fact that the faults affect different layers, with different number of faults, than the one affected by parameter injections. Finally, it must be noted that the memory cost of the mask is not reported in the table, as all the FIs take advantage of this technique. However, the presence of a mask is not very impactful in terms of memory footprint.

VI. Conclusions

Given that DNNs are rapidly growing in size, a faster tool to analyze their reliability to faults is needed. In this work, we introduced SCI-FI: an extremely adaptable and fast software-level fault injector, capable of injecting faults both in the parameters and in the neurons of a DNN. By using the Fault Dropping and Delayed Start techniques, SCI-FI offers a trade-off between video memory footprint and execution time, offering speed-up that can improve performances up to 70% in large networks with large batch sizes.

References

[1] T. M. Niermann, W.-T. Cheng, and J. H. Patel, "Proofs: a fast, memory-efficient sequential circuit fault simulator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, p. 198 – 207, 1992.

[2] L. Berrojo, I. Gonzalez, F. Corno, M. Reorda, G. Squillero, L. Entrena, and C. Lopez, "New techniques for speeding-up fault-injection campaigns," in *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, 2002, pp. 847–852.

[3] A. Ruospo, E. Sanchez, L. Luza, L. Dilillo, M. Traiola, and A. Bosio, "A survey on deep learning resilience assessment methodologies," *Computer*, vol. 56, no. 02, pp. 57–66, feb 2023.

[4] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. USA: ACM, USENIX Association, 2016, p. 265–283. [Online]. Available: https://dl.acm.org/doi/10.5555/3026877.3026899

[5] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach *et al.*, Eds. Vancouver, Canada: Curran Associates, Inc., 2019, pp. 8024–8035.

[6] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "Tensorfi: A flexible fault injection framework for tensorflow applications," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. Coimbra, Portugal: IEEE, Oct. 2020, pp. 426–435. [Online]. Available: https://doi.org/10.1109/ISSRE5003.2020.00047

[7] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "PyTorchFI: A runtime perturbation tool for DNNs," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. Valencia, Spain: IEEE, 2020, pp. 25–31. [Online]. Available: https://doi.org/10.1109/DSN-W50199.2020.00014

[8] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference*. San Francisco, California, USA: Association for Computing Machinery, 2018, pp. 1–6. [Online]. Available: https://doi.org/10.1145/3195970.3195997

[9] A. Lotfi *et al.*, "Resiliency of automotive object detection networks on gpu architectures," in *2019 IEEE International Test Conference (ITC)*, 2019, pp. 1–9.

[10] G. Li *et al.*, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126964

[11] A. Ruospo, E. Sanchez, M. Traiola, I. O'Connor, and A. Bosio, "Investigating data representation for efficient and reliable convolutional neural networks," *Microprocessors and Microsystems*, vol. 86, p. 104318, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933121004786

[12] F. Angione *et al.*, "Test, reliability and functional safety trends for automotive system-on-chip," in *2022 IEEE European Test Symposium (ETS)*, 2022, pp. 1–10.

[13] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[14] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[15] E. M. El Mhamdi and R. Guerraoui, "When neurons fail," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017, pp. 1028–1037.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[17] A. Ruospo *et al.*, "Assessing convolutional neural networks reliability through statistical fault injections," in *2023 Design, Automation Test in Europe Conference*, 2023, In press.