2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) | 978-1-6654-7679-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISSREW55968.2022.00074

TensorFI+: A Scalable Fault Injection Framework for Modern Deep Learning Neural Networks

Sabuj Laskar
University of Iowa
IA, USA
sabuj-laskar@uiowa.edu

Md Hasanur Rahman
University of Iowa
IA, USA
mdhasanur-rahman@uiowa.edu

Guanpeng Li
University of Iowa
IA, USA
guanpeng-li@uiowa.edu

***Abstract*—Deep Neural Networks (DNNs) are widely deployed in various applications such as autonomous vehicles, healthcare, space applications. TensorFlow is the most popular framework for developing DNN models. After the release of TensorFlow 2, a software-level fault injector named TensorFI is developed for TensorFlow 2 models, which is limited to inject faults only in sequential models. However, most popular DNN models today are non-sequential. In this paper, we are the first to propose TensorFI+, an extension to TensorFI to support for non-sequential models so that developers can assess resiliency of any DNN model developed with TensorFlow 2. For the evaluation, we conduct a large-scale fault injection experiment on 30 sequential and non-sequential models with three popularly used classification datasets. We observe that our tool can inject faults in any layer for any sequential or non-sequential DNN model, and fault-injected inference incurs only 7.62× overhead compared to fault-free inference.**

***Index Terms*—Deep Neural Networks (DNNs), Transient Hardware Faults, Non-sequential DNN Models, Reliability Assessment**

I. Introduction

Over the last decade, Deep Learning Neural Networks (DNNs) have been widely deployed in many areas such as computer vision, natural language processing, and autonomous vehicles (AVs) [1], [2]. With the ever-increasing demands of DNNs, parallel processing units such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) have been developed to accelerate the DNN inference [3], [4]. Many machine learning (ML)-based applications, such as AVs, health care, and space applications, require high reliability, low latency, and high throughput during the inference phase [2], [5], [6]. Therefore, the ability of DNNs to deliver accurate performance critically depends on their resiliency to faults [7], [8].

One of major dependability threats today in computer systems is transient hardware faults, which are commonly caused by high energy particles striking electronic devices and resulted in bit flip errors in logic values [9]. Such faults may cause DNN failures, leading to misclassifications in DNN inferences [10]. The straightforward way to assess the reliability of a system is fault injection (FI). FI can be conducted at the hardware or software level. Software-Implemented FI (also known as SWiFI) has lower costs, is more controllable, and easier for developers to deploy [11]. Therefore, SWiFI has become the dominant method to assess a system's resilience on the presence of both hardware and software faults.

Due to the increase in popularity of ML-based applications, there are many frameworks developed on top of them to abstract the underlying complex structure and make it easy-to-use for the developers. Such popular frameworks are TensorFlow and Keras. To operate, Keras needs a backend, and TensorFlow has become the default backend of Keras. Thus, most of the developers and researchers use keras with tensorflow to design their DNNs [12].

To assess the reliability, an open-source SWiFI framework called TensorFI2 [13], built on top of TensorFI [14], was developed which is a configurable FI framework to inject faults in models developed with Tensorflow 2 which inherently supports Keras. But this framework is only limited to conduct FIs on sequential DNN models, such as VGGs [15], meaning any layer's input depends only on the immediate preceding layer's output and vice versa. However, most popular deep learning models such as DenseNet [16], ResNet [17] are non-sequential, meaning any intermediate layer's input can depend on multiple preceding layers' outputs and vice versa. Hence, there is an increasing need to develop a fault injector to support non-sequential models.

In this paper, *we are the first to propose* TensorFI+*, a framework to fill-up this gap by extending TensorFI2 [13] to support FIs in non-sequential DNN models to assess the reliability*. However, we face several challenges to support non-sequential models: 1) Because of non-sequentiality, each DNN layer's input may depend on multiple preceding layers' outputs and vice versa. Moreover, structures of Tensorflow operators are not allowed to modify because of negative side effects. Hence, leveraging only high-level details such as API calls to keep track of error propagation into multiple directions is a very difficult task. 2) There are possibly multiple overlapping paths, hence possible repetition in the computation for some layers during fault propagation. Consequently, the performance overhead would be high. With the above challenges in mind, we make the following contributions in this paper.

- We develop a scalable FI framework, TensorFI+, to support FI into non-sequential DNN models developed with Tensorflow 2.
- We propose an optimization strategy to reduce fault-injected inference overhead by efficiently keeping track

978-1-6654-7679-9/22/$31.00 ©2022 IEEE
DOI 10.1109/ISSREW55968.2022.00074

error propagation in non-sequential DNN models in which an error can be propagated in multiple directions.
- We evaluate TENSORFI+ on 30 DNN models in TensorFlow 2 with datasets used to assess reliability of safety-critical context. We also evaluate the performance overhead of our framework compared to fault-free inference.

II. BACKGROUND & USE CASE DISCUSSION

A. Deep Learning Neural Networks

A deep neural network (DNN) consists of multiple computation layers [18] where higher-level abstraction of the input data or a feature map is extracted to preserve each layer's important information. In this work, we consider convolutional neural networks of DNNs, as they are broadly used in DNN applications. The number of convolutional layers in such DNNs can range from a few to thousands of layers. Each convolutional layer applies a kernel (or filter) on the previous layer's feature maps to extract underlying hidden characteristics and generate the corresponding output feature maps. Activation function (ACT) such as ReLU is generally applied on some layers' output results. In some DNNs, a small number of fully-connected (FC), pooling (POOL) layers are typically stacked on the convolutional layers for classification purposes. Once the topology is constructed, DNN is fed with input image data for training. After that, DNN will go through a series of backpropagation to tune the weights associated with feature maps. After training, DNN is ready for image classification with testing image data, which we call as the *inference* phase. The input of inference phase is a digitized image, and the output is a predicted object such as car. We call *misclassification* as a mismatch between predicted output and true label.

B. Fault Model

In this work, we consider transient hardware faults that randomly occur during the execution of the DNN inference. Note that training phase can also be affected by these faults, however, training is usually a one-time process, so we can always verify the results of the trained model. In contrast, any system expects DNN inference to be fast and accurate, as inference is frequently applied in real time, possibly thousands of time, so we can not verify the outcome after each inference. Hence, DNN inference is essential to be resilient against such faults [10], [19].

During each inference (program execution), we randomly inject one fault directly at the output value of a randomly selected layer of DNN models. This fault injection method is aligned with prior studies [19], [20]. We refer to one fault as single bit-flip in the output value of a randomly selected layer. As multiple faults during one program execution are relatively rare event, single bit-flip fault is found as accurate as multiple bit-flips fault in recent studies [20], [21]. A fault can be activated and may cause the DNN inference produce any of four outcomes: SDC, Masked, Crash or Hang. We assume that faults do not modify the state/structure of the model (e.g., change the model's parameters or operators), nor do we consider faults in the model's inputs, as they are extraneous to TensorFI [14] and are outside of the scope of our paper.

C. Terms and Definitions

- *Top-1 accuracy:* Percentage of top-1 correctly classified labels over total number of fault-free DNN inferences.
- *Masked fault:* Even though a fault occurs during inference, DNN output turns out to be same as the fault-free output of the same inference. Here same inference implies inference with same input image.
- *Silent Data Corruption (SDC):* A mismatch between output of a fault injected DNN inference and fault-free output of the same inference.
- *SDC rate:* Percentage of SDC cases over total number of fault injected DNN inference trials.

D. A Real-World Use Case of DNN Failures by Soft Errors

The consequences of soft errors in DNN systems can be catastrophic, especially in the safety-critical context, and often error mitigation is required to meet specific reliability targets. For example, due to soft errors, if an AV misclassifies a transporting truck as completely different object such as a small bird, the AV might take different action policy other than braking. Thus, the AV may not be able to avoid a possible collision. Moreover, most of the DNN models nowadays deployed in AVs are non-sequential. Hence, there is a growing need to systematically understand error propagation in non-sequential models.

III. TENSORFI+ DESIGN

In this section, we first introduce the workflow of TENSORFI [13], [14] that we extend. Then we discuss its limitation in supporting non-sequential DNN models. Finally, we propose our strategy to design TENSORFI+ for injecting faults in non-sequential models.

A. TENSORFI *Workflow and Limitation*

TENSORFI [13], [14] is a FI framework for TensorFlow applications and has been used to simulate transient hardware faults during DNN inference in recent related studies [19], [22]. It supports FI experiments on DNN models implemented using the TensorFlow framework [23] which is the most popular used ML framework nowadays [24]. TENSORFI operates on TensorFlow dataflow graphs which contains two main components: (1) operators of computational units (e.g., matrix multiplication), and (2) tensors of data units. One can build ML models using the built-in operators or define their customized operators. TENSORFI duplicates the TensorFlow graph with customized operators, which are designed to be able to not only perform computations as standard operators do, but also inject faults at runtime during model inference.

Nowadays, it is common to use TensorFlow 2 and Keras to deploy DNN models because of the abstraction and easy use of the framework and availability of numerous available pre-trained models in Keras. Recently, TENSORFI has been upgraded [13] to work with TensorFlow 2 and Keras. As

structures of operators are not allowed to change, it leverages the Keras APIs to inject faults into the output value of layers. Unfortunately, it only supports fault injections in sequential DNN models. So, the gap to support widely used non-sequential models such as ResNets [17], DenseNets [16] is still open.

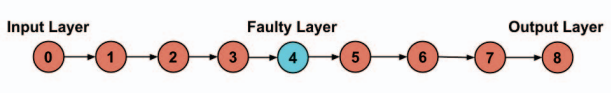

Fig. 1: Fault injection in a sequential model

Figure 1 shows how TENSORFI injects faults into a sequential model. In a sequential model, a layer's input is only sequentially dependent on its previous layer's output. For example, layer 4 is selected for fault injection, then TENSORFI computes the output of layer 4 using output of layer 3 as its input and injects a fault into the output of layer 4. As the error propagates, TENSORFI sequentially keeps track of the propagation by leveraging API calls to compute the outputs from layer 5 to 8 using the faulty input of layer 5. However, if a DNN is non-sequential, in which case a layer's input is dependent on multiple preceding layers' outputs, TENSORFI has no way to trace error propagation in its current implementation.

B. Support for Non-sequential DNN Models

To overcome the above limitation, we propose TENSORFI+ to work with both sequential and non-sequential models. The code is made publicly available[1]. As we see, sequential models have only one input and one output for every intermediate layer. However, a non-sequential model might have multiple inputs and outputs for an intermediate layer. Moreover, changing the structures of Tensorflow operators in Keras has negative effect, so we cannot directly profile them for FIs. Instead, we leverage high-level abstractions such as API calls to build a dependency graph of layers to conduct FIs. This FI method is aligned with TENSORFI [14]. Hence, when we inject a fault to the output of a specific layer of a non-sequential model, we need to propagate that erroneous output to the inputs and outputs of other layers across multiple directions leading up to the final output based on the developed dependency graph in order to provide support for non-sequential models in TENSORFI. We develop such graph by leveraging an internal data structure in Keras which keeps layer mappings in a list. Note that in Keras, it is impossible to find any direct indication of dependency from the sequential pattern of layers. Since each layer and output tensor in Keras has a unique identifier in the data structure, we leverage this information to build the dependency graph that can relate the input and output dependency of each layer of the entire DNN model.

In our dependency graph, each node consists of the following attributes.

1) **Identifier**: A unique identifier of the layer output

[1]https://github.com/sabuj7177/TensorFIPlus

Algorithm 1 FI Method by TENSORFI+

Input: Trained Keras model layers list (L) and an input image (I).
Output: Predicted output by faulty inference.

1: Build dependency graph G from L and get superlayers list S.
2: Call Keras API for output of a randomly selected layer l from G and do fault injection in the output.
3: Get the immediate preceding superlayer ps and next superlayer ns of l from G.
4: $MDict = \emptyset$, and add faulty output tuple (l, output[l]) into $MDict$.
5: Call Keras API to computes outputs of layers from input-layer to ps and store them into $MDict$.
6: Recursively call Keras API to get the output of layer r that are linked to input layers of ns.
7: Apply *Optimization Strategy* (Section III-C) using $MDict$.
8: Save computed output tuple (r, output[r]) of r to $MDict$.
9: Compute output layer value from ns sequentially by calling Keras API.

2) **Input_layers**: List of layer indices on which the layer depends
3) **Output_layers**: List of layer indices which depend on the layer

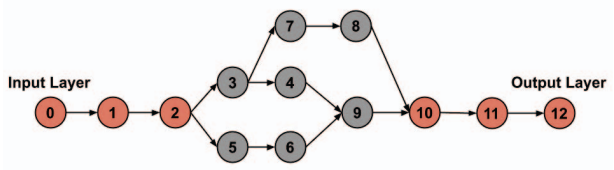

Fig. 2: A Dependency graph with superlayers (marked as red)

Algorithm 1 provides the details of a random FI method by TENSORFI+. We refer to Figure 2 to show an example of a dependency graph (line 1) for non-sequential model. In the example, there are a total of 13 layers. Layer 0 is the input layer, and layer 12 is the output layer. Each node represents a layer. Each arrow represents a layer's dependency. As the model is non-sequential, any layer might have multiple inputs and outputs. There can be multiple branches in the dependency graph such as layers 3 to 10. Recall that we are using Keras API calls for fault propagation. If we inject fault to any layer which are in the branch, we can not directly get output of the last layer from that injected layer because any subsequent layer might need inputs which are not computed yet. For example, if we inject fault in layer 4, we can not sequentially keep track of outputs from layer 4 to layer 12 because layer 9 and 10 also require values from layer 6 and 8 respectively which are not computed yet. The solution is to identify layers which are not part of any branch, so we can directly compute output of any subsequent layer from those layers. We call them *superlayer*s (marked as red in the figure) in the dependency graph. Specifically, any subsequent layer after a superlayer is not dependent on the layers prior to that superlayer. Hence, the layers between 3 and 9 are not superlayers. We call them *non-superlayer*s (marked as grey in the figure). However, layers 0, 1, 2, 10, 11, 12 are superlayers, and we can calculate output value of any subsequent layer from these nodes. If a fault is injected in the output of a *non-superlayer*, we first find

its previous and next immediate superlayer (line 3). We then recursively (line 6) compute the inputs (by the API calls) of that next superlayer by tracing the corrupted value and compute its output. As the dependency graph and superlayers are model-specific, profiling of the model is required before FIs.

We use Figure 2 as a running example to show how a fault can be injected in a non-sequential model. Note that the FI procedure by TensorFI2 is to trace input dependency from immediate neighbours and propagate the value in subsequent layers till the output layer. For example, if we inject a fault at layer 6, then TensorFI2 will: (1) compute layer output from layer 0 to layer 6, (2) inject fault at the output of layer 6, and (3) use the output as the input of layer 9 and propagate the computation from layer 9 to layer 12. However, this approach fails because layer 9 requires output from both layer 4 and layer 6. Although we have the output of layer 6, we also need to compute the fault-free output of layer 4. Similar reason also holds for layer 10. Hence, we propose to first compute the next immediate superlayer of the fault injected layer. In the example, the immediate next superlayer of layer 6 (selected for FI) is layer 10. We then recursively compute the inputs of the superlayer. In other words, we compute outputs of both layer 8 and layer 9. Then, we need to follow the recursive approach because the inputs of the superlayer (e.g., layer 9) might be from multiple layers, and we also need to compute the outputs of all those layers. If we need to compute the inputs of any layer which are not affected by faults (e.g., layer 8), we compute them by using the original inputs of the model.

C. Performance Optimization Strategy

To improve the computing overhead of TENSORFI+during FIs in non-sequential models, we consider two optimization strategies. First, we memorize our computation results. That is, if a layer output is computed earlier, we use the pre-computed value instead of recomputing the layer to make the process more efficient. In Figure 2, layer 8 and layer 9 both depend on the output of layer 3. To save time, we can compute the output of layer 3 once(e.g., to compute output of layer 8) and then use this output if required(e.g., to compute output of layer 9 later).

Second, we consider the longest possible sequence of layers when we recursively compute the the inputs of superlayers. For example, if we inject faults in layer 2, its immediate next superlayer is layer 10, which requires outputs of layer 8 and 9. To get outputs of layer 8 and 9, we need all of the following computations.

- Compute output of layer 3 and 5 using layer 2
- Compute output of layer 4 and 7 using layer 3
- Compute output of layer 6 using layer 5
- Compute output of layer 9 using layer 4 and 6
- Compute output of layer 8 using layer 7

Here, we need to compute for a total of seven layers' outputs. However, if we detect the longest sequential sequences of layers that can be computed only one time, the computation overhead reduces significantly. That is, if we detect the longest sequences such as layer 2-3-7-8, 2-3-4, 2-5-6 and (4,6)-9, we only need to compute four layers' outputs (e.g., layer 8, 4, 6, 9), making our framework efficient for FIs in non-sequential models. As real-world DNN models are non-sequential, more complex, and present opportunities to prune above cases more, these strategies provide significant performance improvement compared to naive recursive approach.

IV. EXPERIMENTAL SETUP

In this section, we evaluate our TENSORFI+ on different DNN models. All of our experiments and evaluations are conducted on a Intel 28-core machine with 32GB memory running Debian Linux.

A. Datasets and DNN Models

We use ImageNet [25], CIFAR-100 [26] and German Traffic Sign Recognition Benchmark(GTSRB) [27] datasets for our experiment. The former two datasets are popularly used for standard classification tasks and the later one contains real-world traffic sign datasets and specifically used in object detection for AVs. We use GTSRB dataset to show that TENSORFI+ can also work with safety-critical datasets and models. We have 13 DNN models for CIFAR-100, 12 DNN models for ImageNet and 5 DNN models for GTSRB. Among them, 8 are sequential and 22 are non-sequential models. For ImageNet, we adopt pre-trained models such as VGG, DenseNets, ResNets, MobileNets, Inception, which are available on Keras[2] . For CIFAR-100 and GTSRB, since the pre-trained models are not directly available, we train the DNN models using pytorch. Then we convert the pre-trained models from pytorch to TensorFlow 2 using pytorch2keras[3] module.

B. Fault-free and Fault-injected Inference Methods

Fault model is described in Section II-B. In our FI experiments, we randomly sample 10,000 images from each ImageNet and GTSRB dataset test sets. As CIFAR-100 has 10000 images in the test set, we use the full test set.

First, we run inference on the sampled 10000 datasets to get the accuracy and fault-free predicted labels for each model. Then, for each model, we inject 3000 random faults to get its SDC rate. 3000 random FIs are found adequate in studying error resilience in previous studies [28], [29].

C. Research Questions

We try to find answers for two research questions (RQ):

- RQ1: What are the SDC rates of both sequential and non-sequential models with different types of datasets?
- RQ2: What is the performance overhead by TENSORFI+?

V. RESULTS

We organize our evaluation results by research questions.

[2]https://keras.io/api/applications/
[3]https://github.com/gmalivenko/pytorch2keras

Dataset	Model	Top-1 accuracy	SDC rate
ImageNet	VGG16 (S)	71.18%	3.53%
	VGG19 (S)	71.35%	3.60%
	ResNet50 (NS)	74.76%	1.43%
	ResNet101 (NS)	76.14%	1.57%
	ResNet152 (NS)	76.38%	2.07%
	MobileNet (NS)	70.25%	0.93%
	MobileNetV2 (NS)	71.07%	0.53%
	DenseNet121 (NS)	75.04%	1.20%
	DenseNet169 (NS)	75.76%	1.10%
	Xception (NS)	78.92%	1.70%
	InceptionV3 (NS)	77.77%	1.73%
	InceptionResNetV2(NS)	80.11%	1.20%
CIFAR-100	VGG11 (S)	69.12%	1.40%
	VGG13 (S)	71.51%	1.73%
	VGG16 (S)	72.33%	1.03%
	VGG19 (S)	71.53%	1.23%
	ResNet18 (NS)	76.35%	1.37%
	ResNet34 (NS)	77.68%	0.90%
	ResNet50 (NS)	78.52%	1.27%
	ResNet101 (NS)	78.93%	1.03%
	ResNet152 (NS)	79.68%	1.40%
	GoogleNet (NS)	76.70%	1.57%
	InceptionV3 (NS)	79.45%	1.50%
	InceptionV4 (NS)	77.80%	1.50%
	Xception (NS)	77.96%	2.00%
GTSRB	VGG16 (S)	97.57%	0.80%
	VGG19 (S)	98.25%	0.90%
	ResNet34 (NS)	98.98%	1.13%
	ResNet50 (NS)	97.91%	0.97%
	ResNet101 (NS)	98.55%	0.67%

TABLE I: Top-1 accuracies and SDC rates for both sequential and non-sequential models across all the datasets. Here *S* and *NS* refer to Sequential and Non-sequential respectively.

A. *RQ1: What are the SDC rates of both sequential and non-sequential models with different types of datasets?*

TensorFI+ can inject faults in both sequential and non-sequential models. Hence, in this RQ, we aim to measure the SDC rates of both sequential and non-sequential models by TensorFI+. In Table I, we show the Top-1 accuracies and SDC rates of different sequential and non-sequential models with three different datasets.

Different VGG models are the common example of sequential models. We can see that TensorFI+ can inject faults in VGG models with ImageNet, CIFAR-100 and GTSRB datasets, and demonstrates an average 3.57% and 1.35% and 0.85% SDC rate. respectively. To test the FI capability of TensorFI+ with the non-sequential model, we experiment with models such as ResNet, MobileNet, DenseNet, Xception, Inception, InceptionResNetV2, GoogleNet, and their variants. These are the most used non-sequential models nowadays. We can see that our framework can successfully inject faults in all of these models, and accordingly demonstrate SDC rates from 0.53% to 2.07% across those models. Moreover, we show that TensorFI+can work with safety-critical datasets, such as GTSRB, by adding support for non-sequential models. Therefore, TensorFI+ can inject fault in any DNN model (irrespective of sequentiality) developed with TensorFlow 2.

Dataset	Model	Fault-Free Inference Time(s)	Fault-Injected Inference Time(s)	Overhead
ImageNet	VGG16 (S)	0.026	0.047	1.81x
	VGG19 (S)	0.030	0.043	1.43x
	ResNet50 (NS)	0.027	0.145	5.37x
	ResNet152 (NS)	0.065	1.098	16.89x
	MobileNet (NS)	0.009	0.099	11.00x
	DenseNet121(NS)	0.027	0.460	17.04x
	Xception (NS)	0.019	0.159	8.37x
CIFAR-100	VGG16 (S)	0.009	0.039	4.33x
	VGG19 (S)	0.009	0.025	2.78x
	ResNet50 (NS)	0.011	0.101	9.18x
	GoogleNet (NS)	0.012	0.223	18.58x
	InceptionV3 (NS)	0.022	0.081	3.68x
	Xception (NS)	0.014	0.144	10.29x
GTSRB	VGG16 (S)	0.029	0.071	2.45x
	VGG19 (S)	0.025	0.065	2.60x
	ResNet50 (NS)	0.016	0.153	9.56x
	ResNet101 (NS)	0.030	0.123	4.10x

TABLE II: FI Overheads of DNN models compared to the fault-free inference

B. *RQ2: Performance overhead by* TensorFI+*?*

In this RQ, we analyze how much computation overhead during inference incurs by TensorFI+ while conducting FIs. In Table II, we show the time required for fault-free inference and fault-injected inference, and provide the overhead (last column) comparison with respect to fault-free inference. To compute the fault-free inference time of a model, we average the times of ten subsequent inferences with the model. We follow a similar approach to measure the average time for fault-injected inference. Note that the FI is random for each of ten subsequent inferences. We can see that, on average, fault-injected inference time is only 7.62× higher than fault-free inference time across all the models. Recall that during a random FI per inference (program execution), we need to randomly choose a layer, compute the output of that layer using layer inputs, inject a fault to the output, propagate the faulty output to subsequent layers leading up to the final layer, and finally compute the model's prediction. We can also see that sequential models such as VGGs have comparatively much lower overhead than non-sequential ones. Recall that for non-sequential models, any layer after the fault-injected layer can have input dependency on layers before the fault injected layer. So, we need to do some extra work, which would significantly increase the computation overhead compared to sequential models. However, thanks to the improvement mentioned in Section III-C, the overhead is still reasonable.

VI. Related Work

With the increasing number of studies related to DNN resilience and accuracy, different fault injectors have been proposed to inject faults in models developed in frameworks such as TensorFlow, PyTorch. To inject faults in TensorFlow-based applications, Li et al. [14] presented a configurable and high-level fault injector tool named TensorFI. Although this tool

can inject faults at both hardware and software levels, it can only work with TensorFlow 1. Mahmoud et al. [30] proposed PyTorchFI, a runtime perturbation tool for injecting faults in DNN models developed in PyTorch. Although TensorFlow 2 has become very popular for DNN model design due to its support of Keras and abstractness in model building, none of the aforementioned fault injection tools can work with this framework. Recently, a fault injector tool named TensorFI2 [13] has been developed, which only supports FIs in linear models with TensorFlow 2. However, our work is a complete and configurable FI framework that can work with all types of linear and non-linear DNN models designed and developed with TensorFlow 2.

VII. Conclusion

In this work, we propose TensorFI+ to support non-sequential DNN models developed with TensorFlow 2. As most developers and researchers today use TensorFlow 2 for designing and developing their DNN models, our framework can help them systematically assess resiliency of DNN models developed with TensorFlow 2. We evaluate TensorFI+ with 30 popularly used DNN models with three datasets. Our evaluation shows that TensorFI+ can work with all types of DNN models (sequential and non-sequential) developed with TensorFlow 2, and its FI overhead during inference is, on average, 2.57× and 10.37× for sequential and non-sequential models respectively compared to fault-free inference.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 06 2016, pp. 770–778.

[2] M. Bojarski, D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 04 2016.

[3] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," ser. ASPLOS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 269–284. [Online]. Available: https://doi.org/10.1145/2541940.2541967

[4] F. Fernandes, L. Weigel, C. Jung, P. Navaux, L. Carro, and P. Rech, "Evaluation of histogram of oriented gradients soft errors criticality for automotive applications," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 4, nov 2016. [Online]. Available: https://doi.org/10.1145/2998573

[5] A. Esteva, B. Kuprel, R. Novoa, J. Ko, S. Swetter, H. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, 01 2017.

[6] P. Rajpurkar, A. Hannun, M. Haghpanahi, C. Bourn, and A. Ng, "Cardiologist-level arrhythmia detection with convolutional neural networks," 07 2017.

[7] N. DeBardeleben, J. Laros, J. T. Daly, S. L. Scott, C. Engelmann, and B. Harrod, "High-end computing resilience: Analysis of issues facing the hec community and path-forward for research and development," *Whitepaper, Dec*, 2009.

[8] B. Schroeder and G. Gibson, "Understanding failures in petascale computers," *Journal of Physics: Conference Series*, vol. 78, 09 2007.

[9] C. Constantinescu, "Intermittent faults and effects on reliability of integrated circuits," *Reliability and Maintainability Symposium*, vol. 0, pp. 370–374, 01 2008.

[10] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "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] M.-C. Hsueh, T. Tsai, and R. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, pp. 75 – 82, 05 1997.

[12] "Why choose keras?" https://keras.io/why_keras/.

[13] "Tensorfi with support of tensorflow 2 and keras," https://github.com/DependableSystemsLab/TensorFI2.

[14] G. Li, K. Pattabiraman, and N. DeBardeleben, "Tensorfi: A configurable fault injector for tensorflow applications," in *2018 IEEE International symposium on software reliability engineering workshops (ISSREW)*. IEEE, 2018, pp. 313–320.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556*, 09 2014.

[16] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2016. [Online]. Available: https://arxiv.org/abs/1608.06993

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 06 2016, pp. 770–778.

[18] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010, pp. 253–256.

[19] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben, "¡i¿binfi¡/i¿: An efficient fault injector for safety-critical machine learning systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356177

[20] C.-K. Chang, S. Lym, N. Kelly, M. B. Sullivan, and M. Erez, "Evaluating and accelerating high-fidelity error injection for hpc," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018, pp. 577–589.

[21] B. Sangchoolie, K. Pattabiraman, and J. Karlsson, "One bit is (not) enough: An empirical study of the impact of single and multiple bit-flip errors," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017, pp. 97–108.

[22] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "Tensorfi: A flexible fault injection framework for tensorflow applications," 2020. [Online]. Available: https://arxiv.org/abs/2004.01743

[23] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "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: USENIX Association, 2016, p. 265–283.

[24] "Tensorflow popularity," https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[26] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[27] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark," in *International Joint Conference on Neural Networks*, no. 1288, 2013.

[28] S. Feng, S. Gupta, A. Ansari, and S. Mahlke, "Shoestring: probabilistic soft error reliability on the cheap," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 1, pp. 385–396, 2010.

[29] Q. Lu, K. Pattabiraman, M. S. Gupta, and J. A. Rivers, "Sdctune: a model for predicting the sdc proneness of an application for configurable protection," in *Proceedings of the 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2014, pp. 1–10.

[30] 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)*, 2020, pp. 25–31.