Characterizing Deep Learning Neural Network Failures between Algorithmic Inaccuracy and Transient Hardware Faults

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

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

Bohan Zhang
University of Iowa
IA, USA
bohan-zhang@uiowa.edu

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

2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC) | 978-1-6654-8555-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/PRDC55274.2022.00020

***Abstract*—Deep Neural Networks (DNNs) have been widely deployed in safety-critical applications such as autonomous vehicles, healthcare, and space applications. Though DNN models have long suffered intrinsic algorithmic inaccuracies, the increasing number of hardware transient faults in computer systems has been raising safety and reliability concerns in safety-critical applications. This paper investigates the impact of DNN misclassifications that caused by hardware transient faults and intrinsic algorithmic inaccuracy in safety-critical applications. We first extend a state-of-the-art fault injector for TensorFlow application, TENSORFI, to support fault injections on modern DNN models in a scalable way, then characterize the outcome classes of the models, analyzing them based on safety related metrics. Finally, we conduct a large-scale fault injection experiment to measure the failures according to the metrics and study their impact on safety. We observe that failures caused by hardware transient faults could have much more significant impact (up to 4 times higher probability) on safety-critical applications than that of the DNN algorithmic inaccuracies, advocating the potential needs to protect DNNs from hardware faults in safety-critical applications.**

***Index Terms*—Deep Neural Networks, Transient Hardware Faults**

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]–[4]. With the ever-increasing usages and demands of DNNs, parallel processing units such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) are being developed to accelerate the DNN inference processes [5]–[7]. Many machine-learning-based applications, such as AVs, health care, and space applications require high reliability, low latency, and high throughput during the inference steps [8]–[12]. Therefore, the ability of DNNs to deliver effective performance critically depends on their dependability [13], [14].

One of major dependability threats in computer systems is transient hardware faults or soft errors, which are commonly caused by high energy particles striking electronic devices and resulted in bit flip errors in logic values [15], [16]. On one hand, such faults may cause DNN failures, leading to misclassifications in DNN inferences [17]–[19]. On the other hand, DNNs are by no means perfect – they do not provide 100% accuracy (say in classification models) in the inference phase even without any transient hardware faults. DNNs often demonstrate some percentage of misclassifications when inferred on previous unseen data samples. As a result, such intrinsic algorithmic inaccuracies of DNN models also lead applications that use the DNNs produce wrong outputs or decisions. Since DNNs often suffer misclassifications due to intrinsic inaccuries during inference phase, researchers have wondered whether there is a strong need to protect DNNs from transient hardware faults which are much less frequent.

This paper takes the first step to answer the above question by characterizing DNN misclassifications caused by DNN intrinsic algorithmic inaccuracy as well as transient hardware faults, in the context of safety-critical applications such as AVs. Our contributions are threefold: (1) We first propose TENSORFI+ – an extension of TENSORFI [20] which is an open-source fault injection framework for DNNs – to support complex DNN models developed with TensorFlow 2 [21]. Our extension also enables us to conduct fault injections in a scalable manner, and so it is possible to study the impact of soft error on the accuracy of large DNN models. (2) We propose an evaluation method that measures DNN dependability with safety-related metrics, rather than simply counting misclassifications or silent data corruptions (SDCs) due to soft errors. This approach allows us to conduct fine-grained analysis of the impact of different DNN failures on safety-critical applications. (3) Based on the method, we evaluate a set of 30 popular DNN models that are used in recent studies and popular applications with respect to DNN intrinsic algorithmic inaccuracy and transient hardware faults, respectively. *To the best of our knowledge, this is the first study that focuses on the comparison of DNN failures between transient hardware faults and model intrinsic inaccuracies based on safety-related metrics.*

Our main results are as follows:

- We observe that, due to intrinsic algorithmic inaccuracy, DNNs demonstrate up to 95% of total misclassifications are non-safety-critical in real-world dataests such as ImageNet and CIFAR-100. Only a small percentage (around

2473-3105/22/$31.00 ©2022 IEEE
DOI 10.1109/PRDC55274.2022.00020

5%) of total misclassifcations are safety-critical.
- However, when we inject transient hardware faults to DNNs, the safety-critical misclassifcation probability is increased to up to 40%, which is significantly higher than that caused by intrinsic algorithmic inaccuracy.
- Based on our evaluation results, due to transient hardware faults on DNNs, we often observe misclassification cases where a DNN misclassifies a safety critical object to a non-safety critical object such as a man to a mushroom, or a baby to a snail. Our results suggest that future DNN systems likely require protections from transient hardware faults even though their intrinsic accuracy is not 100%.

II. Background Knowledge

A. Deep Learning Neural Networks

A deep neural network (DNN) consists of multiple computation layers [22] where higher-level abstraction of the input data or a feature map is extracted to preserve each layer's unique and important information. In this work, we consider convolutional neural networks of DNNs, as they are used in a broad range of DNN applications. The number of convolutional layers in such DNNs can range from three to a few tens of layers [1], [2]. Each convolutional layer applies a kernel (or filter) on the input feature maps to extract underlying visual characteristics and generate the corresponding output feature maps. Every computation result is saved in activations (ACTs) after being processed by an activation function (e.g., ReLU), connected based on the network topology. In some DNNs, a small number (usually less than 3) of fully-connected layers (FC) are typically stacked behind the convolutional layers for classification purposes. Between the convolutional and fully-connected layers, additional layers can be added, such as the pooling (POOL) and normalization (NORM) layers. Once a DNN topology is constructed, the network can be fed with training input data, and the associated weights, abstracted as connections between ACTs, will be learned through a backpropagation process. After training, the DNN is ready for image classification with testing data. This is referred to as the inferencing phase of the network and is carried out several times. The input of the inferencing phase is often a digitized image, and the output is a list of candidates of possible matches such as car, pedestrian, animal, each with a confidence score. Generally, the trained models are deployed to different systems like autonomous vehicles and used for continuous inference tasks.

B. Transient Hardware Faults

Transient hardware faults, commonly known as soft errors, are becoming prevalent in computer systems as the size of transistors keep shrinking [15], [23], [24]. Soft errors occur at circuit level, and then the fault propagates through each system layer. Finally, the fault reach application level and is propagated through program variables. Corrupted program variables may finally affect the program output and change it, leading to silent data corruption (SDC). The consequences of soft errors in DNN systems can be catastrophic as many of the systems are safety-critical, and error mitigation is required to meet specific reliability targets. For example, if the system detects a transporting truck correctly in AVs, it might apply the brakes in time to avoid a collision. However, if the truck is misclassified as different object like a bird because of soft error, the braking action may not be executed in time, especially when the car is operating at high speed. In this paper, we use soft errors and transient hardware faults interchangeably.

C. DNN Intrinsic Algorithmic Inaccuracy

There are different root-causes for a DNN to misclassify objects. One reason is due to DNN intrinsic algorithmic inaccuracy. The inaccuracy is often reflected in the top-1 accuracy of a DNN model, and the vast majority of the studies in machine learning area aim to improve the top-1 accuracy and so minimize the intrinsic algorithmic inaccuracy. The reasons why there is intrinsic inaccuracy of DNN models are many-fold: (i) poor image quality of the target object or in the training dataset, (ii) incapable underlying algorithm/architecture of the DNN model, (iii) inefficient training method, just to name a few. Note that every DNN suffers intrinsic algorithmic inaccuracy regardless of whether there are hardware faults or not, and hence there are no DNNs that can achieve 100% accuracy in real-world object classification datasets such as ImageNet.

D. Fault Model

In this work, we consider transient hardware faults that randomly occur during the execution of the DNN inference phase. The training phase can also be affected by the faults, however, training is usually a one-time process, so we can verify the results of the trained model. DNN inference, on the other hand, is applied frequently at runtime and is hence likely to be affected by such faults [17], [18], [25], [26].

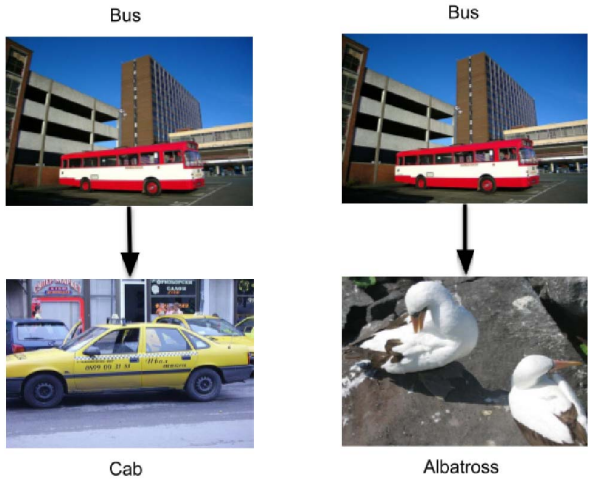
We inject faults directly to the output values of layers of DNN models. This fault injection method is aligned with prior studies [25], [27], [28]. We follow the one-fault-per-execution method as soft errors are relatively rare events with respect to the typical time of a program's execution, which is a common assumption in the literature [28]–[31]. We inject fault to only one layer of the model as faults in multiple layers are relatively rare events. Soft errors can occur in the software layer as single or multiple bit-flips. However, recent studies [27], [32] have shown that multiple bit-flip errors demonstrate similar error propagation patterns and Silent Data Corruption (SDC) probabilities as single bit-flip errors (at the program level) do, showing that studying single bit-flip fault injection is sufficient for drawing conclusions about the error resilience of DNN models.

We assume that faults do not modify the state/structure of the model (e.g., change the model's parameters [33], [34]), nor do we consider faults in the model's inputs, as they are extraneous to TensorFlow [35] and are outside of the scope of our paper.

III. Motivation and Hypothesis

Today, DNNs are being applied in various domains such as AVs, healthcare, space applications, etc. Many of them are safety-critical [36]–[38]. Misclassification during inference is common in DNNs and every DNN suffers intrinsic algorithmic inaccuracy. For example, the accuracy of popular pre-trained DNN models on ImageNet commonly varies from 71.3% to 85.7% [39]. On the other hand, transient hardware faults are becoming more and more prevalent in computer systems [13], [14], [40], which may also lead to DNN misclassifications. Various protection techniques [25], [41], [42] have been proposed to protect DNNs from hardware faults. These proposed techniques incur non-negligible performance overheads at runtime (up to 48%) even at fault-free executions [43].

In the past, most works in the area evaluate DNN resiliency based on the metrics such as SDC (e.g., misclassifications in DNN output). While SDC is a widely accepted metric to gauge system resiliency, in safety critical systems, SDCs in DNNs may not always lead to safety violations or critical failures. In this paper, we investigate the problem based on the impact of DNN failures caused by hardware faults and intrinsic inaccuracy respectively using our proposed safety-related metric.



(a) A bus is misclassified by an AV as a cab, but it will unlikely raise safety concerns because for both cases brake will be applied in the AV

(b) A bus is misclassified by an AV as an albatross, it will likely raise a serious safety concern because brakes may not be applied by the AV

Fig. 1: Impact of safety-critical misclassification in AVs

Not all misclassifications are equal. Although misclassification is a common phenomenon (due to either intrinsic algorithmic inaccuracy or transient hardware faults) during DNN inferences, misclassifications may not have a similar impact on system's behaviours. For example, in Figure 1, if an AV detects a bus as a cab, it will take the necessary steps such as applying brakes which are commonly required for other four-wheelers. However, if it predicts the bus as an albatross, the AV may consider a different driving decision instead of applying brake. So, its action might not be decisive for a four-wheeler in the latter case, which may consequently leads to serious accidents. Prior works in the field mainly focus on SDCs, which do not distinguish the failures according to safety concerns. Instead, they only look at whether a fault leads to misclassification or not. Neither do they compare the failures with those caused by DNN intrinsic inaccuracy.

Our hypothesis: Our intuition is that DNN misclassifications due to intrinsic algorithmic inaccuracy may not change the original prediction to a great extent from the perspective of safety concerns. For example, DNNs may misclassify a cat as a dog or a television as a computer screen. Since the misclassifications tend to stay within similar categories, the failures may not be safety critical. However, a transient hardware fault may misclassify the prediction to a great extent during the DNNs inference since the fault with bit-flip may lead to a drastic change in the numerical value of computations. Consequently, the fault may cause the inference to predict an output object significantly different from the true output class. *In light of this, we hypothesize that the safety impact of intrinsic algorithmic inaccuracies is much lower than that of transient hardware faults in a DNN, and hence DNNs most likely require protection from hardware faults if deployed in safety-critical applications.*

IV. TensorFI Fault Injection Framework

In this section, we first introduce the workflow of TensorFI [44] that we extend for fault injections. Then we discuss its limitation in supporting non-sequential DNN models. Finally, we propose our method to design TensorFI+ for injecting transient hardware faults into non-sequential models.

A. Workflow and Limitation

In order to validate our hypothesis, we use TensorFI to conduct fault injection experiments. TensorFI [44] is a fault injection framework for TensorFlow applications and has been used to simulate hardware faults in DNN model inferences in recent related studies [25], [35]. We choose it because it supports fault injection experiments on DNN models implemented using the TensorFlow framework [21] which is the most popular machine learning framework in use today [45]. TensorFI operates on TensorFlow dataflow graphs which contains two main components: (1) operators of computational units (e.g., matrix multiplication etc), and (2) tensors of data units. One can build machine learning models using the built-in operators or define their customized operators. TensorFI duplicates the TensorFlow graph with customized operators, which are designed to not only be able to perform computations as standard operators do, but also inject faults at runtime of model inferences.

As TensorFI injects faults on different operators, to support all types of DNN models, we need to implement a large number of operators when deploying TensorFI in the models. Nowadays, it is popular to use TensorFlow 2 and Keras to deploy DNN models because of the easy use of

the framework and numerous available pre-trained models in Keras. Recently, TENSORFI has been upgraded to work with TensorFlow 2 and Keras. It uses the Keras model APIs to inject static faults in the output states of layers, which only supports fault injections in sequential models. The limitation of TENSORFI still remains in the support of non-sequential models such as ResNets, DenseNets, InceptionNet, GoogleNet which are widely used in numerous applications nowadays.

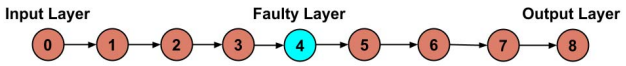


Fig. 2: Fault injection in a sequential model

Figure 2 shows how TENSORFI injects faults into a sequential model. In a sequential model, a layer's input is sequentially dependent on its previous layer's output. In the example, say layer 4 is the selected layer for fault injection, then TENSORFI computes the output of layer 4 from layer 1 using the inputs of layer 1 and injects the fault into the output of layer 4. As the fault propagates, TENSORFI computes the output by computing the outputs of layer 8 from layer 5 using the corrupted input of layer 5 which receives input from layer 4. However, if a DNN is non-sequential, in which case a layer's input is connected to multiple-layers' outputs, TENSORFI has no way to trace error propagation in its current implementation.

B. Support for Non-sequential DNN Models

We propose TENSORFI+ to work with both sequential and non-sequential DNN models. As we can 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 [44], 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 [44]. Hence, when we inject faults to a specific layer of a non-sequential model, we need to propagate the erroneous output of the injected layer to the subsequent layers across different directions to the final output. Thereby we need to keep track of the inputs and outputs of each layer and formulate a dependency graph based on the DNN structure in order to provide support for non-sequential models in TENSORFI+. We do so by leveraging an internal data structure in Keras which keeps sequential 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 output identifier in the data structure, we use this information to build our dependency graph that can relate the input and output dependency for every layer in the entire DNN model.

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

1) **Identifier**: A unique identifier of the layer output
2) **Input_layers**: List of layer indices on which the layer depends
3) **Output_layers**: List of layer indices which depend on the layer

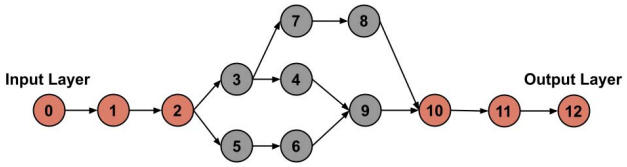


Fig. 3: Example dependency graph with supernodes (marked as red)

Figure 3 shows an example of a dependency graph for a 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 is a representative of a layer in the dependency graph. Each arrow represents a layer dependency in the graph. As the model is non-sequential, any layer might have multiple inputs and outputs. So there can be multiple branches in the dependency graph like layers 3 to 10. Recall that we are using Keras API calls for fault propagation. If we inject faults 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, we can not get output of layer 12 from layer 4 because layer 8 and 10 required value of 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 supernodes (marked as red in the Figure) in the dependency graph. Specifically, any subsequent layer after a supernode is not dependent on the layers prior to that supernode. Hence, the layers between 3 and 9 are not supernodes. We call them non- supernodes (marked as grey in the figure). However, layers 0, 1, 2, 10, 11, 12 are supernodes, and we can calculate output value of any subsequent node from these nodes.

In this way, if one can compute the inputs of a supernode, we can use them to calculate the output of final output layer. If a fault is injected in a non-supernode (marked as grey in the Figure), we first find its next immediate supernode. We then recursively compute the inputs (by the API calls) of the supernode by tracing the corrupted output value and compute the output. As the dependency graph and supernodes are model-specific, profiling of the model is required before fault injections.

We use Figure 3 as a running example to show how a fault can be injected in a non-sequential model. The current fault injection procedure in sequential models is to trace input dependency from immediate neighbours and propagate the value in subsequent layers till the output layer. Say we inject a fault at layer 6, then the current procedure is: (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. Additionally, we cannot use the output of layer 9 only to compute the final output of layer 12 because layer 10 also requires the output of layer 8 which we need to compute as well. In contrast, we propose to first compute the next immediate supernode of the layer where we want to inject the fault. In the example, the immediate next supernode of layer 6 is layer 10. We then recursively compute the inputs of the supernode. In other words, we compute outputs of both layer 8 and layer 9. This is because the inputs of the supernode (e.g., layer 9) might be from multiple layers and we need to compute the outputs of all those layers also. If we need to compute the inputs of any layers which are not affected by faults(e.g., layer 8), we compute them by using the original inputs of the model.

C. Efficiency Improvement

To improve the efficiency of TENSORFI+ when injecting faults in non-sequential models, we take two major design improvements. 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 faster and more efficient. In Figure 3, layer 8 and layer 9 both depends on the output of layer 3. To save time, we can compute the output of layer 3 once (e.g., during output computation of layer 8) and then use this output if required(e.g., to compute output of layer 9 later).

In addition, we compute the longest possible sequence of layers when we recursively compute the inputs of super nodes. For example, layer 10 requires outputs of layer 8 and 9 when injecting faults on layer 2. 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

We can see that we need to compute for a total of seven layers' outputs. However, if we detect the longest sequences of layers that can be computed at a 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 need to compute only four layers' outputs(e.g., layer 8, 4, 6, 9), making TENSORFI+ more efficient and faster in fault injections of non-sequential models. As real-world DNN models are usually complex and non-sequential, more opportunities are presented to prune more longest sequences. Thus, both of the above strategies provide significant performance improvement compared to naive recursive approach.

V. METHODOLOGY

In this section, we formulate supergroups for DNN output classes based on safety metrics in the context of AVs, and discuss how we determine DNN failures in our experiments. We make our code publicly available[1].

A. Supergroup Formation

We focus on two popular datasets in this paper, they are ImageNet [46] and CIFAR-100 [47] datasets. Both datasets contain images reflecting common objects of diverse types. Our first step is to group similar objects of DNN output classes based on their characteristics and safety severity. Since our analysis target is safety in AVs, we then merge these groups into different supergroups from the perspective of AV decisions in driving scenarios. For example, a truck and a cab are two different output classes in both datasets. However, an AV will likely react to a truck and a cab in a similar way during the self-driving and hence they will be in the same supergroup in our formulation. However, the actions will be likely different if an AV detects a person as a clock. So, we put person and clock in different groups. In this way, we categorize all the output classes into multiple groups in each dataset respectively and then finally combine these groups into two supergroups, named as Supergroup A and Supergroup B.

Our aim is to keep all the safety critical classes in Supergroup A which infers that inter-group misclassifications of Supergroup A are safety-critical, but intra-group ones are not. For example, classes such as person, vehicles, big animals, large objects are in supergroup A. So, predicting a boy as a girl (both are person) is not safety-critical, but predicting a boy (person) as a car (vehicle) is safety-critical. Supergroup B contains non-safety-critical classes such as household chores, small animals etc. Any misclassification within Supergroup B is not considered to be safety-critical. For example, predicting a flower as a spider is not safety critical, because AVs reactions to these two are likely similar. Additionally, misclassification of a object from supergroup A to some object in supergroup B will be safety-critical but the opposite is not true. For example, predicting a boy (in supergroup A) to a flower (in supergroup B) might cause accident. However, predicting a flower (in supergroup B) to a boy (in supergroup A) results in AV overcautious reactions, which is not safety-critical. We use the two supergroups as the metrics in our fault injection evaluation. We list the details of each supergroup in ImageNet and CIFAR-100 in Table II in the Appendix.

1) CIFAR-100 Supergroups: CIFAR-100 dataset has a total of 100 output classes which fall into a total of 20 categories according to the official documentation [48]. According to our proposed method of formulating supergroups, we combine these CIFAR-100 categories into the two supergroups (details are in Table II in Appendix). The first supergroup contains significantly distinct objects where misclassification between them is safety-critical. For example, *People, Large Animals, Two Wheelers, Four Wheelers, and Large Outdoor Objects* are

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

in the Supergroup A. On the other hand, *Household Objects, Small Animals, and Trees* are in Supergroup B. We provide the main rationale behind the CIFAR-100 supergrouping below.

- We consider *people* as an unique group in Supergroup A because they are different from *vehicles, animals, buildings*, etc. So the actions related to a person should be different for a AV.
- Generally, *vehicles* are moving objects. So, actions related to a vehicle are different than any other objects and so they are in Supergroup A. Additionally, *two-wheelers and four-wheelers* lead to different AV reactions because *two-wheelers* need significantly reduced space than *four-wheelers* on the road. So misclassifying a *four-wheeler* to a *two-wheeler* might lead to severe consequences. So, they are two different groups in Supergroup A.
- *Large outdoor objects* such as *bridges, houses, and mountains* are huge in size. If an AV predicts a *house* (static) as a *car* (dynamic), the reactions might be different. So, they are considered as a different group in Supergroup A.
- Actions related to *large animals* such as *elephants, tigers* are likely different than that of *vehicles, persons, or outdoor objects*. So they are considered to be in Supergroup A.
- Misclassifications of *small household objects and small animals* are unlikely safety-critical. So, they all are placed in Supergroup B.

2) ImageNet Supergroups: ImageNet dataset has a total of 1,000 classes. There is an hierarchy of classes for the dataset in the documentation [49]. Using the hierarchy of this dataset, we divide the output classes into 10 different groups based on safety concern of AVs. We further classify them into two supergroups. Details are in Table II in Appendix. Below we provide the major rationale behind the supergrouping in ImageNet.

- Similarly to CIFAR-100, *two-wheelers* are considered different from *four-wheelers* and placed them to two different groups in Supergroup A.
- *Emergency vehicles* are considered in different group from *four-wheelers* because *emergency vehicles* often need more precedence than normal cars in traffic signals.
- Different types of *buildings, bridges, fountains, mountains* are considered into a different group in Supergroup A because of their large size and structure.
- Similar to CIFAR-100, *persons* are also considered as a different group in Supergroup A.
- *Planes and birds* are in different supergroups because AVs will likely take different actions to each.
- Different types of *tools and household objects*, *water vehicles*, *small animals* are placed in Supergroup B because misclassification among these classes will not be safety-critical.

B. Failure Outcomes

In this subsection, we define different types of failure outcomes in our experiment.

Correct Classification Probability: During fault-free inference, the images which are correctly predicted by the model.

Masking Probability: Masking probability is the probability of predictions that are the same as the original fault-free predictions.

SCM and Non-SCM Probability: Safety-critical-misclassification (SCM) probability can be computed during both fault-free inference and fault injection experiments. During fault-free inference, we refer SCM probability as the misclassification probability when the original label is in Supergroup A and the predicted label is in Supergroup B or they are from different classes within Supergroup A. In this case, the SCMs are caused by intrinsic algorithmic inaccuracy of the DNN model. During fault injection, this probability is computed between fault-free prediction and fault-injected prediction. On the other hand, non-SCM probability complements to SCM probability, and they add up to 100%.

C. Fault Injection Methods

In our fault injection experiments, we randomly sample 10,000 images from each ImageNet [46] and GTSRB [50] dataset test sets. For both these datasets, the sampled dataset has around similar distribution of category A and B as the original dataset. For example, the original test set of ImageNet contains around 25% and 75% data which are from Supergroup A and Supergroup B, respectively. After random sampling the resulting sampled dataset has around 30% and 70% data from Supergroup A and Supergroup B, respectively. As CIFAR-100 [47] has 10000 images in the test set, we use the full test for our experiment.

First, we run inference on the sampled datasets to get the accuracy (and so the intrinsic algorithmic inaccuracy) for all the models. We calculate the SCM and non-SCM probabilities for each model due to intrinsic model inaccuracies. Then we inject 3,000 random faults on both correctly classified as well as misclassified images to determine the SCM and non-SCM probabilities. We follow the one-fault-per-execution model, a common practice in the related studies in the literature [28]–[31]. For each fault injection run, we randomly sample one tensor from the randomly chosen layer of the model and randomly flip a bit.

VI. Experimental Setup

A. Datasets and DNN Models

We use ImageNet [46], CIFAR-100 [47] and German Traffic Sign Recognition Benchmark(GTSRB) [50] 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 work with datasets and models related to AVs. In our experiments, we have 13 DNN models for CIFAR-100, 12 DNN models for ImageNet and 5 DNN models for GTSRB. For ImageNet, we adopt pre-trained models which

are available on Keras[2] such as VGG, DenseNets, ResNets, MobileNets, Inception. 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. The top-1 accuracy and masking probability of all these models are shown in Table I.

B. Hardware

All of our experiments are conducted on Linux machines running Ubuntu 20.04 with Intel CPUs.

C. Research Questions

We conduct our evaluation by asking 2 research questions below:

- RQ1: What is the DNN SCM and non-SCM probability due to intrinsic algorithmic inaccuracy?
- RQ2: Is DNN SCM probability caused by transient hardware faults different compared with that of intrinsic algorithmic inaccuracy?

VII. Results

We organize our results by research questions (RQs).

A. *RQ1: What is the DNN SCM and non-SCM probability due to intrinsic algorithmic inaccuracy*

In this RQ, we study how intrinsic algorithmic inaccuracy raises safety concern for DNN models. As mentioned, we run inference over 10,000 randomly sampled images for each DNN model of each dataset and identify the accuracy. Among the misclassified images, we summarize how many of them are SCMs and non-SCMs.

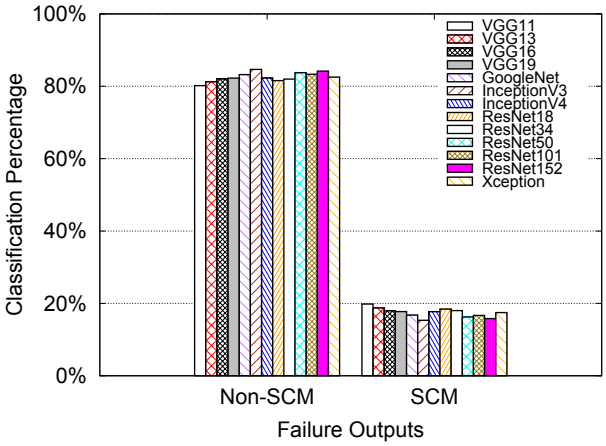


Fig. 4: SCM and non-SCM due to intrinsic algorithmic inaccuracy in DNNs with CIFAR-100 dataset

The results are shown in Table I. We can see that the accuracy of CIFAR-100 dataset without any fault injections

[2]https://keras.io/api/applications/

[3]https://github.com/gmalivenko/pytorch2keras

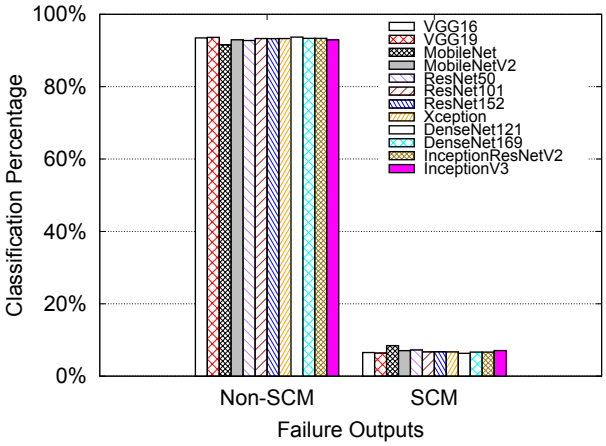


Fig. 5: SCM and non-SCM due to intrinsic algorithmic inaccuracy in DNNs with ImageNet dataset

is between 65% to 80% across all the models. That means, there are around 20% to 35% of images that are misclassified due to the model intrinsic algorithmic inaccuracy. We further examine the probabilities of SCMs and non-SCMs. In Figure 4, we show the SCM and non-SCM probabilities among the misclassified images in CIFAR-100 dataset. We can see that around 80% of misclassified images are non-SCMs, that is, they are unlikely safety-critical concerns in AVs.

On the other hand, from Table I, we observe there are 20% to 30% of misclassifications in the DNNs with ImageNet dataset. In Figure 5, we can see that, among the misclassified images, more than 90% of the misclassifications are non-SCMs, leaving less than 10% misclassifications SCMs. Based on the results, we report that although SCMs are non-negligible in DNNs due to intrinsic algorithmic inaccuracy, most of misclassifications in DNNs tend to be non-SCMs, without injecting transient hardware faults.

Table III in Appendix lists examples (ResNet101) of misclassifications (both SCMs and non-SCMs) due to algorithmic intrinsic inaccuracy.

We sample several object classes in CIFAR-100 dataset, and see how the DNN misclassifies the objects.

As reported, for apple object, all the 11 misclassifications we observe result in other objects like mushrooms and pears. These objects are closely similar to apple in shapes and properties and will unlikely cause any safety concerns for AVs. Apple does not have any predictions that result in SCMs. Thereby the SCM probability in apple object is minimal. On the other hand, other sampled objects tend to have both SCMs and non-SCMs among the misclassifications, but as mentioned, the non-SCMs probability is significantly higher than SCMs. Recall that the non-SCMs tend to result in closely similar and related objects as the original object in these samples. For example, *girl, boy, man, and woman* are non-SCMs when the DNN misclassifies a baby. Similarly, *streetcar, pickup trucks, and tanks* are non-SCM of a bus.

Model Name	Dataset	Top-1 Accuracy	Masking Probability on Correctly Classified	Masking Probability on Misclassified
VGG16	ImageNet	71.18%	96.57%	96.13%
VGG19	ImageNet	71.35%	96.73%	96.27%
ResNet50	ImageNet	74.76%	98.33%	98.33%
ResNet101	ImageNet	76.14%	98.50%	98.43%
ResNet152	ImageNet	76.38%	98.03%	98.23%
MobileNet	ImageNet	70.25%	98.23%	98.77%
MobileNetV2	ImageNet	71.07%	99.37%	98.83%
DenseNet121	ImageNet	75.04%	98.63%	98.80%
DenseNet169	ImageNet	75.76%	98.63%	98.47%
Xception	ImageNet	78.92%	97.97%	98.33%
InceptionV3	ImageNet	77.77%	97.67%	98.63%
InceptionResNetV2	ImageNet	80.11%	98.73%	98.93%
VGG11	CIFAR-100	69.12%	98.60%	98.67%
VGG13	CIFAR-100	71.51%	98.70%	98.60%
VGG16	CIFAR-100	72.33%	98.97%	98.90%
VGG19	CIFAR-100	71.53%	99.20%	98.50%
ResNet18	CIFAR-100	76.35%	98.63%	99.03%
ResNet34	CIFAR-100	77.68%	99.23%	98.83%
ResNet50	CIFAR-100	78.52%	98.57%	98.60%
ResNet101	CIFAR-100	78.93%	99.17%	98.63%
ResNet152	CIFAR-100	79.68%	98.63%	98.93%
GoogleNet	CIFAR-100	76.70%	98.80%	98.60%
InceptionV3	CIFAR-100	79.45%	98.80%	98.77%
InceptionV4	CIFAR-100	77.80%	98.57%	98.30&
Xception	CIFAR-100	77.96%	97.76%	98.00%
VGG16	GTSRB	97.57%	99.36%	98.83%
VGG19	GTSRB	98.25%	99.1%	99.1%
ResNet34	GTSRB	98.98%	98.43%	98.5%
ResNet50	GTSRB	97.91%	98.93%	98.93%
ResNet101	GTSRB	98.55%	99.26%	98.76%

TABLE I: Top-1 accuracy and masking probability on initially correct and incorrect classifications

B. RQ2: Is DNN SCM probability caused by transient hardware faults different compared with that of intrinsic algorithmic inaccuracy

In Table I, we show the SCM and non-SCM probabilities of DNNs when we inject transient hardware faults. We do so by injecting faults on both initially correctly classified and initially misclassified images in the two datasets with all the DNN models.

For CIFAR-100, when we injected faults on correctly classified images, about 98% to 99% faults are masked. Overall, there is around an average of 1% to 2% SDCs observed across DNNs. From Figure 6 and 7, we can see that, among all the SDCs, an average of 20% to 50% resulted in SCMs. More specifically, there are 30% to 40% SCMs observed in initially correct classifications, whereas it is 20% to 50% of SCMs in initially misclassified images. On average, the SCM probability across DNNs in CIFAR-100 is around 37%. Examples of SCMs and non-SCMs due to transient hardware faults can be found in Table IV in Appendix.

In ImageNet, we observe a very similar pattern. There are 1% to 4% SDCs as results of fault injections. Among all the SDCs, we observe there are 10% to 38% SCMs in the initially correct classifications across DNNs, whereas it is 14% to 36% in the initially misclassified images(Figure 8 and 9). The overall SCM probability is 26% across DNNs in ImageNet, which is similar to what we measured in CIFAR-100.

Finally, we observe that the SCM probability due to transient hardware faults is about 2.3 times and 4.1 times higher than that of intrinsic algorithmic inaccuracy in CIFAR-100 and ImageNet respectively, indicating that transient hardware faults may likely result in more SCMs and causing safety-related concerns in AVs.

As the accuracy of DNNs continues to improve while the soft error rates keep increasing, we expect the tend will only widen in the future.

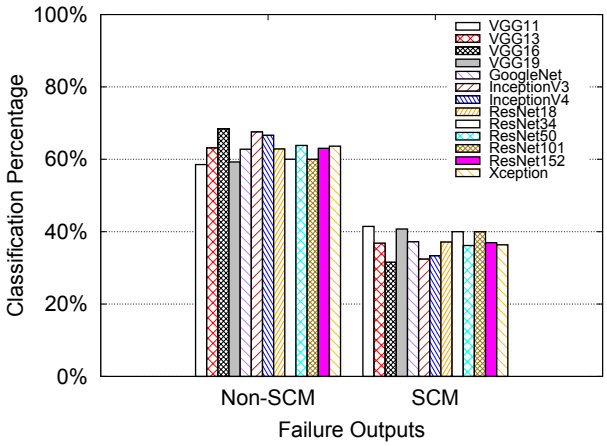


Fig. 6: SCM and non-SCM due to transient hardware faults on initially correct classification in DNNs with CIFAR-100 dataset

VIII. Related Work

There have been many works that studies DNN resilience and accuracy. Tian et al. [51] used transformation matrices (e.g., scale, rotate the image) to automatically generate corner case images to trigger unexpected behaviors of the model. Ma et al. [52] designed a mutation framework to fuzz DNN models, the tecnique was then used to evaluate the test data quality. These studies focus on improve DNN accuracy.

Rubaiyat et al. [53] built a strategic software FI framework, which leverages hazard analysis to identify potential unsafe scenarios to prune the injection space. However, they did not consider transient hardware faults. Li et al. [18] build a fault injector to evaluate transient hardware faults in DNN applications and studied their resilience.

Chen et al. propose BinFI [25], an efficient fault injector for finding the safety-critical bits of ML applications. These papers measured the DNN resilience under hardware faults and reported SDCs in DNNs. However, unlike our study, none

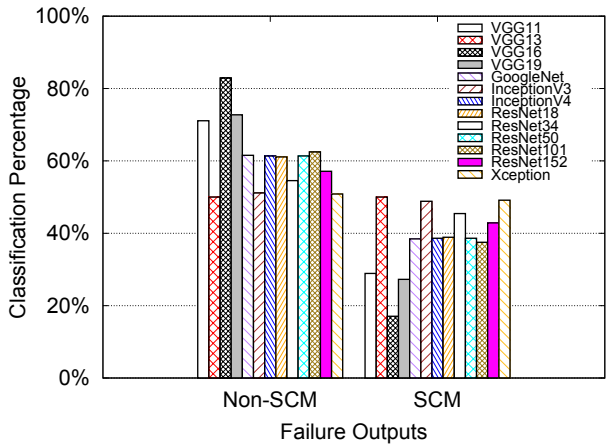


Fig. 7: SCM and non-SCM due to transient hardware faults on initially misclassified images in DNNs with CIFAR-100 dataset

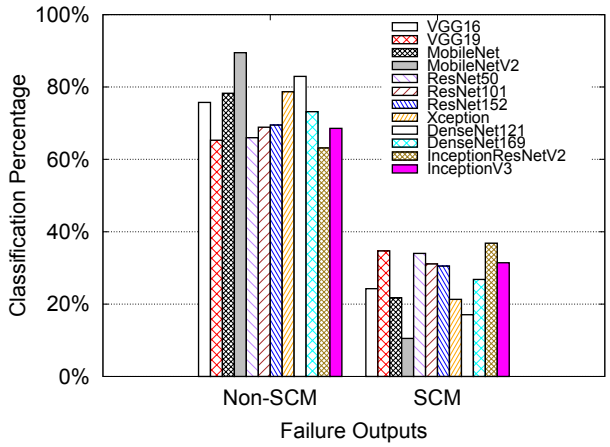


Fig. 8: SCM and non-SCM due to transient hardware faults on initially correct classification in DNNs with ImageNet dataset

of them compared the impacts of DNN failures caused by intrinsic algorithmic inaccuracy and hardware faults.

IX. Conclusion

In this work, we investigate how the hardware transient faults change the prediction of DNN models based on safety-critical metrics compared to algorithmic intrinsic inaccuracy. Our intuition is that intrinsic algorithmic inaccuracy may not change the original prediction to a great extent. But a transient hardware fault may modify any bit position and lead to a drastic change in the numerical value of computations. For this experiment, we use two popularly used classification datasets named ImageNet and CIFAR-100 and group the labels based on safety-related metrics. We evaluate our analysis on 25 popularly used DNN models. Our evaluation demonstrates that without hardware faults, the safety-critical misclassification probability is around 5% to 10%, but under hardware faults,

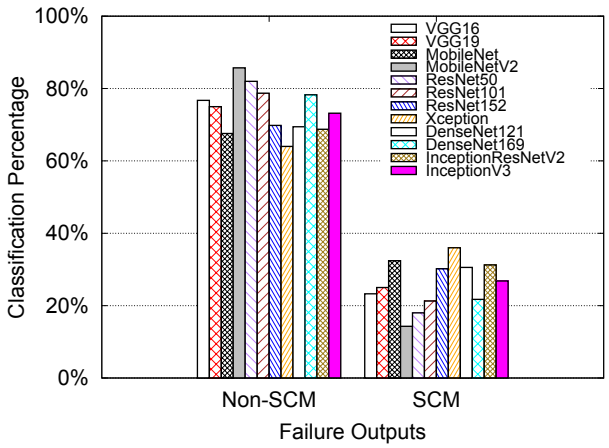


Fig. 9: SCM and non-SCM due to transient hardware faults on initially misclassified images in DNNs with ImageNet dataset

this probability drastically increases around 4 times to up to 40%, indicating DNN systems in safety-critical systems likely need to be protected from transient hardware faults even though the model intrinsic accuracy in not 100%.

References

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

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[3] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

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

[5] 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

[6] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47. USA: IEEE Computer Society, 2014, p. 609–622. [Online]. Available: https://doi.org/10.1109/MICRO.2014.58

[7] 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

[8] 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.

[9] 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.

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

[11] H.-J. Yoon, A. Ramanathan, and G. Tourassi, "Multi-task deep neural networks for automated extraction of primary site and laterality information from cancer pathology reports," 10 2017, pp. 195–204.

[12] Z. Xiong, M. Stiles, and J. Zhao, "Robust ecg signal classification for the detection of atrial fibrillation using novel neural networks," 09 2017.

[13] 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.

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

[15] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," vol. 25, no. 6, p. 10–16, nov 2005. [Online]. Available: https://doi.org/10.1109/MM.2005.110

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

[17] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: a framework for quantifying the resilience of deep neural networks," 06 2018, pp. 1–6.

[18] 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

[19] F. dos Santos, C. Lunardi, D. Oliveira, F. Libano, and P. Rech, "Reliability evaluation of mixed-precision architectures," 02 2019, pp. 238–249.

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

[21] 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.

[22] 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.

[23] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *11th International Symposium on High-Performance Computer Architecture*. IEEE, 2005, pp. 243–247.

[24] 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.

[25] 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

[26] M. Sabbagh, C. Gongye, Y. Fei, and Y. Wang, "Evaluating fault resiliency of compressed deep neural networks," in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, 2019, pp. 1–7.

[27] 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.

[28] G. Li, K. Pattabiraman, S. K. S. Hari, M. Sullivan, and T. Tsai, "Modeling soft-error propagation in programs," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 27–38.

[29] R. A. Ashraf, R. Gioiosa, G. Kestor, R. F. DeMara, C.-Y. Cher, and P. Bose, "Understanding the propagation of transient errors in hpc applications," in *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–12.

[30] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "Gpu-qin: A methodology for evaluating the error resilience of gpgpu applications," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 221–230.

[31] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, "Quantifying the accuracy of high-level fault injection techniques for hardware faults," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 375–382.

[32] 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.

[33] S. Jha, S. S. Banerjee, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "Avfi: Fault injection for autonomous vehicles," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2018, pp. 55–56.

[34] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepmutation: Mutation testing of deep learning systems," 2018. [Online]. Available: https://arxiv.org/abs/1805.05206

[35] 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

[36] L. Fridman, D. E. Brown, M. Glazer, W. Angell, S. Dodd, B. Jenik, J. Terwilliger, A. Patsekin, J. Kindelsberger, L. Ding, S. Seaman, A. Mehler, A. Sipperley, A. Pettinato, B. D. Seppelt, L. Angell, B. Mehler, and B. Reimer, "Mit advanced vehicle technology study: Large-scale naturalistic driving study of driver behavior and interaction with automation," *IEEE Access*, vol. 7, pp. 102 021–102 038, 2019.

[37] T. Ozturk, M. Talo, E. A. Yildirim, U. B. Baloglu, O. Yildirim, and U. Rajendra Acharya, "Automated detection of covid-19 cases using deep neural networks with x-ray images," *Computers in Biology and Medicine*, vol. 121, p. 103792, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010482520301621

[38] V. Kothari, E. Liberis, and N. D. Lane, "The final frontier: Deep learning in space," 2020. [Online]. Available: https://arxiv.org/abs/2001.10362

[39] "Keras applications," https://keras.io/api/applications/.

[40] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. Debardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, "Addressing failures in exascale computing," *Int. J. High Perform. Comput. Appl.*, vol. 28, no. 2, p. 129–173, may 2014. [Online]. Available: https://doi.org/10.1177/1094342014522573

[41] S. K. S. Hari, S. V. Adve, and H. Naeimi, "Low-cost program-level detectors for reducing silent data corruptions," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, 2012, pp. 1–12.

[42] G. Li, Q. Lu, and K. Pattabiraman, "Fine-grained characterization of faults causing long latency crashes in programs," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 450–461.

[43] A. Mahmoud, S. K. S. Hari, C. W. Fletcher, S. V. Adve, C. Sakr, N. Shanbhag, P. Molchanov, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Optimizing selective protection for cnn resilience," in *32nd IEEE International Symposium on Software Reliability Engineering, ISSRE 2021*. IEEE Computer Society, 2021, pp. 127–138.

[44] 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.

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

[46] 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.

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

[48] "Cifar-100 dataset description," https://www.cs.toronto.edu/~kriz/cifar.html.

[49] "Imagenet hiererchy," https://observablehq.com/@mbostock/imagenet-hierarchy.

[50] 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.

[51] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 303–314. [Online]. Available: https://doi.org/10.1145/3180155.3180220

[52] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepmutation: Mutation testing of deep learning systems," 2018. [Online]. Available: https://arxiv.org/abs/1805.05206

[53] A. H. M. Rubaiyat, Y. Qin, and H. Alemzadeh, "Experimental resilience assessment of an open-source driving agent," in *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, dec 2018. [Online]. Available: https://doi.org/10.1109%2Fprdc.2018.00016

APPENDIX

TABLE II: Supergroup formation

CIFAR-100	SG A	People	baby, boy, girl, man, woman
		Large Animals	bear, lion, tiger, wolf, dolphin, whale, camel, cattle, chimpanzee, elephant
		Two Wheelers	bicycle, motorcycle
		Four Wheelers	bus, pickup truck, train, lawn-mower, rocket, streetcar, tank, tractor
		Large Outdoor Objects	bridge, castle, house, road, skyscraper, forest, mountain, sea
	SG B	Household Objects	apples, oranges, roses, tulips, bottles, bowls, plates, clock, television, bed, table
		Small Animals	aquarium fish, ray, bee, beetle, cockroach, snail, spider, lizard, snake, mouse, rabbit
		Trees	maple, oak, palm, pine, willow
ImageNet	SG A	Emergency Vehicle	ambulance, fire engine, police van, stretcher, army tank
		Two Wheeler	tandem bicycle, moped, motor scooter, mountain bike, tricycle, unicycle
		Four Wheeler	amphibious vehicle, cab, jeep, limousine, school bus, sports car, tractor
		Person	ballplayer, groom, scuba diver
		Big Animals	ostrich, Mexican hairless, timber wolf, cougar, hippopotamus, African elephant
		Geological Formation and Structure	bakery, barn, castle, fountain, palace, cliff, seashore, valley, volcano
	SG B	Aerial Objects	magpie, kite, peacock, hummingbird, airliner, airship, balloon, warplane
		Water Vehicles	aircraft carrier, fireboat, container ship, pirate ship, submarine, speedboat
		Small Animals	goldfish, bullfrog, green lizard, Komodo dragon, king snake, snail, mongoose
		Tools and Household Chores	baseball, binoculars, birdhouse, can opener, cello, cloak, desktop computer

TABLE III: Misclassification examples due to intrinsic algorithmic inaccuracy

Dataset Name	**Model Name**	**Original Class**	**Total SCM count**	**Sample SCM Predictions**	**Total non-SCM count**	**Sample non-SCM Predictions**
CIFAR-100	ResNet101	apple	0	-	11	mushroom, sweet pepper, pear
		baby	8	bed, television, chimpanzee, tractor, tulip, snail, butterfly, orchid	26	girl, boy, woman, man
		bear	6	hamster, bowl, lobster, bee, mouse, shrew	25	otter, elephant, beaver, chimpanzee, lion, camel, seal, cattle, whale, kangaroo
		bus	6	pine tree, television, can, motorcycle, couch, bridge	18	streetcar, pickup truck, tank
ImageNet	DenseNet121	green lizard	0	-	8	common iguana, alligator lizard, whiptail, american chameleon
		minivan	2	ambulance, odometer	5	cab, racer, beach wagon, pickup
		screen	0	-	11	television, desktop computer, desk, iron, home theater, notebook
		passenger car	2	gas pump, fire engine	3	electric locomotive, steam locomotive

TABLE IV: Misclassification examples due to transient hardware faults

Dataset Name	Model Name	Initial Classification Type	Original Class	Initial Predicted Class	Total SDC	SCM labels after FI	non-SCM labels after FI
CIFAR-100	ResNet101	Correct	Train	Train	2	Clock(1), Girl(1)	-
			Tank	Tank	10	Keyboard(4), Trout(1), Clock(4), Apple(1)	-
		SCM	Man	Chimpanzee	5	Mushroom(1), Possum(1), Clock(2), Keyboard(1)	-
			bridge	streetcar	4	Keyboard(1), Clock(2), Flatfish(1)	-
		Non-SCM	Woman	Girl	4	Camel(1), Keyboard(1), Tractor(1), Lawn-mower(1)	-
			Bus	Streetcar	5	Clock(2), Crocodile(1), Tulip(1), Keyboard(1)	-
ImageNet	DenseNet121	Correct	Police Van	Police Van	10	Tench(6), Pencil Sharpener(1), Broccoli(1), Husky(1), Stethoscope(1)	-
			Passenger Car	Passenger Car	6	American lobster(1), Mixing bowl(1), Waffle iron(1), Saltshaker(1), Ping-pong ball(1), Microwave(1)	-
		SCM	scuba diver	snorkel	10	-	Cradle(1), Broccoli(1), Ping-pong ball(1), Papillon(1), Tench(1), Scuba diver(1), Strainer(1), Microwave(1), Bassinet(1)
			Street Sign	Parking Meter	14	-	Tench(5), Letter opener(1), Horizontal bar(1), Pay-phone(1), Ping-pong ball(1), Teddy(1), Cassette player(1), Street sign(2), Trash can(1)
		Non-SCM	Limousine	Minibus	11	Tench(3), Stethoscope(1), Television(1), Pool table(1), Ping-pong ball(1), Seat belt(1), Saltshaker(1)	Forklift(1), Limousine(1)
			Timber Wolf	Coyote	7	Screwdriver(1), Ping-pong ball(3), Television(1), Tench(2)	-