Received: 27 June 2023 | Revised: 26 November 2023 | Accepted: 2 January 2024

DOI: 10.1002/stvr.1873

SPECIAL ISSUE PAPER

WILEY

Investigating the impact of transient hardware faults on deep learning neural network inference

Md Hasanur Rahman | **Sabuj Laskar** | **Guanpeng Li**

University of Iowa, Iowa City, Iowa, USA

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

Summary

Safety-critical applications, such as autonomous vehicles, healthcare, and space applications, have witnessed widespread deployment of deep neural networks (DNNs). Inherent algorithmic inaccuracies have consistently been a prevalent cause of misclassifications, even in modern DNNs. Simultaneously, with an ongoing effort to minimize the footprint of contemporary chip design, there is a continual rise in the likelihood of transient hardware faults in deployed DNN models. Consequently, researchers have wondered the extent to which these faults contribute to DNN misclassifications compared to algorithmic inaccuracies. This article delves into the impact of DNN misclassifications caused by transient hardware faults and intrinsic algorithmic inaccuracies in safety-critical applications. Initially, we enhance a cutting-edge fault injector, TensorFI, for TensorFlow applications to facilitate fault injections on modern DNN non-sequential models in a scalable manner. Subsequently, we analyse the DNN-inferred outcomes based on our defined safety-critical metrics. Finally, we conduct extensive fault injection experiments and a comprehensive analysis to achieve the following objectives: (1) investigate the impact of different target class groupings on DNN failures and (2) pinpoint the most vulnerable bit locations within tensors, as well as DNN layers accountable for the majority of safety-critical misclassifications. Our findings regarding different grouping formations reveal that failures induced by transient hardware faults can have a substantially greater impact (with a probability up to 4× higher) on safety-critical applications compared to those resulting from algorithmic inaccuracies. Additionally, our investigation demonstrates that higher order bit positions in tensors, as well as initial and final layers of DNNs, necessitate prioritized protection compared to other regions.

KEYWORDS

deep neural networks, safety-critical misclassifications, transient hardware faults

1 | INTRODUCTION

Over the last several years, deep learning neural networks (DNNs) have been widely deployed in various fields including computer vision, natural language processing, and autonomous vehicles (AVs) [1-4]. As the usage and demand for DNNs continue to rise significantly, researchers have suggested the utilization of specialized hardware accelerators like graphics processing units (GPUs) and tensor processing units (TPUs) to expedite the inference process of DNNs [5-7].

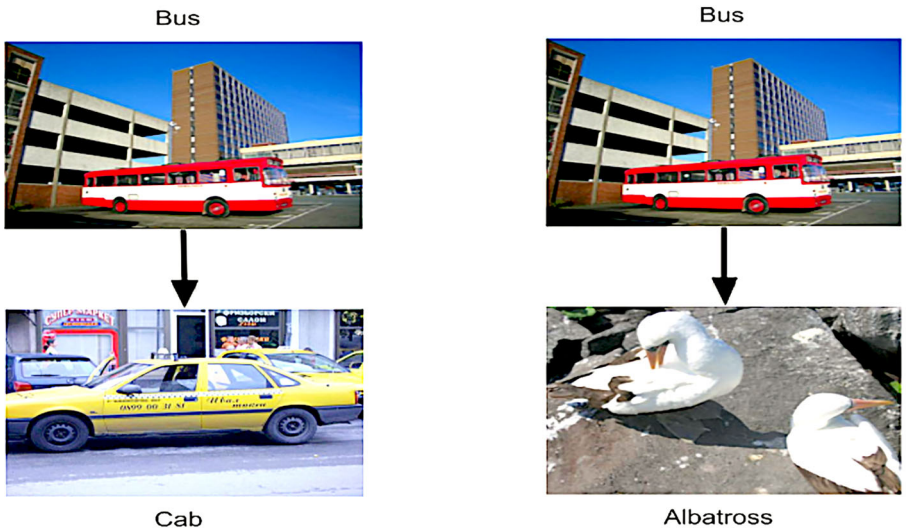
Md Hasanur Rahman and Sabuj Laskar are the co-first authors.

In addition to the high throughput and low latency, many machine-learning-based applications such as AVs, healthcare, and space technology demand a high degree of reliability during the inference [8-12]. Hence, the dependability of DNNs plays a critical role in ensuring their effective and accurate performance [13,14].

Transient hardware faults, also known as soft errors, poses a major threat to the dependability of computing systems. These errors occur when the high energy particles strike electronic devices, causing bit flip errors in logic values, making faults propagate through the system and potentially affect the system output [15,16]. As a result, they can lead to misclassifications during the inference of DNNs [17-19]. On the other hand, during the inference phase, DNNs also usually exhibit a certain degree of misclassifications due to *intrinsic algorithmic inaccuracy* when confronted with previously unseen data samples. Consequently, the intrinsic algorithmic inaccuracies of DNN models can lead to erroneous outputs or wrong decisions in applications that rely on them. For example, even most recent classification models cannot yet achieve 100% accuracy on popular ImageNet dataset [20]. Given that DNNs already show some degree of misclassifications due to intrinsic algorithmic inaccuracies, some researchers have wondered the necessity of protecting them from the relatively less frequent transient hardware faults.

In our previous study [21], we answered 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 hypothesis posited that misclassifications arising from transient hardware faults pose a greater safety concern than those caused by intrinsic algorithmic inaccuracies. To validate our hypothesis, we chose object classes from two widely used datasets such as ImageNet [22] and CIFAR-100 [23] and categorized these object classes into different groups based on their similarity. For example, a bus was considered in the same group as a truck because those two are four-wheelers. Finally, we proposed a method to classify these groups into two *supergroup*s based on the safety-critical context regarding the DNN misclassifications. More specifically, we construct two supergroups: first one was comprised of groups where inter-group misclassification could lead to safety-critical consequences, and second one was consisted of groups where inter-group misclassification was not safety-critical. Finally, we proposed a methodology to measure the reliability of DNNs using our proposed safety-related metric—safety-critical misclassification (SCM) probability—based on the above supergroup formation. SCM probability measures the fraction of DNN misclassifications that lead to inter-group misclassifications for the first supergroup.

Figure 1 motivates the reason behind the above characterization of DNN misclassifications in terms of safety-critical behaviour. As in Figure 1a, if an AV detects a bus as a cab, it would take the necessary steps such as applying brakes, which are commonly required for other four-wheelers. However, as shown in Figure 1b, if the AV predicts the bus as an albatross, it 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. Other related works in this field of study mainly focus on SDCs, which do not particularly characterize the DNN misclassifications in terms of



(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

FIGURE 1 Impact of safety-critical misclassification in AVs.

safety concerns. Instead, they only look at whether a fault leads to misclassification or not. Neither do they distinguish the failures from those caused by DNN intrinsic inaccuracy.

In delineating the distinctions between this study and our earlier work by Laskar et al. [21], we note that our prior research did not delve into the intricate dynamics of how diverse supergroup formations and the varying vulnerability of distinct DNN regions contribute to the probability of SCMs. This article, serving as a substantially revised and expanded edition of our previous work, offers a more profound comprehension of the multifaceted factors influencing the likelihood of SCMs across different DNN models. To advance future research probing the root causes of DNN misclassifications, we provide a finer-grained analysis, scrutinizing various characteristics that contribute to safety-critical implications arising from misclassifications due to transient hardware faults and algorithmic inaccuracies. In this article, we present a comprehensive understanding of the followings which are not considered in our previous study.

- Given that SCM probability is influenced by how we categorize object classes into supergroups, we conduct additional experiments to investigate whether the SCM probability resulting from transient hardware faults still remains significantly higher than that arising from intrinsic algorithmic inaccuracies, considering different combination of supergroup formations based on application requirements (Sections 5.4 and 7.3).
- According to the inherent nature of soft errors, SDCs occur when some bit value of a tensor is flipped and hence affect the DNN output during the inference phase. Therefore, we also conduct rigorous analysis on potential bit positions of DNN tensors that are more susceptible to soft errors, leading to a higher SCM probability (Section 7.4).
- Any DNN model usually consists of many potential intermediate layers such as CONV, MaxPool, FullyConnected etc. We meticulously investigate whether all layers have similar SCM probability or certain layers exhibit higher SCM probabilities than the others (Section 7.5).
- Based on our observations, analysis and evaluation results, we also discuss potential mitigation strategies that can be integrated to develop more reliable DNN models, which are particularly effective in safety-critical applications such as AVs (Section 8).

By delving into these areas, we provide a more comprehensive understanding of the impact of soft errors on DNN misclassifications and their associated safety concerns. Our main contributions and results are as follows:

- To evaluate the effect of supergroup formation, we create multiple combinations of supergroups for safety-critical objects based on the analysis conducted in our previous work. From our analysis from fault injection experiments, we show that transient hardware faults often lead to significantly higher probabilities of safety-critical misclassifications compared to intrinsic algorithmic inaccuracy regardless of the formation of supergroups.
- In order to examine the sensitivity of soft errors in different bit positions leading to safety-critical misclassifications, we conduct comprehensive fault injections into all bit positions across eight DNN models based on both the ImageNet and CIFAR-100 datasets. Our results indicate that bitflips occurring in the most significant bits of DNN tensors predominantly contribute to the safety-critical misclassifications.
- Through fault injection experiments conducted on each layer of various DNN models, we profile the SCM probabilities across all layers of eight DNN models based on both the ImageNet and CIFAR-100 datasets. Our results show that the initial and final layers significantly exhibit higher SCM probabilities compared to that of other layers. Generally, there is a decreasing trend in SCM probability across the layers. However, specific layers such as *MaxPool* tend to have comparatively higher SCM probabilities than other layers such as *CONV*.

The remainder of this paper is organized as follows. Section 2 provides a detailed background on DNNs, transient hardware faults. In Section 3, we present a motivating example followed by our hypothesis. Section 3 is followed by Section 4, where we introduce our new fault injection framework. Section 5 discusses our methodology especially how we form different supergroup sets, which is followed by Sections 6 and 7. We present our experiment setup and evaluation results in those sections, respectively. Finally, we provide a discussion of overall implications and potential mitigation strategies in Section 8, related studies in Section 9, and conclusion in Section 10.

2 | BACKGROUND

2.1 | Deep learning neural networks

A deep neural network (DNN) consists of multiple computation layers [24] where higher level abstraction of the input data or a feature map is extracted to preserve each layer's unique and important information. In our study, we focused on convolutional neural networks (CNNs), which are widely used in various DNN applications. CNNs can have

anywhere from three to several hundreds of convolutional layers [1,2]. Each convolutional layer uses multiple kernels (or filters) to find visual features in the input and generate output feature maps. The results of these computations are stored as activations (ACTs) after passing through an activation function like ReLU. Also, there are a few fully connected layers (FC) following the convolutional layers, mainly used for classification purposes. Additional layers like pooling (POOL) and normalization (NORM) layers can also be added between the convolutional and fully connected layers. Once the DNN architecture is set, the network is trained using input data and associated weights. These weights, represented as connections between ACTs, are learned through a process called backpropagation. After training, the DNN is ready for image classification during the inferencing phase. During inferencing, the network takes digitized images as input and produces a list of potential matches, such as cars, pedestrians, or animals, along with confidence scores. Trained models are commonly deployed in various systems like autonomous vehicles for continuous inference tasks.

2.2 | Transient hardware faults

Soft errors, also known as transient hardware faults, are increasingly common in computer systems due to the shrinking size of transistors [15,25,26]. These errors originate at the circuit level and can propagate through different layers of the system. Eventually, they can reach the application level and affect program variables. When program variables become corrupted, they can alter the program output, resulting in silent data corruption (SDC).

The impact of soft errors in DNN systems can be severe, particularly in safety-critical systems where reliable operation is essential. To ensure specific reliability targets are met, it becomes necessary to mitigate these errors. For instance, in autonomous vehicles, correctly detecting a truck and applying the brakes in time can prevent collisions. However, if a soft error causes misclassification, such as identifying the truck as a bird, the braking action may not be executed promptly, especially when the vehicle is travelling at high speeds.

The issue of silent data corruption is of great importance, as it often leads to non-compliance with standards like ISO 26262 [27], which deals with the functional safety of road vehicles. According to ISO 26262, the System on Chip (SoC) responsible for DNN inferencing hardware in self-driving cars must maintain a soft error FIT[2] rate below 10 FIT [28], regardless of the DNN algorithm used and achieved accuracy. Considering that the DNN inferencing hardware occupies only a fraction of the overall SoC area, the FIT allowance for the DNN accelerator should be significantly lower than 10 in self-driving cars. However, without proper protection mechanisms, a standalone DNN system could potentially surpass the total FIT rate requirement for the entire SoC [18].

In this paper, the terms 'soft errors' and 'transient hardware faults' are used interchangeably to refer to the same phenomenon.

2.3 | DNN intrinsic algorithmic inaccuracy

There are multiple reasons why a deep neural network (DNN) can make mistakes when classifying objects. One of these reasons is the intrinsic algorithmic inaccuracy of the DNN itself. This inaccuracy is often reflected in the top-1 accuracy, which measures how often the DNN correctly identifies the most probable class for an object. Many studies in the field of machine learning focus on improving this top-1 accuracy to reduce the intrinsic algorithmic inaccuracy.

There are several factors that contribute to the intrinsic inaccuracy of DNN models. These include poor image quality of the target object or the training dataset, limitations in the underlying algorithm or architecture of the DNN model, and ineffective training methods, among others. It is important to note that every DNN inherently experiences algorithmic inaccuracy, regardless of whether there are hardware faults or not. Therefore, there are no DNN models that can achieve 100% accuracy in real-world object classification datasets like ImageNet [20].

2.4 | Fault model

In this study, our focus is on transient hardware faults that occur randomly during the execution of the deep neural network (DNN) inference phase. While faults can also impact the training phase, it is typically a one-time process that allows us to verify the results of the trained model. In contrast, DNN inference is performed frequently at runtime, making it more susceptible to these faults [17,18,29,30].

[2]Failure-in-Time rate: 1 FIT = 1 failure per 1 billion hours.

To simulate these faults, we directly inject them into the output values of the DNN layers. This fault injection approach aligns with previous research in the field [29,31,32]. We follow the 'one-fault-per-execution' method, assuming that soft errors are rare events compared to the full runtime of a program, which is a common assumption in the literature [32-36]. We inject faults into a single layer of the model, as faults occurring in multiple layers are relatively uncommon. Soft errors can manifest as single or multiple bit-flip errors at the software level. However, recent studies [31,37] have demonstrated that multiple bit-flip errors exhibit similar error propagation patterns and probabilities of Silent Data Corruption (SDC) as single bit-flip errors at the program level. Therefore, focusing on single bit-flip fault injection is sufficient for drawing conclusions about the error resilience of DNN models.

It's important to note that we assume these faults do not modify the state or structure of the model, such as changing the model's parameters [38,39]. Additionally, we do not consider faults in the model's inputs as they are external to TensorFlow [40] and fall outside the scope of our paper.

3 | MOTIVATION AND OUR HYPOTHESIS

3.1 | Motivation

DNNs have gained widespread adoption in various fields, encompassing diverse domains such as autonomous vehicles (AVs), healthcare, and space applications. Many of these applications are safety-critical [41-43], necessitating the utmost precision and reliability. Nevertheless, misclassification during inference remains a frequent issue in DNNs, and inherent algorithmic inaccuracies often persist throughout all DNN models. For instance, popular pre-trained DNN models applied to ImageNet typically exhibit accuracy ranges from 71.3% to 85.7% [20]. Furthermore, the ever-increasing rate of transient hardware faults in computer systems has further compounded the issue [13,14,44], by additionally increasing the misclassification probability. To address this concern, various protection techniques [29,45,46] have been proposed to safeguard DNNs against hardware faults. However, these protective measures come at a cost. At runtime, these techniques introduce notable performance overheads, which can reach up to 48%, even during fault-free executions [47].

In the past, most works in the area evaluate DNN resiliency based on the metrics such as SDC (e.g., wrong DNN output). While SDC is a widely accepted metric to gauge system resiliency, SDCs in DNNs may not always lead to safety violations or critical failures in safety-critical systems as discussed in Section 1. In this article, 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.

3.2 | 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 due to 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*. The following observations form the basis for our hypothesis: (1) Transient hardware faults are temporary and may not be easily detectable or replicable during testing, whereas intrinsic algorithmic inaccuracies are generally reproducible and can be systematically addressed through model updates and improvements. (2) A bit-flip caused by transient hardware faults may occur in higher bit positions of the DNN tensors, significantly affecting the tensor computations. This can ultimately lead to a substantial increase in DNN misclassifications, potentially surpassing the misclassification threshold introduced solely by intrinsic algorithmic inaccuracies.

4 | IMPROVEMENT OF TensorFI FAULT INJECTION FRAMEWORK

In this section, we first introduce the workflow of TensorFI [40] 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+ [48] for injecting transient hardware faults into non-sequential models.

4.1 | Workflow and limitation

In order to validate our hypothesis, we use TensorFI to conduct fault injection experiments. TensorFI[40] is a fault injection framework for TensorFlow applications. It has been utilized in recent studies [29,40] to simulate transient hardware faults during DNN model inferences. The reason for selecting TensorFI is its compatibility with fault injection experiments on DNN models implemented using the TensorFlow framework [49], which happens to be the most extensively embraced machine learning framework [50].

TensorFI operates on TensorFlow dataflow graphs which contains two main components. Firstly, there are operators responsible for computational units such as matrix multiplication. Secondly, there are tensors representing data units. Users have the flexibility to construct machine learning models using the built-in operators or create their own customized operators. TensorFI duplicates the TensorFlow graph while incorporating customized operators. These operators are designed not only to perform computations similar to standard operators but also to inject faults during the 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, and GoogleNet, which are widely used in numerous applications nowadays.

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.

4.2 | Support for non-sequential DNN models

In our previous study, we proposed TensorFI+ [48] 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 [40], 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 [40]. 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

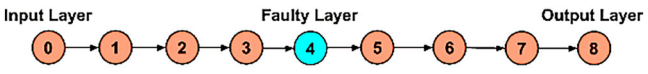


FIGURE 2 Fault injection in a sequential model.

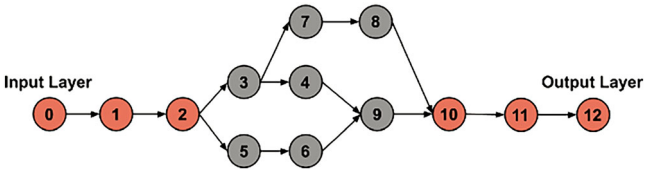


FIGURE 3 Example dependency graph with supernodes (marked as red)

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 cannot get output of layer 12 from layer 4 because layers 9 and 10 require value of layers 6 and 8, respectively, which are not computed yet.

The solution of this problem 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). Layers 0, 1, 2, 10, 11, and 12 are supernodes, and we can calculate output value of any subsequent node from these nodes.

In this way, if we 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 as follows: (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 layers 9 to 12. However, this approach fails because layer 9 requires output from both layers 4 and 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 layers 8 and 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.

4.3 | Performance improvement

To improve the runtime performance 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, layers 8 and 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 layers 8 and 9 when injecting faults on layer 2. To get outputs of layers 8 and 9, we need all of the following computations.

- Compute output of layers 3 and 5 using layer 2
- Compute output of layers 4 and 7 using layer 3

- Compute output of layer 6 using layer 5
- Compute output of layer 9 using layers 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., layers 8, 4, 6, and 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 present to prune more longest sequences. Thus, both of the above strategies provide significant performance improvement compared to naive recursive approach.

5 | METHODOLOGY

In this section, we first discuss the formulation of the formation of supergroups for DNN output classes based on safety metrics in the context of AVs from our previous study. We then present a new method to form different formation of supergroups for this article. We implement this new design and make our code publicly available.[3]

5.1 | Supergroup formation

To distinguish between safety-critical misclassifications and those that are not, we organize similar objects within a dataset into groups where misclassification raises no safety concern. In our study, we categorize objects based on the safety implications for autonomous vehicles (AVs). So, we divide the dataset into multiple groups where similar items are grouped together. For instance, people such as men, women, boys, and girls are placed in a group labelled as *people*, while various four-wheeled vehicles like sedans, SUVs, and trucks are categorized as *Four Wheelers*. As a result, misclassifications within these groups typically not safety-critical. For example, misclassifying a boy for a baby is not safety-critical because both scenarios require the AV to apply brakes. Likewise, misidentifying a truck as an SUV does not significantly impact safety since the AV actions remain largely similar.

However, misclassifications between different groups can be either safety-critical or not. Predicting a boy (from the *person* group) as a car (from the *vehicle* group) is safety-critical because AV responses may differ significantly. On the other hand, predicting an apple(from the *household objects* group) as a baseball (from the *tools* group), despite being from different groups, might not pose significant safety risks. Finally, since there can be numerous groups based on the variety of classes in a dataset, we classify all groups into two supergroups—Supergroup A (SG A) and Supergroup B (SG B). We make these groups such a way that misclassifications between groups in Supergroup A are considered safety-critical, while those in Supergroup B are generally not considered to pose significant safety concerns.

We focus on two popular datasets in this paper; they are ImageNet [22] and CIFAR-100 [23] datasets. Both datasets contain images reflecting common objects of diverse types. We first group similar objects of DNN output classes based on their characteristics and safety severity. We then merge these groups into different supergroups from the perspective of AV decisions in driving scenarios. 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 and small animals. Any misclassification within Supergroup B, even if they are from different groups, 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. In Tables 1 and 2, we list the corresponding groups and some example classes of each supergroups for both CIFAR-100 and ImageNet dataset, respectively.

[3] https://github.com/hasanur-rahman/investigating_soft_errors_impact_on_DNN

TABLE 1 CIFAR-100 SuperGroup example.

SuperGroup Name	Group name	Example classes
SuperGroup 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
SuperGroup 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

TABLE 2 ImageNet SuperGroup example.

SuperGroup Name	Group name	Example classes
SuperGroup 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
SuperGroup 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

5.1.1 | Forming 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 [51]. According to our proposed method of formulating supergroups, we combine these CIFAR-100 categories into the two supergroups (examples are in Table 1). 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 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*, and so forth. 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 and vice versa. 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* and *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.

5.1.2 | ImageNet supergroups

ImageNet dataset has a total of 1000 classes. There is an hierarchy of classes for the dataset in the documentation [52]. 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 shown in Table 2. Below, we provide the major rationale behind the supergrouping in ImageNet.

- Similar 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*, and *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*, and *small animals* are placed in Supergroup B because misclassification among these classes will not be safety-critical.

5.2 | 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.
- **Safety-critical misclassification (SCM) probability:** The SCM probability denotes the likelihood of misclassifications that raises safety concerns. Such misclassifications can occur during fault-free inference or due to transient hardware faults. In fault-free scenarios, if the original and predicted labels belong to different groups within Supergroup A, or if the original label is in Supergroup A and the predicted label is in Supergroup B, we classify these misclassifications as safety-critical. In these cases, SCMs are caused by inherent algorithmic inaccuracies within the DNN model. Similarly, during fault injection, if the fault-free prediction is from Supergroup A and the faulty prediction is in Supergroup B, or if both predictions belong to different groups within Supergroup A, we label these as safety-critical occurrences.
- **Non safety-critical misclassification (Non-SCM) probability:** Non safety-critical misclassifications refers to all the misclassifications that do not pose safety risks in fault-free or fault-injected experiments. This refers to scenarios where both predictions come from Supergroup B, or if the original label (fault-free prediction during fault injection experiments) is from Supergroup B and the predicted label (fault-injected prediction during fault injection experiments) is from Supergroup A. The Non-SCM probability serves as the complement to the SCM probability, collectively summing up to 100%.

5.3 | Fault injection methods

In our fault injection experiments, we randomly sample 10,000 images from each ImageNet [22] and GTSRB [53] dataset test sets. For both these datasets, the sampled dataset has around similar distribution of categories 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 [23] has 10,000 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 3000 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 [32-35]. For each fault injection run, we randomly sample one tensor from the randomly chosen layer of the model and randomly flip a bit.

5.4 | Method for forming different supergroup sets

In Section 5.1, we discuss our method of forming supergroups from various image classes of the CIFAR-100 and ImageNet datasets. This is the default supergroup formation that we used in our previous study [21]. However, in our previous study by Laskar et al. [21], although we carefully analysed and created the formation of the supergroups in the

TABLE 3 Formation of different supergroup sets.

GroupSet name	SuperGroup name	Groups
Supergroup Set 1	SuperGroup A	Big Animals, Emergency Vehicle, Two Wheeler, Four Wheeler, Geological Formation and Structure, Person
	SuperGroup B	Small Animals, Tools and Household Chores, Aerial Objects, Water Vehicles
Supergroup Set 2	SuperGroup A	Big Animals, Vehicles, Geological Formation and Structure, Person
	SuperGroup B	Small Animals, Tools and Household Chores, Aerial Objects, Water Vehicles
Supergroup Set 3	SuperGroup A	Domestic Animals, Ferocious Animals, Big Water Animals, Other Big Animals, Emergency Vehicle, Two Wheeler, Four Wheeler, Weapon, Geological Formation and Structure, Person
	SuperGroup B	Small Animals, Tools and Household Chores, Aerial Objects, Fish, Big Ship, Boat

context of safety critical behaviour, it is important to note that the configuration of these groups may vary depending on the specific requirements of the end-user. Therefore, researchers are further interested to find the answer of the following research question in finer granularity: Would the probabilities of SCM and non-SCM remain consistent with the findings in our previous study by Laskar et al. [21], especially when we alter the supergroup formation while still prioritizing safety critical perspectives?

In this article, we perform thorough analysis to answer the above question. We first provide an example method showing different ways to form the supergroups. For instance, in Section 5.1.2, we categorize road vehicles into three groups: *emergency vehicles*, *two-wheelers*, and *four-wheelers*. Since all of these groups are placed within *supergroup A*, any misclassification occurring among these groups is considered safety-critical. Nonetheless, if the end-user's requirements dictate that these groups should be regarded as a single entity, any misclassifications occurring among them would no longer be deemed as safety-critical. The reason is that those groups are now being treated as a single atomic group rather than three. This adjusted group formation may possibly affect the SCM and non-SCM probability.

Following the process of group formation from the above example, in this article, we investigate the SCM probability by creating three sets of *supergroup*s based on reorganizing the existing groups from the ImageNet dataset in different combinations. Note that ImageNet is chosen for this purpose due to its extensive collection of 1000 image classes, which provides greater flexibility to form different *supergroup*s. As the patterns observed in CIFAR-100 and ImageNet datasets are similar (shown in our previous study), we only focus on the ImageNet dataset to answer the above mentioned question.

We present different combinations of group formation in Table 3. Specifically, *Supergroup Set 1* denotes the default grouping we deployed throughout all experiments, unless stated otherwise. In fact, we used *Supergroup Set 2* for all evaluation results in our previous study. Moreover, *Supergroup Set 2* differs from *Supergroup Set 1* in that it merges *emergency vehicles*, *two-wheelers*, and *four-wheelers* into a single group named *Vehicles*.

As for *Supergroup Set 3*, we deploy a more exhaustive grouping approach by subdividing certain larger groups into smaller ones. For instance, *Big Animals* in *Supergroup Set 1* is divided into *Domestic Animals*, *Ferocious Animals*, *Big Water Animals*, and *Other Big Animals*. Similarly, *Weapon* is separated from *Tools and Household Chores* and placed in *Supergroup A*. Likewise, *Supergroup B* is also divided into multiple smaller groups.

6 | EXPERIMENTAL SETUP

6.1 | Datasets and DNN models

We use ImageNet [22], CIFAR-100 [23], and German Traffic Sign Recognition Benchmark (GTSRB) [53] 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[4] such as VGG, DenseNets, ResNets, MobileNets, and 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[5] module. The top-1 accuracy and masking probability of all these models are shown in Table 4.

[4] https://keras.io/api/applications/
[5] https://github.com/gmalivenko/pytorch2keras

TABLE 4 Top-1 accuracy and masking probability on initially correct and incorrect classifications.

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%

6.2 | Hardware

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

6.3 | Research questions

We conduct our evaluation by asking five 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?
- RQ3: Will there be a change in SCM and non-SCM probability if supergroup formation is altered?
- RQ4: Which bit positions in DNN tensors are more vulnerable in terms of SCM probability due to transient hardware faults?
- RQ5: What is the distribution of SCM and non-SCM probability across different DNN layers due to transient hardware faults?

7 | EVALUATION RESULTS

Our experiment results are organized and presented according to the research questions (RQs) we formulate.

7.1 | 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.

The results are shown in Table 4. We can see that the accuracy of CIFAR-100 dataset without any fault injections 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 4, 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.

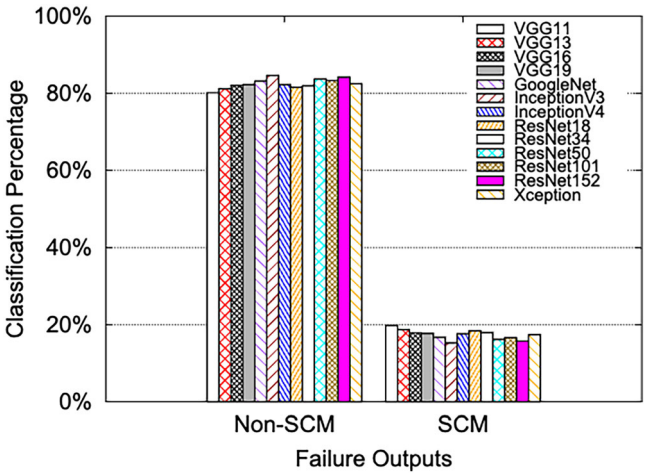


FIGURE 4 SCM and non-SCM due to intrinsic algorithmic inaccuracy in DNNs with CIFAR-100 dataset.

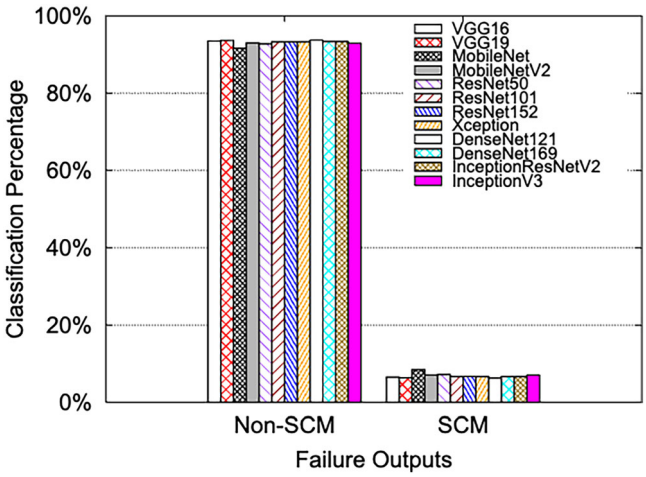


FIGURE 5 SCM and non-SCM due to intrinsic algorithmic inaccuracy in DNNs with ImageNet dataset.

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 A1 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.

7.2 | RQ2: Is DNN SCM probability caused by transient hardware faults different compared with that of intrinsic algorithmic inaccuracy?

In columns 4 and 5 of Table 4, we demonstrate 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 6a,b, 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 A2.

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 7a,b). 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. With the enhanced prediction accuracy of contemporary DNN models resulting in diminishing inherent algorithmic inaccuracies and a simultaneous increase in the soft error probability due to smaller chip sizes, we anticipate the trend of increasing SCM probability due to transient hardware faults will widen in the future.

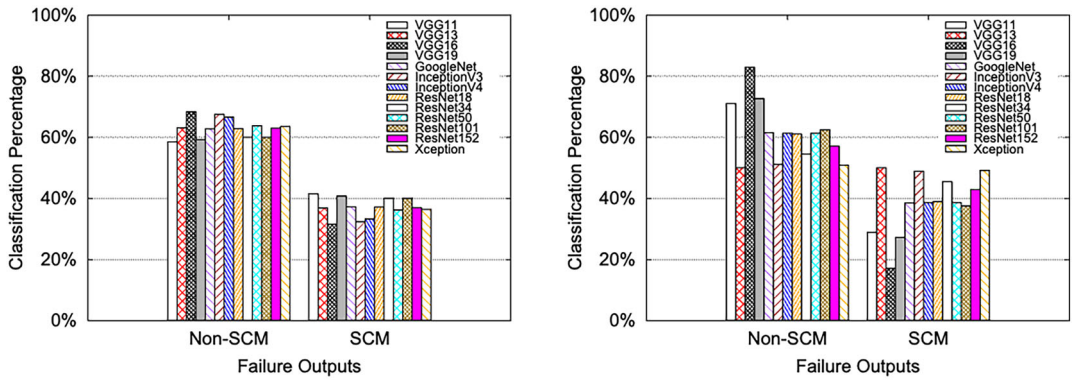


FIGURE 6 SCM and non-SCM probability due to transient hardware faults in DNNs with CIFAR-100 dataset.

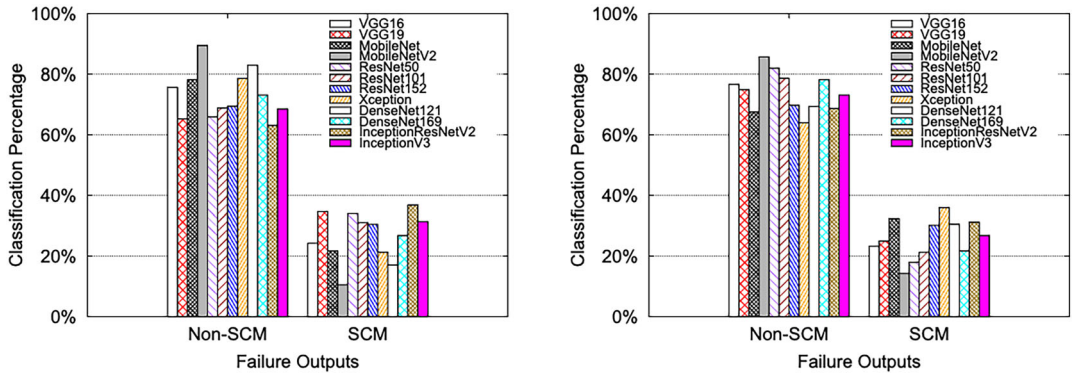


FIGURE 7 SCM and non-SCM probability due to transient hardware faults in DNNs with ImageNet dataset.

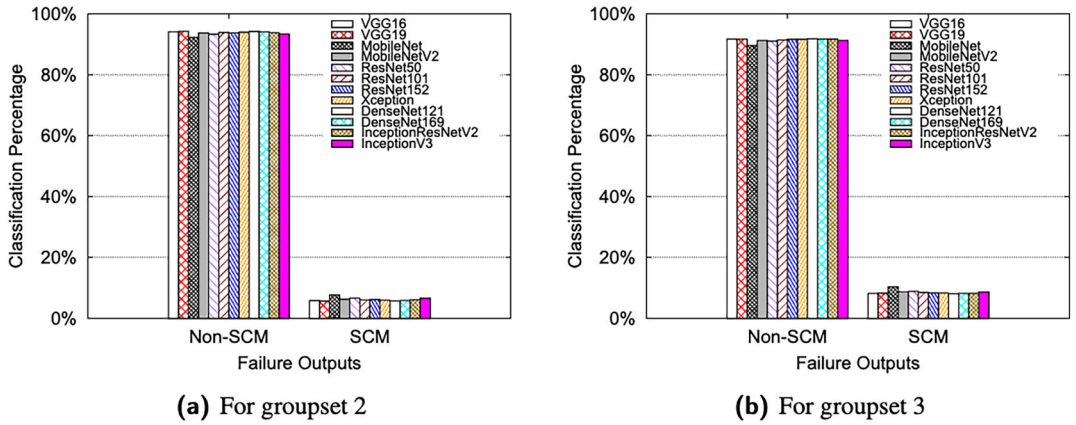


FIGURE 8 SCM and non-SCM due to intrinsic algorithmic inaccuracy for ImageNet dataset.

7.3 | RQ3: Will there be a change in SCM and non-SCM probability if supergroup formation is altered?

In Section 5.4, we outline our approach to address this question by creating three distinct sets of supergroups. In each case, we start with determining the SCM and non-SCM probabilities due to intrinsic algorithmic inaccuracies for the ImageNet dataset. Subsequently, we inject faults in both the initially correctly classified images and initially misclassified images. Finally, we again measure the SCM and non-SCM probabilities for those two cases respectively.

Figures 5 and 8a,b illustrate the SCM and non-SCM probabilities due to intrinsic algorithmic inaccuracy for *GroupSet 1*, *GroupSet 2*, and *GroupSet 3*, respectively based on ImageNet datasets. The SCM probability ranges from approximately 6% to 8% for *GroupSet 1*, 5% to 7% for *GroupSet 2*, and 8% to 10% for *GroupSet 3*. Across all cases, the average SCM probability remains below 10%. In GroupSet 3, as the number of groups within supergroup A increased, the SCM probability shows a little increase. However, in the context of intrinsic algorithmic inaccuracy across all cases, the non-SCM probability still significantly outweighs the SCM probability.

However, after introducing fault injections, the SCM probability increases substantially for all scenarios. Figures 7a, 9a, and 10a depict the SCM probability after fault injections for initially correctly classified images in GroupSet 1, GroupSet 2, and GroupSet 3, respectively. The SCM probability ranges from 10% to 37%, 5% to 40%, and 21% to 40% across all models. A similar trend is observed after injecting faults in initially misclassified images, where the SCM probability ranges from 14% to 37%, 13% to 37%, and 21% to 43% across all models for GroupSet 1, GroupSet 2, and GroupSet 3, respectively (shown in Figures 7b, 9b, and 10b). Therefore, in the context of safety-critical behaviour, we can say that SCM probability due to transient hardware faults always mark higher impact than that of intrinsic algorithmic inaccuracy even when the formation of the groups is changed.

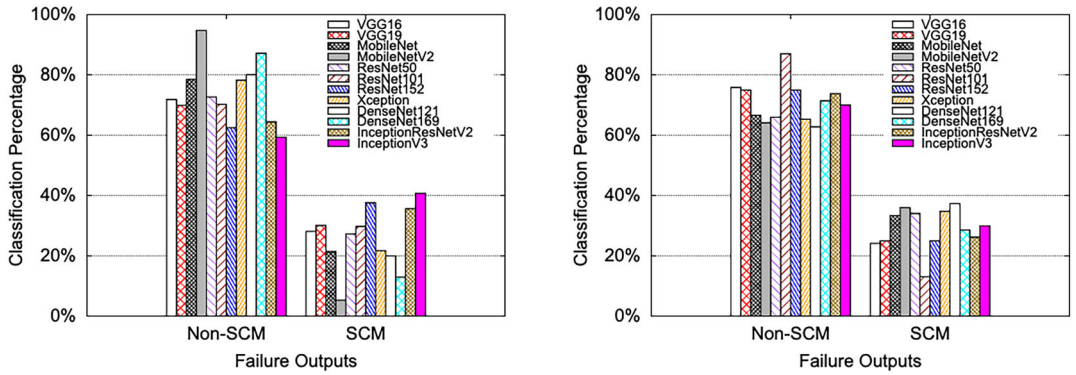


FIGURE 9 SCM and non-SCM due to transient hardware faults in DNNs for ImageNet dataset with groupset2

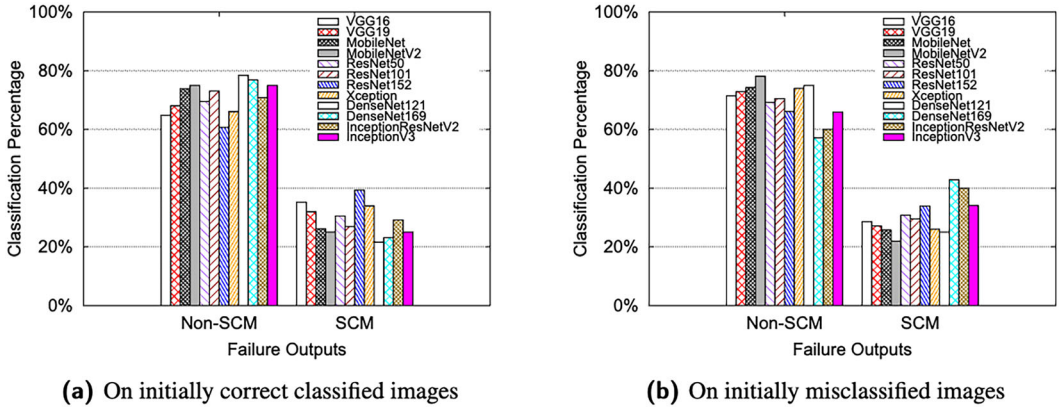


FIGURE 10 SCM and non-SCM due to transient hardware faults in DNNs for ImageNet dataset with groupset3

7.4 | RQ4: Which bit positions in DNN tensors are more vulnerable in terms of SCM probability due to transient hardware faults?

In this RQ, we aim to identify the most vulnerable bit positions in DNN tensors that are responsible for majority of the safety-critical misclassifications. To investigate this, we perform fault injection experiments on VGG16, VGG19, ResNet50, and Xception DNN models using both the ImageNet and CIFAR-100 datasets. As tensors in DNNs models are 32 bits long, we analyse the SCM vulnerabilities of each of the 32 bits due to the transient hardware faults. Specifically, we inject 1000 faults in each bit position of the models to assess their susceptibility to SCM. To inject faults at a particular bit position, the fault injection process involves randomly selecting a layer and a tensor, followed by injecting a fault (also known as a bit flip) into that specific bit position of that tensor. This fault injection method is consistent with existing studies [18,29].

Figure 11 illustrates the SCM probability for different bit locations of different DNN models based on the ImageNet dataset. We show results for four DNN models VGG16, VGG19, ResNet50, and Xception for the sake of brevity. Results with other DNN models based on ImageNet dataset show similar trend. Specifically, Figure 11 reveals that faults at higher-order bits exhibit greater SCM probability. For example, bit positions 26 to 30 for VGG16, bit positions 25 to 30 for VGG19, bit positions 26 to 30 for ResNet50, and bit positions 26 to 30 for Xception can identified as more sensitive to SCMs. In contrast, the SCM probability generally decreases for lower order bits. The key reason is that changing the value of higher order bits from 1 to 0 or vice versa (also known as bit flip) significantly affects the tensor values, leading to a potentially misclassified outcome, hence the higher SCM probability. However, bit flip on lower order bits usually exhibit minimal impact on the tensor values, mitigating the effect of soft errors.

We observe similar results based on the CIFAR-100 dataset, as shown in Figure 12. Across all models, bit positions 24 to 30 are most susceptible to SCMs, with bit 30 being the most vulnerable. Therefore, if selective protection is deployed to meet a reliability target, we emphasize that higher order bits in the DNN tensors should be prioritized first for the protection from soft errors than the lower order bits in terms of safety-critical context.

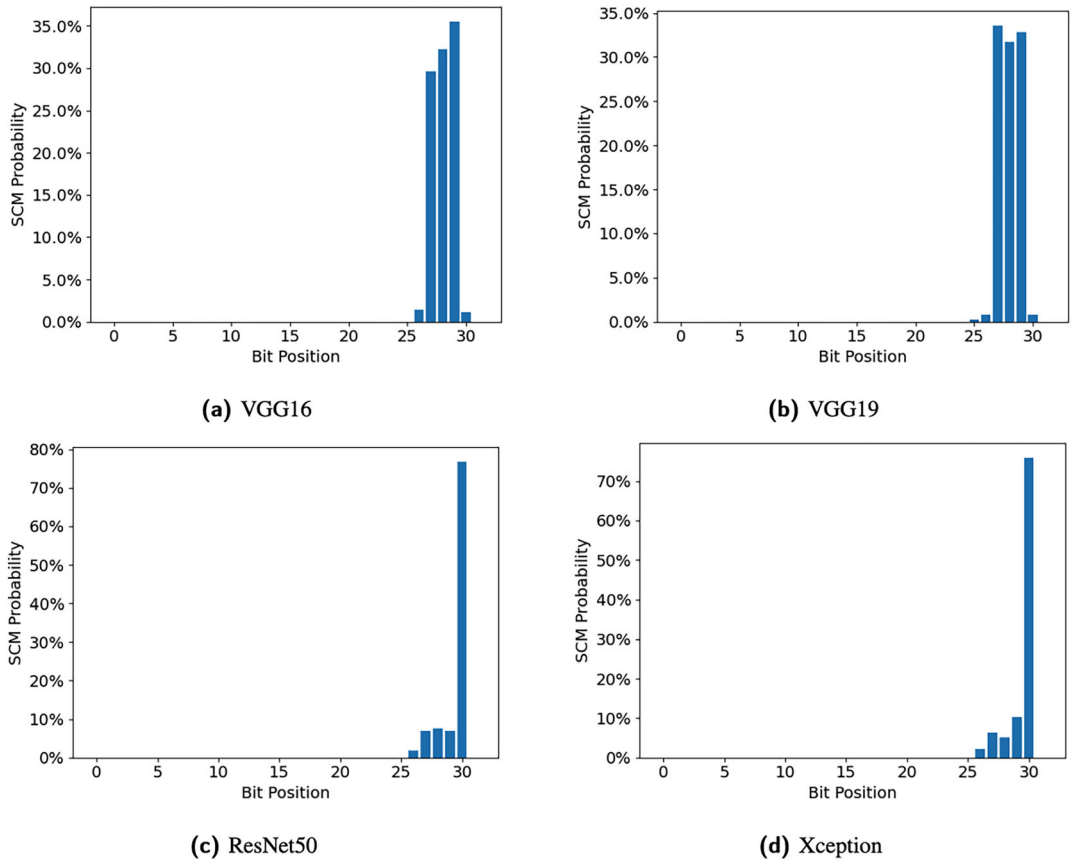


FIGURE 11 Distribution of SCM probabilities across different bit positions in DNN models based on ImageNet dataset. Here, a bit position refers to a specific region in the computational value of DNN tensors across all layers.

7.5 | RQ5: What is the distribution of SCM and non-SCM probability across different DNN layers due to transient hardware faults?

In this RQ, we want to determine the DNN layers that are most vulnerable to safety-critical misclassifications. To achieve that, we conduct fault injection experiments on VGG16, VGG19, ResNet50, and Xception models using both the ImageNet and CIFAR-100 datasets. We choose four models for the sake of brevity. Other DNN models show similar results. Specifically, we inject 1000 faults in each layer by randomly selecting a tensor of that layer and introducing a fault at a randomly chosen bit position. This fault injection method aligns with previous studies in this area [18,29].

Table 5 presents the number of layers of the four DNN models for both the ImageNet and CIFAR-100 datasets. For the ImageNet dataset, we use the pre-trained model from TensorFlow 2. However, as mentioned in Section 6.1, for the CIFAR-100 dataset, we convert the PyTorch model to TensorFlow 2 using the already available *pytorch2keras* module. Consequently, the number of layers for the same model under these two datasets in our experiment differ slightly. The key reason is that certain layers such as *Activation* and *ZeroPadding* are separated from the *CONV* layers during the conversion with *pytorch2keras* module, resulting in varying number of layers for the same models based on the two datasets.

Figure 13 illustrates the SCM probability for different layers of VGG16, VGG19, ResNet50, and Xception using the ImageNet dataset. In all cases, we observe that the initial layers exhibit a higher SCM probability compared to the subsequent layers. This can be attributed to the fact that faults occurring in the earlier layers have a greater likelihood of propagation, ultimately leading to a higher SCM probability at those layers. This observation is aligned with the trend of SDC propagation across layers [18]. Although the SCM probability decreases with the progress of layers, the last few layers still demonstrates a comparatively higher SCM probability than the intermediate layers do. This is because the last layers typically consist of *FullyConnected* layers, where errors have the greater flexibility in that they can propagate across different layer outputs immediately, resulting in a higher probability of output alteration and subsequently the SCM probability.

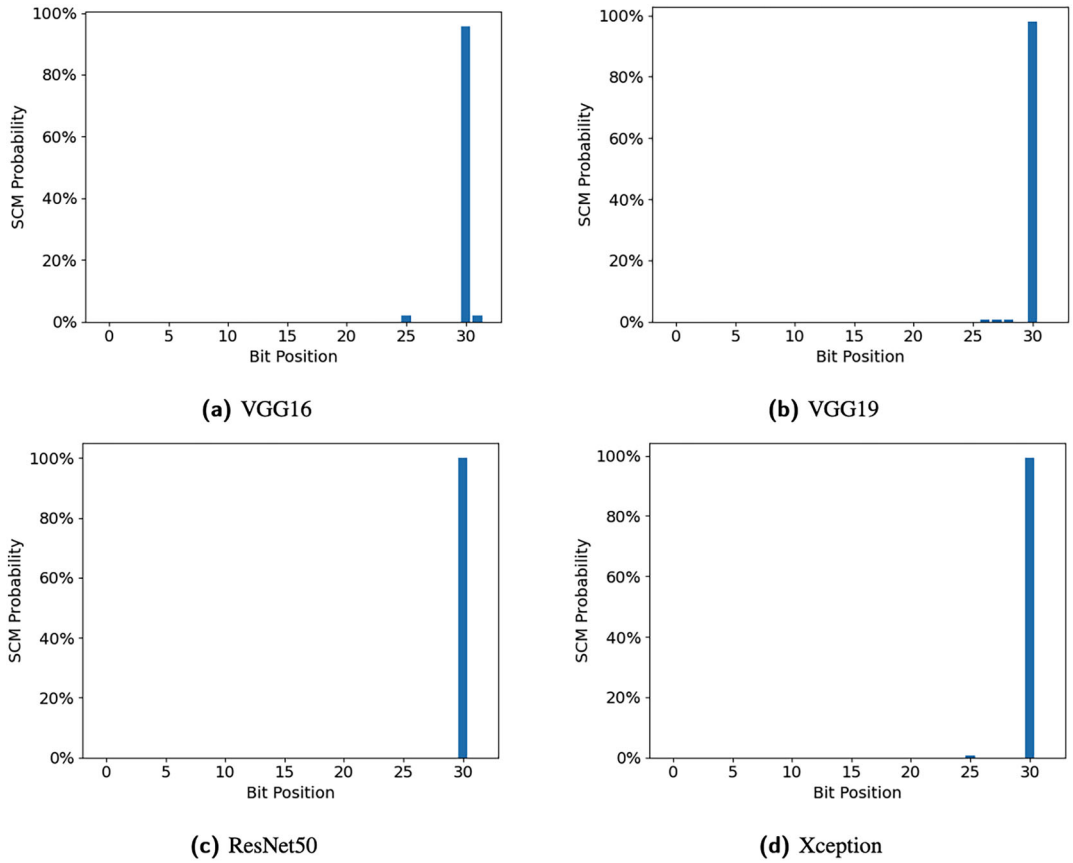


FIGURE 12 Distribution of SCM probabilities across different bit positions in DNN models based on CIFAR-100 dataset. Here, a bit position refers to a specific region in the computational value of DNN tensors across all layers.

TABLE 5 Layer numbers for four DNN models of ImageNet and CIFAR-100 dataset.

	ImageNet	CIFAR-100
VGG16	22	51
VGG19	25	60
ResNet50	176	141
Xception	133	170

As an example, we provide the layer information for the VGG16 model deployed in our experiment in Table 6. Comparing the layers with the results from Figure 13, although the SCM probability for *CONV* layers demonstrates a decreasing trend, the *MaxPool* layers following the CONV layers exhibit relatively higher SCM probability. This is due to the fact that *MaxPool* layers have the ability to completely discard or overwrite immediate layers' outputs, making introduced errors in these layers more likely to have significant impact on the overall inference process and hence on the SCM probability.

In Figure 14, we also present the SCM probability of different layers for the four DNN models based on the CIFAR-100 dataset. Similar to the findings based on the ImageNet datasets, results based on the CIFAR-100 dataset follows a comparable pattern. As previously mentioned, the models in the CIFAR-100 dataset possess some additional layers such as *Activation* and *ZeroPadding*. Since activation is applied after the convolution operation, errors in these layers significantly alter the outputs and contribute to the increase in the SCM probability. Additionally, likewise as results based on ImageNet datsets, DNN models with CIFAR-100 dataset also exhibit higher SCM probability in the initial and final layers, while the probability is relatively lower in the other layers.

Based on our above results with both CIFAR-100 and ImageNet datasets, we can say that SCM probabilities are relatively higher in the initial and final layers of the DNN models.

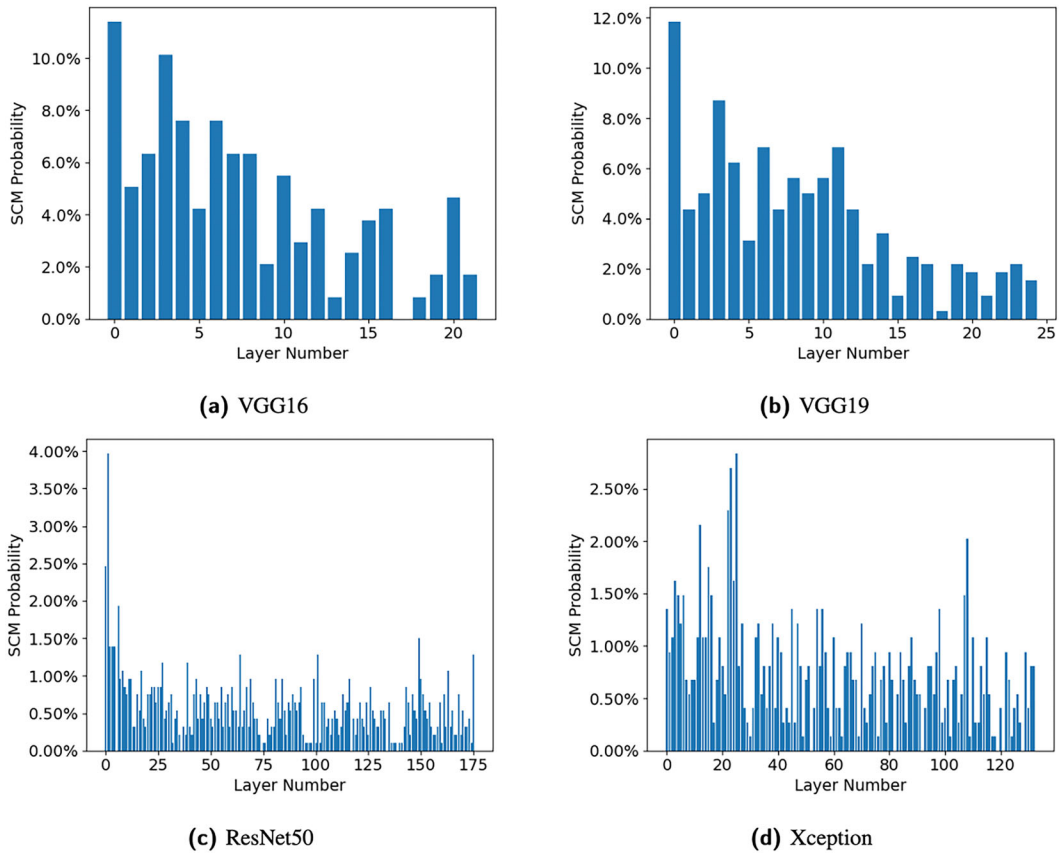


FIGURE 13 SCM probability by injecting faults in different layers of ImageNet dataset.

TABLE 6 VGG16 model layer details for ImageNet dataset.

Layer 0	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	Layer 7	Layer 8	Layer 9	Layer 10
Input	CONV	CONV	MaxPool	CONV	CONV	MaxPool	CONV	CONV	CONV	MaxPool
Layer 11	Layer 12	Layer 13	Layer 14	Layer 15	Layer 16	Layer 17	Layer 18	Layer 19	Layer 20	Layer 21
CONV	CONV	CONV	MaxPool	CONV	CONV	CONV	MaxPool	Flatten	Dense	Dense

8 | DISCUSSION BASED ON EVALUATION RESULTS

8.1 | Overall summary

In summary, the research investigates the safety implications of DNNs especially deployed in AVs by addressing five key research questions. The first question delves into the impact of intrinsic algorithmic inaccuracies on DNNs, revealing that around 20% to 30% of images across various models and datasets are misclassified due to this factor. Notably, the majority of these misclassifications are non-SCMs. Moving to the second question, the study compares the probability of SCMs resulting from transient hardware faults with those due to intrinsic algorithmic inaccuracy. The findings indicate that transient hardware faults significantly elevate the SCM probability, emphasizing their potential to cause safety-related concerns in AVs.

The third research question explores the influence of supergroup formation on SCM and non-SCM probabilities. The study demonstrates that, regardless of changes in group formation, SCM probability due to transient hardware faults usually surpasses that of intrinsic algorithmic inaccuracy, reinforcing the prominence of hardware faults in safety-critical scenarios. The fourth question identifies vulnerable bit positions in DNN tensors concerning SCM probability due to transient hardware faults. The research reveals that higher order bits are more susceptible to soft errors, as a bit-flip in these positions can lead to substantial changes in tensor values, increasing the likelihood of SCMs.

Finally, the fifth question investigates the distribution of SCM and non-SCM probabilities across different DNN layers due to transient hardware faults. The study indicates that initial layers exhibit a higher SCM probability,

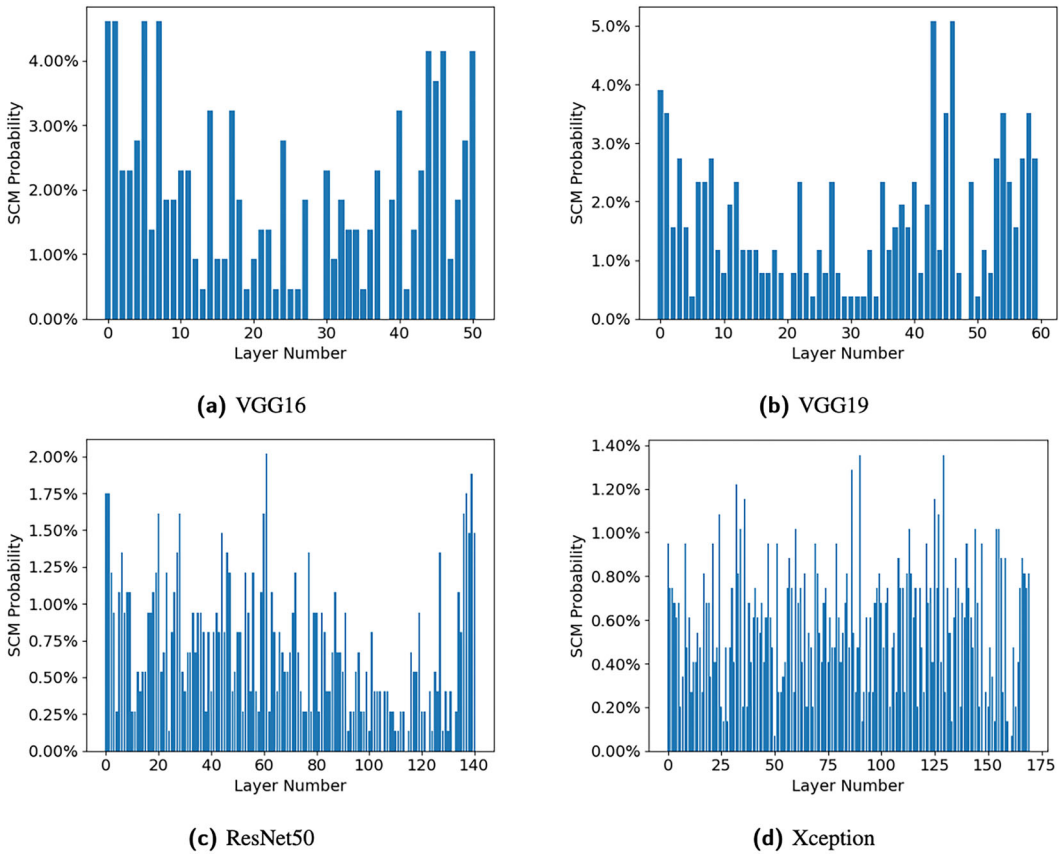


FIGURE 14 SCM probability by injecting faults in different layers of CIFAR-100 dataset.

attributed to the greater likelihood of error propagation. While SCM probability generally decreases across layers, the last few layers, often composed of FullyConnected layers, still demonstrate a comparatively higher SCM probability due to their potential for immediate error propagation across different layer outputs.

Overall, the research underscores the significance of transient hardware faults in driving safety-related concerns in AVs, even as intrinsic algorithmic accuracy improves. The findings provide valuable insights into the vulnerabilities of DNNs, emphasizing the need for robust fault-tolerant strategies in the development and deployment of DNNs in safety-critical applications such as AVs.

8.2 | Potential mitigation strategies

Our evaluation results suggest several potential mitigation strategies against SCMs in deep neural networks (DNNs). First, considering the substantial impact of transient hardware faults on SCM probability, implementing fault-tolerant designs and mechanisms becomes crucial. Focusing on the identified vulnerable bit positions, developers can explore error-correcting codes, redundant computations, or hardware-level protections to mitigate the effects of soft errors. This may involve introducing redundancy in critical areas, employing error-detection and correction techniques, or utilizing fault-tolerant hardware components. Additionally, understanding the distribution of SCM probabilities across different layers suggests a targeted approach. Prioritizing the improvement of fault tolerance in the initial layers, where SCM probability is higher, can be an effective strategy. This could involve employing more robust error-handling mechanisms, such as redundancy or dynamic reconfiguration, in these critical layers. Furthermore, the consistent elevation of SCM probability due to transient hardware faults, irrespective of supergroup formations, highlights the need for system-wide fault-tolerant strategies rather than relying solely on specific group configurations. Developers should consider comprehensive fault injection testing across various scenarios to ensure robustness under different conditions. In conclusion, the mitigation strategy against SCM probability in DNNs should involve a multifaceted approach. This includes ongoing efforts to improve intrinsic algorithmic accuracy, targeted enhancements in fault tolerance at vulnerable bit positions, and system-wide fault-tolerant designs that account for variations in supergroup formations.

Integrating these strategies would contribute to the development of more reliable and safer DNNs, particularly in safety-critical applications such as AVs.

9 | RELATED WORK

There have been many works that studies DNN resilience and accuracy. Tian et al. [54] used transformation matrices (e.g., scale, rotate the image) to automatically generate corner case images to trigger unexpected behaviours of the model. Ma et al. [39] designed a mutation framework to fuzz DNN models, the technique was then used to evaluate the test data quality. These studies focused on improving DNN accuracy.

Rubaiyat et al. [55] 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] built a fault injector to evaluate transient hardware faults in DNN applications and studied their resilience.

Chen et al. proposed BinFI [29], 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 of the above studies distinguished between DNN failures caused by intrinsic algorithmic inaccuracy and transient hardware faults in terms of safety-critical behaviour.

10 | CONCLUSION AND FUTURE WORK

Modern computing systems such as AVs require the underlying deployed DNN models to correctly classify the target object; otherwise, incorrect classification may have severe safety consequences. In this article, we investigate how the transient hardware faults contribute to the misclassification of DNN models based on safety-critical metrics compared to that by intrinsic algorithmic inaccuracy. We show that misclassifications due to transient hardware faults have a greater impact on safety-critical consequences than those due to intrinsic algorithmic inaccuracy. We also investigate whether altering the initial target image class groupings still yield similar results. Our evaluation on different formation of grouping the image classes in the safety-critical context demonstrates that safety-critical misclassification probability due to soft errors yield a consistent increase of 4×, on average, than that of intrinsic inaccuracy regardless of group formation. In this article, we also perform a comprehensive study on the potential bit locations and DNN layers responsible for the majority of safety-critical misclassifications. Our results show that higher order bit positions in DNN tensors as well as initial and final DNN layers are the most critical regions for the cause of safety-critical misclassifications regardless of DNN models under consideration.

In future work, we plan to fully automate the process of supergroup formation by considering the safety-critical behaviour and the end-user requirements.

DATA AVAILABILITY STATEMENT

We share our code artefacts in the following url: https://github.com/hasanur-rahman/investigating_soft_errors_impact_on_DNN.

ORCID


Md Hasanur Rahman https://orcid.org/0009-0002-5540-8751

REFERENCES

1. He K., Zhang X., Ren S., Sun J.. Deep residual learning for image recognition, 2016; 770–8.
2. Krizhevsky A., Sutskever I., Hinton G.E.. 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, Pereira F., Burges C. J. C., Bottou L., Weinberger K. Q. (eds)., vol. 25 Curran Associates, Inc.
3. Silver D., Huang A., Maddison C., Guez A., Sifre L., Driessche G., et al. Mastering the game of go with deep neural networks and tree search. Nature. 2016;529:484–9.
4. Simonyan K., Zisserman A.. Very deep convolutional networks for large-scale image recognition, 2014. arXiv 1409.1556.
5. Chen T., Du Z., Sun N., Wang J., Wu C., Chen Y., Temam O.. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Association for Computing Machinery*, ASPLOS '14: New York, NY, USA, 2014; 269–84.
6. Chen Y., Luo T., Liu S., Zhang S., He L., Wang J., et al. DadianNao: a machine-learning supercomputer. In *IEEE Computer Society*: USA, 2014; 609–22.
7. Fernandes F., Weigel L., Jung C., Navaux P., Carro L., Rech P.. Evaluation of histogram of oriented gradients soft errors criticality for automotive applications. ACM Transactions on Architecture and Code Optimization (TACO). 2016;13(4):1–25. https://doi.org/10.1145/2998573
8. Bojarski M., Testa D., Dworakowski D., Firner B., Flepp B., Goyal P., et al. End to end learning for self-driving cars, 2016.
9. Esteva A., Kuprel B., Novoa R., Ko J., Swetter S., Blau H., Thrun S.. Dermatologist-level classification of skin cancer with deep neural networks. Nature. 2017;542:115–8. https://doi.org/10.1038/nature21056

10. Rajpurkar P., Hannun A., Haghpanahi M., Bourn C., Ng A.. Cardiologist-level arrhythmia detection with convolutional neural networks, 2017.
11. Yoon H.-J., Ramanathan A., Tourassi G.. Multi-task deep neural networks for automated extraction of primary site and laterality information from cancer pathology reports, 2017. 195–204.
12. Xiong Z., Stiles M., Zhao J.. Robust ecg signal classification for the detection of atrial fibrillation using novel neural networks, 2017.
13. DeBardeleben N., Laros J., Daly J.T., Scott S.L., Engelmann C., Harrod B.. High-end computing resilience: Analysis of issues facing the hec community and path-forward for research and development, 2009. Whitepaper, Dec.
14. Schroeder B., Gibson G.. Understanding failures in petascale computers. Journal of Physics: Conference Series. 2007;78:12022. https://doi.org/10.1088/1742-6596/78/1/012022
15. Borkar S.. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. Ieee Micro. 2005;25(6):10–6. https://doi.org/10.1109/MM.2005.110
16. Constantinescu C.. Intermittent faults and effects on reliability of integrated circuits. Reliability and Maintainability Symposium. 2008;0:370–4. https://doi.org/10.1109/RAMS.2008.4925824
17. Reagen B., Gupta U., Pentecost L., Whatmough P., Lee S., Mulholland N., et al. Ares: a framework for quantifying the resilience of deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference*, 2018; 1–6.
18. Li G., Hari S. K.S., Sullivan M., Tsai T., Pattabiraman K., Emer J., Keckler S.W.. 2017. 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, SC '17 Association for Computing Machinery: New York, NY, USA.
19. dos Santos F., Lunardi C., Oliveira D., Libano F., Rech P.. Reliability evaluation of mixed-precision architectures. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019; 238–49.
20. Keras applications. https://keras.io/api/applications/
21. Laskar S., Rahman M.H., Zhang B., Li G.. Characterizing deep learning neural network failures between algorithmic inaccuracy and transient hardware faults. In *2022 ieee 27th pacific rim international symposium on dependable computing (PRDC)*, IEEE, 2022; 54–67.
22. Deng J., Dong W., Socher R., Li L.-J., Li K., Fei-Fei L.. Imagenet: a large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009; 248–55.
23. Krizhevsky A.. Learning multiple layers of features from tiny images, 2009.
24. LeCun Y., Kavukcuoglu K., Farabet C.. Convolutional networks and applications in vision. In *Proceedings of 2010 ieee international symposium on circuits and systems*, 2010; 253–6.
25. Mukherjee S.S., Emer J., Reinhardt S.K.. The soft error problem: an architectural perspective. In *11th International Symposium on High-Performance Computer Architecture*, IEEE, 2005; 243–7.
26. Feng S., Gupta S., Ansari A., Mahlke S.. Shoestring: probabilistic soft error reliability on the cheap. ACM SIGARCH Computer Architecture News. 2010;38(1):385–96.
27. Safety standard 2016. Iso-26262: road vehicles functional safety, 2016. Retrieved Oct. 2016 from https://en.wikipedia.org/wiki/ISO_26262
28. Autonomous car facts 2016. keynote: autonomous car a new driver for resilient computing and design-for-test, 2016. Retrieved Oct. 2016 from https://nepp.nasa.gov/workshops/etw2016/talks/15WED/20160615-0930-Autonomous_Saxena-Nirmal-Saxena-Rec2016Jun16-nasaNEPP.pdf
29. Chen Z., Li G., Pattabiraman K., DeBardeleben N.. Binfi: an efficient fault injector for safety-critical machine learning systems. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 2019; 1–23.
30. Sabbagh M., Gongye C., Fei Y., Wang Y.. Evaluating fault resiliency of compressed deep neural networks. In *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, 2019; 1–7.
31. Chang C.-K., Lym S., Kelly N., Sullivan M.B., Erez M.. Evaluating and accelerating high-fidelity error injection for HPC. In *Sc18: International conference for high performance computing, networking, storage and analysis*, 2018; 577–89.
32. Li G., Pattabiraman K., Hari S. K.S., Sullivan M., Tsai T.. Modeling soft-error propagation in programs. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018; 27–38.
33. Ashraf R.A., Gioiosa R., Kestor G., DeMara R.F., Cher C.-Y., Bose P.. 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; 1–12.
34. Fang B., Pattabiraman K., Ripeanu M., Gurumurthi S.. 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; 221–30.
35. Wei J., Thomas A., Li G., Pattabiraman K.. 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; 375–82.
36. Rahman M.H., Shamji A., Guo S., Li G.. PEPPA-X: finding program test inputs to bound silent data corruption vulnerability in HPC applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021; 1–13.
37. Sangchoolie B., Pattabiraman K., Karlsson J.. 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; 97–108.
38. Jha S., Banerjee S.S., Cyriac J., Kalbarczyk Z.T., Iyer R.K.. Avfi: fault injection for autonomous vehicles. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2018; 55–6.
39. Ma L., Zhang F., Sun J., Xue M., Li B., Juefei-Xu F., et al. Deepmutation: mutation testing of deep learning systems, 2018. https://doi.org/10.48550/ARXIV.1805.05206
40. Li G., Pattabiraman K., DeBardeleben N.. Tensorfi: a configurable fault injector for tensorflow applications. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, IEEE, 2018; 313–20.
41. Fridman L., Brown D.E., Glazer M., Angell W., Dodd S., Jenik B., et al. MIT advanced vehicle technology study: large-scale naturalistic driving study of driver behavior and interaction with automation. IEEE Access. 2019;7:102021–38. https://doi.org/10.1109/ACCESS.2019.2926040
42. Ozturk T., Talo M., Yildirim E.A., Baloglu U.B., Yildirim O., Rajendra Acharya U.. Automated detection of COVID-19 cases using deep neural networks with x-ray images. Computers in Biology and Medicine. 2020;121:103792. https://doi.org/10.1016/j.compbiomed.2020.103792
43. Kothari V., Liberis E., Lane N.D.. The final frontier: deep learning in space, 2020.
44. Snir M., Wisniewski R.W., Abraham J.A., Adve S.V., Bagchi S., Balaji P., et al. Addressing failures in exascale computing. International Journal of High Performance Computing Applications. 2014;28(2):129–73. https://doi.org/10.1177/1094342014522573
45. Hari S. K.S., Adve S.V., Naeimi H.. Low-cost program-level detectors for reducing silent data corruptions. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, 2012; 1–12.

46. Li G., Lu Q., Pattabiraman K.. Fine-grained characterization of faults causing long latency crashes in programs. In *2015 45th annual ieeelifip international conference on dependable systems and networks*, 2015; 450–61.
47. Mahmoud A., Hari S. K.S., Fletcher C.W., Adve S.V., Sakr C., Shanbhag N., et al. Optimizing selective protection for CNN resilience. In *32nd IEEE International symposium on Software Reliability Engineering, ISSRE 2021*, IEEE Computer Society, 2021; 127–38.
48. Laskar S., Rahman M.H., Li G.. Tensorfi+: a scalable fault injection framework for modern deep learning neural networks. In *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, IEEE, 2022; 246–51.
49. Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., et al. 2016. Tensorflow: a system for large-scale machine learning. In Proceedings of the 12th usenix conference on operating systems design and implementation USENIX Association: USA; 265–83.
50. Tensorflow popularity. https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a
51. Cifar-100 dataset description. https://www.cs.toronto.edu/~kriz/cifar.html
52. Imagenet hiererchy. https://observablehq.com/@mbostock/imagenet-hierarchy
53. Houben S., Stallkamp J., Salmen J., Schlipsing M., Igel C.. Detection of traffic signs in real-world images: the German Traffic Sign Detection Benchmark, *International joint conference on neural networks*, 2013.
54. Tian Y., Pei K., Jana S., Ray B.. 2018. Deeptest: automated testing of deep-neural-network-driven autonomous cars. In Proceedings of the 40th international conference on software engineering, ICSE '18 Association for Computing Machinery: New York, NY, USA; 303–14. https://doi.org/10.1145/3180155.3180220
55. Rubaiyat A. H.M., Qin Y., Alemzadeh H.. Experimental resilience assessment of an open-source driving agent, *2018 IEEE 23rd pacific rim international symposium on dependable computing (PRDC)*. IEEE, 2018. https://doi.org/10.1109/prdc.2018.00016

How to cite this article: Rahman MH, Laskar S, Li G. Investigating the impact of transient hardware faults on deep learning neural network inference. Softw Test Verif Reliab. 2024. e1873. https://doi.org/10.1002/stvr.1873

APPENDIX A

TABLE A1 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 theatre, notebook
		passenger car	2	gas pump, fire engine	3	electric locomotive, steam locomotive

TABLE A2 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)	-