IEEE Open Journal of the Computer Society

Received 9 February 2024; revised 14 April 2024; accepted 25 April 2024. Date of publication 14 May 2024; date of current version 9 January 2025. The review of this article was arranged by Associate Editor W. Liu.

Digital Object Identifier 10.1109/OJCS.2024.3400696

ALBERTA: ALgorithm-Based Error Resilience in Transformer Architectures

HAOXUAN LIU, VASU SINGH, MICHAŁ FILIPIUK, AND SIVA KUMAR SASTRY HARI (Senior Member, IEEE)

NVIDIA, Santa Clara, CA 95051 USA

CORRESPONDING AUTHOR: HAOXUAN LIU (e-mail: steveliu@nvidia.com).

ABSTRACT Vision Transformers are being increasingly deployed in safety-critical applications that demand high reliability. Ensuring the correct execution of these models in GPUs is critical, despite the potential for transient hardware errors. We propose a novel algorithm-based resilience framework called ALBERTA that allows us to perform end-to-end resilience analysis and protection of transformer-based architectures. First, our work develops an efficient process of computing and ranking the resilience of transformers layers. Due to the large size of transformer models, applying traditional network redundancy to a subset of the most vulnerable layers provides high error coverage albeit with impractically high overhead. We address this shortcoming by providing a software-directed, checksum-based error detection technique aimed at protecting the most vulnerable general matrix multiply (GEMM) layers in the transformer models that use either floating-point or integer arithmetic. Results show that our approach achieves over 99% coverage for errors (single bit-flip fault model) that result in a mismatch with <0.2% and <0.01% computation and memory overheads, respectively. Lastly, we present the applicability of our framework in various modern GPU architectures under different numerical precisions. We introduce an efficient self-correction mechanism for resolving erroneous detection with an average of less than 2% overhead per error.

INDEX TERMS Correction, detection, protection, resilience, robustness, safety, transformers.

I. INTRODUCTION

Deep learning has become the key technology in several safety-critical fields like autonomous vehicles (AVs) [18], medical image analysis [34], and drug design [3]. While convolutional neural networks (CNNs) were the predominant architecture of choice for deep learning in the last decade in computer vision tasks, the focus is rapidly shifting to transformer-based networks. Following recent breakthroughs in self-attention and model training optimizations, vision transformers (ViTs) have demonstrated their effectiveness in adapting to a wide variety of vision problems ranging from image classification [11], object detection [24], image generation [38], and semantic segmentation [21].

In most ViT-based architectures, latent dependencies between embedded image tokens are learned in parallel via the inter-/intra-token calculations computed by the self-attention modules. Processing an image in such a fashion remains computationally expensive today, primarily due to the high computation and memory cost of large matrix operations.

It is important to understand the effect of transient hardware errors on various components of the transformer architecture. Such understanding can provide insights to develop resilient transformers without high computational overhead for safety-critical applications.

Prior work on resilience in deep neural networks utilize different techniques like range-based bounds checking [6], redundant layer computation [26], and algorithm-based fault tolerance (ABFT) techniques [17]. These methods are either not sufficient or pose significant overhead for vision transformers. We investigate the vulnerability of individual layers of transformers and whether we can provide high resilience in vision transformer models with minimal overhead.

In this article, we present a framework called ALBERTA (ALgorithm-Based Error Resilience in Transformer Architectures) for end-to-end resilience analysis and enhancement framework for vision transformers. While ALBERTA can be employed using any deep learning framework, we prototype it using PyTorch [29]. To the best of our knowledge, this is

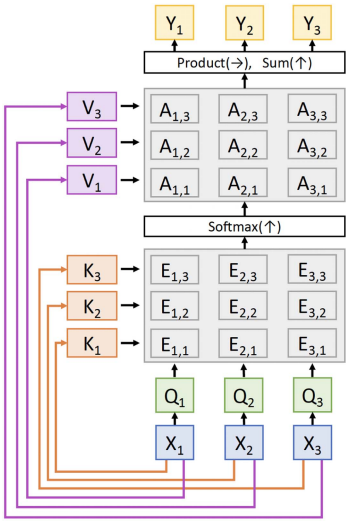


FIGURE 1. Overview of the self-attention module. Source: [20].

the first article to perform a detailed resilience analysis of vision transformer models (an important class of perception models) using floating-point and integer data types, provide novel insights into the vulnerable components, and propose a low-cost protection technique to significantly increase its resilience with ultra-low overheads. The following are the main contributions of this article:

- We implement the vulnerability analysis in ALBERTA using an error injector module in PyTorch that allows the user to import custom datasets and select desired injection modes for transformer model analysis. The set of injection locations supported by the injector module include layer input, output, and weights, while the set of enabled injection types include single bit flip for integers, single bit in the exponent or mantissa for floating-point values, and replacement with random or user-specified value.
- We investigate the vulnerability of different layers in a transformer model and gather interesting insights: (1) Our chosen transformer architecture exhibits a significant jump in resilience from the third to eighth multi-head attention blocks (out of twelve blocks), which is a counter-intuitive pattern compared to the findings on CNNs that earlier layers are resilient [22], [26]. (2) Injections in the prediction head of the transformer result in the highest probability of mismatch in the network output. (3) Selective layer duplication provides coverage of about 90% at an overhead of more than 30%.
- The error-resilience algorithm in ALBERTA is also implemented as a Pytorch module for GPUs. We implement optimized checksum computation for floating-point (FP) based models. It is well-known that such checksums cannot be bitwise precise due to FP non-associativity. So, the solutions have to rely on conservative thresholds for error checking, which can compromise error coverage. We present a methodology that derives the GPU kernel-specific thresholds such that high error coverage is maintained.
- Our proposed error-resilience algorithm achieves over 99% coverage of all injected errors that result in network misclassification at $<0.2\%$ computation overhead and $<0.01\%$ memory overhead. While false detections are rare, we introduce an efficient self-correction mechanism with an average overhead of $<2\%$ to resolve each erroneous detection. These results are based on a single bit-flip fault model.

The article is organized as follows. Section II formalizes our problem statement and related work. Section III describes the ALBERTA framework and Section IV presents the results we obtain on different vision transformers using ALBERTA. Section VI concludes the article and discusses further directions for future work.

II. PROBLEM STATEMENT

A. BACKGROUND

Vision Transformers: Transformer-based models [36] first gathered attention in the context of natural language processing. After their huge success, they started to gain traction in different areas [14], [39], computer vision including. The Vision Transformer (ViT) was one of the first transformer models for vision: it is pretrained (supervised) on the ImageNet-21 k dataset at a resolution of 224x224 pixels, and later finetuned on ImageNet [8] at the same resolution [11]. Images are presented to the model as a sequence of 16x16 fixed-size patches that are linearly and positionally embedded. For image classification tasks, the model also contains an extra classification token that will eventually be passed through a final linear layer in order to produce the network result. Studies have demonstrated that transformers do not generalize well when trained on insufficient amounts of data, and the training of these models involved extensive computing resources [35]. As a result, DeiT was created as a more efficiently trained transformer that utilizes a novel distillation procedure. Specifically, the process involves a distillation token aimed at reproducing the label estimated by a teacher model (often a pretrained ConvNet).

In our work, we included both the original ViT and DeiT models, alongside the DeiT-Tiny to determine the effect of the network's size and complexity on its error resilience. For all transformers included in our studies, the architecture follows a pattern of stacked self-attention functions and fully connected layers. The self-attention functions can be thought of as mapping a query vector and a set of key-value pairs to a vector output, while the fully connected layers serve to project and modify the output vector to be used as input tokens to the next layer. Similar to the concept used in database retrieval, the vector output of self-attention is computed as a weighted sum of the input values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key [36]. As shown in Fig. 1, the self-attention function first computes the corresponding query, key, and value matrices for the input tokens, and subsequently performs scaled dot-products between the generated matrices to arrive at the output vector. Since these GEMM operations

make up the majority of the computation in vision transformers, our studies focus primarily on analyzing and protecting these operations.

Formulation to Quantify Resilience: We focus on understanding the resilience of different transformers used in vision in the context of transient hardware errors at inference time. Specifically, we employ a single-bit flip error model that is commonly used as an abstraction for modeling hardware faults [2]. Within this error model, we focus on transient errors that successfully alter the originally correctly chosen classification class of an inference. We refer to this as a classification mismatch and it can be used as the baseline vulnerability metric for selective protection. Since memory is often protected by ECC, the aim is to model hardware faults in logic using a layer-level fault model. Prior research demonstrated that transient or permanent faults in logic of GPUs may not always manifest in software as single-bit flips [15], [31]. The use of fault models that take into the consideration the effect of cross-layer error propagation can improve the quality of the vulnerability assessment [28]. However, such methods often come with complexity and overhead resulting in an accuracy versus complexity/speed trade-off. While ALBERTA is evaluated using a single-fault model, the methodology (GoldenEye [27]) can be enhanced to consider other fault models that can better represent the underlying hardware faults and is an interesting future direction.

CNN error resilience is a well-studied topic [16], [19], [23], [30], [32], [33]. Similar to a prior work [26], we also formulated a transformer layer's vulnerability metric based on the size of the layer and the likelihood of an error in the layer propagating to the output. Specifically, we define a layer's vulnerability as $V_{layer}[i] = V_{orig}[i] \times P_{prop}[i]$, where $V_{orig}[i]$ represents the probability that a transient hardware fault corrupts the output of the i-th layer and P_{prop} represent the likelihood that the corrupted output will propagate to and modify the final model output. $V_{orig}[i]$ depends heavily on the hardware details such as raw failure rates, numerical precision, size of the layer (e.g., number of MACs and buffers used to execute the work in the layer). It can effectively be approximated as proportional to the number of MAC operations used to compute the output of the i-th layer, assuming that all MAC operations within the system have an equal likelihood of faulting and the large storage structures (e.g., FIFOs, buffers) are protected with ECC/parity. Without loss of generality, we compute and use the relative vulnerability for $V_{orig}[i]$ in this article. It is computed as the number of MACs used in one forward pass of the i-th layer divided by the total number of MACs used in one forward pass of the entire model. By definition, $0.0 < V_{orig}[i] < 1.0$.

A layer's propagation vulnerability, on the other hand, can be computed using either the number of mismatches that occur during our injection campaign of said layer, or a continuous metric we refer to as $\Delta Loss$:

$$\Delta Loss_{layer} = \sum_{i=1}^{N} [\ell_{golden} - \ell_i]/N$$

TABLE 1. Comparison of ALBERTA to Related Works

Technique	Relevance of Evaluation	Floating-point Error Analysis	Runtime Overhead	Recovery
FILR [26]	Classification CNN	Fixed-point only	High	N/A
ABFT [17]	GEMM-based models	Rigorous	Very high	Very high
CNN ABED [16]	Classification CNN	Fixed-point only	High	N/A
Ma, et al. [25]	Transformers on small datasets	Perliminary	Medium	High
ALBERTA	SOTA transformers and datasets	Extensive	Low	Very high

where ℓ_{golden} is the cross-entropy loss for an error-free inference and ℓ_i stands for the cross-entropy loss for the corresponding i^{th} error-injected inference across N total error injections. Similar to the case for $V_{orig}[i]$, the larger the value of $\Delta Loss_{layer}$, the more vulnerable the layer is to transient errors. The advantage of $\Delta Loss_{layer}$ is that it converts the binary metric of classification mismatch into a continuous metric, which is shown to significantly boost the speed of model analysis, especially when mismatches occur sparsely.

B. RELATED WORK

For CNNs, different techniques have been explored to make inferences resilient. Exploding neuron values were identified as one of the primary reasons for inference output corruptions [6], [22]. A prior work profiled the value ranges in each layer during fault-free executions and derived a range-based bounds checker to detect significant deviations [22]. Ranger extended the concept and proposed an automated transformation to selectively restrict the value ranges in DNNs, which reduces the large deviations caused by critical faults and transforms them to benign faults that can be tolerated by the inherent resilience of the DNNs [6]. This work builds on top of these methods and only considers residual errors. We assume that such detectors or range restrictors are in place or the quantized data types used during deployment will inherently restrict the value ranges. We only consider errors that will perturb the value of the neuron within the profiled value ranges as computed using the training set. We describe the related methods below and show a summary in Table 1.

Mahmoud et al. [26] introduced FILR, an overarching procedure that can be broken down into two separate resilience methods – feature-map level resilience (FLR) and inference level resilience (ILR) – as each method targets a different dimension when providing selective resilience for CNNs. During FLR, the network layers are ranked using a given vulnerability metric, and the layers are selectively protected such that it will simultaneously maximize the vulnerability coverage and minimize the overhead of redundancy.

ILR, on the other hand, is used to determine the thresholds on network confidence to determine which inferences are vulnerable and need a rerun. Metrics to determine vulnerability for non-classification models (e.g., segmentation or detection) are not studied, which makes it difficult to apply ILR to such applications. While FLR is attractive (and has similarities to our solution), the duplication of layers as a protection method is too expensive and often results in a high-overhead solution. We overcome that challenge by leveraging a significantly lower-cost algorithm-based protection method.

Since most of the compute-intensive (convolution and fully connected) layers are linear operations and are executed as GEMMs (general matrix multiplications) on GPUs, algorithm-based fault tolerance (ABFT) techniques that verify and correct computation using checksums are applicable [17]. These techniques compute checksums for input data, store them with the original data, perform the original and redundant computation, verify outputs, and correct errors inline. While the extra compute operations ABFT introduces are a small fraction of the number of the original computations, prior ABFT implementations have achieved about 20 percent runtime overheads for square matrices [9]. A recent study reported that the overheads can be much higher (>50%) for the non-square matrices that are typically used in CNNs [16].

Recent studies applied the approach to detect errors in convolutions [12], [16] and reported much lower overheads (6−23%) without the inline error correction capabilities (referred as Algorithm-Based Error Detection or ABED). In complex safety-critical systems such as AVs, preventing silent data corruption (SDC) is more important than the ability to correct the error inline. Some of the prior solutions focus on fixed-point computations (e.g., int8 convolutions in [16]) where a precise checksum computation is viable and sufficient. In most practical applications, the use of floating-point data types (e.g., FP16/E5M10 or Bfloat16/E8M7) remains prevalent. Floating-point computations are known to introduce numerical error that makes checksum-based checking challenging.

Range restriction is another interesting method that has been applied to CNNs as well as transformers. This technique offers high error coverage for datatypes with large ranges, but for quantized datatypes with inherently restricted ranges, the benefit of this approach becomes negligible. One such implementation provides significant fault mitigation for Swin transformers but with very high runtime overheads of nearly 68% [13]. The overheads for ALBERTA, on the other hand, are significantly lower while also offering relatively high coverage irrespective of the datatype.

There is parallel work to ALBERTA that employs similar methodologies of providing transformer error resilience through checksum-based algorithms [25]. In our assessments, ALBERTA provides several advantages when compared to the alternative study. In terms of performance, ALBERTA achieves a substantial improvement in error recovery from 96% to 99%. When evaluating the experiment setup and results, ALBERTA evidently provides better versatility in handling production level architectures and dataset configurations. As opposed to analyzing CIFAR-10 with less relevant images in [25], ALBERTA utilizes the full ImageNet dataset and allows for error injection campaigns of arbitrary sizes. Moreover, ALBERTA provides detailed studies of numerical discrepancies encountered during checksum computation, which is crucial to providing error resilience for floating pointing models. Combined with an easy extension proposed to address broader SDCs from hardware and systemic faults, the findings and error correction capabilities provided by ALBERTA translates directly to production settings characterized by substantial variations in inference data.

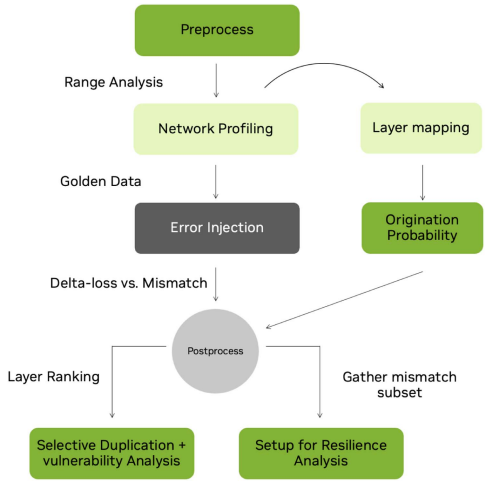


FIGURE 2. Pipeline Overview for ALBERTA vulnerability analysis.

C. RESEARCH QUESTIONS

Given these challenges, the primary objective of this article is to answer the following four key research questions.

- RQ1: How resilient are vision transformers to transient errors when performing image classification?
- RQ2: What per-layer resilience patterns and variations exist, if any, for vision transformers?
- RQ3: Can we protect vision transformer models in a way to achieve high resilience with minimal computation and memory overhead?
- RQ4: How do we design a reliable checksum-based error detection mechanism that maintains high error coverage for floating-point models?

III. THE ALBERTA FRAMEWORK

We present ALBERTA, **al**gorithm-**b**ased **e**rror **r**esilience in **t**ransformer **a**rchitectures, a framework to analyze and improve the resilience of (vision) transformer models. We focus on vision transformers owing to their increasing use in safety-critical applications. We first present ALBERTA's vulnerability analysis followed by the resilience analysis.

A. VULNERABILITY ANALYSIS

We quantify the vulnerability of individual layers with a pipeline that is divided into the following four stages. Fig. 2 summarizes the pipeline. (1) The *preprocess* step records the numerical lower and upper bounds for every layer in the network during inference to define the expected values ranges of neuron outputs. This information is later used in the error injection to rule out errors that can be protected by techniques like clamping and range restriction [6], [22]. (2) The *profiling* step records the network architecture details (layer sizes, parameter count, etc.) in addition to collecting the set of images that the error-free pre-trained model correctly classifies. We

refer to this subset as *golden data* and will use it as the sample space for error injection and resilience. (3) The *error injection* step performs data type dependent random error injections for every layer within the network. Specifically, for every injection experiment, we flip a random bit at a randomly chosen neuron for a random image – as constrained by the numerical range determined during the preprocess step – and record the location of the error injection with the corrupted output. (4) The *postprocess* step compiles the results from injection and profiling steps into pandas dataframes for selective layer level duplication.

In the error injection step, we perform 102400 injections for very low error in the measurement at high confidence. With the above sample size, the error bar is <0.24% at 99% confidence level (using 90% as population proportion). Depending on the desired error tolerance in the results, fewer injections may be sufficient. During each individual error injection, we sample from the set of Golden Data containing 5017600 images that the model correctly classifies and select batches of size 128 or user specified number of images for analysis. For each selected image, the output activation of the injected layer is modified using the target injection type and the corrupted classification loss – alongside other injection metadata – is recorded during inference for downstream analysis.

With the objective of protecting the model with the lowest possible overhead, we focus on using the vulnerability data collected from the pipeline to rank the network layers, such that employing a chosen protection algorithm, such as selective layer duplication or checksum style error detection, will simultaneously maximize the resilience and minimize the overhead of redundancy. In the case for selective layer duplication, the coverage provided by protecting a target layer is computed using the equation for V_{layer}, while the overhead of duplicating a target layer depends on the amount of work in the layer (e.g., number of MACs, which is proportional to V_{orig} according to the formulation described in Section II-A).

The majority of the computation in transformer architectures is performed in linear (fully connected) layers that are a part of each transformer block. To provide comprehensive error protection, we select four layers to be included in our analysis pipeline for every transformer block. This consists of the linear layer used to compute key query value matrices (Q, K, V) as shown in Fig. 1, the linear projection layer that outputs the result of multi-head attention, and two accompanying linear layers used by the feed-forward MLP. The matrix multiplication that occurs within the self-attention computation (dot product between query and key matrix and subsequently the product between the softmax output and the value matrix) are not considered in this approach and will be addressed in future work due to user-end implementation complexity. As of Pytorch version v2.0.0, all efficient error injection methods are done through Pytorch hooks, whose applications are limited to well defined Pytorch modules as opposed to general operations such as matrix multiplication. Although it is possible to use a user-defined self-attention module with custom matrix multiplication layers, doing so would render the framework not generalizable where each target model must be individually modified.

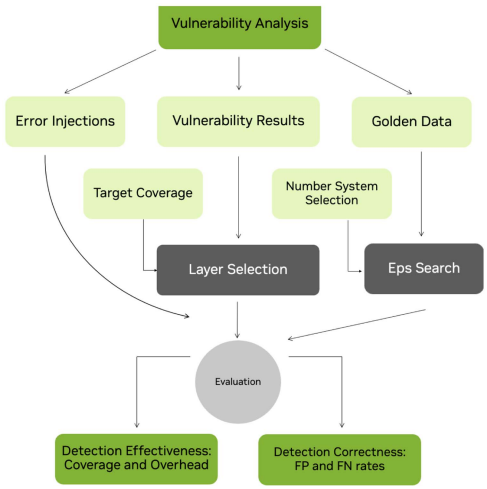


FIGURE 3. Overview of ALBERTA's resilience implementation.

B. RESILIENCE ANALYSIS

This section presents our resilience improvement technique in the layers identified as vulnerable based on the above analysis. We start with the description of the Pytorch wrapper class that integrates ALBERTA into different transformer models. Next, we describe the execution and evaluation process which includes layer selection, parameter tuning, and generating the false positive and negative rates for error detection. Lastly, we provide the details of the correction mechanism that can be deployed alongside ALBERTA for forward error correction to complete the end-to-end protection solution. Fig. 3 summarizes the flow.

1) RESILIENCE IMPLEMENTATION

ALBERTA's resilience implementation is defined as a Pytorch module. It provides a set of functionalities that enables an in-depth resilience analysis of transformer based architectures. Apart from the basic utility functions such as custom loading of models and datasets, the resilience implementation contains several key functionalities that sets it apart from previous works as listed below.

Checksums: The core function of ALBERTA is to enable checksum style protection for the subset of layers that are deemed most vulnerable according to prior layer level analysis. In the case for most transformer models, such subsets consists solely of GEMM layers whose output activations are obtained by performing batched matrix multiplication between the input features and the transposed hidden weight matrix, while also adjusting the result by the layer bias. As part of ALBERTA's core functionality, we perform checksum style verification on this general matrix multiply (GEMM) operation and successfully avoid the overhead of traditional ABFT checksum matrices. To accomplish this, we forfeit the capabilities of inline error correction and instead compute

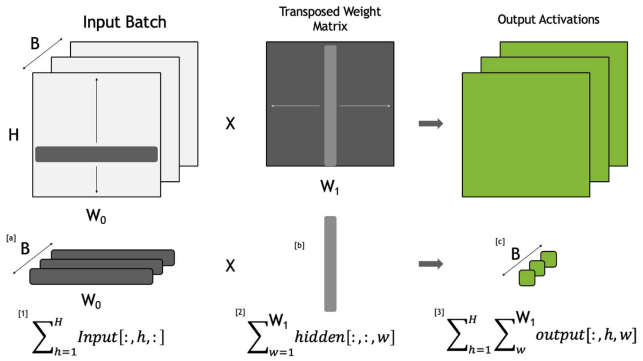


FIGURE 4. The error detection checksum compares the value of step-3 against the green vector in step-c element-wise for any numerical discrepancy and computes the batch average difference for distribution modeling during eps search.

row and column checksum vectors for the input and layer activation, respectively. Specifically, as depicted in Fig. 4, the batched row checksum vector is computed by summing along the column dimension of the input batch, while the column checksum vector is computed by reducing the row dimension of the transposed hidden weight matrix. To perform error detection, we compare the dot product of the row and column checksums against the sum of layer activations for numerical discrepancy. It is worth noting that (1) this process offers a significant overhead reduction when compared to traditional techniques such as fmap duplication and modular redundancy – as shown later in the results section and (2) it can be optimized such that the column checksum of the transposed hidden weight matrix can be computed offline and stored by ALBERTA's resilience implementation prior to deployment.

While the checksum itself can be naively implemented using torch tensor operations, some challenges must be addressed for this detection mechanism to be generalizable.

1) As introduced in the findings of [16], the increasing use of reduced-precision data types (e.g., 8- and 4-bit integers) during inference accelerators introduces new challenges for checksum-based error detection techniques, as successive sums across large layer activations have a high likelihood for overflow. Although floating-point data types do not exhibit the extreme overflow behaviors observed in integers, significant numerical imprecision still occurs in the case where depending on the rounding mode – overly large results will be represented as max float (RTZ) or Inf (RNE), while underflown results are often rounded to zero. In situations where such rounding errors necessitate significant increases in the discrepancy tolerance levels used during error detection, ALBERTA's resilience implementation includes additional functionalities for automatically selecting the appropriate numerical precision used for checksum computations based on the numerical range of network weights and the input dataset.

2) An equality check is sufficient when dealing with models that use fixed point data types. However, a more nuanced checksum verification is needed for floating-point models, as discrepancies in the checksum can arise due to the nondeterministic behavior of floating-point arithmetic, as opposed to transient errors. ALBERTA's resilience implementation addresses this issue by providing an intuitive, confidence based mechanism for identifying whether a discrepancy is caused by transient error or numerical imprecision. We now describe this epsilon generation process.

Eps Generation: One significant limitation of prior works in algorithm based fault tolerance (ABFT) is the restricted application to fixed datatypes where the checksum computations are deterministic. Our work provides a solution to this issue by introducing a confidence based error detection framework capable of achieving close to 100% coverage of transient errors that result in model misclassification. In order to accomplish this, ALBERTA's resilience implementation includes a numerical analysis function that empirically models – for each layer – a unique distribution describing the probability that a discrepancy in the layer's online checksum is due to numerical imprecision rather than transient error. More specifically, this process involves first iterating through the subset of data that the error free network correctly classifies, while collecting during every inference a set of differences between the computed checksum and the true layer activation sum. Using this set of discrepancies which are unique to each layer's trained parameters and dimensions, a layer specific distribution can then be generated through regression and used for error detection during deployment, where if the observed checksum discrepancy is above a user set confidence threshold, an error would be raised and the correction mechanism would be triggered.

2) EXECUTION AND EVALUATION

Aside from allowing the users to select their desired numerical precision for the checksum independently from the model itself, ALBERTA offers two additional tuning mechanisms that the user can use prior to the evaluation step:

1) The option to either select custom layers for resilience or provide a desired theoretical error coverage and have ALBERTA's resilience implementation dynamically select the layers necessary for achieving it.
2) Selecting the target distribution (defaults to standard normal) and confidence threshold of checksum discrepancies based on which error detection will be performed.

As we show in the vulnerability analysis, it is not uncommon for transformer architectures to have layers that are extremely resilient to transient errors. As a result, we can reduce overhead by omitting these layers. By simply providing a desired coverage level, ALBERTA's resilience implementation will automatically perform this optimization through a knapsack inspired mechanism. It is important to note that since ALBERTA behaves independently for each layer of the network, selecting a subset of layers for protection means error occurring in any unprotected layers cannot be detected; making it important to select layers with the highest error

propagation rate and overall vulnerability for protection. In other words, when we decide to not include a layer for checksum protection, we are choosing to forefeit all error coverage for that layer in exchange for a reduction in network overhead. Furthermore, the optimal subset of layers returned by ALBERTA is based on estimated layer vulnerabilities that are dataset dependent. As a result, the resilience bounds we provide in the results section are not strictly true when fine-tuning models on new target datasets, but can always serve as a statistically accurate approximation.

3) CORRECTION

In addition to providing the necessary tools to integrate and evaluate the effectiveness of checksums as a protection mechanism for arbitrary transformer models, the wrapper also offers the choice of two correction mechanisms to be performed after error detection – replay and skip forward.

To perform a *replay* upon error detection, we first save the activation values of every layer during inference, and if the subsequent layer raises an error, we simply read the error-free activations from the previous layer and rerun the inference step. This mechanism incurs constant space overhead – since only one activation needs to be saved at a time during evaluation – while also resulting in minimal runtime overhead as shown later in the results section.

To perform a *skip forward* upon error detection, we have the following three options – jump ahead to the next layer with the same input size, skip to the next transformer block, or simply jump to the inference step while passing forward the saved, error free activation as input. This mechanism can be used for any layer inside the transformer blocks with zero overhead but will likely have a negative impact on the model accuracy. For this reason, we believe it is applicable only for special use cases and have designed ALBERTA's resilience implementation to default to replay.

IV. RESULTS

A. EVALUATION METHODOLOGY

We perform our evaluation with a focus on DeiT-base and DeiT-tiny models, which are both pretrained and fine-tuned with distillation on ImageNet-1 k (1 million images, 1,000 classes). We use the Pytorch Framework and obtained pretrained models from the Facebook research repository with patch 16 and 224×224 pre-training resolution. All experiments are run on a standard Linux server with NVIDIA A6000 GPUs and Intel i9-10980XE CPU. We obtained the results for vulnerability analysis using an adapted version of the GoldenEye injection module [27] and perform all other testing using custom Pytorch hooks. During vulnerability analysis, all inferences are performed using full fp32 precision, while inferences during resilience analysis are executed with half-precision at fp16. We evaluate checksum operations comprehensively at fp16, fp32, and fp64. However, due to limitations mentioned regarding checksums, the best accuracy during error detection are obtained using fp64 and are included here in both overhead and coverage related plots.

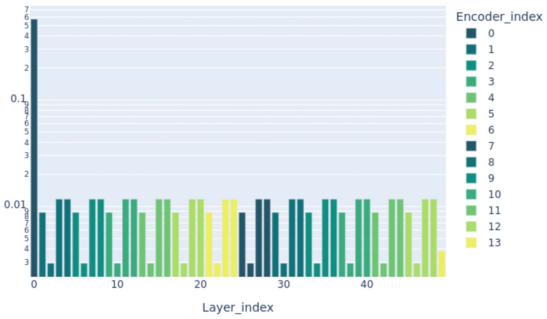


FIGURE 5. V_{orig} for DeiT-base is shown here. The x-axis shows the layer number. The y-axis shows the V_{orig} value. $V_{orig}[i]$ is computed as the fraction of MAC operations used in the i^{th} layer over the total number used by the network. The results pattern is similar for DeiT-tiny (not shown here).

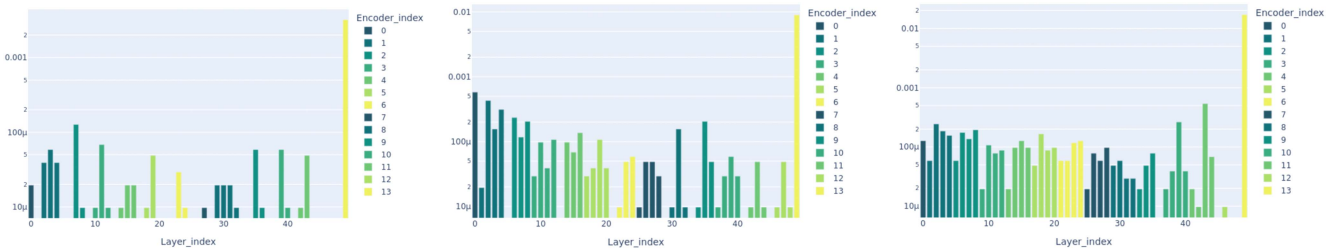
B. VULNERABILITY ANALYSIS

We first present the results from ALBERTA's vulnerability analysis on DEiT models. We excluded the original ViT results as they are similar to base DeiT. We note that the original ViT performs significantly worse than its DeiT counterpart – its error propagation rate and (as a result) vulnerability exceeds that of DeiT across all layers in the network. The cause for this behavior can be attributed to the network parameters learned during distillation versus regular training.

Origination Probability Comparison: During the profiling step of our pipeline, we compute V_{orig} as the fraction of MAC operations used in the forward pass of each layer (shown in Fig. 5). We observe that the largest value corresponds to the positional embedding layer (convolution), which is about 100× bigger than the intermediate layers inside the transformer block. Furthermore, the layer sizes are proportionally identical among both models included in our study.

Propagation Probability Comparison: Using mismatch as the ground truth vulnerability metric for computing V_{prop}, we note that both the DeiT models exhibit a similar behavior where error injections performed at the final linear prediction head resulted in the highest probability of mismatch in the output. This is illustrated in Fig. 6.

More importantly, it can be observed that a significant number of layers within the DeiT-base model are incredibly resistant to transient errors. Moreover, DeiT-base has significantly better built-in resilience to transient error propagation when compared to DeiT-tiny. Out of 102400 error injections, zero random bit flip resulted in mis-classification of the network output for 22 out of the 50 layers. Comparisons between the results of DeiT-base and DeiT-tiny support the conclusion that a larger network size contributes to a lower error propagation probability. In practice, this behavior can be intuitively understood as the result of larger transformer networks containing higher number of normalization and activation layers that can likely suppress errors during inference.



(a) DeiT-base (b) DeiT-tiny (c) ViT-Base

FIGURE 6. $V_{prop}[i]$ as computed using mismatch. The x-axis represents the layer number within the corresponding architecture, while the y-axis represents the V_{prop} value.

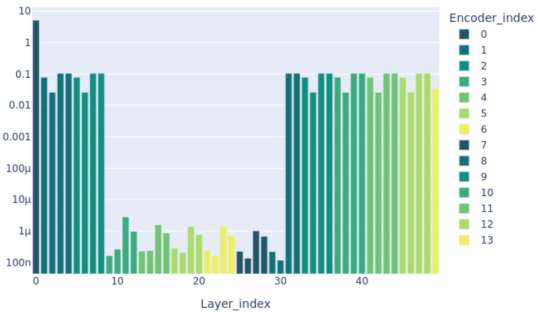


FIGURE 7. V_{layer} of DeiT-base computed as $V_{orig}[i] \times P_{prop}[i]$.

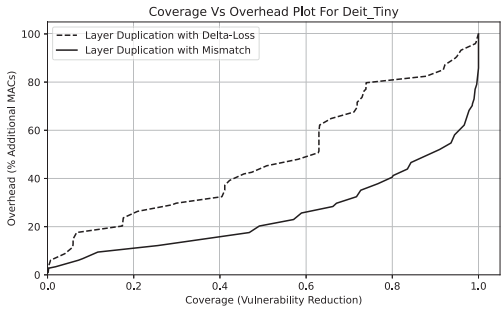


FIGURE 8. The coverage result of DeiT-tiny as computed from optimized selective layer duplication; maximizes the vulnerability / coverage of selected layers while minimizing the overhead of redundancy. The prediction head is not considered during this evaluation, as it should always be protected.

Delta Loss: $\Delta Loss$ converts binary classification mismatch into a continuous metric. We observe that the error propagation graph of DeiT-base exhibits a considerable dip in V_{layer} value starting from the 3rd to the middle of the 8th transformer block as shown in Fig. 7. Although this is consistent with the claim that DeiT-base has significantly better resilience than its DeiT-tiny counterpart, we do not fully understand how the random sparsity difference in the Mismatch graph is transformed into a contiguous block of minimal vulnerability in the DeiT-base architecture. Noting that both $\Delta Loss$ and mismatch measure the same latent layer vulnerability, we propose alternate error models based on finer-grained error injections as part of future investigation into this statistical divergence.

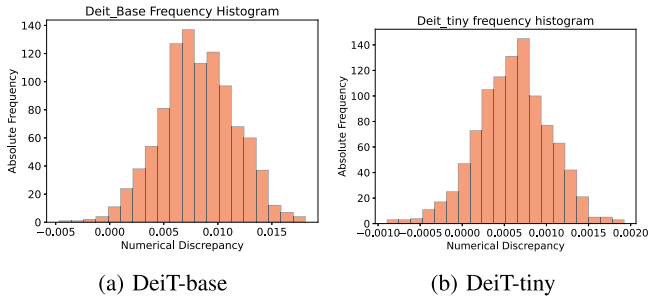
Selective Layer Duplication: The coverage versus overhead results from selective layer duplication for DeiT-Tiny are shown in Fig. 8, where x and y axes represent the total coverage and cumulative overhead, respectively. The dotted line represents the result of using the $\Delta Loss$ derived layer vulnerability, while the solid oracle line is the result of using mismatch. Due to the prediction head of transformer architectures being consistently the most vulnerable layer (as measured using mismatch), we assume it is always protected by the algorithm and instead plot the coverage vs. overhead from protecting the remaining layers.

We present here two main takeaways from this study using Mismatch as the ground truth vulnerability. First, using the intersection between $x = 90\%$ and the coverage graphs, it can be observed that DeiT-Tiny is able to achieve 90% of the remaining vulnerability coverage at approximately 49% overhead. Second, a significantly steeper overhead can be seen when trying to obtain the remaining 10% coverage. To reach 99% coverage for instance, select layer duplication would incur approximately 45% additional overhead, which is prohibitively high in most high performance systems. While 90% may be sufficient for architectures that have high fault tolerance, safety critical applications often demand higher reliability that are closer to full 100% coverage. As a solution to this shortcoming, we introduce the alternative checksum-based protection for vision transformers.

C. PROTECTING VISION TRANSFORMERS.

We now present the main results from our experiments on error resilience, as divided into the following points of focus. (1) The empirical statistics of numerical discrepancies observed during floating-point checksums. (2) The coverage and overhead from selecting optimal subsets of layers for checksum protection. (3) The overall effectiveness of ALBERTA's confidence-based error detection and correction.

Empirical statistics of numerical discrepancies: At its core, the basis for using confidence-based error detection is the observation that the empirical discrepancies encountered during checksum form a normal-like distribution for all Imagenet samples at inference. As shown in Fig. 9, the same linear layer in DeiT-base and DeiT-tiny – sizes (768,3072) and (192,



(a) DeiT-base (b) DeiT-tiny

FIGURE 9. Histogram of sample layer's discrepancy values in DeiT-base and DeiT-tiny.

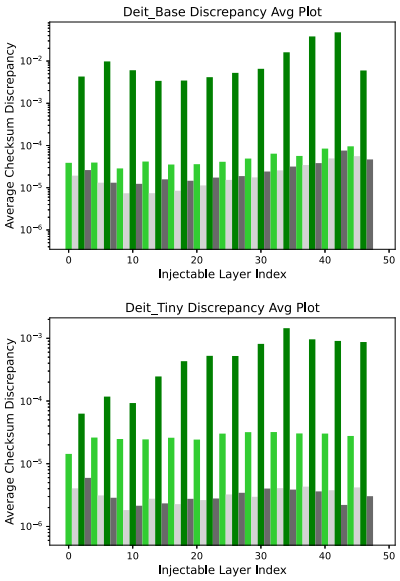


FIGURE 10. Average checksum discrepancies observed during Epsilon Search for DeiT-base & DeiT-tiny.

768) – both exhibit a normal-like behavior, with the overall size decrease of 16× (3072 / 192) translating to a similar magnitude drop in observed discrepancies.

At the network level, similar trends can be seen in Fig. 10, where we observe a positive correlation between layer size and observed discrepancy during checksum. Across all layers in the network, the observed discrepancy for DeiT-base is approximately 10x higher than that of DeiT-tiny – going from a range of $(10^{-5}, 10^{-2})$ to $(10^{-6}, 10^{-3})$

ALBERTA Coverage versus Overhead: As observed earlier from the layer analysis results of DeiT-base and similar models in Fig. 6, the last layer of most transformer models is by far the most vulnerable to transient errors – contributing to close to 50% of all mismatches and 88% of the total vulnerability score – and as a result, we have designed ALBERTA such that the last layer will always be included during layer selection for protection. Similar to the setup in Fig. 8, the coverage & overhead offered by protecting the last linear layer is omitted in Fig. 11. Although it cannot be directly observed in Fig. 11, the plots for ALBERTA – lying flat along the x-axis – exhibit

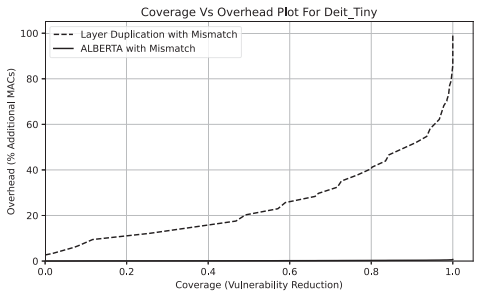


FIGURE 11. DeiT-Tiny overhead vs. coverage plot. The graph for ALBERTA lies almost flat along the x-axis and distinctly outperforms layer duplication.

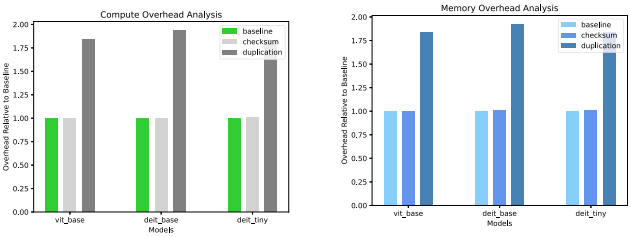


FIGURE 12. Network analysis of compute and memory overhead as obtained using FP64 checksums and at 99% coverage.

a similar pattern to that of selective layer duplication, but is consistently over 100× more efficient. While this value depends on the dimensions of the protected layers, for most transformer models, this value is well above 100 even without considering the boosts from offline checksum generation.

In previous works such as selective duplication, serious concerns were raised due to non-negligible model performance drops that resulted from high computational and/or memory overhead. As illustrated in Fig. 12, ALBERTA marks a significant improvement from existing works where the computation overhead is less than 0.2% across all of our target models, while the memory overhead reaches a negligible maximum of 0.01% during checksum computation. The memory overheads are so low because only the protected layers need additional memory and that the additional memory incurred is a order of magnitude lower than the layer's original memory footprint. Specifically, the network memory overhead is computed as the sum of two components – offline and online checksums. The offline checksum term is computed as the total memory allocated to storing the column checksum of weight matrices that are computed prior to inference 4. The online checksum term, on the other hand, is computed as the per-layer maximum of memory allocated to input checksum vector + memory allocated to computing the output checksum during verification. In cases where kernel fusion is possible, the overhead can be significantly lower because the output checksum computation will be fused with the epilogue.

ALBERTA Performance Analysis: Furthermore, assessments using *torch.cuda.max_memory_allocated()* indicate that our target models' maximum memory usage during

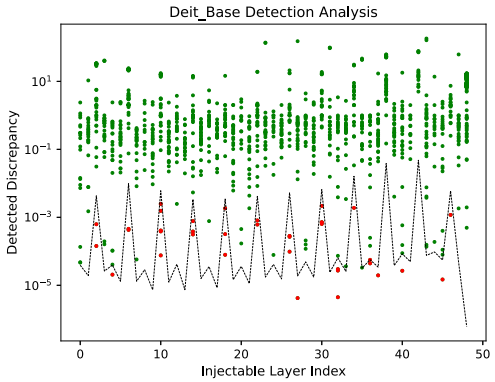


FIGURE 13. DeiT-base detection analysis plot. Green points represent true positive errors that are sucessfully detected, red points represent the false negatives that the detection missed, and the black line plots the per-layer discrepancy thresholds.

inference is >30% less than that of training. Given the efficient and accelerated training support on modern GPUs, models running ALBERTA for error resilience would see little to no performance impact even in resource constrained systems.

Effectiveness of confidence based error detection and correction: In related works that focus on models with fixed data types, layers protected by checksum-based algorithms easily achieves 100% coverage of single-bit flip errors. In floating-point models, a range-based equality check at the 99.9% confidence level does not catch all injected errors but consistently achieves over 99% coverage of errors that result in mismatches. Fig. 13 shows that out of 3890 error injections that resulted in mismatches, our detection algorithm achieves a false negative rate of less than 1%, with 31 missed errors. Please note that error detection with ALBERTA has near zero false positives thanks to the earlier analysis of per-layer numerical discrepancies in Fig. 9. By setting the per layer threshold at 99.99% confidence level or greater, we completely eliminate false positives in our detection, as any discrepancies recorded above our threshold have less than 0.01% chance of occurring due to numerical imprecision. In our studies, errors injections that do not cause a significant change in layer activation – magnitude of less than 1e-5 – are not detected by our proposed framework in layers with high error-free discrepancies, but have also rarely been detected to cause a mismatch. As a general rule, there exists a correlation between how much an error offsets the original activation value and the probability of it affecting the network output. In our case with eps of less than 1e-3, such situation becomes even more unlikely to occur.

D. ANSWERING THE RESEARCH QUESTIONS

Based on the above results, we directly answer the research questions listed in Section II-C here.

RQ1: Vision transformers are extremely resilient to transient errors when performing image classification, with injections in several layers resulting in zero mismatches. Using the origination probability and per-layer mismatches as shown in Figs. 5 and 6, we compute the probability that a transient error results in an SDC for DeiT-base and DeiT-tiny to be 0.98% and 2.49%, respectively. While a fault in a DeiT-tiny's operation is more likely to cause an SDC, it has 13.9× fewer operations than DeiT-base.

RQ2: Using mismatch as the ground truth vulnerability metric, we find that the prediction head of vision transformers is most vulnerable to transient errors, while many layers across the middle of the network are highly resilient. We observe a positive correlation between network size and layer level resilience (deit-base vs. deit-tiny), and that training techniques such as distillation can have a significant impact in network resilience (deit-base vs. vit-base).

RQ3: Existing modular redundancy techniques provides good protection but at the cost of prohibitively high overheads. Despite the challenges, ALBERTA achieves over 99% coverage for transient errors that result in a mismatch at less than 0.2% computation overhead and on average <0.01% memory overhead for vision transformer. We introduce several optimizations in the future work section that can further reduce the memory overhead.

RQ4: We design a reliable checksum-based error detection mechanism for floating-point networks by modeling the numerical discrepancies that occur during error-free inferences and creating per layer confidence thresholds that can be used for error detection and correction.

V. DISCUSSION

Duplication vs. ALBERTA: Using mismatch as the ground truth vulnerability metric, performing selective layer duplication for DeiT-base provides 90% coverage at around 40% computation overhead and 92% memory overhead. ALBERTA reduces the computation overhead to approximately 0.2% and the memory overhead to less than 0.01%, while also increasing the overall coverage to 99%. By using replay as the self-correction mechanism for transient hardware faults, we are able to resolve all erroneous detection with an overhead of less than 2% – making the framework itself resilient to future architecture and library changes that may affect the false positive rate for error detection.

Optimizations: While the compute and data transfer overheads are small, the checksum computations can be memory intensive. Several optimizations can be employed to reduce (and eliminate) this overhead. (1) L2 cache eviction policy hints (e.g., evict-last hint) available in the latest GPUs starting Ampere [1] can be provided to keep the inputs (and outputs) persistent in L2 to minimize DRAM traffic for checksum generation. (2) Merge the checksum generation with the layer that produces the tensor. This way the tensor need not be fetched twice, eliminating extra memory traffic originating from checksum generation. (3) Lastly, hardware support to perform near L2 cache (or memory) reduction will eliminate data fetching from L2 (or memory) to streaming multiprocessors and pollute local caches with read-once data, and speed up checksum generation.

Other fault types and devices: Methods proposed in ALBERTA can be used to guard against SDCs from other fault sources, e.g., permanent and intermittent hardware faults, and potentially software bugs. There is increased interest in the industry to address SDCs caused by faults in processors deployed in datacenters, where the source is primarily attributed to manufacturing defects and aging-related faults [10], [37]. While the checksum-based error detection approach will be effective in detecting such faults, some aspects need modification: the selection strategy to determine what needs protection, a diagnostic procedure to root-cause the fault type (permanent or transient), and the recovery mechanism to relaunch the work on a different node to maintain forward progress.

ALBERTA may also be extended to other processing modules such as CPUs and FPGAs. The core components of ALBERTA are error injections + correction and floating point rounding error analysis. Since we use a high-level fault injector [27], it can be tailored to fit the device with a custom error manifestation model. The rounding error analysis is device- and kernel-specific and needs to be updated to reflect the implementation.

Faults in cryptography: While ALBERTA targeted fault detection in vision transformers, fault detection in traditional and post-quantum cryptographic algorithms is becoming increasingly important. A prior research proposed using Cyclic Redundancy Check (CRC) for error detection in finite field inversions over GF(2 m) on a polynomial basis [4]. In separate studies, Canto et al. [5], [7] investigated error detection schemes applicable to multipliers in lattice-based key encapsulation mechanisms, a post-quantum cryptography technique, when implemented on FPGAs and using cyclic codes. Further research in this direction seems promising.

VI. CONCLUSION

We investigate the error resilience of the transformer architecture, specifically for the DeiT-base and DeiT-tiny implementations. Our investigations show that the embedding layer is most susceptible to error origination due to its large size, while the final prediction head is most vulnerable to error propagation due to its proximity to network output. Moreover, comparisons between the results for DeiT-base and DeiT-tiny serve as evidence that a larger network size contributes to lower average error propagation probability in transformers, while also highlighting the significant jump in layer resilience inside DeiT-base.

REFERENCES

[1] *CUDA C. Program. Guide–Device Memory L2 Access Manage.*, Nov. 2024. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programmingguide/index.html#device-memory-l2-access-management

[2] A. Rizwan et al., "Understanding the propagation of transient errors in HPC applications," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2015, pp. 1–12.

[3] H. Askr et al., "Deep learning in drug discovery: An integrative review and future challenges," *Artif. Intell. Rev.*, vol. 56, pp. 5975–6037, Nov. 2023.

[4] A. C. Canto, M. M. Kermani, and R. Azarderakhsh, "CRC-based error detection constructions for FLT and ITA finite field inversions over GF(2m)," *IEEE Trans. Very Large Scale Integration. Syst.*, vol. 29, no. 5, pp. 1033–1037, Mar. 2021.

[5] A. C. Canto, A. Sarker, J. Kaur, M. M. Kermani, and R. Azarderakhsh, "Error detection schemes assessed on FPGA for multipliers in lattice-based key encapsulation mechanisms in post-quantum cryptography," *IEEE Trans. Emerg. Topics Comput.*, vol. 11, no. 3, pp. 791–797, Jul.-Sep. 2023.

[6] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2021, pp. 1–13.

[7] A. Cintas-Canto, M. M. Kermani, and R. Azarderakhsh, "Reliable architectures for finite field multipliers using cyclic codes on FPGA utilized in classic and post-quantum cryptography," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 31, no. 1, pp. 157–161, Jan. 2023.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[9] C. Ding, C. Karlsson, H. Liu, T. Davies, and Z. Chen, "Matrix multiplication on GPUs with on-line fault tolerance," in *Proc. IEEE 9th Int. Symp. Parallel Distrib. Process. Appl.*, 2011, pp. 311–317.

[10] H. Dixit et al., "Keytone: Silent data corruptions at scale," in *Proc. IEEE 29th Int. Symp. On-Line Testing Robust Syst. Des.*, Crete, Greece, 2023, pp. 1–2, doi: 10.1109/IOLTS59296.2023.10224872.

[11] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. 9th Int. Conf. Learn. Representations*, Virtual Event, Austria, May 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[12] D. Dosovitskiy, N. Margomenos, N. Mitianoudis, C. Nicopoulos, and G. Dimitrakopoulos, "Low-cost online convolution checksum checker," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 30, no. 2, pp. 201–212, Feb. 2022.

[13] G. Gavarini, A. Ruospo, and E. Sánchez, "Evaluation and mitigation of faults affecting swin transformers," in *Proc. 29th Int. Symp. On-Line Testing Robust Syst. Des.*, 2023, pp. 1–7.

[14] Y. Gong, Y.-A. Chung, and J. Glass, "PSLA: Improving Audio Tagging with Pretraining, Sampling, Labeling, and Aggregation," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, 2021. [Online]. Available: https://doi.org/10.1109/TASLP.2021.3120633

[15] J. D. G. Balaguera, J. E. R. Condia, F. F. Dos Santos, M. S. Reorda, and P. Rech, "Understanding the effects of permanent faults in GPU's parallelism management and control units," in *Proc. Int. Conf. High Perform. Comput., Netw. Storage Anal.*, 2023, Art. no. 46.

[16] S. K. S. Hari, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Making convolutions resilient via algorithm-based error detection techniques," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 4, pp. 2546–2558, Jul./Aug. 2022.

[17] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. C-33, no. 6, pp. 518–528, Jun. 1984.

[18] Y. Huang and Y. Chen, "Autonomous driving with deep learning: A survey of state-of-art technologies," 2020, *arXiv:2006.06091*.

[19] Y. Ibrahim et al., "Soft error resilience of deep residual networks for object recognition," *IEEE Access*, vol. 8, pp. 19490–19503, 2020.

[20] J. Johnson, "Deep learning for computer vision, Lecture 17: Attention," 2022. [Online]. Available: https://web.eecs.umich.edu/justincj/slides/eecs498/WI2022/598_WI2022_lecture17.pdf

[21] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 1–41, 2022.

[22] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2017, pp. 1–12.

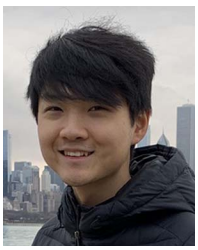
[23] F. Libano et al., "Selective hardening for neural networks in FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 66, no. 1, pp. 216–222, Jan. 2019.

[24] Z. Liu et al., "Swin transformer: Hierarchical vision transformer using shifted windows," 2021, *arXiv:2103.14030*.

[25] K. Ma, C. Amarnath, and A. Chatterjee, "Error resilient transformers: A novel soft error vulnerability guided approach to error checking and suppression," in *Proc. IEEE Eur. Test Symp.*, 2023, pp. 1–6.

[26] A. Mahmoud et al., "Optimizing selective protection for CNN resilience," in *Proc. IEEE 32nd Int. Symp. Softw. Rel. Eng.*, 2021, pp. 127–138.

[27] A. Mahmoud, T. Tambe, T. Aloui, D. Brooks, and G.-Y. Wei, "GoldenEye: A platform for evaluating emerging numerical data formats in DNN accelerators," in *Proc. 52nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2022, pp. 206–214.
[28] G. Papadimitriou and D. Gizopoulos, "Demystifying the system vulnerability stack: Transient fault effects across the layers," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Architecture,* 2021, pp. 902–915.
[29] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf
[30] M. Rios, F. Ponzina, G. Ansaloni, A. Levisse, and D. Atienza, "Error resilient in-memory computing architecture for CNN inference on the edge," in *Proc. Great Lakes Symp.*, 2022, pp. 249–254.
[31] F. F. dos Santos, J. E. Rodriguez Condia, L. Carro, M. S. Reorda, and P. Rech, "Revealing GPUs vulnerabilities by combining register-transfer and software-level fault injection," in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2021, pp. 292–304.
[32] C. Schorn, T. Elsken, S. Vogel, A. Runge, A. Guntoro, and G. Ascheid, "Automated design of error-resilient and hardware-efficient deep neural networks," *Neural Comput. Appl.*, vol. 32, no. 24, pp. 19, Dec. 2020, pp. 18327–18345, doi: 10.1007/s00521-020-04969-6.
[33] C. Schorn, A. Guntoro, and G. Ascheid, "An efficient bit-flip resilience optimization method for deep neural networks," in *Proc. Des. Autom. Test Europe Conf. Exhib.*, 2019, pp. 1507–1512.
[34] D. Shen, G. Wu, and H.-I. Suk, "Deep learning in medical image analysis," *Annu. Rev. Biomed. Eng.*, vol. 19, pp. 221–248, Mar. 2017.
[35] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proc. Int. Conf. Mach. Learn.*,, 2021, pp. 10347–10357.
[36] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.
[37] S. Wang, G. Zhang, J. Wei, Y. Wang, J. Wu, and Q. Luo, "Understanding silent data corruptions in a large production CPU population," in *Proc. 29th Symp. Operating Syst. Princ.*, 2023, pp. 216–230.
[38] B. Zhang et al., "StyleSwin: Transformer-based GAN for high-resolution image generation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 11294–11304, doi: 10.1109/CVPR52688.2022.01102.
[39] H. Zhou et al., "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proc. 35th AAAI Conf. Artif. Intell.*, vol. 35, no. 12, 2021, pp. 11106–11115.



HAOXUAN LIU received two bachelor's degrees in computer science and data science from the University of Michigan, Ann Arbor, MI, USA. He is currently a Deep Learning Libraries Engineer with NVIDIA, Santa Clara, CA, USA. His research interests include autonomous systems, deep learning safety, and robotics, with prior experience in Event-based Stereo Visual SLAM and GNSS, and sensor fusion algorithms for precise 3D mapping and localization.

VASU SINGH received the Ph.D. degree in software verification from Computer Science Department, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland. He is the Director of Software Engineering in the AI Compute Safety group with NVIDIA, Santa Clara, CA, USA. He works in the field of safety standards for autonomous AI based systems. His research interests include across AI safety and formal verification for accelerated AI computing.

MICHAŁ FILIPIUK received the master's degree in computer science from the Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Warsaw, Poland. He is currently a Senior Deep Learning Engineer with NVIDIA, Munich, Germany. He specializes in testing and certifying deep learning software for autonomous vehicles. His research interests include deep learning resiliency, encompassing architectural properties, low-level kernel execution, and deployment on embedded platforms.

SIVA KUMAR SASTRY HARI (Senior Member, IEEE) received the Ph.D. degree from the Computer Science Department, University of Illinois Urbana-Champaign, Champaign, IL, USA. He is currently a Principal Research Scientist with the Architecture Research Group, NVIDIA, Santa Clara, CA, USA. His research interests include computer architecture, artificial intelligence, and systems, with current focus on autonomous and high-performance computing systems.