2023 Design, Automation & Test in Europe Conference (DATE 2023)
Special Initiative "Autonomous Systems Design"

Efficient Software-Implemented HW Fault Tolerance for TinyML Inference in Safety-critical Applications

Uzair Sharif, Daniel Mueller-Gritschneder, Rafael Stahl, Ulf Schlichtmann
Chair of Electronic Design Automation, Technical University of Munich (TUM), Munich, Germany
Email: {uzair.sharif, daniel.mueller, r.stahl, ulf.schlichtmann}@tum.de

***Abstract*—TinyML research has mainly focused on optimizing neural network inference in terms of latency, code-size and energy-use for efficient execution on low-power micro-controller units (MCUs). However, distinctive design challenges emerge in safety-critical applications, for example in small unmanned autonomous vehicles such as drones, due to the susceptibility of off-the-shelf MCU devices to soft-errors.**

We propose three new techniques to protect TinyML inference against random soft errors with the target to reduce run-time overhead: one for protecting fully-connected layers; one adaptation of existing algorithmic fault tolerance techniques to depth-wise convolutions; and an efficient technique to protect the so-called epilogues within TinyML layers. Integrating these layer-wise methods, we derive a full-inference hardening solution for TinyML that achieves run-time efficient soft-error resilience.

We evaluate our proposed solution on MLPerf-Tiny benchmarks. Our experimental results show that competitive resilience can be achieved compared with currently available methods, while reducing run-time overheads by $\sim$120% for one fully-connected neural network (NN); $\sim$20% for the two CNNs with depth-wise convolutions; and $\sim$2% for standard CNN. Additionally, we propose selective hardening which reduces the incurred run-time overhead further by $\sim 2\times$ for the studied CNNs by focusing exclusively on avoiding mispredictions.

***Index Terms*—TinyML, safety, error detection, soft-error**

I. Introduction

TinyML aspires to be a key enabler of pervasive Machine Learning (ML), by synergizing deep learning with low-power micro-controller units (MCUs). Research in this field has mainly focused on optimizing neural network inference in terms of latency, code-size and energy-use for efficient execution on MCUs. Due to this, TinyML is particularly suitable for deploying DNNs on systems such as unmanned autonomous vehicles (UAVs), e.g., drones, which can only fit small MCU-type compute devices due to energy and space constraints. Yet, in such applications, also safety concerns need to be considered. Progressive technology scaling has improved transistor-density adequately within such MCUs. However, such advances in technology scaling also make the devices increasingly prone to reliability concerns, such as random radiation-induced *soft-errors*. To overcome such errors, special safety MCUs are available that employ on-board HW-based safety mechanisms such as lock-step processors, clock monitors or error correction logic. However, such devices are usually quite costly such that in many applications custom-off-the-shelf (COTS) MCU devices are preferred. For such COTS MCUs, *software-implemented hardware fault tolerance* (SIHFT) methods [1] are attractive for hardening TinyML inference as they are highly flexible and offer competitive soft error resilience. Among existing works, instruction duplication (ID) based approaches (such as [2]–[4]) can adequately tackle soft-errors by ensuring near zero silent data corruptions (SDCs). However they incur excessive run-time overheads (RTO) that results in high energy usage, rendering them ill-suited for hardening TinyML computations especially in real-time applications such as drones.

Recent works on SIHFT for large embedded ML systems have proposed alternate algorithmic-based fault tolerance (ABFT) methods to reduce RTO while ensuring high soft error resilience. However, our core insight is that such techniques need to be adopted and extended for TinyML applications, which have much severe resource and run-time constraints. For example, TinyML layers often employ so-called *epilogue* [5], that allow performing post-processing tasks (such as bias-addition, re-quantization, activation, range-clipping etc.) on each computed output value within the layer itself, to reduce data movement during inference. Similarly, Tiny convolution neural networks (CNNs) regularly carry out convolutions using efficient depth-wise separable convolution procedure [6] that employs depth-wise convolution layers that are more uncommon in larger embedded ML systems. To protect such specific operations, existing SIHFT solutions proposed to resort to using ID causing high RTO. Furthermore, we find the established ABFT strategy to protect the fully-connected layers to cause excessive RTO (at par with ID methods) for TinyML inference.

In this work, we analyze the limitations of existing methods for TinyML and propose an efficient SIHFT approach for TinyML inference. For this,

1) we propose an ABFT solution to protect the matrix-vector product operation in fully-connected TinyML layers,
2) we adapt an existing ABFT method for convolution operation to also protect depth-wise convolution operation,
3) we propose source-level transformations to protect epilogue computations within layers.

Combining the newly proposed SIHFT methods with existing works (for the rest of the TinyML inference task), we derive a full-inference hardening solution for TinyML systems to achieve run-time efficient soft-error resilience. Based on a RISC-V simulation platform, our experimental results show that compared with baseline SIHFT solution (existing SIHFT approaches for entire inference), our solution offers competitive soft-error resilience for all studied benchmarks. In terms of RTO, our solution offers considerable benefit. Maximum potential can be seen in hardening a fully-connected neural

978-3-9819263-7-8/DATE23/© 2023 EDAA

network by reducing RTO requirements by 120%. Meanwhile, we observe RTO reduction by atleast 20% and 2% for CNNs wih depth-wise convolutions and standard CNNs respectively. Moreover, our approach offers flexibility via selective hardening to further gain $\sim 2\times$ reduction in RTO by focusing solely on avoiding mispredictions instead of all possible data corruptions.

II. Background and Related Works

FC: Full-Checksum ABFT scheme for fully-connected layers. Huang et al. [7] presented an elegant ABFT approach to protect general matrix-matrix multiplication $\mathbf{AB} = \mathbf{C}$ computations. The proposed ABFT method entails carrying out following *left-sided* and *right-sided* checks (see [8]):

$$(\mathbf{v}^T\mathbf{A})\mathbf{B} \stackrel{?}{=} \mathbf{v}^T\mathbf{C} \tag{1}$$

$$\mathbf{A}(\mathbf{Bw}) \stackrel{?}{=} \mathbf{Cw} \tag{2}$$

where $\mathbf{v}, \mathbf{w}$ are so-called checksum vectors. We refer to this scheme as *Full-Checksum* (FC) method to distinguish with other ABFT approaches in the rest of the paper. In [7], $\mathbf{v}, \mathbf{w}$ are chosen as all ones, which simplifies the required dot-products $\mathbf{v}^T\mathbf{A}$ and $\mathbf{v}^T\mathbf{C}$ in (1) to be the column-summation vectors of $\mathbf{A}$ and $\mathbf{C}$ respectively. Similarly row-summation vectors of $\mathbf{B}$ and $\mathbf{C}$ represent respectively the dot-products $\mathbf{Bw}$ and $\mathbf{Cw}$.

A fully-connected layer computation involves performing the matrix vector product (MVP), between its inputs and its weights, followed by epilogue to generate output. As MVP is a specialized case for matrix-matrix product, existing SIHFT solutions for ML networks have opted to use the FC method to protect the MVP in fully-connected layers.

CoC-D, FIC: ABFT schemes for convolution layers. Convolution layers such as Conv2D perform linear convolution operations ($\otimes$) between the input feature-maps (fmaps) $\{\mathbf{D}_n\}$ and the filter-array $\{\mathbf{W}_k\}$, to generate output fmaps $\{\mathbf{O}_n\}$. Zhao et al. [9] presented comprehensive work on ABFT schemes for hardening the convolution layers. The *Checksum-of-Checksum Detection* (CoC-D) scheme is proposed to detect errors in convolution operation during inference. An input-checksum fmap $\mathbf{C^d}$ and filter-checksum $\mathbf{C^w}$ are generated, which respectively represent the depth-wise sum of all input-fmaps $\{\mathbf{D}_n\}$ and all filters $\{\mathbf{W}_k\}$. The convolution operation between $\mathbf{C^d}$ and $\mathbf{C^w}$ should match with the output-checksum.

Hari et al. [5] propose their *Filter Input Checksum* (FIC) scheme which, in essence, extends the ABFT principles of CoC-D by leveraging the transformation proposed in [10] to match the size of $\mathbf{C^d}$ to the size of $\mathbf{C^w}$. Due to size match, the required convolution operation for the ABFT part can be performed cheaply via simpler vector-vector dot-product operation. The resulting scalar c^o should match with reduced output s^o obtained via summing all elements of $\{\mathbf{O}_n\}$.

ID: Instruction duplication schemes for other layers. Neural networks employ other layers such as pooling, softmax etc. Compiler-based ID methods, such as EDDI [2], NZDC [3] or REPAIR [4], can be employed for hardening these layers. Further, the so-called layer epilogues and activation function also involve non-linear operations. Hari et al. [5] have proposed to use ID methods for hardening these operations.

Ranger: Layer-wise range checks. In contrast to layer-specific SIHFT solutions, Ranger [11] proposes performing additional range-restriction operations at the end of each layer to avoid the propagation of large deviations resulting from computation errors. Small deviations are allowed to pass as they are expected to be tolerated inherently by neural-networks.

III. Tailored SIHFT for TinyML Inference

We first analyze the limitations of existing SIHFT shemes for TinyML inference and then propose an efficient tailored TinyML SIHFT scheme.

A. Limitations of Existing SIHFT for TinyML

Most TinyML networks perform byte-quantization on the trained model parameters. This effectively approximates each determined floating-point parameter value to certain levels within the integer-range [-128,127] to enable optimized inference on MCUs. Due to this quantization, the Ranger [11] method becomes ineffective as the network activations span the entire range of allowed data values within TinyML networks.

Employing the FC method [7] to protect the fully-connected layers in TinyML inference leads to excessive RTO. Specifically, carrying out FC check (2) essentially entails carrying out the entire main MVP computation again. Thus, FC method at least doubles the run-time, which is in line with ID costs for protecting TinyML fully-connected layers.

Popularized by Howard et al. [6], depth-wise convolution layers feature regularly in TinyML networks. These layers perform convolution operations on the input-activations $\{\mathbf{D}_n\}$ with the layer filter $\mathbf{W}$ in a depth-wise manner. We have found recent ABFT works [5], [9] to provide high resilience in protecting convolution operations in TinyML networks, however, these methods were not described for depth-wise convolutions.

Hari et al. [5] propose to protect the layer epilogues by ID schemes. Partitioning the layer computations into non-ID and ID regions reflect no major concerns for GPU-based platforms as they offer substantially high thread-level parallelism. However, for single-threaded MCU execution environments, selective ID requires a context-switch between ID and non-ID regions, which exacerbates the RTO.

In the following, we show how SIHFT can be efficiently implemented for TinyML.

B. PC Scheme for Fully-connected Layers

Fully-connected layers within TinyML networks employ the MVP operation on input-vector $\mathbf{d}$ and weights-matrix $\mathbf{W}$ to generate intermediate output-vector $\mathbf{o}$ which is subsequently fed into the epilogue to generate final layer output. To protect the MVP operation cheaply, we modify the FC scheme to perform only the left-sided check (1) and avoid the right-sided check (2). The resulting ABFT approach, referred to as *Partial-Checksum* (PC), is illustrated in Fig. 1 to protect ($\mathbf{Wd} = \mathbf{o}$) MVP.

Gunnels et al. [8] performed an analytical analysis to study the propagation of errors during matrix-matrix product computation when protected under FC method. We extend their analysis for the considered MVP case when protected via PC method in the following:

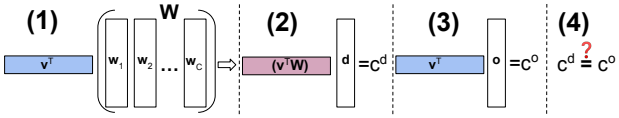

Fig. 1. PC scheme (1-4) to protect MVP operation.

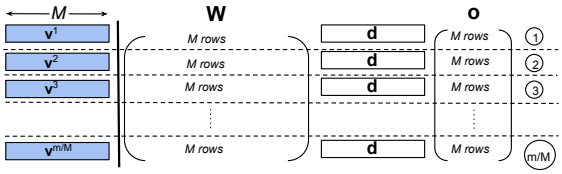

Fig. 2. Partitioned implementation of MVP operation.

In case of erroneous computation, the MVP generates $\mathbf{o}'$ instead of reference $\mathbf{o}$. Let $\mathbf{f}{=}(\mathbf{o}'{-}\mathbf{o})$, then $||\mathbf{f}||{\neq}0$ indicates erroneous MVP operation. To perform this check at run-time cheaply, the PC check introduces $\mathbf{e}{=}(\mathbf{v}^T\mathbf{o}'{-}(\mathbf{v}^T\mathbf{W})\mathbf{d}){=}\mathbf{v}^T\mathbf{f}$ and computes $||\mathbf{e}||$ online alongside MVP as shown in Fig. 1. Table I derives the relation between $||\mathbf{e}||$ and $||\mathbf{f}||$, in case errors occur (in $\mathbf{W}, \mathbf{d}$ or $\mathbf{o}$), to serve as a guideline in choosing $\mathbf{v}$:

- $\mathbf{v}$ must have non-zero elements i.e. $|\mathbf{v}_i| \neq 0$, so that PC protects fully against errors in $\mathbf{W}$ and $\mathbf{o}$ as $||\mathbf{e}||{=}0$ implies $||\mathbf{f}||{=}0$.
- $\mathbf{v}$ must not be normal to any column-vectors of $\mathbf{W}$ i.e. $\mathbf{v}^T\mathbf{w}_i{\neq}0$ for all i, so that PC protects fully against errors in $\mathbf{d}$ as $||\mathbf{e}||{=}0$ implies $||\mathbf{f}||{=}0$.

Implementation Aspects. Pure SW based implementations of MVP iterate over elements of $\mathbf{d}$ while processing a $\mathbf{W}$ row. Motivated by SIMD based implementations, TinyML kernels perform the MVP in a partitioned fashion, for which (parametrizable) M rows of the filter-matrix $\mathbf{W}$ are processed with one cached input $\mathbf{d}$ to generate partial outputs for the considered rows. This optimizes load operations and reduces data movement. The process is repeated over all M-sized blocks of rows of $\mathbf{W}$ to get the complete output $\mathbf{o}$. Fig. 2 depicts this conceptually.

As $\mathbf{d}$ has to be reloaded for each partition/block, the error in $\mathbf{d}$ can only persist for one block instead of persisting for entire MVP computation. Due to this, the analysis shown in Tab. I remains valid only for one block and we have to choose $\mathbf{v}$ for each block separately. As highlighted on left of Fig. 2, we choose $\mathbf{v}^m$ for each partition m of $\mathbf{W}$ such that it has non-zero elements, and it is not normal to any column-vector $\mathbf{w}_i^m$ in that partition. The chosen $\mathbf{v}^m$ vectors are then augmented together to form $\mathbf{v}$ so as to perform PC method outlined in Fig. 1.

For the MVP employed in TinyML's fully-connected layers, the filter-matrix $\mathbf{W}$ is known prior to TinyML inference, thereby enabling:

- checksum-vector $\mathbf{v}$ can be determined (offline) prior to inference instead of at inference-time.
- filter-summation vector $(\mathbf{v}^T\mathbf{W})$ can be determined also offline and stored in memory along with $\mathbf{W}$.

TABLE I
GUIDELINE TO SELECT CHECKSUM-VECTOR $\mathbf{v}$ FOR PC METHOD
η: MAGNITUDE OF ERRORS, $\mathbf{u}_i$: UNIT VECTOR WITH i^{th} COMPONENT AS 1

	w_{ij} corrupted	o_i corrupted	d_i corrupted
$\mathbf{o}'$	$(\mathbf{W}+\eta\mathbf{u}_i\mathbf{u}_j^T)\mathbf{d}$	$\mathbf{o}+\eta\mathbf{u}_i$	$\mathbf{W}(\mathbf{d}+\eta\mathbf{u}_i)$
$\|\|\mathbf{f}\|\| = \|\|\mathbf{o}'-\mathbf{o}\|\|$	$\eta\|d_j\|$	η	$\eta\|\|\mathbf{w}_i\|\|$
$\|\|\mathbf{e}\|\| = \|\|\mathbf{v}^T\mathbf{f}\|\|$	$\eta\|v_i\|\|d_j\| = \|v_i\|\|\|\mathbf{f}\|\|$	$\eta\|v_i\| = \|v_i\|\|\|\mathbf{f}\|\|$	$\eta\|\mathbf{v}^T\mathbf{w}_i\| = \frac{\|\mathbf{v}^T\mathbf{w}_i\|}{\|\|\mathbf{w}_i\|\|}\|\|\mathbf{f}\|\|$

Due to this, for an $m \times n$ weight-matrix $\mathbf{W}$, the PC method requires m and n additional multiply-accumulate operations for computing $\mathbf{v}^T\mathbf{o}$ and $(\mathbf{v}^T\mathbf{W})\mathbf{d}$ respectively at run-time. The required overhead of PC is deemed to be much cheaper than double run-time of the FC method to protect MVP operation.

C. FICdw Scheme for Depth-wise Convolution Layers

The efficient FIC scheme [5] was so far only described for convolutional layers. We propose an adoption for depth-wise convolution layers, which perform the convolution operation depth-wise on input fmaps $\{\mathbf{D}_n\}$ with the layer filter $\mathbf{W}$ to generate output fmaps $\{\mathbf{O}_n\}$. An input-checksum $\mathbf{C^d} = \sum_n(\mathbf{D}_n)$ is first generated at run-time that sums up all input fmaps depth-wise. Afterwards, output-checksum $\mathbf{C^o}$ is computed by performing a depth-wise convolution operation ($\circledast$) between $\mathbf{C^d}$ and $\mathbf{W}$:

$$\mathbf{C^o} = \mathbf{C^d} \circledast \mathbf{W} = \sum_{n=1}^{N}(\mathbf{D}_n) \circledast \mathbf{W}$$

The depth-wise convolution operation, as standard convolution, is a linear operation that satisfies the distributive property for (depth-wise) addition, hence

$$\mathbf{C^o} = \sum_{n=1}^{N}(\mathbf{D}_n \circledast \mathbf{W}) = \sum_{n=1}^{N}(\mathbf{O}_n)$$

An ABFT check can be carried out at run-time that compares $\mathbf{C^o}$ with sum of all generated output fmaps:

$$(\mathbf{C^d} \circledast \mathbf{W}) \stackrel{?}{=} \sum_{n=1}^{N}(\mathbf{O}_n)$$

To perform the check cheaply, we can equivalently compare the reduced-sums of these fmaps at run-time. Denoting reduced-sum of generated outputs as $s^o = \sum_{x,y}\sum_n(\mathbf{O}_n)$, we get

$$\sum_{x,y}(\sum_{ch.}(\mathbf{C^d} \circledast \mathbf{W})) \stackrel{?}{=} \sum_{x,y}(\sum_{n=1}^{N}(\mathbf{O}_n))$$

$$\sum_{x,y}(\sum_{ch.}(\mathbf{C^d} \circledast \mathbf{W})) \stackrel{?}{=} s^o$$

Here the quantity $\sum_{ch.}(\mathbf{C^d} \circledast \mathbf{W})$ in effect represents the convolution operation $(\mathbf{C^d} \otimes \mathbf{W})$, hence the above check can be expressed as:

$$\sum_{x,y}(\mathbf{C^d} \otimes \mathbf{W}) \stackrel{?}{=} s^o$$

The quantity $\sum_{x,y}(\mathbf{C^d} \otimes \mathbf{W}) = c^o$ can be efficiently computed at run-time using similar procedure to FIC method. We refer to application of FIC method to depth-wise convolution operation as FICdw scheme in remainder of the paper.

Implementation Aspects. We illustrate the process to perform the FICdw check in Fig. 3. For TinyML focused implementations, the shown procedure can be carried out efficiently in the following manner: As inputs $\{\mathbf{D}_n\}$ are processed sequentially, an accumulator can be setup for $\mathbf{C^d}$ that sums up the inputs depth-wise for **(1)**. Similarly, as each output element is generated it is also fed into a scalar accumulator s^o for **(2)**. On receiving the last input $\mathbf{D}_N$, we perform the reduction (per [10]) on accumulated $\mathbf{C^d}$ for **(3)**. Vector-vector dot-product on this reduced channel is then performed with the filter $\mathbf{W}$ for **(4)**. We perform the final **(5)** check between c^o and s^o.

D. EPI and EPIsel Scheme for Layer Epilogues

In this work, we propose source-level transformations, referred collectively to as EPI hereinafter, geared towards protecting the layer epilogues as a more efficient alternative compared to ID approaches. The EPI transformations, as highlighted in Fig. 4, include:

(i) Duplicate epilogue. All the key tasks in epilogue computation are duplicated at source-level. As shown, each output value generated from layer's main computation (such as convolution, MVP etc.) is duplicated at run-time. The epilogue task-pipeline is then executed twice using these redundant values as inputs. This yields redundant final output values `epi-out, epi-out*`.

(ii) Protect store to output-buffer. After performing the final store of `epi-out` to the output-buffer, EPI performs an additional load afterwards to the same location in the buffer. The subsequent load, however, uses the copy of output-buffer's base-address. The loaded value is compared with the duplicated `epi-out*` value to protect the store to output-buffer. In case the offset to the base-address is corrupted before the final store such that the effective address and its copy are corrupted, then the above mentioned load-back would fail to detect the error. To overcome this, EPI introduces a checksum value that accumulates offsets for all stores to the output-buffer. After the entire layer was executed, the offset-checksum is compared with a reference value (from an error-free scenario) to detect address errors.

(iii) Duplicate layer parameters. Layer parameters are constants that are live (i.e. are in scope) during entire execution of the layer. They are used in computing each output value in the layer. For example, the `out-offset` adds a constant value to the output element before the final store. EPI protects such parameters by duplicating them before executing the layer.

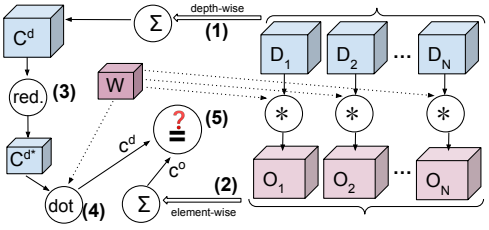

Fig. 3. FICdw (1-5) for ABFT protection of Depth-wise Convolutions.

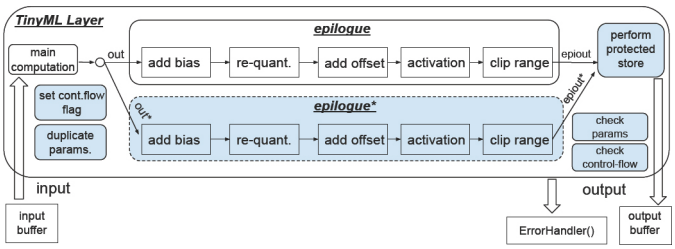

Fig. 4. EPI transformations for a given TinyML layer.

After executing the layer, these parameters are matched with shadow values to detect errors.

(iv) Check control-flow to detect early return. EPI introduces a simple layer-scope control flow monitoring flag, which is set to '0' at start of layer execution. After the layer is executed successfully, this flag is set to '1'. The flag is checked before returning from the layer function to make sure the return is legal in terms of control-flow.

Selective EPIsel hardening. For relaxed safety requirements, some mechanisms in the proposed EPI approach could be disabled to reduce RTO. The focus lies on ensuring that no classification errors (misprediction) occur rather than no Silent Data Corruptions (SDCs) (corruption in output class probabilities). We noticed that often more than one activation has to be corrupted to cause a misprediction. On the other hand, corruptions to single activation elements can be often tolerated inherently by CNNs (see also [11]). With this insight, we can disable EPI mechanisms **(ii)**, **(iii)** as they focus strictly on ensuring that all individual output elements are computed without error. This selective variant of EPI is referred to as EPIsel in remainder of the paper. EPIsel still include mechanisms, which are responsible for protecting layer parameters and impact all output values. However, these mechanisms bear very low overheads.

E. SIHFT Configuration for Hardening TinyML Inference

We integrate our proposed layer-specific SIHFT approaches into a combined configuration to achieve cost-effective soft error resilience for TinyML inference tasks. Table II provides a comparison of this solution, in terms of how different operations are protected during the inference, to the baseline solution that is obtained by combining the state-of-the-art layer-wise SIHFT solutions.

As shown, we employ the ABFT schemes FIC and the newly proposed FICdw for protecting the convolution and depth-wise convolution operations respectively in convolutional layers. The MVP in fully-connected layers is protected via the newly proposed PC method. Add layers employ an element-wise add operation to its input tensors. Due to element-wise nature of the operation, we can extend our EPI method (see Fig. 4) to protect the entire layer by bringing the layer-computation (simple add) into the sphere-of-replication (i.e. before the re-quantization step within epilogue) as well. Apart from this, we employ EPI to protect epilogues of already discussed convolution and fully-connected layers. Moreover, we allow to replace EPI with

TABLE II
FULL-INFERENCE SIHFT SOLUTIONS FOR TINYML

Layer	Baseline	Ours
Convolution	**FIC** (Conv.) + **ID** (Epilogue)	**FIC** (Conv.) + **EPI / EPIsel** (Epilogue)
Depth-wise convolution	**ID** (Layer)	**FICdw** (DW Conv.) + **EPI / EPIsel** (Epilogue)
Fully-connected	**FC** (MVP) + **ID** (Epilogue)	**PC** (MVP) + **EPI / EPIsel** (Epilogue)
Add	**ID** (Layer)	**EPI / EPIsel** (Layer)
Average Pool & Softmax	**ID** (Layer)	**ID** (Layer)

EPIsel to seek RTO improvements without degrading error-coverage for relaxed safety requirements. For other layers, such as average-pool layers, we resort to using an ID scheme.

IV. EVALUATION

A. Experimental Setup

Benchmarks. We use the MLPerf-Tiny [12] suite for benchmarks to evaluate our proposed methods on practical TinyML networks, which span a range of varying model topologies and carry out diverse learning tasks. The suite comprises three CNNs — Audio Wakeup Word (AWW), Video Wakeup Word (VWW) and RESNET for speech recognition, video recognition, and image classification respectively. The CNNs of AWW, VWW use depth-wise separable convolutions. In addition, the suite provides an Anamoly Detection (AD) network, which employs primarily fully-connected layers.

Implementation. We use the Tensor Flow Lite for Microcontrollers (TFLM) framework [13] to setup the execution environment for deploying pre-trained MLPerf-Tiny networks on MCUs. The TFLM run-time is configured to delegate the deployed network's operations to the CMSIS-NN [14] kernels library for inference. We implement ABFT and our EPI solution by hardening the C-Versions of the CMSIS-NN kernels. Further, we implement NZDC method [3] to realize ID hardening.

Evaluation Methodology. The inference tasks are compiled for execution on an Instruction Set Simulator [15] with fault injection support, which simulates a standard RISC-V processor (`RV32IMAC`) at instruction-accurate level.

We employ statistical fault injection [16] to carry out a Monte Carlo simulation based analysis for estimating the Silent Data Corruption (SDC) rates for the studied TinyML tasks. For each simulation run, one input is picked randomly from a subset of the data set of size 30. During inference, a random instruction point is chosen, whereby a *bit-flip* (soft error) is injected into one of the simulated processor's architectural registers. At the end of simulation, we compare the network output to the error-free reference values. In case of an undetected output corruption, we mark the outcome of the run as SDC. We perform 200,000 such runs to reliably assess SDC vulnerability of unprotected as well as various hardened variants of studied TinyML networks. We compare efficacy of various implemented SIHFT solutions in terms of incurred run-time overheads (RTO) relative to the run-time (r) of unprotected programs as under:

$$\text{RTO } (\%) = \frac{r_{sihft} - r_{unprot.}}{r_{unprot.}} \cdot 100 \tag{3}$$

where run-time (r_x) is measured as number of instructions simulated for running task x on the simulator. After conducting N_{trial} trials, the SDC-rate for a given SIHFT can then be determined as per Schirmeier correction [17]:

$$\text{SDC } (\%) = \frac{N_{sdc}}{N_{trial}} \cdot \frac{r_{sihft}}{r_{unprot.}} \cdot 100 = \frac{N_{sdc}}{N_{trial}} \cdot (\text{RTO} + 100) \tag{4}$$

B. Layer-wise Evaluation

First we evaluate the proposed SIHFT methods on a per-layer basis. Throughout this section, the value $\text{SDC}_{(x)}$ represents the SDC rate that is caused due to soft errors within layer of type x. Further please note: we omit results for TinyML benchmarks that do not have a layer of the studied layer type.

Convolution Layers. The existing methods CoC-D, FIC (see Sec. II) harden convolution layers efficiently as can be seen in Table III. Both CoC-D, FIC schemes exceptionally reduce the $\text{SDC}_{(\text{conv})}$ by at least one order of magnitude. In terms of costs, FIC offers less RTO of $\sim$1.65$\times$ than CoC-D. Further, we inspect the remaining $\text{SDC}_{(\text{conv})}$ and find them entirely to be caused due to soft errors during epilogue computation.

For hardening the epilogues, we compare our EPI method with ID. The corresponding SDC improvements are shown on the right-side columns of Table III. Both ID and EPI are effective in eliminating most of the remaining $\text{SDC}_{(\text{conv})}$, however, our solution offers a clear RTO benefit.

Depth-wise Convolution Layer. Table IV reports the SDC evaluation for FICdw method in comparison with ID. As seen, both solutions offer reliable SDC coverage by removing almost all of $\text{SDC}_{(\text{dwconv})}$. The proposed FICdw approach offers significant benefit in terms of RTO.

Fully-connected Layer. Table V reports the SDC rate of the proposed PC scheme compared to the existing FC scheme for hardening MVP. We report this analysis for the AD network

TABLE III
EVALUATION OF SIHFT FOR CONVOLUTION LAYER
SDC-RATE [%], RTO [%]

		Unprt.	CoCD	FIC	FIC + ID	FIC + EPI
AWW	$\text{SDC}_{(\text{conv})}$	13.98	1.347	1.313	0.0255	0.0252
	$\text{RTO}_{(\text{conv})}$	0	5.084	3.391	17.71	13.78
VWW	$\text{SDC}_{(\text{conv})}$	10.46	1.143	1.114	0.0238	0.0265
	$\text{RTO}_{(\text{conv})}$	0	7.087	3.595	25.12	17.86
RES-NET	$\text{SDC}_{(\text{conv})}$	26.97	2.002	1.886	0.0863	0.0815
	$\text{RTO}_{(\text{conv})}$	0	7.054	4.685	13.56	10.49

TABLE IV
EVALUATION OF SIHFT FOR DEPTH-WISE CONVOLUTION LAYER
SDC-RATE [%], RTO [%]

		Unprt.	ID	FICdw + EPI
AWW	$\text{SDC}_{(\text{dwconv})}$	3.356	0.0021	0.0034
	$\text{RTO}_{(\text{dwconv})}$	0	152.1	61.16
VWW	$\text{SDC}_{(\text{dwconv})}$	3.104	0.0026	0.0029
	$\text{RTO}_{(\text{dwconv})}$	0	152.1	56.07

TABLE V
EVALUATION OF SIHFT FOR FULLY-CONNECTED LAYER
SDC-RATE [%], RTO [%]

		Unprt.	FC + ID	PC + EPI
AD	$SDC_{(fully\text{-}conn)}$	35.37	0.0015	0.0013
	$RTO_{(fully\text{-}conn)}$	0	135.8	10.49

as it solely relies on fully-connected layers. As shown, the PC method achieves competitive SDC resilience compared to the FC method at substantially lower RTO.

C. Evaluation for Full Inference

We evaluate the overall SDC resilience for the full inference of the benchmark networks using the proposed full hardening solutions (see Sec. III-E). The findings are reported in Table VI.

Run-time efficient SIHFT compared with baseline. As can be seen, our proposed TinyML solution provides competitive SDC resilience at lower RTO across all studied benchmarks. The majority of the remaining observed SDCs are caused due to soft errors in unprotected firmware modules such as the TFLM run-time, and model I/O setup.

Selective EPI variant. Table VI also reports, for the studied CNNs, the rates of *mispredictions*, which represent the probability of inferring wrong classifications in response to soft-errors. As observed, the selective EPI variant constructively leverage the inherent resilience of CNN topologies to offer almost zero loss in prediction accuracy, relative to the full solution, yet with inferior SDC resilience. Further, the selective EPI scheme offers even more than $2\times$ additional savings in RTO.

Memory overhead. Further, we also report the memory overheads incurred by implemented SIHFT solutions on studied TinyML networks. These costs arise primarily due to increased code-size (for redundancy) and increased static data (for checksum) in ROM use. Further, less available registers in ID schemes could lead to more stack usage in RAM space. As seen in Table VI, our solution does not impose high memory requirements ($< 5\%$ for all benchmarks). Further, the incurred memory overheads are inline with the baseline solution.

TABLE VI
EVALUATION OF SIHFT FOR FULL TINYML INFERENCE
SDC-RATE [%], RTO [%], MISPRED.-RATE [%], MEMORY OV. [%]

		Unprt.	Base-line	Ours	Ours (Sel.)
AD	SDC	35.54	0.2011	0.1947	-
	RTO	0	135.8	10.51	-
	Memory ov.	0	2.694	4.768	-
AWW	SDC	17.49	0.0216	0.0287	1.224
	RTO	0	45.05	23.53	11.27
	Mispred.	6.224	0.0018	0.0021	0.0025
	Memory ov.	0	3.533	3.294	2.718
VWW	SDC	13.71	0.1744	0.1750	0.952
	RTO	0	48.92	24.64	12.56
	Mispred.	2.405	0.0028	0.0039	0.0041
	Memory ov.	0	1.613	1.490	1.361
RESNET	SDC	27.81	0.0536	0.0679	1.527
	RTO	0	17.07	14.42	5.949
	Mispred.	7.944	0.0010	0.0011	0.0011
	Memory ov.	0	4.874	4.357	4.182

V. CONCLUSION

In this paper, we presented dedicated SIHFT solutions for soft-error resilience that focus on reducing the associated RTO specifically for TinyML inference.

ACKNOWLEDGMENT

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project Scale4Edge under contract no. 16ME0131.

REFERENCES

[1] O. Goloubeva, M. Rebaudengo, M. Sonza Reorda, and M. Violante, *Software-Implemented Hardware Fault Tolerance*. Springer US, 2006.

[2] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Error detection by duplicated instructions in super-scalar processors," *IEEE Transactions on Reliability*, vol. 51, no. 1, pp. 63–75, 2002.

[3] M. Didehban and A. Shrivastava, "nzdc: A compiler technique for near zero silent data corruption," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[4] U. Sharif, D. Mueller-Gritschneder, and U. Schlichtmann, "Repair: Control flow protection based on register pairing updates for sw-implemented hw fault tolerance," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5s, sep 2021.

[5] S. K. S. Hari, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Making convolutions resilient via algorithm-based error detection techniques," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2546–2558, 2022.

[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

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

[8] J. A. Gunnels, D. S. Katz, E. S. Quintana-Orti, and R. A. V. de Gejin, "Fault-tolerant high-performance matrix multiplication: theory and practice," in *2001 International Conference on Dependable Systems and Networks*, 2001, pp. 47–56.

[9] K. Zhao, S. Di, S. Li, X. Liang, Y. Zhai, J. Chen, K. Ouyang, F. Cappello, and Z. Chen, "Ft-cnn: Algorithm-based fault tolerance for convolutional neural networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1677–1689, 2021.

[10] T. Marty, T. Yuki, and S. Derrien, "Enabling overclocking through algorithm-level error detection," in *2018 International Conference on Field-Programmable Technology (FPT)*, 2018, pp. 174–181.

[11] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 1–13.

[12] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau *et al.*, "Mlperf tiny benchmark," *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[13] "Tensorflow lite for micro-controllers," 2022. [Online]. Available: https://github.com/tensorflow/tflite-micro

[14] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," 2018.

[15] D. Mueller-Gritschneder, K. Devarajegowda, M. Dittrich, W. Ecker, M. Greim, and U. Schlichtmann, "The extendable translating instruction set simulator (etiss) interlinked with an mda framework for fast risc prototyping," in *Proceedings of the 28th International Symposium on Rapid System Prototyping*, ser. RSP '17, 2017, p. 79–84.

[16] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation, Test in Europe Conference and Exhibition*, 2009, pp. 502–506.

[17] H. Schirmeier, C. Borchert, and O. Spinczyk, "Avoiding pitfalls in fault-injection based comparison of program susceptibility to soft errors," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 319–330.