UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

GEANCARLO ABICH

Early Soft Error Reliability Assessment of Convolutional Neural Networks Executing on Resource-constrained IoT Edge Devices

Thesis presented in partial fulfillment of the requirements for the degree of Doctor of Microelectronics

Advisor: Prof. Dr. Ricardo Augusto da Luz Reis
Coadvisor: Prof. Dr. Luciano Ost

Porto Alegre
April 2022

CIP — CATALOGING-IN-PUBLICATION

Abich, Geancarlo

Early Soft Error Reliability Assessment of Convolutional Neural Networks Executing on Resource-constrained IoT Edge Devices / Geancarlo Abich. – Porto Alegre: PGMICRO da UFRGS, 2022.

159 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2022. Advisor: Ricardo Augusto da Luz Reis; Coadvisor: Luciano Ost.

1. Reliability, Modeling, Simulation, Soft Errors, Fault Injection, Virtual Platform Simulator, Microprocessors, Machine Learning, IoT, Mitigation, Neural Networks, Microelectronics. I. Reis, Ricardo Augusto da Luz. II. Ost, Luciano. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos André Bulhões
Vice-Reitora: Prof^{a}. Patricia Pranke
Pró-Reitor de Pós-Graduação: Prof. Júlio Otávio Jardim Barcellos
Diretora do Instituto de Informática: Prof^{a}. Carla Maria Dal Sasso Freitas
Coordenador do PGMICRO: Prof. Tiago Roberto Balen
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

"You may never know what results come from your action, but if you do nothing there will be no results."

— Mahatma Gandhi

AGRADECIMENTOS

Agradeço a Deis em primeiro lugar por me acompanhar em todos os momentos da minha vida, principalmente durante as dificuldades.

A família é a principal estrutura na vida de um grande homem e a minha está sempre ao meu lado. Agradeço a meus pais CARLOS ALBERTO DORNELLES ABICH, vulgo Dornellão, e ELANE ABICH, minha mãe, por me ensinar que com trabalho duro, respeito, dignidade e caráter se atinge qualquer objetivo. Vocês literalmente fizeram um baita doutor.

PAULA ANDRESSA FISCHER, toda vez que eu preciso falar de ti eu sinto que fui abençoado em te ter ao meu lado. Se pudesse desejar algo melhor para esse mundo, desejaria que existissem mais pessoas como você. Só nós sabemos o que passamos nestes últimos anos. Obrigado por existir, e obrigado por estar ao meu lado, sempre, independente dos obstáculos que temos no nosso caminho. A tua companía da mais sentido a minha trajetória. Com esta conquista, até que enfim tu vai poder casar... Te amo.

Caros RICARDO AUGUSTO DA LUZ REIS e LUCIANO COPELLO OST, haja insistência, paciência e café comigo né! Mas vamos aos fatos. À quem ler este pedaço da tese, entenda que essas duas pessoas são incríveis. Não estou falando de curriculo e parte técnica. O Reis é meu segundo pai, literalmente, acho que a admiração que um filho tem por um pai ja é o suficiente para exemplificar o que eu sinto. Ost, é como um irmão mais experiênte (pra não dizer mais velho), que sempre buscou que eu não cometesse os mesmos erros que ele, que eu evoluisse a cada passo, que me dava a mão para levantar a cada tropeço e um pontapé para não ficar resmungando ao invés de trabalhar. Sem contar as viagens, os 20 dias que passei com o Ost em Leicester e os churrascos do grupo. Obrigado por tudo isso. Agora vamos além!

Aos colegas de laboratório. O RAFAEL FRAGA GARIBOTTI, obrigado pelos auxilios nos artigos e pelas conversas, certamente me esclareceram muita coisa com tua experiência de vida profissional e de estudos. JONAS FOGLIARINI GAVA, e sua paciência, valeu pelos auxilios nos artigos e pela parceria. O MATEUS PAIVA FOGAÇA, meu irmão de pós graduação, obrigado pela amizade, conversas e apoio durante todo este trabalho. Ao VITOR VIANA BANDEIRA e ao FELIPE TODESCHINI BORTOLLON, pelas caronas, trabalhos, churras e parceria. Leo Brendler e Leo Barlette pelas parcerias e pelas conversas descontraídas no laboratório. Mais além, Mateus, Alexandra, Calebe, Jucemar, Ygor, Graci, Felipe Rosa, Walter, Guilherme Paim, Leandro Rocha, Gustavo e

Jean Hamerski, pelo apoio e incentivo em todas as horas.

Por fim, agradeço a todos os demais que de alguma forma contribuíram para este trabalho, tanto os que apoiaram quanto os que dificultaram, pois sempre é um incentivo para seguir em frente. Muito obrigado a todos.

ABSTRACT

Machine learning (ML) algorithms have provided straightforward solutions to a wide range of applications. The high computational demand of such algorithms limits their adoption in resource-constrained devices, which typically rely on reduced memory footprint and low-power components (e.g., microcontrollers and processors). While performance improvement, customized, and reduced-precision implementations of ML algorithms have been studied extensively, their susceptibility to soft errors caused by radiation particles is still an open question. In this regard, due to their flexibility and high simulation performance, researchers are using virtual platform (VP) frameworks to assess the soft error reliability of complex systems considering several software stack components running on top of commercial processors. While the gain in simulation speed is trivially observed in VP simulators based on just-in-time (JIT) dynamic binary translation, the soft error assessment consistency of underlying fault injection frameworks remains unclear. In this regard, the *main contribution* of this Thesis is to provide, at early design phases, a consistent and extensive soft error reliability assessment of ML algorithms developed with specialized libraries that enable the execution of such applications in resource-constrained Arm processors. The *first goal* of this Thesis is to analyze the consistency of the soft error reliability assessment of a JIT-based fault injection framework (SOFIA) against fault injection campaigns conducted with event-driven simulators (i.e., more realistic and accurate platforms) considering single-processor architectures. Considering the consistency of the results conducted with SOFIA, the *second goal* of this Thesis is to early investigate and identify the correlation between fault injection results, NN optimized kernels, and reduced precision parameters of convolutional neural networks (CNNs) executing on resource-constrained IoT devices. Such a study aims at evaluating the balance between relative performance and reliability to promote the use of software-based mitigation techniques to improve soft error reliability. Understanding that adopted CNNs are vulnerable to soft errors, the *third goal* of this Thesis is to evaluate the impact of soft errors in the code, parameters, and data stored in the memory units of IoT edge devices considering the optimized libraries and the reduced precision used in such ML models. Besides that, we also developed a parallel CNN version as an attempt to increase performance while evaluating the impact of multi-threaded parallelism in the soft error reliability w.r.t. the original sequential version. In this sense, the results conducted in this Thesis comprise more than 14.8 million of fault injections considering distinct case studies, architectures,

number of cores, OSs, and parallelization libraries. The consistency evaluation shows that SOFIA is more than 1000× faster than cycle-accurate simulators while preserving the soft error analysis accuracy (i.e., mismatch below to 10%).The early soft error reliability assessment of CNN executing on resource-constrained IoT Edge devices shows that the occurrence of critical faults varies depending on the instruction set architecture, the layer where the faults are injected, and the precision bitwidth of the convolutional layers. With that in mind, promoting the lightweight register allocation mitigation technique (RAT) gives the best relative performance, memory utilization, and soft error reliability trade-offs w.r.t. a more traditional replication-based mitigation approach. Furthermore, results from fault injections in memory sections show that stored binaries and trained parameters tend to have more critical faults than register files. Moreover, this Thesis's contributions enable us to advance our study to different multiprocessor platforms to gain performance while maintaining low energy costs during execution, which implies different reliability parameters to be considered both in execution and in data stored in memory.

Keywords: Reliability, Modeling, Simulation, Soft Errors, Fault Injection, Virtual Platform Simulator, Microprocessors, Machine Learning, IoT, Mitigation, Neural Networks, Microelectronics.

Avaliação antecipada da influência de erros transientes em redes neurais convolucionais executando em dispositivos IoT Edge de recursos limitados

RESUMO

Os algoritmos de aprendizado de máquina (ML) têm fornecido soluções diretas para uma ampla gama de aplicações. A alta demanda computacional de tais algoritmos limita sua adoção em dispositivos com restrição de recursos, os quais normalmente são constituídos por memória reduzida e componentes de baixo consumo de energia (por exemplo, microcontroladores e processadores). Embora implementações personalizadas, melhorias de desempenho e precisão reduzida de modelos de ML tenham sido estudadas extensivamente, sua suscetibilidade a erros transientes causados por partículas de radiação ainda é uma questão em aberto. Nesse sentido, devido à sua flexibilidade e alto desempenho de simulação, os pesquisadores estão usando *frameworks* baseados em plataformas virtuais (VPs) para avaliar a confiabilidade de sistemas complexos expostos a erros temporários, considerando vários componentes de pilha de software rodando em processadores e microcontroladores comerciais. Embora o ganho na velocidade de simulação seja observado trivialmente em simuladores VP baseados em tradução binária dinâmica *just-in-time* (JIT), a consistência da avaliação de erros temporários dos *frameworks* de injeção de falha subjacentes permanece incerta. Nesse sentido, a *principal contribuição* desta Tese é permitir, em fases iniciais de projeto, uma avaliação consistente e extensa da suscetibilidade à erros transientes de modelos de ML desenvolvidos com bibliotecas especializadas que permitem sua execução em processadores Arm com recursos limitados. Neste contexto, o *primeiro objetivo* desta Tese é analisar a consistência da avaliação de ocorrência erros transientes de um *framework* de injeção de falhas baseado em JIT (SOFIA) comparando com campanhas de injeção de falha conduzidas com simuladores orientados a eventos (isto é, plataformas mais realistas e precisas) considerando arquiteturas de um único processador. Considerando a consistência dos resultados conduzidos com SOFIA, o *segundo objetivo* desta Tese é investigar e identificar a correlação entre os resultados de injeção de falha, bibliotecas NN otimizadas e parâmetros de precisão reduzida de redes neurais convolucionais (CNNs) executando em dispositivos IoT com recursos limitados. Este estudo visa avaliar o equilíbrio entre desempenho relativo e confiabilidade para promover o uso de técnicas de mitigação baseadas em software para melhorar a confiabilidade destes modelos de ML. Compreendendo que as CNNs adotadas são vulneráveis à erros

transientes, o *terceiro objetivo* desta Tese é avaliar o impacto das falhas no código, parametros e dados armazenados nas unidades de memória destes dispositivos considerando as bibliotecas otimizadas e a precisão reduzida utilizada em tais modelos de ML. Além disso, neste trabalho também foi desenvolvido uma versão paralela da CNN como uma tentativa de aumentar o desempenho e avaliar o impacto do paralelismo multi-thread na susceptibilidade a erros transientes comparando com a versão sequencial original. Nesse sentido, os resultados conduzidos nesta Tese compreendem mais de 14,8 milhões de injeções de falhas considerando distintos estudos de caso, arquiteturas, número de núcleos, OSs, e bibliotecas de paralelização. A avaliação de consistência mostrou que o SOFIA é mais de 1000× mais rápido do que os simuladores com precisão de ciclo, preservando a precisão da análise de susceptibilidade a erros transientes (ou seja, diferença abaixo de 10%). A avaliação inicial da susceptibilidade a erros transientes da CNN executando em dispositivos IoT da borda com recursos limitados mostra que a ocorrência de falhas críticas varia dependendo da arquitetura do conjunto de instruções, da camada em que as falhas são injetadas e da largura de bits de precisão das camadas convolucionais. Com isso em mente, a promoção da técnica de mitigação baseada na alocação de registradores (RAT) oferece o melhor desempenho relativo, utilização de memória e compensações de confiabilidade em comparação com uma abordagem de mitigação mais tradicional baseada em replicação. Além disso, os resultados de injeções de falhas em seções de memória mostram que códigos binários e parâmetros pré-treinados armazenados nas unidades de memória tendem a ter mais falhas críticas do que o banco de registradores. Além disso, as contribuições desta Tese permitem avançar o estudo da susceptibilidade a erros transientes para diferentes plataformas multiprocessadoras visando ganhar desempenho mantendo baixos custos de energia durante a execução, o que implica em diferentes parâmetros de confiabilidade a serem considerados tanto na execução quanto nos dados armazenados em memória.

Palavras-chave: Confiabilidade, Modelage, Simulação, Falhas Temporárias, Injeção de Falhas, Simulador de Plataforma Virtual, Processadores Comerciais, Aprendizado de Máquina, Internet das Coisas, Mitigação.

LIST OF ABBREVIATIONS AND ACRONYMS

ABFT	Algorithm-Based Fault Tolerance
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interfaces
Arm	Advanced RISC Machines
ASIC	Application-Specific Integrated Circuits
AVF	Architecture Vulnerability Factor
AVX	Advanced Vector Extensions
BITFLIPS	Basic Instrumentation Tool for Fault Localized Injection of Probabilistic SEUs
BNN	Binarized Neural Network
CIFAR	Canadian Institute for Advanced Research
CMOS	Complementary Metal-Oxide-Semiconductor
CMSIS	Common Microcontroller Software Interface Standard
CMSIS-NN	CMSIS API for Neural Networks
CMix-NN	CMSIS-based Mixed precision API for Neural Networks
CMR	Conditional Modular Redundancy
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DFI	Direct Fault Injection
DNN	Deep Neural Network
DSP	Digital Signal Processor

ECC	Error Correcting Code
EDDI	Error Detection by Duplicated Instructions
EAFC	Extrapolated Absolute Failure Count
FinFET	Fin Field Effect Transistor
FDSOI	Fully Depleted Silicon On Insulator
FI	Fault Injections
FIM	Fault Injection Modules
FINN	Fast Inference for binarized Neural Networks
FIT	Failure-in-Time
FPGA	Field-Programmable Gate Arrays
FPU	Floating-Point Unit
GCC	GNU Compiler Collection
GEMS	General Execution-driven Multiprocessor Simulator
GPU	Graphic Processing Units
HDL	Hardware Description Languages
HPC	High Performance Computers
HWC	Height-Width-Channel
IBM	International Business Machines
ICN	Inter-Channel Normalization
IoT	Internet of Things
IR	Intermediate Representation
ISA	Instruction Set Architectures
JIT	Just-In-Time
kNN	K-Nearest Neighbor

KIPS	Kilo Instructions Per Second
LLVM	Low Level Virtual Machine
MAC	Multiply-And-Accumulate
MARSS	MIPS Assembler and Runtime Simulator
MIPS	Million Instructions Per Second
ML	Machine Learning
MMU	Memory Management Unit
MNIST	Modified National Institute of Standards and Technology
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MPI	Message Passing Interface
MWTF	Mean Work To Failure
M*DEV	Multicore/Multiprocessor Software Development Kit
NLP	Natural Language Processing
NN	Neural Networks
NPB	NAS Parallel Benchmark
nZDC	near Zero silent Data Corruption
OpenMP	Open Multi-Processing
OMM	Output Memory Mismatch
ONA	Output Not Affected
OS	Operating Systems
OVPsim	Open Virtual Platform Simulator
PC	Per-Channel
PL	Per-Layer
P-TMR	Partial Triple Modular Redundancy

QCL	Quantized Convolutional Layer
QEMU	Quick Emulator
RAM	Random Access Memory
RAT	Register Allocation Technique
RECCO	REliable Code COmpiler
ResNet	Residual Network
ReLU	Rectified Linear activation Unit
RPi2	Raspberry Pi 2
RTOS	Real-Time Operating System
RTL	Register-Transfer Level
SASSIFI	SASSI-based Fault Injector
SBU	Single Bit Upsets
SDC	Silent Data Corruptions
SEB	Single Event Burnout
SEE	Single Event Effects
SEGR	Single Event Gate Rupture
SEL	Single Event Latchup
SET	Single Event Transient
SEU	Single Event Upset
SIMD	Single Instruction Multiple Data
SNN	Spiking Neural Network
SMLAD	Signed Multiply Accumulate Long Dual
SOFIA	Soft error Fault Injection Analysis
SPARC	Scalable Processor Architecture

SVM	Support Vector Machine
SWAR	SIMD Within a Register
SWIFI	Software Implemented Fault Injection
SWIFT	SoftWare Implemented Fault Tolerance
TMR	Triple Modular Redundancy
TPU	Tensor Processing Unit
UART	Universal Asynchronous Receiver Transmitter
UT	Unexpected Termination
VP	Virtual Platforms
WCET	Worst-Case Execution Time

LIST OF FIGURES

Figure 1.1 Contributions of this Thesis considering the soft error assessment flow in the SOFIA framework..27

Figure 2.1 Different types and applications of ML exploited for solving computer science problems..32
Figure 2.2 Bias and variance using bulls-eye diagram..34
Figure 2.3 Model complexity curve defined by Equation (2.1) 35
Figure 2.4 Examples of clustering algorithms. 36
Figure 2.5 Example of dimensionality reduction using matrix decomposition technique.37
Figure 2.6 1D Kernel density estimation example. 38
Figure 2.7 Trade-off between approaches used to enable deep inference in resource-constrained IoT Edge devices...39
Figure 2.8 Sketch of the Earth's radiation environment...41
Figure 2.9 SEU and SET effects occurring in a circuit example. 44

Figure 4.1 Fault injection campaign flow applicable to the three fault injection approaches, i.e., RTL, gem5 and SOFIA. 65
Figure 4.2 Fault injection types available in SOFIA...66
Figure 4.3 Proposed extension in the physical memory FI technique............................69

Figure 5.1 Speedup of SOFIA over the RTL-FIM, considering the Cortex-M0 executing 26 benchmarks in bare-metal...76
Figure 5.2 Boxplot of fault mismatch between SOFIA and RTL-Fault Injection Modules (FIM) considering different architectures and software stacks. Outlier cases (represented by circles) are at least two standard deviations from the mean...78
Figure 5.3 Average faults for compilers and their optimization flags. The red line represents the average number of Vanished faults. 81
Figure 5.4 Average vanished faults for compilers and their optimization flags. 82
Figure 5.5 Average execution time and executed instructions *vs* number of Vanishes considering different compiler sets. 82

Figure 6.1 Brief illustration of the adopted CIFAR-10 CNN topology..........................85
Figure 6.2 Overview of the CMSIS-NN kernel structure. 86
Figure 6.3 CNN timeline with CMSIS-NN Kernel executing on Cortex-M3 and Cortex-M4 processors...89
Figure 6.4 Fault classification when injecting faults in the CNN Layers........................90
Figure 6.5 Normalized MWTF for fault injections in different layers of CNN in Cortex-M3...91
Figure 6.6 Results showing fault classifications for Register File, Flash, and RAM memory sections for the CIFAR-10 CNN considering two Arm processors..........93
Figure 6.7 Fault effects on Flash memory sections considering the CNN object code. .95
Figure 6.8 Fault classifications for CIFAR-10 CNN parameters and data stored in Flash and RAM memory sections for Arm Cortex-M3 and M4 processors...........96
Figure 6.9 CMSIS-NN extension to provide a parallel convolution kernel. 98
Figure 6.10 Speedup results for 1, 2, 3 and 4 thread implementations for Thumb and SIMD instruction sets...99
Figure 6.11 Results from FI campaigns evaluating the thread parallelism with single, dual, and quad-core configurations. 101

Figure 6.12 Brief description of the MobileNet CNN topology. 103
Figure 6.13 CMix-NN quantized convolutional layer. 105
Figure 6.14 CNN timeline with CMix-NN library executing on Arm Cortex-M7 processor. 107
Figure 6.15 Results from fault injection campaigns when injecting faults in the CNN layers with different precision bitwidth configurations. 108
Figure 6.16 Results showing fault classifications for Register File, Flash, and RAM memory sections for the MobileNet CNN according to the precision bitwidth. .. 112
Figure 6.17 Relative trade-off between the normalised *Mean Work To Failure* (MWTF) and memory-saving, considering different precision bitwidth configurations, comparing *Code MWTF* (CM), *Parameters MWTF* (PM), *Data MWTF* (DM), *Flash-Saving* (FS), and *RAM-Saving* (RS). 113
Figure 6.18 Results showing fault classifications comparing MobileNet CNN without protection, with P-TMR, and RAT mitigation techniques. The red dots indicate the normalized MWTF (right y-axis). 118
Figure 6.19 Results of fault classification by layer of the two most affected precision bitwidth configurations. 119
Figure 6.20 MobileNet execution time overhead considering P-TMR and RAT. 120
Figure 6.21 Relative trade-off between P-TMR and RAT mitigation techniques considering *w8a4* and *w8a8* precision bitwidth configurations, comparing *Mean Work To Failure* (MWTF), *Performance Overhead* (PO), *Footprint Overhead* (FO), *Accuracy* (AC), and *Tolerable Faults* (TF). 121

Figure A.1 M*DEV scheduling policy varying the quantum size for a dual-core processor executing the same workload. 145
Figure A.2 Simulation speedup and scalability of the two virtual platforms, showing the performance gain achieved by using the SOFIA. 146
Figure A.3 Mismatch between gem5-FIM and SOFIA varying the number of cores in Arm Cortex-A9 while executing the Rodinia Benchmarks. 148
Figure A.4 Mismatch between gem5-FIM and SOFIA varying the number of cores in Arm Cortex-A9 while executing NPB Benchmarks. 149
Figure A.5 Mismatch between gem5-FIM and SOFIA considering different programming models. 151
Figure A.6 Mismatch between the gem5-FIM (ψ) and SOFIA quantum sizes: 448,000 (λ); 4,480 (γ); 448 (β); and 44 (δ). 154
Figure A.7 Multi-core mismatch between gem5-FIM and SOFIA with smallest quantum size (a 44-instruction block). 157

LIST OF TABLES

Table 3.1 Related works considering virtual platform fault injection simulators. 49
Table 3.2 Related works considering soft error assessment of ML techniques. 61

Table 5.1 RTL vs. SOFIA Experimental Setup. The * means a sub-flag used in conjunction with other standard flags (e.g., O0, O1, O2, O3). 75
Table 5.2 Mean and worst-case absolute mismatch percentages for different cross-compilers and their optimization flags in Arm Cortex-M3. The * means a combined flag equivalent to *Ofast*...81

Table 6.1 Contributions organized by Case study. 84
Table 6.2 Layer parameters for the CIFAR-10 CNN..86
Table 6.3 Experimental Setup 88
Table 6.4 Percentages of tolerable and critical faults in the CNN.................................91
Table 6.5 Experimental Setup 92
Table 6.6 On-chip Memory Occupation (MO) for the inference CNN models, considering code, parameters and data 93
Table 6.7 Percentages of memory occupation (MO), execution lifespan (ELS), and classification of faults occurring on CIFAR-10 CNN functions object code stored in Flash memory section 94
Table 6.8 CNN runtime percentages according to layer type...97
Table 6.9 Validation Experimental Setup..99
Table 6.10 FI Experimental Setup..100
Table 6.11 Layer parameters for the Mobilenet CNN. 103
Table 6.12 Experimental Setup. 106
Table 6.13 Percentages of tolerable and critical faults in the CNN considering different precision bitwidth configurations. 109
Table 6.14 Experimental Setup 111
Table 6.15 On-chip Memory Occupation (MO) for 8-bit precision inference CNN models, considering code, parameters and data..111
Table 6.16 Experimental Setup 114
Table 6.17 Percentages of tolerable and critical faults on MobileNet CNN considering different precision bitwidth configurations..116
Table 6.18 Normalized MobileNet footprint overhead when applying soft error mitigation techniques. 120
Table 6.19 Percentage of use and relative increase in executed instructions considering different instruction sets..122

Table 7.1 Publications Summary..126

Table A.1 gem5 vs. SOFIA Experimental Setup. 143
Table A.2 Mismatch comparison of programming models of SOFIA with default (DF) and small quantum size (Q) in relation to gem5-FIM. 155

CONTENTS

1 INTRODUCTION..20
1.1 Hypothesis to be demonstrated in Thesis..24
1.2 Thesis Goal..25
1.3 Original Contributions of this Thesis..26
1.3.1 Evaluation of SOFIA consistency w.r.t. RTL..27
1.3.2 Early soft error assessment of ML models executing on Resource-constrained IoT edge devices..28
1.3.2.1 Applying Lightweight Soft Error Mitigation Techniques to Embedded Mixed Precision Deep Neural Networks..29
1.3.2.2 Consistent and Extensive Evaluation..29
1.3.3 Legacy of tools integrated into the SOFIA framework..30
1.4 Thesis Outline..30
2 BACKGROUND IN ML MODELS AND RADIATION EFFECTS..32
2.1 Basic Concepts of ML models..32
2.1.1 Supervised learning..33
2.1.2 Unsupervised learning..36
2.1.3 ML for the IoT Edge devices..39
2.2 Radiation Environment an its Effects on Semiconductors Devices..41
2.2.1 Classification of SEEs..42
2.2.1.1 Hard Errors..43
2.2.1.2 Soft Errors..44
3 RELATED WORKS..46
3.1 Soft Error Assesment Considering Virtual Platforms..46
3.1.1 Contribution in Soft Error Assessment Considering Virtual Platforms..50
3.2 Soft Error Reliability Assessment of Machine Learning Techniques..52
3.2.1 Review of System-level Soft Error Mitigation Techniques..57
3.2.2 Contribution in Machine Learning Soft Error Assessment and Mitigation..59
4 SOFT ERROR ASSESSMENT METHODOLOGY..64
4.1 Fault Injection Frameworks..64
4.1.1 RTL Fault Injection Module..64
4.1.2 SOFIA Framework..65
4.1.2.1 SOFIA: gem5 Fault Injection Module..67
4.1.2.2 SOFIA: OVPsim Fault Injection Module..67
4.2 Fault Classification..69
4.3 Assessment Metrics..71
4.3.1 Supported Soft Error Mitigation Techniques..72
5 EARLY SOFT ERROR CONSISTENCY ASSESSMENT..74
5.1 Soft Error Consistency Assessment for Single-core Processors..74
5.1.1 Experimental Setup..74
5.1.2 FI Simulation Performance of SOFIA w.r.t. RTL..75
5.1.3 Soft Error Reliability Mismatch..76
5.1.3.1 Mismatch Analysis Considering Different Processor Architectures..77
5.1.3.2 Mismatch Analysis Considering Cross-compilers..79
5.1.4 Closing Remarks..83

6 SOFT ERROR RELIABILITY ASSESSMENT OF ML INFERENCE MODELS EXECUTING ON RESOURCE-CONSTRAINED IOT EDGE DEVICES....84
6.1 Soft Error Reliability Assessment of the CIFAR-10 CNN....85
6.1.1 CIFAR-10 CNN Developed with CMSIS-NN....85
6.1.2 Soft Error Reliability Assessment of CIFAR-10 CNN Execution on Resource-constrained IoT Devices....87
6.1.2.1 Experimental Setup....87
6.1.2.2 CIFAR-10 CNN Execution Lifetime....88
6.1.2.3 CIFAR-10 CNN Soft Error Reliability Assessment....89
6.1.3 The Impact of Soft Errors in Memory Units of Edge Devices Executing CIFAR-10 CNN....92
6.1.3.1 Experimental Setup....92
6.1.3.2 Soft Error Reliability Analysis....93
6.1.4 Impact of Thread Parallelism on the Soft Error Reliability of CIFAR-10 CNN....96
6.1.4.1 Parallel Convolution Kernels....97
6.1.4.2 Multi-thread Speedup Analysis Experimental Setup....98
6.1.4.3 Soft Error Reliability Analysis Experimental Setup....99
6.1.4.4 Soft Error Reliability Assessment....100
6.2 Soft Error Reliability Assessment of the MobileNet CNN....102
6.2.1 MobileNet CNN Developed with CMix-NN....102
6.2.2 The Impact of Precision Bitwidth on the Soft Error Reliability of the MobileNet CNN Execution on Resource-constrained IoT Devices....105
6.2.2.1 Experimental Setup....105
6.2.2.2 MobileNet CNN Execution Lifetime....106
6.2.2.3 Soft Error Reliability Analysis....108
6.2.3 The Impact of Soft Errors in Memory Units of Edge Devices Executing the MobileNet CNN....110
6.2.3.1 Experimental Setup....110
6.2.3.2 Soft Error Reliability Analysis....110
6.2.4 Applying Lightweight Soft Error Mitigation Techniques to Embedded Mixed Precision Deep Neural Networks....113
6.2.4.1 Experimental Setup....114
6.2.4.2 Soft Error Reliability Assessment of the MobileNet CNN Considering the Precision Bitwidth....115
6.2.4.3 Applying mitigation techniques to MobileNet CNN....117
6.2.4.4 Trade-off Between Performance and Reliability....118
7 CONCLUSIONS AND FUTURE WORK....123
REFERENCES....127
APPENDIX A — SOFT ERROR CONSISTENCY ASSESSMENT FOR MULTICORE PROCESSORS....142
A.1 Experimental Setup....142
A.2 FI Simulation Performance of SOFIA w.r.t. gem5-FIM....143
A.3 Soft Error Mismatch Analysis....147
A.3.1 Mismatch Analysis Considering the Number of Cores....147
A.3.2 Mismatch Analysis Considering Parallel Programming Models....150
A.3.3 Mismatch Analysis Considering the SOFIA Quantum Parameter....153
A.4 Closing Remarks....156
APPENDIX B — LIST OF WORKS PUBLISHED BY THE AUTHOR....158

1 INTRODUCTION

Computing is a reality in human daily lives since computers are used from wearable devices to complex High Performance Computers (HPC). Relevant improvements in internet protocols and the computational efficiency of emerging technologies have made communication between different devices more accessible than before (MAHDAVINEJAD et al., 2018). Recent reports, from communication technology enterprises (CISCO, 2021) estimate that there will be around 25 to 32 billion devices connected to the Internet in the coming years (i.e., ~ 3.6 networked devices *per capita* by 2023). However, the popularity of such systems generated a high demand for data to be processed in a timely manner (LABRINIDIS; JAGADISH, 2012; NAJAFABADI et al., 2015). In this sense, researchers started to use Machine Learning (ML) models to identify patterns in large-scale datasets, aiming to classify or predict the behaviour of complex systems in several fields, such as Natural Language Processing (NLP), autonomous driving, and smart healthcare (SAMUEL, 1959) (LI; OTA; DONG, 2018). ML is a branch of Artificial Intelligence (AI), which comprises the study of computer algorithms that are used to improve performance through automated recognition of patterns and regularities in data sets (SAMUEL, 1959). Technology giants such as Google, Microsoft, Facebook, Amazon, Nvidia, and others have invested heavily with their powerful computing resources to boost AI research, mainly aiming at breakthroughs and improving ML techniques. Such research efforts evince that ML models will improve both current and next generation of computing systems.

While emerging learning techniques fulfill a broad range of applications, the accuracy of underlying ML models comes at the expense of high computational and memory requirements for both the training and the inference phases. Training a ML model is space and computationally expensive due to the high set of parameters, which are iteratively refined over multiple executions. An inference model might still be complex due to the potentially high density of the input data (e.g., a high-resolution image) and the vast number of computations that need to be performed during its execution. Even with such computational requirements, recent works denote an emerging trend in creating inference versions of ML models that aim at enabling their execution in edge devices, which rely on some security, reliability, resource, and power constraints (CHEN; RAN, 2019).

ML models are used in complex safety-critical systems (e.g., self-driving vehicles) and have, more recently, been incorporated in resource-constrained Internet of Things (IoT)

edge devices. IoT edge devices combine embedded technologies such as wired and wireless communications, sensor and actuator devices, and the physical objects connected to the internet (ATZORI; IERA; MORABITO, 2010; CECCHINEL et al., 2014). Such systems are able to access raw data from different resources over the network and analyze a high set of information to extract knowledge (MAHDAVINEJAD et al., 2018). Currently, the ML models most employed in resource-constrained IoT systems are the pre-trained Neural Networks (NN), such as Convolutional Neural Networks (CNN)s, since they do not perform the complex training phase on the target device (JACOB et al., June 2018). For instance, CNNs are used to detect, identify, or classify patterns at the edge of IoT world (KHAN et al., 2019). However, apart from robust algorithms and high-performance hardware, executing such complex algorithms under resource-constrained devices is still a challenging task (ADI et al., 2020).

Aiming to enable the execution of such computationally intensive CNNs under resource-constrained IoT edge devices, researchers and industrial leaders are investigating software libraries and Application Programming Interfaces (API)s (LEE et al., 2016). Such approaches rely on the development of lightweight and optimized NN kernels, allowing their execution on low-power and less efficient processors typically found in edge devices. Besides that, the inference of such kernels is based on reduced precision quantization schemes that exploits the trade-off between accuracy and data size to reduce the memory footprint (JACOB et al., June 2018). In this sense, the main challenges to enable the execution of CNNs in such devices are the follows:

- (i) reduce the memory footprint, considering both application object code and CNN memory parameters;
- (ii) provide optimised and accurate NN inference models taking into account the architecture particularities, such as Single Instruction Multiple Data (SIMD);
- (iii) validate the resultant model in real world environment considering a target IoT edge device.

Although the broad applicability provided by such lightweight and optimized approaches, their changes need to meet performance and reliability requirements considering different environments.

Challenges in existing CNNs that targets the execution in IoT systems must consider several threats that affect both the reliability and the efficiency of the entire system (PUNITHAVATHI et al., 2019). The deployment of such models in market leader

IoT edge devices requires robustness features, since soft errors and system malfunctions might have potentially fatal implications in safety-critical application areas (KRIEBEL et al., 2018). For example, while the slight inaccuracy in ML models like NLP does not have any severe consequences, a small error in safety-critical applications like autonomous driving vehicles and smart healthcare can lead to catastrophic effects. In contrast with high-accuracy ML models, there is a significant need for robust CNN models that can generate reliable and trustworthy results in the presence of faults while also preserving safety and healthy. For this reason, software engineers must develop lightweight and reliable applications aiming to guarantee a fail-safe critical IoT edge system. In this sense, it is essential to evaluate the soft error susceptibility of such applications executing on resource-constrained IoT edge devices. With that in mind, researchers have started to investigate the impact of radiation-induced soft errors on the reliability of ML models.

In the reliability aspect, the soft error resilience is emerging as a key design metric due to the increasing susceptibility of electronic computer systems to the occurrence of soft errors caused by radiation effects (BAUMANN, 2005). In IoT edge computing, one open problem is how to reliably mine real-world data from a noisy and complex environment that confuses conventional CNNs (LI; OTA; DONG, 2018). The complexity of CNNs executing on IoT edge systems impose both software and hardware exploration challenges, including:

- (*i*) conduct a large number of Fault Injections (FI) campaigns within a reasonable time;
- (*ii*) provide engineers with detailed observation of a system's behavior in the presence of soft errors occurring both system execution (i.e., registers) and memory parameters (i.e., Flash and RAM);
- (*iii*) investigate the impact of soft errors in optimized kernel characteristics according to platform specific parameters;
- (*iv*) investigate the impact of soft errors in reduced precision NN memory parameters;
- (*v*) investigate the impact of soft errors when considering thread parallelism optimizations targeting existing NN kernels;
- (*vi*) identify relationships or associations between application characteristics and specific platform parameters in large data sets resulting from the fault campaigns.

The resulting scenario calls for faster and more efficient means to assess the soft error

resilience of such complex systems with minimal overhead in time-to-market (MUKHERJEE; EMER; REINHARDT, 2005; LI et al., 2007; CHATZIDIMITRIOU et al., 2019; ROSA et al., 2019; BODMANN et al., 2021; ABICH et al., 2021).

With that in mind, current researches are proposing virtual platform FI frameworks to cover such response time constraints (HARI et al., 2012; KALIORAKIS et al., 2015; TANIKELLA et al., 2016). Virtual Platforms (VP) frameworks have gained popularity over the years, not only in academia but also in many industrial sectors, due to their design flexibility, debugging, and simulation performance capacities. In this sense, works incorporated FI capability into VP frameworks (PARASYRIS et al., 2014; ROSA et al., 2018), enabling the analysis of complex software stacks and processor architectures at early design phases. However, to ensure the failsafe functionality of emerging resource-constrained IoT systems, engineers should be able to not only assess and identify, but also to promote efficient alternatives to mitigate the occurrence of soft errors (GAVA; REIS; OST, 2020). Furthermore, while the gain in simulation speed is trivially observed in VP simulators based on Just-In-Time (JIT) dynamic binary translation, the soft error assessment consistency of underlying FI frameworks remains unclear. The preceding context motivates this Thesis, which aims at investigating the consistency of JIT-based FI techniques and use these tools to assess and mitigate the occurrence of soft errors in resource-constrained IoT devices executing CNN models.

This work first focuses on elucidating the consistency of the results gathered with the Soft error Fault Injection Analysis (SOFIA) framework (BANDEIRA et al., 2019), when compared to an Register-Transfer Level (RTL) FI approach. With this consistency assessment, this thesis thus focuses on enhancing SOFIA capability by proposing two extensions: (*i*) a realistic fault classification according to CNN output predictions; and (*ii*) an automated FI technique that isolates specific memory sections in order to evaluate the impact of soft errors on different memory parameters of CNN models executing on resource-constrained IoT edge devices. The proposed extensions move SOFIA beyond the traditional frameworks, thereby furthering potential advantages that enable engineers to evaluate specific aspects of emerging CNN models. Furthermore, this work extends existing NN kernels employing parallel capabilities in order to evaluate the impact of thread parallelism on the performance and reliability of an CNN model executing on IoT edge device. Finally, this Thesis presents different case studies that evaluate the relative trade-off between different soft error mitigation techniques considering reliability, performance, and precision of the adopted CNN model.

1.1 Hypothesis to be demonstrated in Thesis

This Thesis relies on three hypotheses:

- The first hypothesis of this Thesis is that virtual platform FI frameworks provide truthful results with significant performance gains when compared to the most classical ones. A fast and consistent FI tool enable engineers to ensure the accuracy of soft errors and failures to assess the reliability of complex computing systems (i.e., real software stacks, state-of-the-art Instruction Set Architectures (ISA)s), ensuring a realistic results at early design phases. Furthermore, the proof of this hypothesis allow us to enable us to provide a consistent assessment of the soft error reliability of CNN models, which would be unfeasible in traditional models due to the simulation time.
- With the use of architecture specific kernel optimizations as well as reduced precision to enable the execution of CNN models on IoT systems, engineers are more likely to identify meaningful relationships or associations between FI results, software optimizations, and parameters quantizations. In this regard, the second hypothesis of this thesis is to early investigate and identify the correlation between FI results, NN optimized kernels, and reduced precision parameters of CNNs executing on resource-constrained IoT edge devices. The proof of this hypothesis allow us to distinguish soft errors occurring in such models and apply lightweight and non-intrusive mitigation techniques to improve the reliability and lifetime of CNN models executing on resource-constrained IoT edge systems.
- When dealing with those parameters, the existing evaluation techniques in the literature are not sufficient to correlate the real impact of faults in the soft error reliability of CNN models considering the IoT device storage. With that in mind, the third hypothesis of this Thesis is that the reduced precision optimizations impacts not only in the CNN execution, but also in the CNN code, parameters, and data stored in memory. The proof of such hypothesis allow us to isolate and identify the most vulnerable memory sections of a CNN deployed to a resource-constrained IoT edge device.

1.2 Thesis Goal

In order to address the reliability of ML models using VPs, the strategic goal of this Thesis is first to perform an in-depth and statistical significance soft error consistency evaluation of a JIT-based FI framework called SOFIA (BANDEIRA et al., 2019; ROSA et al., 2019). This work aims to further contribute to the use of virtual platform FI frameworks in the design flow of safety-critical industry applications considering soft error reliability aspect. The second strategic goal of this Thesis is to combine existing features support by SOFIA (e.g., FI techniques) along with new tools and mitigation techniques aiming to assess the soft error reliability of CNN models executing on resource-constrained IoT edge devices considering different case studies, model optimizations (e.g., precision quantization, thread parallelism), processor models and ISAs. Finally, the third strategic goal of this thesis comprises to evaluate the impact of soft errors on memory units of IoT edge devices executing reduced precision ML models.

The following specific objectives should be fulfilled to accomplish the first strategic goal:

- Port several benchmarks from embedded and IoT domains, considering standard open-source Clang and GNU Compiler Collection (GCC) compilers and commercial Advanced RISC Machines (Arm) compilers, along with standard optimization flags to cover a wide range of probabilities in soft error consistency evaluation.
- Port FI techniques and scripts that enable us to trace, evaluate, and identify the particular source of errors in RTL processor descriptions.
- Evaluate the soft error assessment consistency of a VP FI tool w.r.t. a real single-core commercial processors at the RTL with a discrete event full system simulator.

The second goal requires the following task to be achieved:

- Port reduced and mixed precision CNN models for the SOFIA environment to enable the soft error assessment with a significant number of FI campaigns.
- Propose a realistic fault classification aiming to evaluate the real impact on the output parameters of CNN models.
- Investigate the impact of reduced precision and architecture specific optimizations on soft error reliability of CNN models executing on resource-constrained IoT systems.
- Employ the use of soft error software-based mitigation techniques to improve the reliability of CNN models executing on resource-constrained IoT systems.

- Assess both performance and reliability impact when using lightweight and non-intrusive software-based mitigation techniques in such CNN models.
- Evaluate the relative trade-off between soft error software-based mitigation techniques considering performance, reliability, and precision.

Finally, the third goal comprises the following milestones:

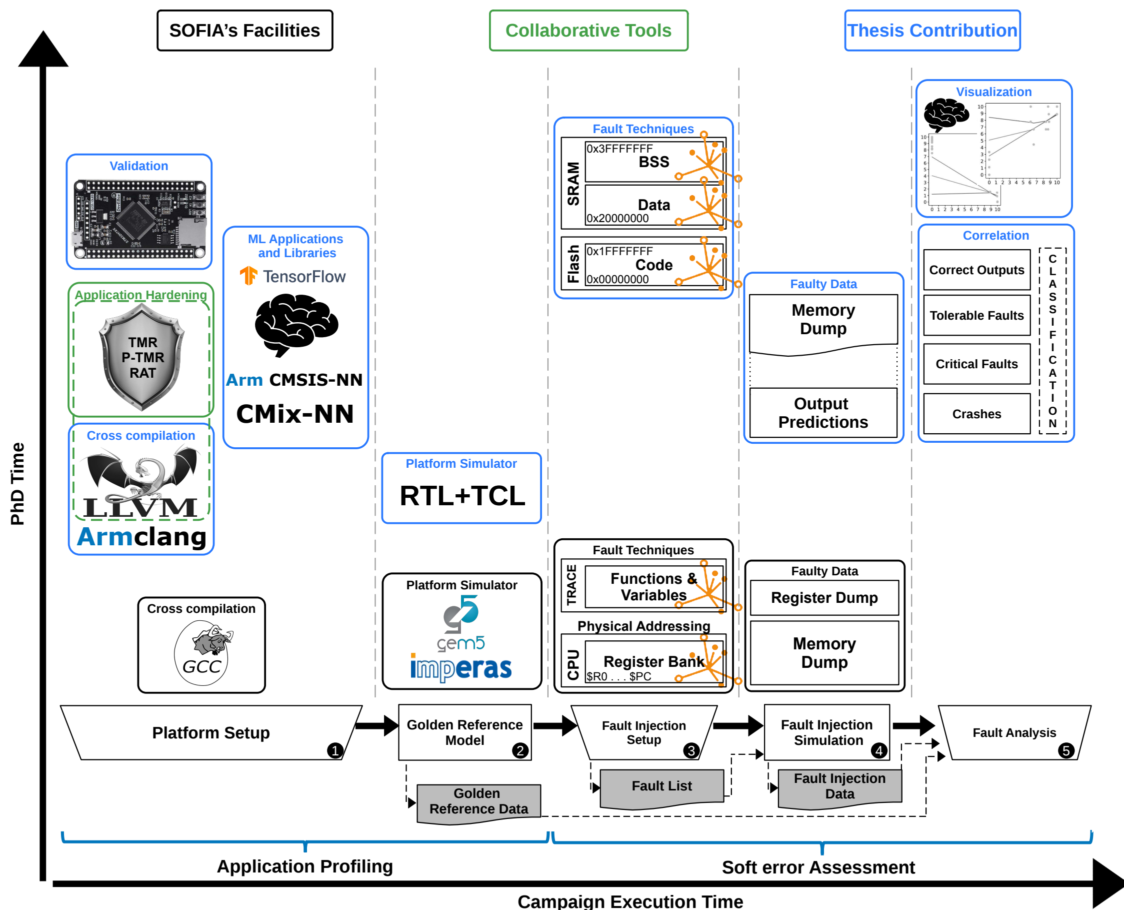
- Evaluate different aspects of CNN models executing on IoT systems using novel FI techniques that enable us to isolate specific system execution moments and memory sections.
- Investigate the impact of reduced and mixed precision memory parameters on the soft error reliability of the adopted CNN models.
- Investigate the impact of thread parallelism on soft error reliability of existing NN kernels.

1.3 Original Contributions of this Thesis

Figure 1.1 illustrates the main contributions of this work, which are joined into a soft error analysis flow that includes: the FI techniques and extensions, fault classification, and soft error mitigation techniques (fully described in Chapter 4); the cross compilation and platform simulators used to evaluate the consistency of the SOFIA framework (described in Chapter 5); the case studies and libraries adopted to asses soft error reliability of ML models executing on resource-constrained IoT edge devices (described in Section 6.1 and Section 6.2); the validation boards; the automated framework flow that allowed to evaluate different aspects of soft error reliability in a feasible time, i.e., Section 6.1.2 and Section 6.2.2 considering the ML model execution (e.g., isolated layers, target ISA optimization, precision bitwidth), Section 6.1.3 and Section 6.2.3 considering the ML model storage, Section 6.1.4 considering thread parallelism; and the adopted non intrusive soft error software-based mitigation techniques used as alternative to improve reliability in the adopted case studies with lower performance costs in Section 6.2.4.

The contributions of this Thesis are summarized as follows:

Figure 1.1 – Contributions of this Thesis considering the soft error assessment flow in the SOFIA framework.



Source : The Authors

1.3.1 Evaluation of SOFIA consistency w.r.t. RTL

The design flexibility, debugging and simulation performance capabilities of virtual platform frameworks led to the increase the popularity in both academia and industrial sectors. While the gain in simulation speed is trivially observed in VP simulators based on JIT dynamic binary translation, the soft error assessment consistency of underlying FI frameworks remains unclear. In order to investigate the soft error consistency of VP simulators, recent works demonstrate that higher level FI approaches provide more flexibility and simulation performance at the cost of accuracy, mostly resulting from the lack of microarchitecture and timing modelling aspects (KALIORAKIS et al., 2015; CHATZIDIMITRIOU et al., 2019; ROSA et al., 2017). In this direction, one of the *main contributions* of this thesis is an in-depth and statistical significance soft error consistency evaluation of SOFIA framework(BANDEIRA et al., 2019) against FI campaigns conducted

with event-driven simulators (i.e., more realistic and accurate platforms) considering Arm Cortex-M processor architectures. Reference single-core FI campaigns are performed on RTL descriptions of Arm Cortex-M0 and M3 processors. Campaigns consider different open-source and commercial compilers as well as real software stacks including FreeRTOS kernel and 26 applications. Such contribution have already published at ABICH et al. 2021.

1.3.2 Early soft error assessment of ML models executing on Resource-constrained IoT edge devices

ML algorithms have provided straightforward solutions to a wide range of applications. The high computational demand of such algorithms limits their adoption in resource-constrained devices, typically relying on reduced memory footprint, low-power, and low performance processors. While performance improvement, customized, and reduced-precision implementations of ML models have been studied extensively, their susceptibility to soft errors caused by radiation particles is still an open research question. In this regard, the *second main contribution* of this thesis relies on the soft error reliability assessment of reduced and mixed precision CNNs executing on resource-constrained IoT edge devices. In order to cover a wide range of aspects from such models, this study evaluates the soft error reliability considering the fault impact in:

- isolated critical function layers;
- different processor architectures;
- reduced and mixed precision quantizations;
- isolated memory sections;
- multi-threaded execution.

In this contribution, we *propose* a more realistic fault classification to evaluate the impact of soft errors on the output probabilities of the target models. The evaluated results employs a FI technique that isolates the critical layers' functions of the adopted CNNs. The second case study(ABICH; REIS; OST, 2020) investigates the impact of precision bitwidth on the soft error reliability of the MobileNet (HOWARD et al., 2017) CNN developed based on the CMSIS-based Mixed precision API for Neural Networks (CMix-NN) library(CAPOTONDI et al., 2020) and executed on an Arm Cortex-M processor. In the subsequent study, we *propose* an extension to the physical memory FI technique avail-

able in SOFIA (BANDEIRA et al., 2019) that isolates the memory sections, evaluating and comparing the fault impact on the object code and the CNN weights, bias, and buffers. Furthermore, our late approach aims to assess the soft error reliability of a multi-threaded version of a CNN model developed based on the Arm CMSIS API for Neural Networks (CMSIS-NN) kernels considering 1, 2 and 4 cores. Such contributions have already published at ABICH et al. 2020, ABICH; REIS; OST 2020, ABICH et al. 2021, and ABICH et al. 2022, and accepted to publication ABICH et al. 2022.

1.3.2.1 Applying Lightweight Soft Error Mitigation Techniques to Embedded Mixed Precision Deep Neural Networks

While CNNs' precision and performance can vary and are essential, it is also vital to deploy trained models that provide high reliability at low cost. To achieve an unyielding reliability and safety level, it is imperative to provide electronic computing systems with appropriate mechanisms to tackle soft errors. In this sense, the *third main contribution* of this Thesis is to investigate the relationship between soft errors, performance, and model accuracy while promoting the use of a Register Allocation Technique (RAT) GAVA; REIS; OST that allocates the critical CNN function/layer to a pool of specific general-purpose processor registers. This study comprises an extensive soft error assessment of the MobileNet model considering 2, 4, and 8 precision bitwidth variations running on an Arm Cortex-M processor. Such results have already published (ABICH et al., 2021).

1.3.2.2 Consistent and Extensive Evaluation

In order to provide a consistent and extensive evaluation, the results comprise more than 14.8 millions of FIs considering distinct case studies, architectures, number of cores, OS, parallelization libraries. Different from other works, the promoted framework flow uses a realistic software stack comprising bare-metal applications and running over an operating system (e.g., Linux and FreeRTOS), parallelization libraries (e.g., OpenMP), and ML libraries (e.g., CMSIS-NN and CMix-NN). Furthermore, to reinforce the experiments, all adopted case study applications (i.e., benchmarks and CNNs) have been validated by running them on a real board or on the available RTL desciption, which also provide some performance measurements.

1.3.3 Legacy of tools integrated into the SOFIA framework

In addition to the evaluation of soft error consistency the contributions of this work are fully integrated into SOFIA automated soft error analysis flow, which leverages the high simulation performance of M*DEV (IMPERAS, 2021a) to efficiently acquire representative error/failure-related data considering state-of-the-art single and multi-core processor architectures, from ARMv6-M to recent ARMv8-A. Such contributions also comprise the support to:

- compiler scripts considering GCC, Clang, and Arm compilers;
- ML libraries and models targeting resource-constrained IoT edge devices;
- TCL scripts to assess the soft error reliability in RTL processor descriptions;
- a fault injection technique that isolates memory sections;
- a fault classification considering the critical faults affecting the output probabilities of ML models.

Furthermore, the visualization tools have been improved to meet the proposed assessments to correlating faulty results and find some particular behavior according to each case study.

1.4 Thesis Outline

This Thesis is organized into seven Chapters:

Chapter 1 - Introduction: this Chapter introduces the current issues found in the literature to enable a consistent reliability assessment of ML models executing on resource-constrained IoT edge devices and summarizes the contributions of this Thesis. The following paragraphs present a succinct Thesis summary Chapter by Chapter.

Chapter 2 - Background: this Chapter presents the required background in works related to ML models (Section 2.1) and Radiation induced soft errors (Section 2.2).

Chapter 3 - Related Works: this Chapter presents the works related to the challenges of evaluating soft errors in ML models using VPs. In this sense, Section 3.1 presents the state-of-the-art on soft error assessment using VPs. Next, Section 3.2 presents the state-of-the-art on soft error assessment of ML models considering different FI approaches. Finally, we place the contributions of this work regarding the ones found in the literature in Section 3.2.2 and Section 3.1.1.

Chapter 4 - Methodology: this Chapter describes the adopted methodology in this

study considering the FI approaches, the fault classification, and the assessment metrics. First, Section 4.1 details the adopted FI approaches and their different level of accuracy. Then, Section 4.1.1, Section 4.1.2.1, and 4.1.2.2 details each adopted FI modules for RTL, gem5, and Open Virtual Platform Simulator (OVPsim) respectively. Section 4.2 details the proposed fault classification according the ones existing in the literature. Finally, Section 4.3 presents the assessment metrics used to asses the soft error reliability from the adopted case studies, and Section 4.3.1 present a brief description of the mitigation techniques promoted in this work.

Chapter 5 - Results on Soft Error Consistency Assessment: this Section presents the conducted experiments considering different FI approaches to evaluate the consistency of SOFIA against RTL and gem5 FI approaches. First, Section 5.1 presents the results of the soft error consistency assessment for single-core processors. Next, Appendix A presents the results of the soft error consistency assessment for multi-core processors. In each of those analyses, we present a detailed consistency evaluation addressing different aspects of adopted architectures and simulators.

Chapter 6 - Soft Error Reliability Assessment of ML Models on Resource-constrained IoT edge devices: this Chapter presents the proposed case studies considering the execution of ML models in resource-constrained IoT edge devices. Section 6.1 and Section 6.2 details the ML kernels and APIs used to deploy the adopted case studies. Then, following sub sections comprise the results from conducted FI campaigns considering different soft error reliability aspects on the execution of CNN models in resource-constrained IoT edge devices.

Chapter 7 - Conclusions: this Chapter summarizes the contributions of this while highlights the results published in significant journal papers and peer-reviewed international conferences. Finally, this Chapter also points out the possible future works enabled by the contributions reported in this Thesis.

2 BACKGROUND IN ML MODELS AND RADIATION EFFECTS

This Chapter aims to introduce some necessary backgrounds and works related to this Thesis, the soft error assessment of ML models applied to IoT systems. First, Section 2.1 presents a background in ML models and the challenges to enable the execution of such models in IoT edge devices. Further, Section 2.2 addresses the basic concepts regarding radiation-induced errors and their impact on electronic computing system devices.

2.1 Basic Concepts of ML models

Figure 2.1 – Different types and applications of ML exploited for solving computer science problems.

Source : Adapted from KATO et al. (2017).

ML is a special branch of artificial intelligence that acquires knowledge from training data based on known facts without explicit programming (MAHDAVINEJAD et al., 2018). As described by SAMUEL in 1959, ML is a *"Field of study that gives computers the ability to learn without being explicitly programmed"*. Learning is a matter of finding statistical regularities or other patterns in the data, and the performance

analysis of such learning models can give insight into relative difficulty of learning in different environments (AYODELE, 2010). ML models are organized into a taxonomy presented in Figure 2.1 (KATO et al., 2017). The literature classifies ML models into three broad categories such as: supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning is the learning method where the instances are labeled in the training phase generating a function that maps inputs to desired outputs. *Unsupervised learning* is a type of ML model, which models a set of inputs without labeled examples. *Reinforcement learning* means a computer interacting with an environment to achieve a certain goal considering a policy of how to act given an observation of the real world scenario. Every action has some impact in the environment, and the environment provides feedback that guides the learning algorithm.

According to reviewed works (SHANTHAMALLU et al., 2017; MAHDAVINEJAD et al., 2018; CHEN; RAN, 2019; MARCHISIO et al., 2019), supervised and unsupervised learning have been and are still widely applied in smart data analysis. In this sense, the following subsections present a review focused on these ML types (Section 2.1.1 and Section 2.1.2) thus focusing on ML models commonly used in IoT devices (Section 2.1.3).

2.1.1 Supervised learning

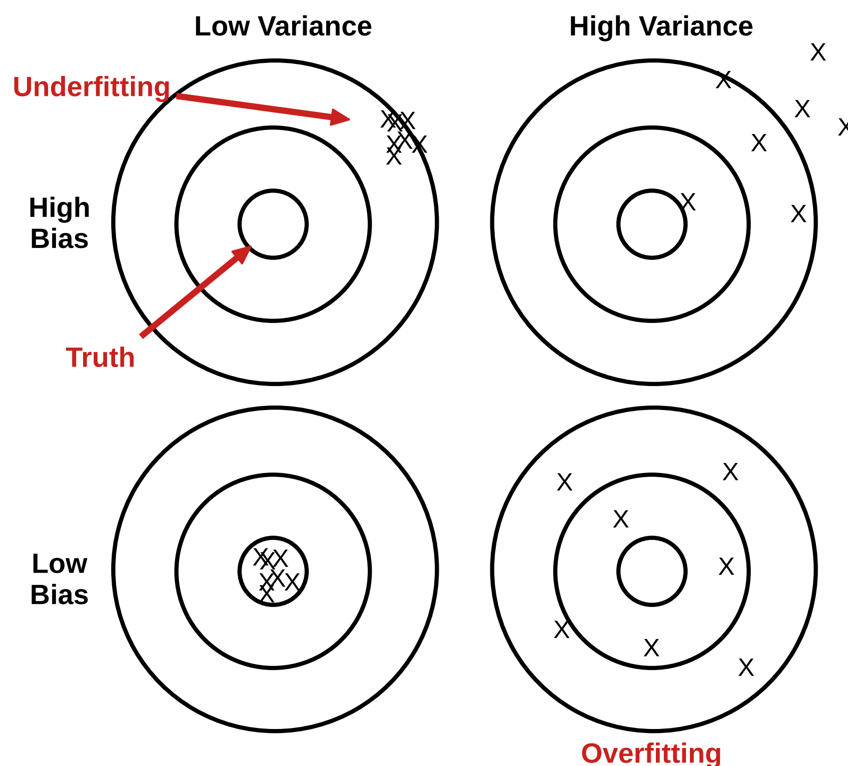
Supervised learning is the most common subbranch of ML (AYODELE, 2010). The objective of such learning method is to learn how to predict the appropriate output vector for a given input vector. The supervised learning means that the training phase needs supervisor interaction. During the training phase, the algorithm will search for patterns in the data that correlate with the desired outputs. The training data will consist of inputs paired with the correct outputs. After training, a supervised learning algorithm will take in new unseen inputs and will determine which label the new inputs will be classified as based on prior training data.

There are several supervised learning techniques known in the literature such as: Artificial Neural Network (ANN), Bayesian Statistics, Gaussian Process Regression, Lazy learning, Nearest Neighbor, Support Vector Machine (SVM), Hidden Markov Model, Bayesian Networks, Decision Trees(C4.5,ID3, CART, Random Forrest), K-Nearest Neighbor (kNN), Boosting, Ensembles classifiers (Bagging, Boosting), Linear Classifiers (Logistic regression, Fisher Linear discriminant, Naive Bayes classifier, Perceptron), and

Quadratic classifiers.

According to CARUANA; NICULESCU-MIZIL (2006), supervised learning can be split into two subcategories: *classification* and *regression*. Applications where the target labels consist of a finite number of discrete categories are known as classification tasks (MORENTE-MOLINERA et al., 2017). Cases where the target labels are composed of one or more continuous variables are known as regression tasks (XIE; LIU, 2010). In both regression and classification, the goal is to find specific relationships or structure in the input data that allow us to effectively produce correct output data. In this sense, a correct output is determined entirely from the training data. However, noisy or incorrect data labels can reduce the effectiveness of a given model in real-world situations. Thus, to provide efficient data labels, the main considerations when conducting supervised learning are the model complexity (JAIN et al., 2007; MORENTE-MOLINERA et al., 2017) and the bias-variance trade-off (VALENTINI; DIETTERICH, 2004; BELKIN et al., 2019; REPPEN; SONER, 2020).

Figure 2.2 – Bias and variance using bulls-eye diagram.

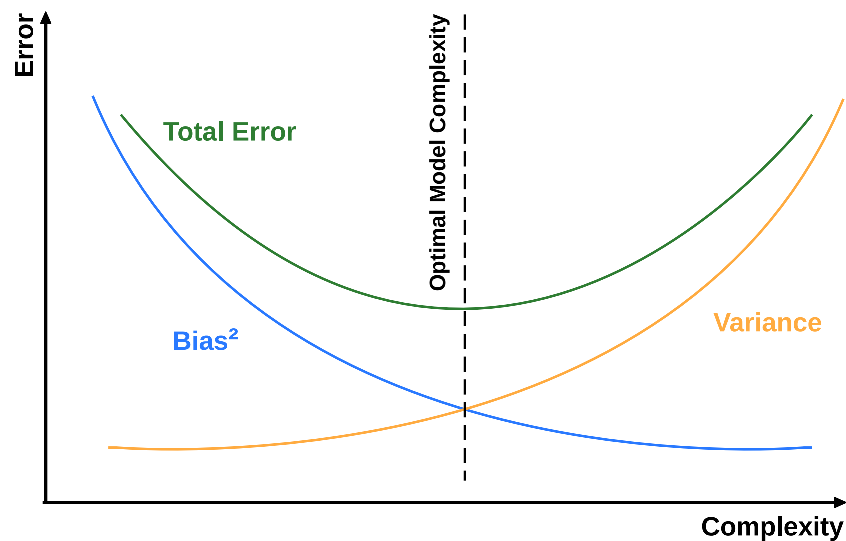


Source : Adapted from GUDIVADA; APON; DING (2017).

In ML, the model complexity often refers to the number of features or terms included in a given predictive model. High complexity models are harder to interpret and have a high probability of overfitting, consequently these models are computationally

expensive. Overfitting refers to learning a function that overly fits the training data, and essentially captures the noise and outliers. Underfitting happens when a model is unable to capture the underlying pattern of the data. As consequence, the model will not accurately predict cases that are not represented by the training data. There are some methods to control or reduce model complexity including linear modeling and subset selection, regularization, and dimensionality reduction. Such methods, essentially, keep all features, but reduce (or penalize) the effect of some features on the model's predicted values.

Figure 2.3 – Model complexity curve defined by Equation (2.1)



Source : Adapted from GUDIVADA; APON; DING (2017).

Figure 2.2 shows the bias-variance trade-off, where the center of the target is a model that perfectly predicts correct values. Bias is the difference between the average model's prediction and the correct value which must be predicted. Variance is the variability of the model's prediction for a given data point or a value that shows the dispersion of the dataset. While models with high bias oversimplify the trained model, models with high variance increases the model complexity. In both cases the resultant model leads to high error rates on the test data. This trade-off must find the right/good balance without overfitting and underfitting the data.

$$Total\ Errors = Bias^2 + Variance + IrreducibleError \tag{2.1}$$

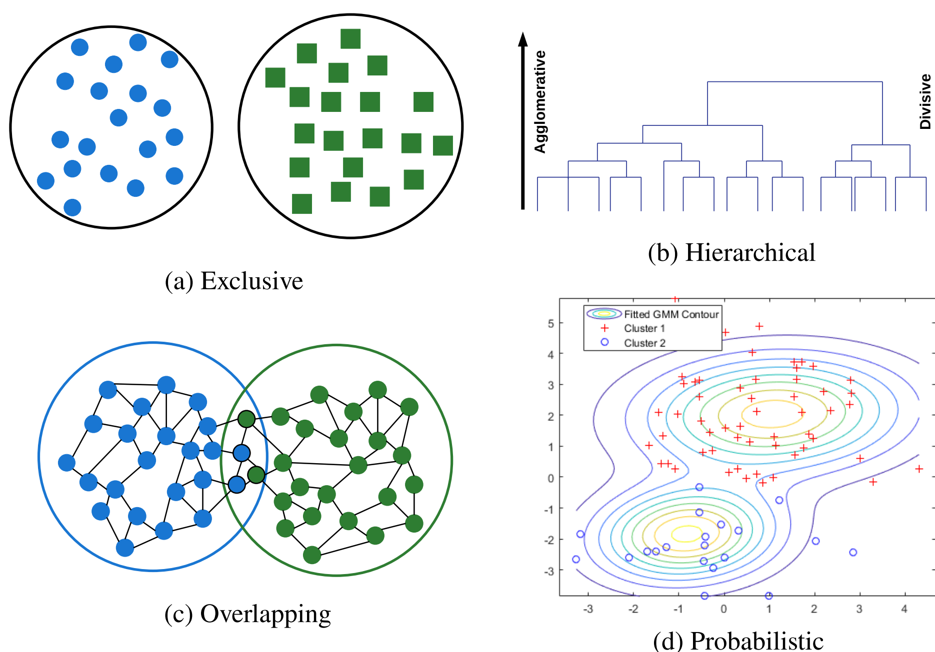
An optimal balance of bias and variance would never overfit or underfit the trained model. To build a good model, it is necessary to find a good balance between the bias and the variance such that it minimizes the total error defined by Equation (2.1). The Total Errors is the sum of Bias^2, variance and the irreducible error. Irreducible error is the error

that cannot be reduced by creating good models (i.e., amount of noise in training data). Finally, Figure 2.3 shows the bias-variance trade-off curve where the center represents the optimal model complexity.

2.1.2 Unsupervised learning

Unsupervised learning techniques are regarded as self-learning algorithms that possess the capacity to explore and locate the previously unknown patterns in a dataset (BARLOW, 1989). While supervised learning (Section 2.1.1) requires a labeled dataset, in unsupervised learning algorithms the training data consists of a set of input vectors without any corresponding target values (HASTIE; TIBSHIRANI; FRIEDMAN, 2009). Such learning algorithms are left to their own devises using a set of statistical tools to discover and represent interesting patterns from data structures (CELEBI; AYDIN, 2016). Some of the most common unsupervised learners are: Cluster analysis (K-means clustering, Fuzzy clustering), Hierarchical clustering, Self-organizing map, Apriori algorithm, Eclat algorithm, and Outlier detection. Unsupervised learning algorithms can be further grouped into the following three categories: clustering, dimensionality reduction, and density estimation (e.g., Figure 2.1).

Figure 2.4 – Examples of clustering algorithms.

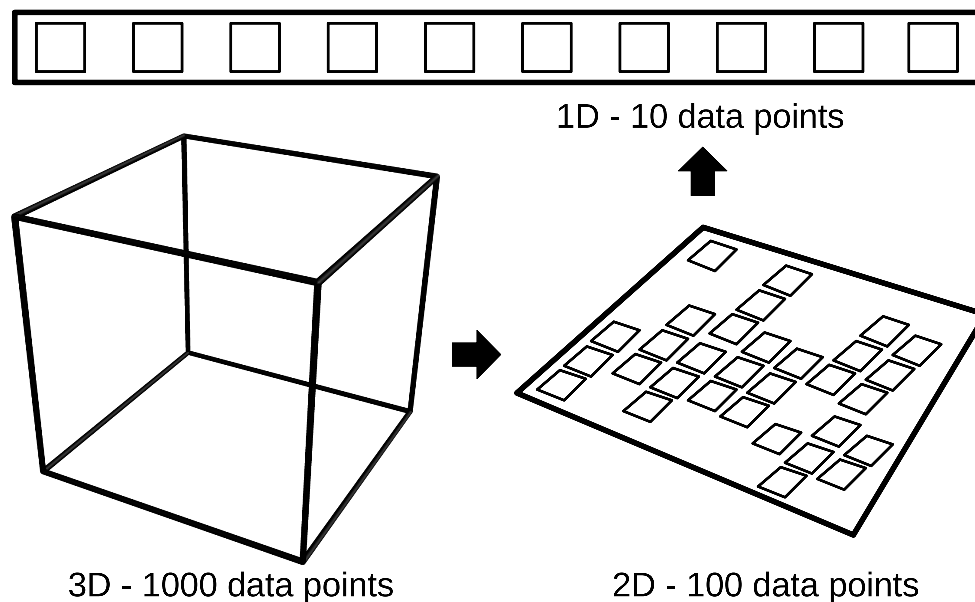


(a) Exclusive (b) Hierarchical (c) Overlapping (d) Probabilistic

Source : Adapted from HASTIE; TIBSHIRANI; FRIEDMAN (2009).

Clustering is the most common unsupervised learning problem, which categorizes the observed dataset into groups that maximize some similarity criterion, equivalently dissimilar to the objects belonging to other clusters. Clustering algorithms may be classified as Exclusive (Figure 2.4a), Hierarchical(Figure 2.4b), Overlapping(Figure 2.4c), and Probabilistic Clustering(Figure 2.4d). In exclusive or partitioning clustering, data are grouped in such a way that one data can belong to one cluster only (KANUNGO et al., 2002). Hierarchical clustering technique considers every data as a cluster and the iterative unions between the two nearest clusters reduce the number of clusters (SONAGARA; BADHEKA, 2014). There are two types of hierarchical clustering, Divisive and Agglomerative. In the divisive (top-down) clustering method, the algorithm assigns all of the observations to a single cluster and then partition the cluster to two least similar clusters. Thus, the algorithm proceed recursively on each cluster until there is one cluster for each observation. In turn, agglomerative or bottom-up clustering method performs the opposite. The overlapping technique use fuzzy sets to cluster data where each point may belong to two or more clusters with separate degrees of membership (HATHAWAY; BEZDEK, 2006; GHOSH; DUBEY, 2013). Probabilistic clustering technique uses probability distribution to create the clusters (FIGUEIREDO; JAIN, 2002; XU; LI, 2008).

Figure 2.5 – Example of dimensionality reduction using matrix decomposition technique.

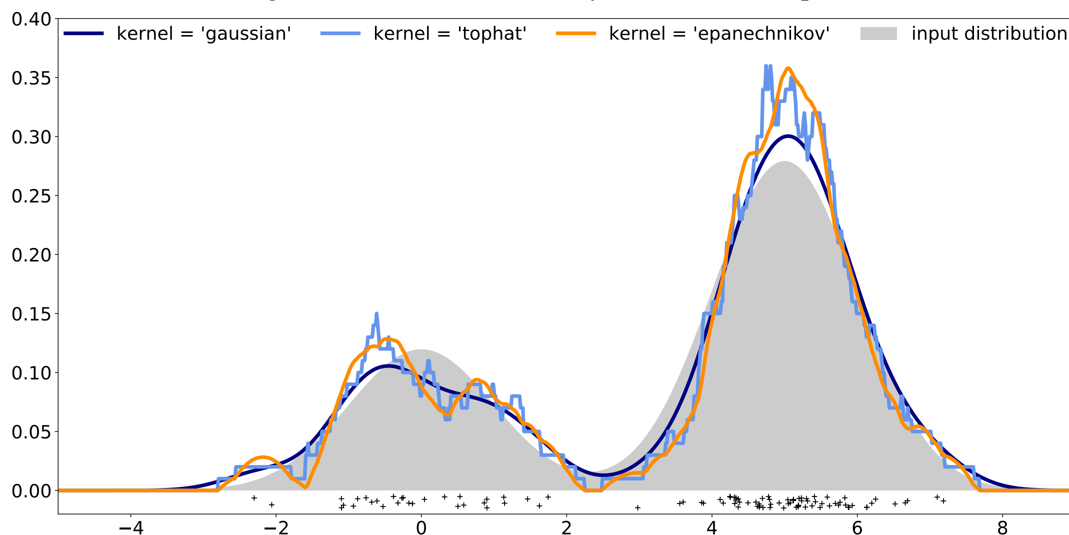


Source : Adapted from TSUGE et al. (2001).

Dimensionality reduction transforms a high-dimensional set of variables into a low-dimensional space where the resultant representation retains some meaningful properties

of the original data, ideally close to its intrinsic dimension (WEINBERGER; SHA; SAUL, 2004). For instance, Figure 2.5 shows an example of matrix decomposition data correlation used to reduce a 3D matrix to a single data vector. Such learning technique correlates the number of features present in the dataset in order to remove redundant information, reducing model's complexity, and avoid overfitting(MAATEN; POSTMA; HERIK, 2009). In this sense, dimensionality reduction splits into two main categories: *feature selection* and *feature extraction* (KHALID; KHALIL; NASREEN, 2014). In feature selection, the algorithm select a subset of features of the original dataset, aiming to obtain a model capable of automatically selecting the subset of features most relevant to a faced problem (DY; BRODLEY, 2004). Feature extraction derives the information from the original dataset to reduce the number of features and build a new feature subspace(HILD et al., 2006). In contrast to the feature selection, the output features will not be the same as the originals(KHALID; KHALIL; NASREEN, 2014). In this sense, the extracted features are combinations of the original features, compressed in a way that they will retain the most relevant information.

Figure 2.6 – 1D Kernel density estimation example.



Source : Adapted from PEDREGOSA et al. (2011).

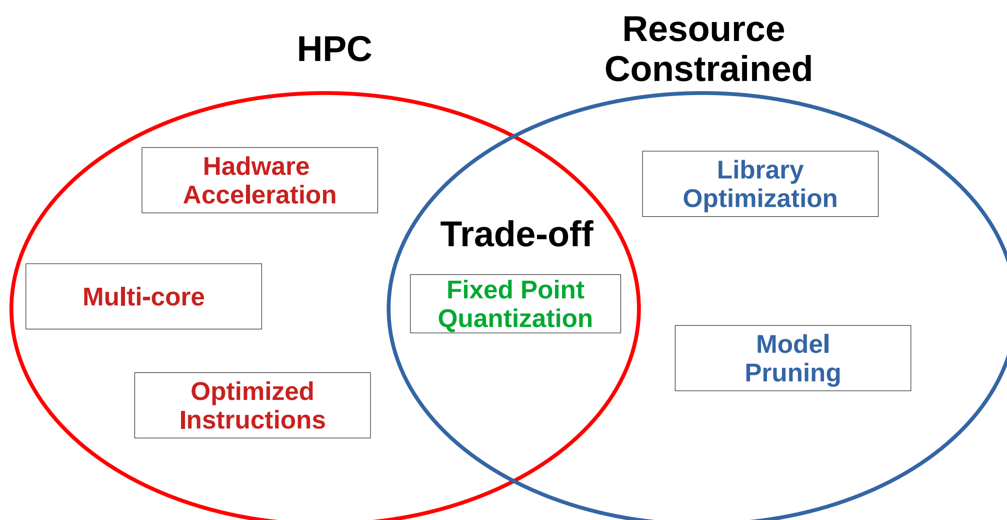
Density estimation uses statistical models to find an underlying probability distribution that gives rise to the observed variables (SMYTH; WOLPERT, 1997). For example, Figure 2.6 shows the kernel density estimation for three choices of kernels considering as input 100 points from a bimodal distribution. The goal in such learning problems is to model the underlying structure or distribution in the data in order to identify the inherent structures of a dataset without using explicitly-provided labels (TRENTIN; FRENO, 2009). Some of the most popular and useful density estimation techniques are mixture

models such as Gaussian mixtures, and neighbor-based approaches (e.g., kernel density estimation) (DINH; SOHL-DICKSTEIN; BENGIO, 2016). Gaussian mixtures are discussed more fully in the context of clustering, because the technique is also useful as an unsupervised clustering scheme.

2.1.3 ML for the IoT Edge devices

The size of computing devices in current technology nodes is largely reducing while their computing capability is drastically improving. Such resultant increase in computing power and advancements have made it possible to map and process much larger and deeper neural networks than was possible in earlier generations(SZYDLO; SENDOREK; BRZOZA-WOCH, 2018). These upgrades encouraged the current shift of ML models to IoT environment, aiming to explore the execution of smart applications on resource-constrained devices(DAWIT; FRISK, 2019). In contrast to other ML models shown in Figure 2.1 that must be highly tuned to solve specific problems and need a plenty of rules for successful operation, IoT edge deep inference techniques must deal with the constraints of power, performance, and resources inherent to such devices (KATO et al., 2017).

Figure 2.7 – Trade-off between approaches used to enable deep inference in resource-constrained IoT Edge devices.



Source : Adapted from QI; LIU (2018).

Figure 2.7 shows the trade-off between the techniques used to promote deep inference in HPC and resource-constrained devices. While the HPC environment capabilities

support hardware acceleration, multi-core execution, and optimized instructions, to enable deep inference tasks on IoT edge side, there are two main considerations: architecture specific optimized software libraries and model pruning. One of the important aspects in the industrial IoT is the response time of such systems, which means that the IoT systems using ML models tend to be moved to the edge of the network. Currently, researchers aim to achieve a feasible performance while using optimizations allowed by the target ISAs. In this sense, to provide the execution of Deep Neural Network (DNN) models on the underlying devices, software libraries and APIs have been proposed (SUN; LIU; GAUDIOT, 2017; LAI; SUDA; CHANDRA, 2018; GAROFALO et al., 2019; STMICROELECTRONICS, 2020; CAPOTONDI et al., 2020). Such libraries/APIs are devoted to streamlining the design and development of embedded deep learning-based applications through the fine-tuning of pre-trained network models, thus enabling their efficient execution in edge-computing platforms (AMOH; ODAME, 2019; MAHDAVINEJAD et al., 2018). In additional, to add more performance to IoT platform researchers also tried to make the ML models lighter with a similar level of performance and accuracy. Such approaches aim to reduce the number of inference parameters (e.g., weights and bias) to reduce the memory requirements from deep inference models while achieving such a performance improvements (HAN; MAO; DALLY, 2015; HE; ZHANG; SUN, 2017).

The resultant trade-off between performance and resource constraints shifts the deep inference models to fixed point integer quantization methods. Such techniques aim to reduce the size of the parameters from deep neural networks and improve inference latency and throughput by taking advantage of high throughput integer instructions (WU et al., 2020). In this sense, researchers already found that quantized models reach almost the same level precision when considering reduced precision of NN parameters, mostly the weights (GUPTA et al., 2015; JUDD et al., 2015). While the resultant precision is degraded (e.g., $< 1\%$ in MobileNet), the memory overhead is drastically reduced by 75% when using integer 8-bit parameters instead of 32-bit respectively. Moreover, with the cooperation of hardware, the quantization can also lead to faster model execution speed. Considering IoT applications targeting Arm processor, a category of SIMD instructions is introduced to accelerate the add and multiply operations (LAI; SUDA; CHANDRA, 2018; CAPOTONDI et al., 2020), which are the essential elements in ML model execution.

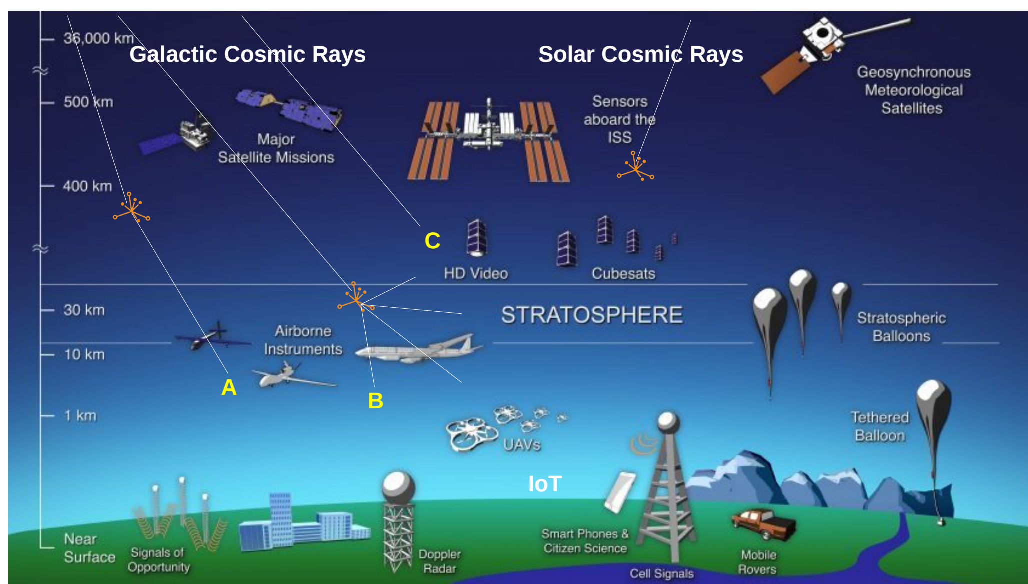
In the domain of resource-constrained devices one can find many implementations of ML models on mobile and embedded devices that cooperate with the cloud computing (SZYDLO; SENDOREK; BRZOZA-WOCH, 2018). For the time being, the majority of

embedded trained models and their inference engines have been evaluated only according to their accuracy and performance over a given dataset (QI; LIU, 2018). In contrast, as DNN become increasingly common in mission-critical applications, ensuring their reliable operation has become crucial nowadays (MITTAL, 2020). For that reason, this work aims to *contribute* by evaluating the soft error reliability aspects that can impact such kind of systems.

2.2 Radiation Environment an its Effects on Semiconductors Devices

Current challenges presented in the semiconductor industry target not only performance but also the system reliability, since the capacity of transistor integration in existing technology nodes has been presenting significant issues provided by highly energetic radiation particles (MANSOUR; VELAZCO, 2013; ISO, 2011). Such particles may induce the occurrence of Single Event Effects (SEE)s on integrated transistor circuits causing the change of data states (e.g., from 0 to 1), which might incur into catastrophic results (i.e., human life losses). In order to comprehend the origin of those effects, this Section presents a brief summary of radiation effects.

Figure 2.8 – Sketch of the Earth's radiation environment.



Source : Adapted from BARTH; DYER; STASSINOPOULOS (2003) and CIANI; CATELANI; VELTRONI (2008).

Figure 2.8 illustrates the possible sources of atmospheric radiation known in the literature (ZIEGLER, 1996; BARTH; DYER; STASSINOPOULOS, 2003; CIANI; CATE-

LANI; VELTRONI, 2008). The main sources of atmospheric radiation are *galactic* and *solar* cosmic rays, which can generate high energy particles according to the particle's trajectory in the Earth's atmosphere, such as: *neutrons*, *protons* and *heavy-ions*. In addition to spacecrafts that suffer with radiation effects, such particles are those that can penetrate the planetary magnetic field, causing SEEs on devices inside the Earth's atmosphere (BARTH; DYER; STASSINOPOULOS, 2003). *Neutrons* (Figure 2.8A) are the primary particles generated by the shock between radiation ions and atoms from the atmosphere (NORMAND, 1996). For that reason, such particles are more likely to generate SEEs at high altitudes. The shock of primary cosmic rays with the atoms in the air can also result in *protons*, *pions*, *kaons*, and *electrons* particles. In turn, most studies consider only *Protons* (Figure 2.8B), since this particle have similar occurrence to *neutrons*, including energy, charging capacity and altitude. Furthermore, the *heavy-ion* particles (Figure 2.8C) are composed by one or more electric charge units with a mass that exceeds the alpha particle. The occurrence of these particles is related to the outer layer of the Earth's atmosphere, or in the combination of high altitudes and high latitudes (polar), where they manage to cross the magnetic belt. Such effect occurs due to the interactions of heavy-ions with the Eath's atmosphere cause fragmentation and thereby remove such ions.

Once one of these ions hits a device, it can generate an SEE which in turn can be classified according to the resulting effect. The next Section describes the SEE types and how they are commonly classified.

2.2.1 Classification of SEEs

This Section aims to review the types of radiation effects that lead to errors occurring in transistor-based devices. The possible types of SEEs are destructive, those that imply in hard errors mostly non recoverable, and nondestructive, soft errors that can be observed in microelectronic devices (digital or analog). Soft errors imply in single particle strikes that generate some noise or crash in common workflow. The types of hard and soft errors can be classified as follows:

- Hard Errors: Single Event Latchup (SEL), Single Event Burnout (SEB), and Single Event Gate Rupture (SEGR);
- Soft Errors : Single Event Upset (SEU) and Single Event Transient (SET).

2.2.1.1 Hard Errors

The hard errors comprise irreversible changes in a device, which may imply in permanent damage even after a restart. Most of the known hard errors are caused by thermal stress generated by the impact of high energy ions (TORO et al., 2013). The aerospace and microelectronic industry have identified the occurrence of SEL on a wide range of devices based on Complementary Metal-Oxide-Semiconductor (CMOS) technology over the last 35 years (SOLIMAN; NICHOLS, 1983; JOHNSTON, 1996; BRUGUIER; PALAU, 1996; HUTSON et al., 2009). With the technology scaling, the decreased transistor size along with reduced nodal capacitance increase the device sensibility to charges induced by radiation particles that can modify the electrical field and create a latchup (SCHWANK et al., 2005; SCHWANK et al., 2006). The SEL consists of charges deposited by striking energetic particles that modify the transistor electrical field and can trigger a parasitic bipolar structure, leading to device destruction as a consequence of thermal stress. In order to reduce the SEL sensitivity, the major existing solutions are based on layout and/or process optimizations. As an alternative, since latchup is known to be temperature dependent, a thermal controller that turns off the device power supply can disable the parasitic structure in case of stress, avoiding the device damage (i.e., turning this effect into a soft error).

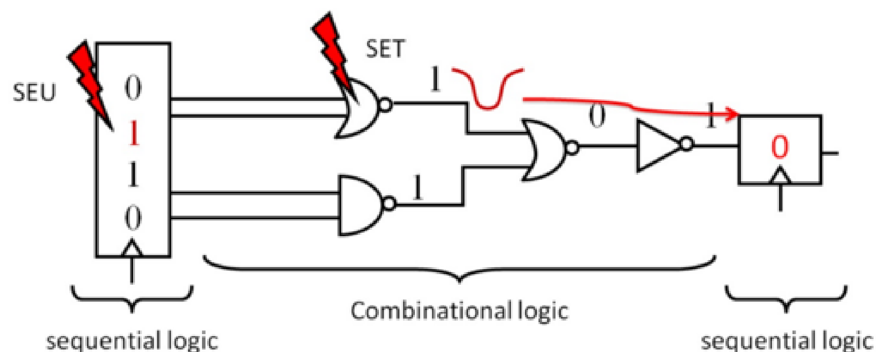
Similarly, several studies over the past 30 years have shown that heavy ions can trigger catastrophic failure modes in power Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET)s (JOHNSON et al., 1996; TITUS, 2013) Power MOSFETs, subjected to cosmic environments or with widespread occurrence of particles, are prone to SEGR and SEB, which can adversely affect a device's performance and can cause system failures (HANDS et al., 2011). In general, an SEB occurs in n-channel MOSFETs, combining the *off* state and abnormal applied voltages. In this situation, when an particle collides with the circuit it forms a parasitic structure that causes a high-density current and consequently the device's transistor burn (WROBEL et al., 1985). Similar to SEB, SEGR effects occur mainly in MOSFETs. In this effect, the impact of accumulated ions holes under the gate, increasing the electric field through the silicon dioxide that cause dielectric break that results in a leakage current, which can also cause a door oxide's thermal failure (TITUS et al., 1998). Furthermore, the literature shows, both experimentally and theoretically, that such effects must be considered at early circuit design phase in order to reduce/avoid their occurrence (BARAK et al., 2008; TITUS, 2013; MUTUEL, 2014).

2.2.1.2 Soft Errors

On the contrary to hard errors, soft errors are non-destructive effects induced by the impact of energetic ions. Soft errors may emerge either during intensive or idle working periods, affecting both critical and non-essential system functionalities. For instance, Processor-based systems working at sea level are expected to experience at least one soft error per day (GRANLUND; GRANBOM; OLSSON, 2003). Figure 2.9 shows the occurrence of both SET and SEU in a given circuit example.

The difference between SET and SEU regards to the affected target and its duration over time. The SET is a momentary voltage spike at a node of an integrated circuit generated by a single energetic particle passing through or near the junctions creating an ionization track. As illustrated in Figure 2.9, the peak voltage affects non-latched elements such as combinational logic(LOVELESS et al., 2012). The resulting effect may propagate any significant distance through the combinational logic and, if it is not masked, can reach a memory element where can be sampled causing a fault (WIRTH; KASTENSMIDT; RIBEIRO, 2008). In this sense, an SET can become an SEU when it reaches an memory element, since it is stored in a memory element the error is no longer transient. Although these effects are recurrent and widely studied, other radiation effects appear with reduced scaling in current technology nodes. For instance, the charge sharing causes signal degradation by transferring charges from one electronic domain to another (FERLET-CAVROIS; MASSENGILL; GOUKER, 2013). Furthermore, the reduced scale CMOS circuits have been showing a significant sensitivity to SETs (DODD et al., 2010; PRINZIE; STEYAERT; LEROUX, 2018). Despite that, recent technology nodes consider either Fin Field Effect Transistor (FinFET) and Fully Depleted Silicon On Insulator (FDSOI) nanotechnologies that show significant improvements in terms of fault resilience (LIU et al., 2016; AGUIAR et al., 2017).

Figure 2.9 – SEU and SET effects occurring in a circuit example.



Source: KASTENSMIDT; RECH (2016).

As shown in Figure 2.9, the SEU differs from SET with regard to the affected target in the integrated circuit and its duration over the time. The SEU phenomenon may occur in a digital circuit when an ionizing particle strike results to a change in data states from a storage element such as latches, flip-flops, Random Access Memory (RAM) cells, or asynchronous memory logic (VARGAS; NICOLAIDIS, 1994). When a SEU occurs, a simple reset or rewriting masks the resultant soft error effect, preserving device's normal behavior. In this sense, logical masking can occur through an operation that instantly overwrites the faulty value in the memory element. Besides that, an electrical masking can occur when the signal interference is not enough to change the logic state. However, in the worst case scenario, the SEU effect can remain several clock cycles, for synchronous logic, or until the next transition of an input signal in asynchronous logic (TABER; NORMAND, 1993). In this case, the soft error could imply in catastrophic results in the circuit operation.

3 RELATED WORKS

The soft error assessment and mitigation literature is abundant, requiring a taxonomy to classify the different approaches. This thesis considers the definitions from AVIŽIENIS; LAPRIE; RANDELL (2004) for fault, error, and failure. A fault is an event that may cause the internal state of the system to change, e.g., a radiation particle strike. When a fault affects the system's internal state, it becomes an error. If the error causes a deviation of at least one of the system's external states, then it is considered as a failure. To achieve compliance with safety and reliability standard requirements, it is utmost importance to provide systems with appropriate mechanisms to tackle systematic, SEU, or SET faults, also known as soft errors.

In this regard, this Chapter presents a literature review of the works related to this Thesis contributions on the soft error reliability assessment of ML models executing on resource-constrained IoT systems. First, Section 3.1 presents a review of fault injector frameworks implemented on the top of VPs. Next, Section 3.2 discusses some related works on soft error reliability assessment of ML models in different scopes. Finally, we distinguish this Thesis from the works found in the literature (Section 3.2.2 and Section 3.1.1).

3.1 Soft Error Assesment Considering Virtual Platforms

Fault injection simulation frameworks are gaining momentum due to their efficiency to obtain prominent results on the soft error resilience of different systems early in the design phase. These FI frameworks can be divided into two classes: (*i*) those focused on the accuracy of the results and (*ii*) those that deal with highly complex systems or have little time to explore the project reliability. Simulation-based soft error analysis at RTL or gate levels are examples of the first class. Within this category, MANSOUR; VELAZCO 2013 presented a method called Direct Fault Injection (DFI) to emulate the consequences of a SEU occurring in the processor's memory cells. However, this approach might, in same cases, require modification of the circuit architecture, which is not easily applied anywhere. ABBASITABAR; ZARANDI; SALAMAT 2012 also investigated the fault susceptibility of the LEON3 processor in RTL, but their approach relies on the Modelsim simulator to inject faults without explicitly changing the reference design. Although both approaches produce accurate results, they are restricted to relatively small or specific sys-

tems, where experimental fault campaigns provide some simulation performance but with low observability (i.e., fault must be detected by the system).

Researchers started investigating other approaches, such as the use of VPs, aiming to boost the soft error analysis of complex systems comprising not only real software stacks but also ISAs and state-of-the-art processors (HARI et al., 2012; PARASYRIS et al., 2014; ROSA et al., 2015; KALIORAKIS et al., 2015). VP simulators facilitate fault injection implementation and analyses due to their design flexibility (e.g., several processor models available) and debugging capabilities (e.g., GDB support). Table 3.1 shows related works considering fault injection capabilities in VP simulators, which are detailed below.

Authors in HARI et al. 2012 present the Relyzer, a hybrid simulation framework for Scalable Processor Architecture (SPARC) core using Simics (MAGNUSSON et al., 2002) and General Execution-driven Multiprocessor Simulator (GEMS) (MARTIN et al., 2005) simulators coupled with a pruning technique to reduce injected faults and targeting architectural integer registers and in the output latches of the address generation units. The low number of injected faults is due to the high-cost simulation time of the simics+GEMS simulator, which can achieve a few hundred Kilo Instructions Per Second (KIPS). Further HARI et al. 2014 presented the GangES approach, a Relyzer extension to more aggressive pruning and reducing the number of faults that must be simulated. With these techniques embedded in the GangES, the authors further reduce this simulation time by half. In GEISSLER; KASTENSMIDT; SOUZA 2014, the authors proposed a fault injection framework based on Quick Emulator (QEMU) (BELLARD, 2005) to inject faults in an x86 architecture running applications in a Real-Time Operating System (RTOS). This approach considers an in-house experimental setup that achieved some performance related to the number of injected faults (i.e., an average of less than one fault per second).

PARASYRIS et al. 2014 introduced the GemFI, a fault injection tool based on the cycle-accurate full-system model of the gem5 simulator (BINKERT et al., 2011). Authors improve the fault injections by creating checkpoints and parallelization strategies to gain simulation performance (i.e., 64.5x faster on average). Moreover, KALIORAKIS et al. 2015 propose two tools: the GeFIN tool, a gem5-based fault injection framework and MaFIN, a MIPS Assembler and Runtime Simulator (MARSS)-based (PATEL et al., 2011) fault injection framework. In this work, faults are injected, randomly in time, in general-purpose registers, caches control registers, and other micro architectural components. (ROSA et al., 2015) presented a fault injection framework based on another VP, the

OVPsim (IMPERAS, 2021b), which has the advantage of supporting parallel simulation to boost up the fault injection process. Such works denote the VP fault injection simulators not only has higher performance but the flexibility to perform techniques that reduce the simulation time (e.g., checkpoints and parallelization) considering much more complex scenarios.

To cover higher fault probabilities, the authors in TANIKELLA et al. 2016 introduces the gemV, a gem5-based (BINKERT et al., 2011) fault injection framework for micro-architectural elements such as instruction queue, reorder buffer, load-store queue, pipeline queue, renaming unit, and register file. Similarly, DIDEHBAN; SHRIVASTAVA 2016 presented a gem5-based fault injection capable of flipping random register file bits, pipeline registers, functional units, and load-store queue. GUAN et al. 2016 presents the P-FSEFI tool, developed around the QEMU (BELLARD, 2005) simulator to injects faults in the Central Processing Unit (CPU) logic units, registers, caches, and memory. The experimental setup consists of seven applications from the NAS Parallel Benchmark (NPB) (BAILEY et al., 1991), each one in a sequential and a parallel version. Recently, BANDEIRA et al. 2019 presented the SOFIA framework, an extension of the works presented in ROSA et al. 2015 and ROSA et al. 2017. The framework's construction considers OVPsim (IMPERAS, 2021b) as a base, and the latest version has been enhanced with the Multicore/Multiprocessor Software Development Kit (M*DEV) simulator capabilities (IMPERAS, 2021a), which allowed us to incorporate the injection of bit-flips in six different scopes: register file, physical memory, application virtual memory, application variables and data structures, function object code, and function lifespan. Such approaches evaluate soft error reliability considering parallel benchmarks and more comprehensive fault coverage considering not only register files but other micro-architectural elements and software structures. Furthermore, SOFIA flexibility allow us to use the same FI approach and FI flow to evaluate the soft error reliability even in low level RTL[1] processor descriptions.

[1]The adopted RTL model must allow to access the register file and memory addressing

Table 3.1 – Related works considering virtual platform fault injection simulators.

Year	Author	Simulator	ISAs	Applications	Guest OS	Level of Detailing	FI Techniques
2014	HARI et al. Relyzer	Simics GEMS	SPARC V9	12: Parsec 2, Splash-2, and SPEC-Int 2006	OpenSolaris	Cycle and Instruction	Architectural integer registers and in the output latches of the address generation units
2014	GEISSLER; KASTENS-MIDT; SOUZA	QEMU	x86	4 in-house applications	RTEMS	Instruction	8 general-purpose registers, 6 segment registers, and instruction pointer
2014	PARASYRIS et al. GemFI	gem5	Alpha ISA	6 baremetal	None or Unknown	Cycle	Architectural integer registers, L1 and L2 cache, and Load/Store Queue
2015	KALIORAKIS et al. MaFIN and GeFIN	MARSS gem5	x86 ARMv7	10 MiBench	None or Unknown	Cycle	Architectural integer registers, L1 and L2 cache, and Load/Store Queue
2016	TANIKELLA et al. gemV	gem5	x86 ARMv7 SPARC ALPHA	10: MiBench and SPEC-Int 2006	None or Unknown	Cycle	11 Microarchitectural components
2016	DIDEHBAN; SHRIVASTAVA	gem5	ARMv8	10 MiBench	gem5 Syscall Mode	Instruction	Register file, pipeline registers, functional units, and load-store queue
2016	GUAN et al. P-FSEFI	QEMU	ARMv7 x86	14 serial NPB	None or Unknown	Instruction	CPU logic units, registers, caches, and memory
2018	MEDEIROS et al.	SystemC Model	Plasma-MIPS	26 Malardalen Benchmarks	None or Unknown	Cycle	Register file
2022	***This work (BANDEIRA et al. SOFIA)***	***M*DEV gem5***	***ARMv6 ARMv7 ARMv8***	***CIFAR-10 CNN, Mobilenet CNN, 26-Malardalen, 29-NPB, 16-Rodinia***	***Linux OS, FreeRTOS, and Baremetal***	***Cycle and Instruction***	***Register file and physical memory address space in both simulators. Application Virtual Memory, Variables and Data Structures; and Function Object Code and Lifespan in the M*DEV-based fault injection.***

Source : The authors

3.1.1 Contribution in Soft Error Assessment Considering Virtual Platforms

Reviewed FI frameworks are also used to investigate different software stacks configurations, such as standard parallelization libraries (ROSA et al., 2018) and compiler optimization flags (LINS et al., 2017; SANGCHOOLIE et al., 2014; MEDEIROS et al., 2018), and their impact on soft error reliability. For instance, authors in SANGCHOOLIE et al. 2014, LINS et al. 2017, and MEDEIROS et al. 2018, investigate the impact of GCC compilation flags (e.g., O0, O3, etc) on the application behavior under fault influence. These works consider either simple (i.e., in-house and bare-metal applications) or small scenarios, where only a single processor is considered. Modern compilers have specific characteristics, which directly impact on applications performance, power-efficiency, and reliability (HOSTE; EECKHOUT, 2008). Taking a step forward in compiler assessment, SERRANO-CASES et al. 2019 proposed a method to find out the best combination of compiler optimizations and parameters to improve the fault tolerance of applications.

Aiming to demonstrate the discrepancies between fault injections conducted at different levels, CHO et al. 2013 evaluate the accuracy trade-offs associated with a variety of high-level fault injection techniques (i.e., RTL) comparing to a flip-flop-level baseline. Similarly, SCHIRMEIER; BREDDEMANN September 2019 apply gate, flip-flop, and ISA level (i.e., register file) FI techniques to evaluate error-rate discrepancies considering the gate-level FI as reference. Authors in CHO et al. 2013 use geometric means to show the mismatch between the FI levels. Although useful, the authors mention that the adopted metric may not capture how mismatch levels vary across various applications. To improve the soft error assessment accuracy analysis, authors in SCHIRMEIER; BREDDEMANN September 2019 use a ranked correlation to evaluate the mismatch between the FI approaches considering the Extrapolated Absolute Failure Count (EAFC) (SCHIRMEIER; BORCHERT; SPINCZYK, September 2015) normalized according to the gate-level FI. Such an approach ranks the results from FI techniques of each application, considering the rank shuffling to evaluate the mismatch between the FI approaches. Authors demonstrate that even with discrepancies in such an approach, ISA-level FI approaches are sufficient to evaluate the soft error reliability of low-resource constraint processors (e.g., Cortex-M0). However, the soft error analysis consistency conducted in this work was made based on the EAFC metric. This metric only considers the occurrence of Silent Data Corruptions (SDC)s, which might lead to an inadequate mismatch assessment between different levels of FIs since not all fault classifications are taken into account.

KALIORAKIS et al. 2015 were the first to be concerned with the accuracy of such FI frameworks based on VPs, comparing FI implementations for gem5 (BINKERT et al., 2011) and MARSS (PATEL et al., 2011). In the same sense, CHATZIDIMITRIOU et al. 2019 proposed to analyze the soft error rate accuracy of a gem5-based FI framework against results obtained from a neutron beam experiment, considering an Arm Cortex-A9 processor and 13 benchmarks. An initial effort to evaluate the soft error assessment consistency of a JIT-based FI framework is described in (ROSA et al., 2017), where a Fault Injection Module (FIM) was integrated into gem5 and OVPsim. Results considering several fault injection campaigns were compared, and a mismatch of up to 20% is reported. In further experiments, the Authors achieved a lower worst-case mismatch of 12% by reducing the simulation granularity of OVPsim, i.e., the number of instructions executed per simulation cycle. Similar to KALIORAKIS et al. 2015, conducted experiments aim to evaluate and provide insights on how to improve the accuracy of FI frameworks based on VP simulators.

In this regard, the work conducted in this Thesis is complementary to SERRANO-CASES et al. 2019 that evaluated x86 processors, as it aims to find out which is the best compiler combined with an optimization flag for Arm processors. It also complements CHATZIDIMITRIOU et al. 2019 because it considers other parameters (e.g., cross-compilers), and the soft error consistency analysis focus on a JIT-based VP. Finally, differs from ROSA et al. 2017 with regard to the investigation of multi-core systems is that we introduce the discussion on the different cross-compilers.

Even with most reviewed approaches in Table 3.1 covering several scenarios of fault injection and application benchmark capabilities, *the real consistency of soft error reliability assessment in VP simulators remains open*.

This Thesis distinguishes from works found in the literature by making an extensive and statistical significance soft error consistency assessment of a JIT-based fault injection framework (the SOFIA OVPsim fault injection module presented in Chapter 5). This work is *the first* to cover so many aspects together, such as: single and multi-core processors (i.e., Arm Cortex-M0, Cortex-M3, and Cortex-A9); two parallel programming models (i.e., Message Passing Interface (MPI) and Open Multi-Processing (OpenMP)) compared with Serial execution; operating system (i.e., FreeRTOS, Linux kernel); five sets of compilers and versions (i.e., GCC 4.9, GCC 7.2, Clang 6, Arm 6.10 and Arm 5.06); optimization flags (i.e., O0, O1, O2, O3, Os and Ofast); and more than 50 applications taken from the NPB (BAILEY et al., 1991), Rodinia (CHE et al., 2009), and the Mälardalen Worst-

Case Execution Time (WCET) benchmarks (GUSTAFSSON et al., 2010). This range of parameters led to more than 12.7 million fault injections, bringing an excellent confidence level to the results. Note that the soft error consistency evaluation in multi-core processors are presented in Appendix A since those results were conducted at ROSA (2018).

3.2 Soft Error Reliability Assessment of Machine Learning Techniques

The trend towards having edge computing in our daily lives is to see a wide variety of promising new applications and devices where data collection and analysis are combined (HAO et al., 2021). In this regard, lightweight and performance-efficient CNN inference models have been deployed in resource-constrained devices thanks to software libraries and Application Programming Interfaces (APIs) (LAI; SUDA; CHANDRA, 2018). These libraries/APIs integrate different kernels and quantization techniques (CAPOTONDI et al., 2020) that are devoted to reducing traditional machine learning (ML) models' memory and computational requirements.

While the bespoke and optimized implementations of ML models have been studied extensively, the susceptibility of such algorithms to soft errors is still an open research question. In this sense, fault injection campaigns must be conducted to assess the reliability of ML models and understand the fault vulnerability in different approaches (BREWER et al., 2019). In this direction, recent works conduct fault injections considering software frameworks (GRANAT et al., 2009; LI; PATTABIRAMAN; DEBARDELEBEN, 2018; CHEN et al., 2019), Field-Programmable Gate Arrays (FPGA)s (LIBANO et al., 2017; SALAMI; UNSAL; KESTELMAN, 2018; TRINDADE et al., 2019; KHOSHAVI; BROYLES; BI, 2020), Graphic Processing Units (GPU) (REAGEN et al., 2018; SANTOS et al., 2018; IBRAHIM et al., 2020), and Application-Specific Integrated Circuits (ASIC) (LI et al., 2017; BREWER et al., 2019).

Aiming to evaluate and improve the reliability of a SVM used in the Mars Odyssey spacecraft, GRANAT et al. 2009 proposed a FI tool built on the Valgrind debugger/profiler (NETHERCOTE; SEWARD, 2007) called Basic Instrumentation Tool for Fault Localized Injection of Probabilistic SEUs (BITFLIPS). Such a tool was used to validate an Algorithm-Based Fault Tolerance (ABFT) technique implemented in the SVM by injecting faults in the application variables. The experiments demonstrate a significant improvement in system reliability while using ABFT techniques in different SVM kernel functions. The authors in LI; PATTABIRAMAN; DEBARDELEBEN 2018 proposed the

TensorFI, a FI tool used to evaluate the reliability of TensorFlow ML applications (ABADI et al., 2016). TensorFI uses a Software Implemented Fault Injection (SWIFI) to inject faults at the inference phase of ML operators in TensorFlow. Furthermore, CHEN et al. 2019 proposed the BinFI, an extension of TensorFI, which uses a binary-search technique to pinpoint the safety-critical bits and measure overall resilience in the TensorFlow framework reducing the execution costs. Although both approaches produce significant results, they are restricted to relatively generic software frameworks and do not consider the execution in real devices.

The flexibility of FPGAs enable engineers to design distinct ML models in order to assess different aspects of the soft error reliability in such approaches. For instance, LIBANO et al. 2017 proposed a reliability evaluation methodology for two Feedforward ANNs implemented in an FPGA. Such work started to evaluate the influence of the layer functions complexity in the NN reliability. The experiments consider both radiation-induced and simulation fault injections to evaluate the impact on the *Sigmoid* activation function considering three discretization levels. The study demonstrates that faults occurring at neurons are propagated to multiple instances of the data set having a higher occurrence in hidden layers when compared to output layers. In SALAMI; UNSAL; KESTELMAN 2018, the authors evaluate the resilience aspects of a typical fully-connected NN accelerator processing the Modified National Institute of Standards and Technology (MNIST) dataset (DENG, 2012). This case study considers faults randomly injected in specific NN registers while streaming the NN input data. The authors compare the results from fault campaigns with the golden output data and correlate the inference errors in the classification outputs to the application and architecture specifications. TRINDADE et al. 2019 evaluate the soft error reliability of an SVM under both radiation-induced and simulation fault injections experiments. Such an approach assesses the reliability of an FPGA-designed SVM while classifying the fault's criticality. Such study demonstrates that the high number of soft errors in the target SVM architecture critically affects the SVM accuracy. The authors in KHOSHAVI; BROYLES; BI 2020 investigate the soft error effects in a Binarized Neural Network (BNN) accelerator implemented in FPGA considering two topologies: a CNN composed with twelve layers used to classify the Canadian Institute for Advanced Research (CIFAR)-10 dataset (KRIZHEVSKY; HINTON et al., 2009); and a Fully-Connected NN with four layers used to classify MNIST dataset (DENG, 2012). The experiments were performed with a modified version of the Fast Inference for binarized Neural Networks (FINN) framework (UMUROGLU et al.,

2017) targeting weights and activations of the NNs. In this reviewed case, the results demonstrate that the classification accuracy in the BNN accelerator can drastically drop in the presence of soft errors for the worst-case scenarios. Furthermore, the reviewed FPGA-based works started to assess the soft error reliability of ML models considering different scopes, and demonstrate that radiation-induced soft errors might critically affect both the reliability and accuracy of ML inference models.

Reliability becomes an increasing concern as researchers started using CNNs running on GPU Compute Unified Device Architecture (CUDA) cores applied to safety-critical environments. In this sense, REAGEN et al. 2018 proposed the Ares tool, a fault injection framework that enables the evaluation of soft error reliability of DNNs executing directly on GPUs. Ares is built on top of Keras (CHOLLET et al., 2018), which takes high-level DNN descriptions using either Theano (AL-RFOU et al., 2016) or TensorFlow (ABADI et al., 2016). In this work, the fault injections were performed at specific DNN design points, including weights, activations, and hidden states. The authors demonstrate that the soft error reliability varies across the DNN concerning different scopes such as the model, layer type, and data structure. SANTOS et al. 2018 use SASSI-based Fault Injector (SASSIFI) tool (HARI et al., 2017) and neutron beam experiments to evaluate the reliability of object detection algorithms in GPUs. The study demonstrates a significant reliability reduction in CNNs once the faults occurring in a GPU tend to propagate to multiple threads. Also, soft errors occurring in CNN layers are most likely to generate critical errors (i.e., errors that could potentially impact safety-critical applications). Similarly, IBRAHIM et al. 2020 evaluate layer and kernel vulnerabilities of Residual Network (ResNet) CNN architectures executing on GPUS. This work's experiments show the vulnerability of ResNet kernel and layers while generating a high number of crashes and critical errors. Although GPUs allow a high parallelism and performance capabilities in the execution of CNNs, this advantage generates a high fault propagation through the NN, which means faults are most likely to generate critical errors.

To understand the reliability implications of using high-performance ML accelerators, researchers started to evaluate the fault impact on such devices. LI et al. 2017 modify the Tiny-DNN framework (TINY-DNN, 2017) to inject faults in the buffers of four CNNs running on the specialized Eyeriss DNN accelerator (CHEN; EMER; SZE, 2016). This work compares the DNN application outputs to evaluate the Failure-in-Time (FIT) rates of the Eyeriss accelerator and classify the output rank deviation (i.e., critical SDCs). The study demonstrates that the sensitivity of the DNN depends on different aspects, such as

the network topology and the data type used. Besides that, the buffers implemented to leverage data locality are hugely affected by soft errors, increasing the overall FIT rate of the DNN accelerator. In BREWER et al. 2019, the authors use radiation-induced experiments to evaluate the reliability of a spontaneously Spiking Neural Network (SNN) from International Business Machines (IBM) TrueNorth neurosynaptic systems (AKOPYAN et al., 2015). Although the high occurrence of false positives and false negatives in the SNN output classifications, the authors found that the overall classification accuracy remains unaffected.

The impact of SEUs in the execution of ML inference models, particularly convolutional ones has received more attention recently SANTOS et al.; KHOSHAVI; BROYLES; BI; TRINDADE et al.. Such works demonstrated that the soft error reliability of CNN models executing on resource-constrained devices depends on (*i*) the instruction set architecture, (*ii*) the layer where the faults are injected, and (*iii*) the adopted precision bitwidth. Furthermore, other works have also demonstrated that SEUs occurring in data or NN parameters (e.g., weights and activation quantizations) stored in memory affect inference models' soft error reliability and accuracy. Memory faults (e.g., bit flips in memory elements), which may result from environment perturbations and radiation-induced soft errors, may change the stored data (e.g., CNN parameters), which may lead to large deviations of the inference results (LI et al., 2017; REAGEN et al., 2018). In this context, some works aimed at protecting memory cells to ensure the reliability of inference results in the presence of register and memory faults (AZIZIMAZREAH et al., 2018; GUAN et al., 2019; JASEMI; HESSABI; BAGHERZADEH, 2020). More recently, some works also assess the fault impact considering hardware independent algorithms in TensorFlow backend (BOSIO et al., 2019; PING; TAN; YAN, 2020). In turn, other works investigated the soft error reliability of weights and activations from DNNs (REAGEN et al., 2018; KHOSHAVI; BROYLES; BI, 2020).

LI et al. 2017 investigated the soft error effects on datapath registers and buffers of a DNN considering a specialised Tensor Processing Unit (TPU). REAGEN et al. 2018 started evaluating the soft error reliability of DNN parameters in the training phase through fault injections in bias, weights, activations, and hidden states. In (REAGEN et al., 2018)(SANTOS et al., 2019) authors assessed the impact of soft errors on weights, bias, and hidden states of DNN models (e.g., CNN) executing on GPUs. In TRINDADE et al. 2019, the authors exposed an FPGA board to radiation effects in order to evaluate the soft error reliability in SVM algorithms. Results show that critical faults present

high probability to propagate in along the inference. Moreover, SANTOS et al. 2019 demonstrate that the dataset is critically affected by a single soft error occuring in memory parameters of the CNN executing in GPUs. Similarly, in KHOSHAVI; BROYLES; BI 2020 and LUZA et al. 2020, the analysis comprise the effects of faults on BNN and CNN parameters stored in FPGA based memory. Other works (KHOSHAVI; BROYLES; BI, 2020; LUZA et al., 2020; CORNELIOU et al., 2021) demonstrate that the occurrence of bit-flips in NN's parameters stored in memory affect both reliability and accuracy. Authors have also evaluated the soft error reliability of NN models taking into account different precision implementations varying from floating-point CNNs (SANTOS et al., 2019; LUZA et al., 2020) to 1-bit precision Binarized Neural Networks (BNNs) (KHOSHAVI; BROYLES; BI, 2020). Researchers are also investigating new technologies and memory designs aiming at protecting memory cells to reduce disturbances in NN parameters and buffers in the presence of soft errors (AZIZIMAZREAH et al., 2018)(JASEMI; HESSABI; BAGHERZADEH, 2020). The reviewed works demonstrate that a single fault occurring in bias, weights, and activations may cause critical errors compromising the entire classification and causing the majority of the input vectors to be misclassified, which has a serious effect on applications' reliability and accuracy.

The development of ML models targeting the edge is non-trivial due to the limited computational capabilities and energy efficiency of resource-constrained IoT edge devices (TABANELLI; TAGLIAVINI; BENINI, 2021). In recent years, both academic and industrial researchers have focused their interest on NNs performance and sustainability trade-off, where emerging parallel IoT processors represent an appealing target for tiny ML models since they enable to meet the ML computational constraints (QI; LIU, 2018). Recent works have been proposed parallel solutions to lift the performance of neural network models (ZHANG et al., 2020; GAROFALO et al., 2020; TABANELLI; TAGLIAVINI; BENINI, 2021). GAROFALO et al. 2020 provide quantized neural network kernels ranging from 8-bit to 1-bit fixed-point integer inferences targeting a multi-core RISC-V based cluster. In TABANELLI; TAGLIAVINI; BENINI 2021, the authors employ non-neural ML kernels to maximize the speedup while using floating-point emulations on a multi-core RISC-V platform. Although underlying approaches enables the execution of ML algorithms on parallel devices, software engineers must consider parallelism as a new issue to be addressed while developing lightweight and performance-efficient software to avoid catastrophes in safety-critical applications. In this sense, while most of the existing works consider HPC or architecture-specific approaches (LI et al., 2017; REAGEN et al.,

2018; SANTOS et al., 2018; KHOSHAVI; BROYLES; BI, 2020), the works SANTOS et al. 2018 and ROSA et al. 2019 demonstrate that the parallelization of ML models affects their soft error reliability. Furthermore, the assessment of the soft error reliability at the level of sequential and parallel ML models might be an emergent research question for the next years.

The reviewed works presented different approaches to evaluate several aspects on the soft error reliability of ML models. However, the next step in the soft error reliability assessment is to improve reliability while protect the target platform against radiation effects. Radiation-induced soft errors can be handled either in hardware or software using mitigation techniques. In this sense, the following Section 3.2.1 presents a brief review considering system-level soft error mitigation techniques rather than technology-specific approaches that require control of the chip fabrication process, which is often outsourced.

3.2.1 Review of System-level Soft Error Mitigation Techniques

With the high adoption of nanotechnology manufacturing process the search of methods to reduce the thread of radiation effects in computing systems has become prominent in recent years (KASTENSMIDT; CARRO; REIS, 2006; AVIRNENI; SOMANI, 2011). Soft error mitigation techniques comprise different methods of protection that might be applied either in hardware, software, or in a hybrid mode. While hardware approaches lead to the area and power overhead, software techniques are generally implemented on a per-application basis that usually incurs performance penalties. Most of such techniques implement any type of redundancy that can vary in terms of time or area/space. The temporal redundancy comprises the same combinational logic used at many separate times, while spatial redundancy replicates the computing element main times (MAVIS; EATON, 2002). However, soft error mitigation techniques implemented in hardware require physical modification of the target device during the design phase.

Aiming at mitigating soft errors that affects the processor's control-flow and dataflow without changing the chip design, some mitigation techniques are developed on a per-application basis. Such approaches are used due to flexibility and proper fault coverage while charging some performance and energy penalties. For instance, NICOLESCU; VELAZCO 2003 propose an error detection technique that is based on the introduction of data and code redundancy using a set of transformation rules applied to high-level code. In turn, BENSO et al. 2000 introduce the REliable Code COmpiler (RECCO), a tool that

exploits code reordering and selective variable duplication to generate hardened C/C++ source code automatically. SERRANO-CASES et al. 2019 use genetic algorithms to find a combination of optimization flags that can increase the final binary reliability while maintaining a reasonable performance and memory utilization trade-off. The authors in RODRIGUES et al. 2016 developed software-based Triple Modular Redundancy (TMR) and Conditional Modular Redundancy (CMR) mitigation implementations, aiming to reduce the occurrence of soft errors in a Cortex-A9 processor running Linux kernel.

Another software-based alternative to mitigate soft errors comes from low-level code protection. Authors in REIS et al. 2005 promote the SoftWare Implemented Fault Tolerance (SWIFT) technique aiming to reduce the overhead associated with Error Detection by Duplicated Instructions (EDDI) (OH; SHIRVANI; MCCLUSKEY, 2002). They remove duplicate store instructions, reducing both memory and performance overhead. The SWIFT technique assumes that the system's memory architecture is protected by some error correction mechanism. Results showed a 14% speed-up over EDDI when tested with an Intel Itanium 2. Furthermore, DIDEHBAN; SHRIVASTAVA 2016 improved SWIFT technique by checking the load instructions right after a store instruction and creating redundant load instructions in critical sections to achieve near-zero the occurrence of SDC. A popular instruction-level mitigation technique introduced by REIS; CHANG; AUGUST 2007 is the SWIFT-R, which implements a TMR to recover from soft errors in the register file. Instead of duplicating instructions, it triplicates, and change the checking points to a voter mechanism.

In FENG et al. 2010, authors presented the Shoestring technique, which exploits a low-cost symptom-based error detection mechanism that focuses on applying instruction duplication to protect only those code segments that are likely to result in user-visible faults and do not exhibit symptomatic behaviour. Results show that Shoestring can recover from an additional 33.9% of soft errors that are undetected by a symptom-only approach. Authors in FENG et al. 2011 present the Encore, a software-based error recovery mechanism (paired with other error detection techniques) that combines program analysis, profile data, and simple code transformations to create code portions, which can recover from faults at a minimal cost. Gathered results show that Encore can recover from 97% of transient faults on average with 14% additional runtime overhead. Another TMR-based technique, called ELZAR, is proposed in KUVAISKII et al. 2016. It triplicates arithmetic and logical operations, and the voting mechanisms are inserted between register operands of memory and control flow operations for recovery. To reduce the

performance overhead introduced by replicated instructions, they utilize Intel Advanced Vector Extensions (AVX), which includes SIMD instructions. The experiments show that the performance overhead is reasonable for CPU-intense applications with many floating-point operations. However, for some case studies, the instruction-level parallelism was inefficient, resulting in a performance penalty that surpassed the SWIFT-R technique.

The NEMESIS technique introduced by DIDEHBAN; SHRIVASTAVA; LOKAM 2017 is a duplication with recovery technique. It replicates instructions and checks the results of memory write operations and branches' direction. If an error is detected, it then recovers to a valid state if possible; otherwise, a power restart is needed. The results, obtained from ten selected applications, show that at least 97% of the detected errors are recoverable. Another error recovery technique is the InCheck (DIDEHBAN; LOKAM; SHRIVASTAVA, 2017), which is an extension of the near Zero silent Data Corruption (nZDC) technique (DIDEHBAN; SHRIVASTAVA, 2016). The proposed technique comprises of error detection, diagnosis, and recovery schemes. Unlike SWIFT-R, the InCheck mitigates faults by protecting error handling routines in addition to the main program instructions. The authors claim that their technique offers complete error coverage for the tested applications.

3.2.2 Contribution in Machine Learning Soft Error Assessment and Mitigation

Most of the reviewed works consider different approaches to evaluate the reliability of ML models in distinct scopes. Some of the reviewed works evaluate the soft error reliability considering the fault injections in ML operators (e.g., data and parameters) through the early phases of the ML model development framework (i.e., TensorFlow, PyTorch, Keras). Either FPGA and GPU-based works consider both radiation-induced and simulation fault injections to evaluate soft error reliability of different ML models and algorithms. Regarding evaluations, specific points of the analysis stand out considering the layers of the NN and the types of data used in the inference. These characteristics are relevant since they are directly related to the effect of failures both in the reliability and accuracy of ML models. Even with an overall resilience, ASIC-based ML accelerators presented a high occurrence of false positives and false negatives, which can mean catastrophic consequences when applied to safety-critical environments. Table 3.2 shows related works considering soft error assessment in ML models and highlights the approach adopted in this Thesis.

Emerging IoT systems are expected to incorporate ML models aiming to recognize patterns and predict how systems (e.g., medical) would react to unexpected circumstances (MAHDAVINEJAD et al., 2018; AMOH; ODAME, 2019). Aiming to enable the execution of such computationally intensive ML models under resource-constrained IoT devices, researchers and industrial leaders are investigating software libraries and APIs. Such libraries/APIs are devoted to support the efficient execution of ML models at reduced memory footprint, which is critical to edge-computing devices (LAI; SUDA; CHANDRA, 2018). Another approach relies on developing bespoke and optimized ML models, allowing their execution on battery constrained edge IoT devices. The resulting benefit comes at the cost of precision, which has crucial importance in the ML models efficiency and applicability.

In the context of soft error assessment, with the exception of TRINDADE et al. 2020 and the work resulting from this Thesis, reviewed approaches do not consider resource-constraint on their experiments. The majority of these works consider either FPGA implementations of ML models (LIBANO et al., 2019; TRINDADE et al., 2019; LUZA et al., 2020) or their execution on GPU (SANTOS et al., 2018), DNN accelerators (LI et al., 2017; REAGEN et al., 2018; KUNDU et al., 2021) or general-purpose processors (ROSA et al., 2019; CHEN et al., 2019). On the soft error mitigation side, traditional partial TMR or specific mitigation techniques have been considered either in FPGA implementations (LIBANO et al., 2019) or applied to specialized hardware accelerators (LI et al., 2017) or more generic GPUs (SANTOS et al., 2018).

Table 3.2 – Related works considering soft error assessment of ML techniques.

Author and Year	Environment	FI Type	ML Algorithm	Datasets	Target FI	Mitigation	Target Evaluation
GRANAT et al. 2009	Framework (Valgrind)	Simulation	SVM	Spacecraft dataset	Application Memory	ABFT	Reliability
LI et al. 2017	Eyeriss accelerator	Simulation	AlexNet, CaffeNet, NiN, ConvNet	CIFAR-10, ImageNet	Mapped hardware components (i.e., datapath registers, buffers)	Symptom-based, Selective Latch Hardening	Reliability (Failure-in-Time rates), accuracy
LIBANO et al. 2017	FPGA	Simulation, radiation-induced	2 Feedforward ANNs	Iris flower, Boston Housing	Flip-flops of the configuration logic block		Reliability (activation functions)
SANTOS et al. 2018	GPU	Simulation, radiation-induced	YOLO, Faster R-CNN, ResNet	PASCAL VOC 2012, Caltech Pedestrian	GPU's ISA visible state (register file, memory)	ABFT	Reliability (safety-critical CNNs on GPUs)
REAGEN et al. 2018	GPU	Simulation	LeNetFC, LeNetCNN, CF-VGG, VGG16, ResNet50, TiGRU	CIFAR-10, ImageNet, MNIST, TIDIGITS	DNN weights, activations, hidden states		DNN reliability (model, layers, data structures)
SALAMI; UNSAL; KESTELMAN 2018	FPGA	Simulation	Fully Connected NN	MNIST	Specific NN registers		Reliability, accuracy
BREWER et al. 2019	IBM's TrueNorth	Radiation-induced	SNN	MNIST	TrueNorth chip		Reliability, accuracy (false positives, false negatives)
BOSIO et al. 2019	Framework (Tiny-CNN)	Simulation	LeNet, Tiny YOLO	MNIST, COCO	Memory (weights)		Reliability
ROSA et al. 2019	Virtual Platform	Simulation	CNN	KITTI Visual Odometry	Register file		correlates multi-core microarchitecture, soft errors
TRINDADE et al. 2019	FPGA	Simulation, radiation-induced	SVM	150 In-house input vectors	SVM architecture nodes		Reliability, accuracy
LIBANO et al. 2019	FPGA	Simulation, radiation-induced	2-layer ANN, 7-layer CNN	Iris flower, MNIST	Flip-flops of the configuration logic block	Partial TMR, Full TMR	Reliability (activation functions)
SANTOS et al. 2019	GPU	Simulation, radiation-induced	YOLO	Caltech Pedestrian	GPU's ISA visible state (register file, memory)		Reliability of double (64 bits), single (32 bits), and half (16 bits) floating point precisions
TRINDADE et al. 2020	STM32 L45RE-P board	Radiation-induced	ANN, SVM	Iris Flower	Register file		Reliability, accuracy
IBRAHIM et al. 2020	GPU	Simulation	ResNet	ImageNet	Register File, Instruction Output Address, Instruction Output Value		Reliability
PING; TAN; YAN 2020	TensorFlow Framework	Simulation	LeNet-5, ResNet-50, CIFAR-10 CNN, VGG-16	MNIST, CIFAR-10 CNN, and ImageNet	Memory (weights, activations)	SEC-DED ECC, Reduntant Layers	Reliability
KHOSHAVI; BROYLES; BI 2020	FPGA	Simulation	2 BNN: CNN and Fully connected NN	CIFAR-10, MNIST	Memory (weights, activations)		Reliability, accuracy (dataset optimizations)
LUZA et al. 2020	FPGA	Radiation-induced	LeNet-5 CNN	MNIST	HyperRAM memory (weights, input data)		Reliability, accuracy (32-bit float, 16-bit and 8-bit integer)
CHEN; LI; PATTABIRAMAN 2021	GPU	Simulation	LeNet, AlexNet, VGG11, VGG16, ResNet-18, SqueezeNet, Nvidia Dave, Comma.ai	MNIST, Cifar-10, GTSRB, ImageNet, Driving	TensorFlow graph (ALUs and pipeline registers)	Selective Range Restriction	Reliability (Reduce Critical SDCs)
ADAM; MOHD; YOUNIS 2021	GPU	Simulation	AlexNet	ImageNet	Register File, Instruction Output Address, Instruction Output Value		Reliability (Program Vulnerability Factor)
CORNELIOU et al.2021	FPGA	Emulation	AlexNet	ImageNet	Register File, Memory (weights, activations)		Reliability (Average Vulnerability Factor)
This work	***Virtual Platform***	***Simulation***	***7-layer CIFAR-10 CNN (CMSIS-NN), 29-layer MobileNet (CMix-NN)***	***CIFAR-10 and ImageNet***	***Register file, Function Lifespan (CNN layers, Activations), and Memory sections (Flash and RAM)***	***P-TMR and RAT***	***Reliability and accuracy of reduced and mixed precision ML algorithms targeting resource constrained IoT edge devices.***

Source : The authors

Regarding the findings from the reviewed works highlighted in Table 3.2, soft errors can affect either the reliability and the accuracy of ML models. In this sense, the authors demonstrate that the fault effects vary according to different aspects of the ML model: the model, the topology (i.e., layers and activations), and the dataset (i.e., data type and data optimizations). Different from the above works mentioned in Table 3.2, this Thesis *contributes* with the soft error assessment of reduced and mixed precision CNNs developed with specialized NN kernels (CMSIS-NN and CMix-NN), which are devoted to improve the performance and minimize the memory footprint of NN -based applications. To better understand how soft errors affect the reliability of the ML models, gathered results had been obtained through fault injection campaigns conducted with SOFIA framework (BANDEIRA et al., 2019). The proposed evaluation comprises both the reduced precision CMSIS-NN (LAI; SUDA; CHANDRA, 2018) kernels, which support 8-bit fixed point quantizations, and CMix-NN (CAPOTONDI et al., 2020) kernels, which supports 8, 4, and 2 bit mixed precision quantizations. The adopted case studies are: *(i)* the CIFAR-10 CNN composed of 7 layers developed with CMSIS-NN and trained with CIFAR-10 dataset (KRIZHEVSKY; HINTON et al., 2009); and *(ii)* the MobileNet CNN composed of 29 layers developed with CMix-NN and trained with ImageNet dataset (DENG et al., 2009). To evaluate the soft error reliability considering the application inference flow, this work employs a fault injection technique that isolates the critical functions of the CNN model's layers. In order to conduct a more relevant assessment, this Thesis also promotes a fault classification, which considers the impact of critical faults on the output predictions of evaluated ML models. Furthermore, to asses the impact of soft errors in the ML model's parameters (e.g., activation values) storage in memory elements, SOFIA was extended to enable the fault injection in isolated memory sections (i.e., Flash and RAM).

In summary, this Thesis innovates from previous work in five key directions:

- First, this is the first work to investigate the relationship between the soft error susceptibility and reduced precision CNN models, which is completely ignored in the works presented in Table 3.2;
- Second, this work evaluates the soft error reliability of reduced and mixed precision CNN applications executing on resource-constrained IoT devices, considering different aspects: register file, layers, activations, multithreads, and memory sections;
- Third, this Theses focuses on reducing the occurrence of soft errors in resource-constraint devices. Therefore, this is the first work to evaluate the benefits of using a lightweight technique, RAT, w.r.t. a partial replication technique;

– Fourth, this work explores the relative performance, memory utilization, and soft error reliability trade-offs of two system-level mitigation techniques considering a microprocessor running different precision bitwidth variations of MobileNet on ImageNet;
– Fifth, extensive and consistent soft error assessment of adopted case studies considering more than 14.8 million fault injections while maintaining acceptable consistency metrics defined in the literature (LEVEUGLE et al., 2009).

4 SOFT ERROR ASSESSMENT METHODOLOGY

This Chapter details the adopted fault injection approaches and their fault injection modules, used to evaluate the soft error resilience of single or multi-core systems (Section 4.1). In this sense, each fault injection module is detailed to show its usefulness, that is, RTL (Section 4.1.1), gem5 (Section 4.1.2.1), and OVPsim (Section 4.1.2.2). Next, Section 4.2 details the adopted fault classification, which is implemented in the three FIMs so that the results are automatically classified. Section 4.3 presents the assessment metrics used in this work, which guarantees the consistency and statistical significance of the results and subsequent conclusions of the research. Finally, Section 4.3.1 details the adopted software-based soft error mitigation techniques used in this Thesis.

4.1 Fault Injection Frameworks

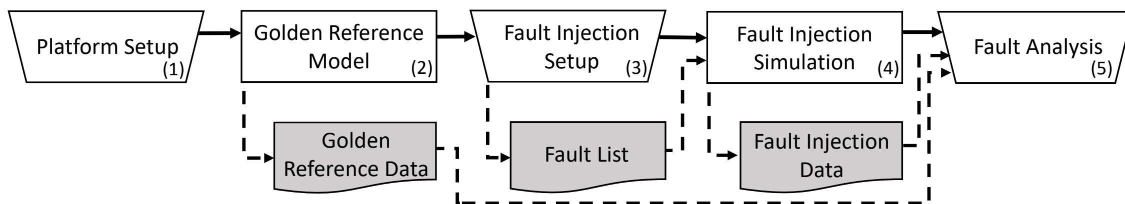
This Section first describes two event-driven fault injection frameworks based on Questa Advanced Simulator (Section 4.1.1) and gem5 (Section 4.1.2.1), the latter is integrated into the SOFIA framework. Such approaches are used in the present work as *references* for the assessment of single and multi-core systems since RTL descriptions represent the synthesizable real processors and gem5 is known as an accurate simulator for complex multi-core systems(BUTKO et al., 2012). The main functionalities of SOFIA are presented in Section 4.1.2.2, and then Section 4.1.2.2 describes the OVPsim-based fault injection module.

4.1.1 RTL Fault Injection Module

This Section depicts the developed fault injection module for RTL descriptions, similar to that presented in (ABBASITABAR; ZARANDI; SALAMAT, 2012; BORTOLON et al., 2018). The main distinction between FIMs is the moment in which the fault is injected. RTL-FIM executes under a discrete event model of computation (e.g., Questa Advanced Simulator), enabling the injection of faults at any or even within half a clock cycle. Therefore, inserted faults may affect the behavior of both the current and the next instructions. Developed RTL fault injection module explores built-in simulator commands and its observability capability to control and monitor the internal signal of a

given processor without requiring any changes in its description.

Figure 4.1 – Fault injection campaign flow applicable to the three fault injection approaches, i.e., RTL, gem5 and SOFIA.



Source : The authors

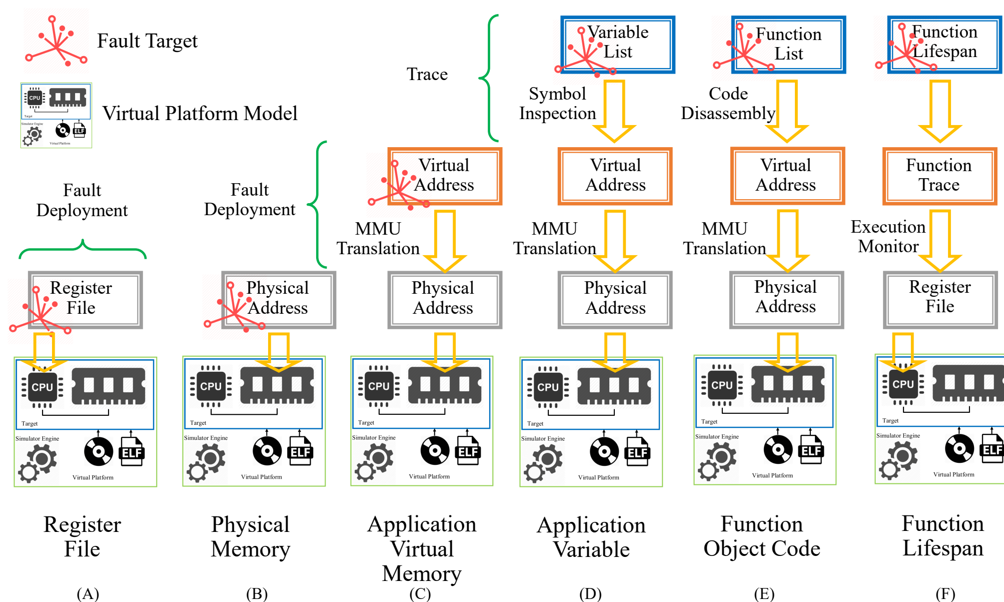
This approach assumes that the fault injection campaigns comprise the five phases illustrated in Figure 4.1: (*1*) *Platform Setup*; (*2*) *Golden Reference Model*; (*3*) *Fault Injection Setup*; (*4*) *Fault Injection Simulation*; (*5*) *Fault Analysis*. First, the Platform Setup defines the parameters used in the fault campaign, such as the target architecture and the evaluated application. In the second phase, the FIM executes the target system to extract its behavior under ideal circumstances (i.e., no presence of faults). To improve the performance of the RTL fault injection campaign, this step generates checkpoints from simulation time slices. Then, in the Fault Injection Setup phase, the engine defines the fault configuration, which consists of its location (e.g., register, memory address), position (e.g., register bit), and its insertion time. The fault injection configuration relies on a random uniform function, which is a well-accepted fault injection technique since it covers the majority of possible faults on a system at a low computation cost(LEVEUGLE et al., 2009; CHO et al., 2013). In the Fault Injection Simulation phase, the FIM loads the checkpoint, executes the target system architecture in the presence of the configured faults (i.e., flipped bits), and extracts its behavior. Throughout the Fault Injection Simulation, the developed engine uses available interrupt signals to detect any unexpected activity. Finally, in the last phase, the FIM compares the results against its golden reference data to automatically classify the occurred faults.

4.1.2 SOFIA Framework

Rather than develop a fault injection (FI) framework from scratch, this Thesis adopted SOFIA - an open-source toolset developed by (BANDEIRA et al., 2019; ROSA, 2018). The SOFIA framework integrates well-accepted FI techniques along with several

facilities (e.g., error tracer module), which enable reliability and software engineers to identify and classify the effects of soft errors on the system's behavior, considering both hardware and software architectures. The adopted framework emulates the occurrence of Single Bit Upsets (SBU)s by injecting faults into pre-selected register or memory locations during the execution of a given software stack (i.e., kernels, drivers, and applications). In this work, SBUs targets only storage elements due to its higher susceptibility to radiation events when compared to logic elements (SEIFERT et al., 2012). Furthermore, register FI modeling is useful to evaluate low-end processors' soft error reliability since it concentrates injection into the more critical structures of the ISA (SCHIRMEIER; BREDDEMANN, September 2019). Nevertheless, the framework supports the injection of bit-flips in the six different scopes presented in Figure 4.2: (a) register file, (b) physical memory, (c) application virtual memory, (d) application variables and data structures, (e) function object code, and (f) function lifespan. The FI techniques provided in SOFIA can be implemented in any simulation environment that provides access to the system Memory Management Unit (MMU) (BANDEIRA et al., 2019). Note that in addition to *random register file*, the *function lifespan* technique consider the injection of bit-flips in the register file during the execution of a selected function, while the remaining techniques consider the MMU translation to inject bit-flips in physical memory addresses. The following Sections describe the fault injection modules available in SOFIA framework.

Figure 4.2 – Fault injection types available in SOFIA.



Source : Adapted from (BANDEIRA et al., 2019)

4.1.2.1 SOFIA: gem5 Fault Injection Module

The SOFIA framework uses the gem5 (BINKERT et al., 2011) among the available cycle-accurate virtual platform simulators due to its open and free availability as well as its support for the Arm processor architectures with four CPU models, which differ in speed/accuracy trade-offs. gem5 also supports a rich set of component models, including processor cores, memories, caches, and interconnections. It targets microarchitecture explorations, which incurs substantial simulation overheads due to the number of modeled aspects; typically, it reports simulation performances varying form 2 to 10 Million Instructions Per Second (MIPS).

Additionally, gem5 is a well-known simulator used in many research projects underlying the soft error assessment systems (PARASYRIS et al., 2014; KALIORAKIS et al., 2015; CHATZIDIMITRIOU et al., 2019; ROSA et al., 2018). Our gem5-based FIM follows the five-phase fault injection flow illustrated in Figure 4.1.

Although the phase split is the same for the three FIMs, each one has its own implementation. The first difference w.r.t. RTL is in the platform setup. Here, the application, the kernel, and the configuration of the target architecture are compiled to simulate together in the first phase of the flow. Another difference is on how to inject a fault. While RTL relies on proprietary Questa Advanced Simulator commands such as *force -deposit*, gem5-FIM employs Python scripts to control the simulation flow and uses C/C++ modules to model the microarchitectural components. The deployed fault injection approach minimizes intrusion into the simulator's engines, allowing any researcher in possession of the original simulator to use, modify, or extend its functionality.

4.1.2.2 SOFIA: OVPsim Fault Injection Module

Due to the high simulation speed (typically at hundreds of MIPS), virtual platform simulators based on JIT dynamic binary translation appear to have an advantage over event-driven simulators (ROSA et al., 2015). The SOFIA framework also considers the M*DEV toolset (IMPERAS, 2021a), an advanced version of the OVPsim (IMPERAS, 2021b) that includes specific tools to improve the development and verification of embedded software, utilizing virtual platforms. Among available JIT-based virtual platforms, OVPsim (IMPERAS, 2021b) distinguishes by its rich number of component models, which includes more than 170 processor variants, memories, Universal Asynchronous Receiver Transmitter (UART), among other components. Note that the JIT-based simulator remains

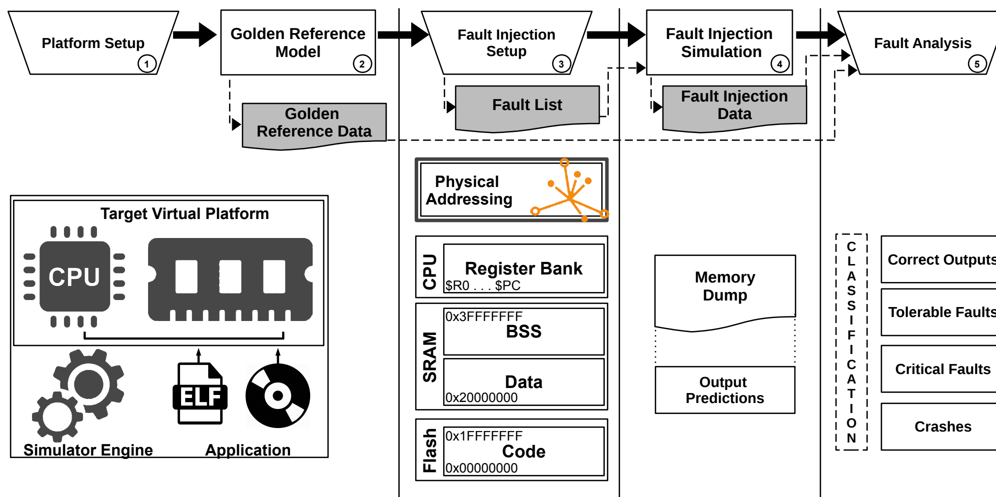
OVPsim and the M*DEV's capabilities are used to enable and improve fault injection techniques and campaigns. In this sense, OVPsim-FIM is an instruction-accurate simulation engine, and thus faults can only occur between instructions. Consequently, injected faults can only influence the behavior of the next instructions. The lack of micro-architecture components and its untimed simulation engine restrict the soft error reliability analysis and impact on the results accuracy. Results demonstrating the loss of accuracy w.r.t. lower level FI simulation approaches is discussed in Sections 4.1.1 and 4.1.2.1.

OVPsim-FIM also relies on the five-phase fault injection flow shown in Figure 4.1. However, OVPsim-FIM has an improved fault injection infrastructure, including two simulation techniques to boost the fault injection assessment: checkpoint and an independent parallel simulation engine. The checkpoint technique consists of collecting platform components context during the Golden Reference Model (phase 1) to restore the appropriate context later during the FI campaign, reducing the amount of re-executed code and, consequently, accelerating the simulation time. This technique is built using M*DEV's save and restore functions, allowing restoring the processor and memory context. In addition, the independent parallel simulation technique benefits from the host's processing capacity. The proposed simulation infrastructure comprises a set of C-based functions (e.g., management) and bash scripts developed according to the OVPsim guidelines (IMPERAS, 2021a). The goal is to allocate one platform model per available host-core, enabling the execution of multiple fault injection campaigns in parallel (ROSA et al., 2015). For example, considering a quad-core processor machine, it is possible to run four platform models injecting 1,000 faults each, reaching 4,000 fault injections in parallel. This being one of the techniques used to enable the assessment of an extremely large number of FI scenarios in this work.

Among the different scopes of bit-flip injections, this work considers the *random register file*, the *function lifespan*, and the *physical memory* to assess complex safety critical applications, for instance, deep inference networks. Aforementioned reviewed works indicate that engineers must be able to early asses the soft error reliability considering both the application execution flow and the storage elements used to accommodate the object code and a given dataset of a neural network application. On the one hand, the *random register file* is a FI technique that homogeneously stimulates all general-purpose registers that execute the CNN application and operating system codes. On the other hand, *function lifespan* reduces the FI spectrum by limiting the insertion time to those small intervals where the target function is active. As the name implies, *physical memory*

reproduces faults coming from radiation particles through bit-flip injections into system memory. In the latter technique, to assess the different effects of soft errors occurring in storage elements, this work extends the OVPsim-FIM to inject faults into specific memory sections (e.g., RAM or Flash). Such facility allowed us to assess the reliability of the CNN data, both stored in register files and memory, as later discussed in Section 6.1.3 and Section 6.2.3.

Figure 4.3 – Proposed extension in the physical memory FI technique.



Source : The authors

Figure 4.3 illustrates the proposed extension inside the framework FI flow. The developed extension uses the application memory map defined by the compiler to inject faults into the memory section selected in the *Platform Setup* phase. At phase 3, the scripts generate the fault list according to the mapped addressing range. Further, it maps the injection address and the application's output variable to verify and classify the impact of the failure. Note that developed extension is totally non-intrusive, thus not influencing the platform simulation process and consequently without impacting the performance at the fault injection campaign.

4.2 Fault Classification

Aiming to achieve a reasonable confidence level, the three fault injection modules execute the FI campaigns according to an adopted statistical analysis (Section 4.3) to characterize the target architecture behavior in the presence of faults, so that additional campaigns do not disturb the result. By default, the gathered results are classified according

to the well accepted classification proposed by CHO et al. 2013, which defines five possible behaviors for a system in the presence of soft errors:

- ***Vanish***: no fault traces are left in both memory and architectural state.
- ***Output Not Affected (ONA)***: the resulting memory is not modified, however, one or more remaining bits of the architectural state is incorrect.
- ***Output Memory Mismatch (OMM)***: the application terminates without any error indication, and the resulting memory is affected.
- ***Unexpected Termination (UT)***: the application terminates abnormally with an error indication.
- ***Hang***: the application does not finish requiring a preemptive removal.

Complementary to Cho's classification, this work *proposes* a bespoke fault classification that evaluates the impact of soft errors on the output probabilities of ML case studies. This classification follows the pattern shown in (LI et al., 2017; TRINDADE et al., 2019; KHOSHAVI; BROYLES; BI, 2020), where the authors identify the faults in the outputs as: *correct output*, *tolerable faults*, *critical faults*, and *crashes*. As shown in Figure 4.3, the proposed classification relies on the application output trace during either golden and fault injection simulations. The resulting faulty data is automatically classified according to the following behavior:

- ***Correct outputs***: are the scenarios where the output probabilities (e.g., recognized top ranked classification) are the same as faultless execution (i.e., Vanished and ONA);
- ***Critical faults***: consider only the OMM faults that affect the output with incorrect probabilities and no predictions (i.e., cases where odds are dispersed, therefore, they have no probabilities in the output data). Incorrect probabilities are those results from OMM, which covers two possibilities: *(i)* when the top-ranked classification differs from the one predicted in the fault-free execution of the ML model; *(ii)* when there is no top-ranked classification (i.e., the odds are dispersed). No prediction cases are those that have no probabilities in the output data.
- ***Tolerable faults***: are the remaining OMMs, which are those that have the top-ranked classification equals to the fault-free execution of a NN.
- ***Crash***: comprises the application that ends abnormally with an error indication or does not finish, requiring a preemptive removal after a threshold execution time (i.e., effects of UT and Hangs).

Note that such proposed fault classification have been developed based on classification of CNN models. However it can be used for any ML model that saves the output (i.e., probabilities or other either information) after it's execution. Furthermore, it is essential to mention that the main difference between critical fault and crash is that the former refers to a silent error (i.e., the application ends without an error signal) while the latter is a detectable one (i.e., an error signal or unexpected behaviour). Silent errors are considered critical in this work as they can be propagated, which might ultimately incur in human life losses for safety-critical applications (e.g., autonomous vehicles). In contrast, detectable errors can be handled by the system as there is the possibility to reset the system or rerun the algorithm to obtain the correct result.

4.3 Assessment Metrics

One of the main concerns when assessing a system's reliability is to develop a precise, well-covered and realistic approach. In this sense, this work sought to ensure that the number of fault injections has a statistical significance by applying the equations developed by (LEVEUGLE et al., 2009). Equation (4.1) shows the minimum number of campaigns needed to cover a certain confidence level with their respective margin of error. While the confidence level assures that if we repeat the experiments, the same results are obtained, and the margin of error indicates the percentage difference between the obtained results and the real value of the population.

$$n = \frac{N}{1 + e^2 \times \frac{N-1}{t^2 \times p \times (1-p)}} \tag{4.1}$$

where: N is the initial population, p is the estimated probability of a failure (defined as 50%), e is the considered margin of error, and t is the minimum required confidence level (usually 95% is eligible), which is calculated from *t-Distribution* table (MCKAY, 1932) considering the number of exposed bits as degree of freedom.

Our initial population is the product of possible spatial and temporal fault injections, i.e., the location (e.g., register, memory address) and the position (e.g., register bit) at which the bit-flip will be applied to and its insertion time (e.g., a random clock cycle), respectively. An essential characteristic of the Equation (4.1) is that when the initial population (N) is large, its increase has little influence on the FI campaign size for a given margin of error and confidence level. For example, for a population greater than 1

million (N), if the chosen confidence level is 99% (in which the calculated t corresponding to 2.575829303549), with a margin of error of 5% ($e = 0.05$), we must perform at least 664 fault injections to provide the necessary confidence in our results.

While maintaining consistency of results, the use of metrics is essential to assess the soft error reliability of a given system. Therefore, this work uses the *Mean Work To Failure (MWTF)* metric (REIS; CHANG et al., 2005) to properly assess the soft error reliability impact on the adopted case studies (Chapter 6). Complementary to the fault classification, the MWTF shows the average amount of work that an application can perform until reaching a failure (i.e., higher values are better). This is a fair metric to either compare or evaluate the effects generated by different mitigation techniques. This metric is evaluated in the *Fault Analysis* step (Figure 4.1). Note that for deep inference networks, the unit work is defined as the relationship between the application's runtime and the most critical vulnerability (i.e., *critical faults*), as shown in Equation (4.2).

$$MWTF = \frac{1}{(execution\ time \times AVF_{CriticalFaults})} \tag{4.2}$$

The Architecture Vulnerability Factor (AVF) is used to measure the probability of a fault result in an error (i.e., SDC or Crash) (MUKHERJEE et al., 2003). The AVF critical considers only the SDCs that actually led to wrong classifications. For example, in safety-critical applications, such as autonomous cars, a critical fault can alter the detection of an obstacle in front of the vehicle, which can lead to an accident. For this reason, this work uses the critical-based AVF ($AVF_{critical\ faults}$).

The execution time metric is not ever available in virtual platform simulators such as OVPsim. In this sense, to provide accurate and consistent analysis all case studies have been validated in an of those environments: evaluation board, RTL description, or gem5 simulator. Such environments provide accurate execution time metrics according to the respective platform, which is defined for each case study at the experimental setup (i.e., platform setup phase).

4.3.1 Supported Soft Error Mitigation Techniques

To ensure failsafe functionality of ML-based systems, reliability engineers should be able not only to identify but also explore efficient mitigation solutions to reduce the

occurrence of soft errors. SOFIA provides support to 2 mitigation techniques, including *Partial Triple Modular Redundancy (P-TMR)* and *RAT*.

The first mitigation technique is based on a replication approach, i.e., a technique that replicates instructions (except stores and branches) and adds majority voters before conditional branches, load, and store instructions on top of the language-independent Low Level Virtual Machine (LLVM) Intermediate Representation (IR) (LATTNER; ADVE, 2004) compilation environment. However, unlike traditional approaches that replicate the entire code, this work uses the *P-TMR* technique that only replicates specific/critical functions, thus minimizing the performance overhead.

The second adopted mitigation technique is the *register allocation technique* (*RAT*) (GAVA; REIS; OST, 2020). This hardening technique restricts the number of available registers used to execute a specific functions, thus reducing the exposed area. Unlike replication approaches, RAT does not involve code redundancy and is an architecture-independent approach. Also, RAT is a compiler-based mitigating technique; thus, it can be associated with other mitigation techniques.

This work uses the underlying techniques as means to improve the soft error reliability of deep inference models as highly explored in Chapter 6.

5 EARLY SOFT ERROR CONSISTENCY ASSESSMENT

This Chapter presents the evaluation of the soft error assessment consistency considering the SOFIA OVPsim-FIM w.r.t. RTL and gem5 FIMs. This Thesis contribution lead to the ABICH et al. 2021 publication. The Section 5.1 and Appendix A detail the case studies adopted to assess the consistency of the results considering single-core and multi-core architectures respectively. As mentioned before, the soft error results' consistency regarding multi-core architectures are presented in the separate Appendix A of this thesis as they were generated by ROSA (2018).

5.1 Soft Error Consistency Assessment for Single-core Processors

This Section aims to thoroughly assess the SOFIA soft error consistency when targeting single-core processors. In this sense, this work considers two real commercial RTL processor descriptions from the Arm Cortex-M family (i.e., Arm Cortex-M0 and Cortex-M3), both available under the Arm University Program (ARM, 2020). SOFIA natively supports both processor models, and the synthesis-ready netlist descriptions of the underlying processors were used to conduct the fault injection campaigns at the RTL level. Section 5.1.1 presents the experimental setup used in the proposed case study. The first results in Section 5.1.2 cover the simulation performance brought by the JIT simulator used in SOFIA. Further, in Section 5.1.3, we analyze the consistency and the reliability of single-core systems considering different ISAs software stacks (Section 5.1.3.1), and cross-compilers (Section 5.1.3.2).

5.1.1 Experimental Setup

To provide trustworthy results, conducted experiments consider more than 8.5 million fault injections using 26 applications and varying parameters such as target processor, software stack, and cross compiler with its optimization flags. Table 5.1 presents the proposed experimental setup used to measure the soft error assessment consistency of SOFIA with respect to the RTL approach.

Selected applications from the Mälardalen WCET benchmark suit (GUSTAFSSON et al., 2010) include: Adpcm, Binary Search, Bit Manipulation, Blowfish, Bubble

Table 5.1 – RTL vs. SOFIA Experimental Setup. The * means a sub-flag used in conjunction with other standard flags (e.g., O0, O1, O2, O3).

Processors	Arm Cortex-M0 and Cortex-M3
Software Stack	Bare-Metal and FreeRTOS
Benchmark Suite	Mälardalen WCET
Number of Applications	26
Compilers	GCC 4.9, GCC 7.2, Clang 6, Arm 6.10 and Arm 5.06
Optimization Flags	O0, O1, O2, O3, Os, Ofast, Ospace and Otime
Number of Compiler Sets (with optimization flag)	32
Number of FI Campaigns	1140
Injections per Campaign	1,000 (Section 5.1.3.1) and 10,000 (Section 5.1.3.2)
Total Fault Injections	208,000 (Section 5.1.3.1) and 8,320,000 (Section 5.1.3.2)

Source : The authors

Sort, Counts, CRC, Data Compression, Dhrystone, Edn, Exponential Integral, Factorial, Fdct, Fibonacci, Hanoi Tower, Harmonic Calculations, Insert Sort, Jfdctint, Matrix Multiplication, MDC, PeakSpeed, Petri Net, Prime Numbers, Switch Cases, Ud, and Usqrt.

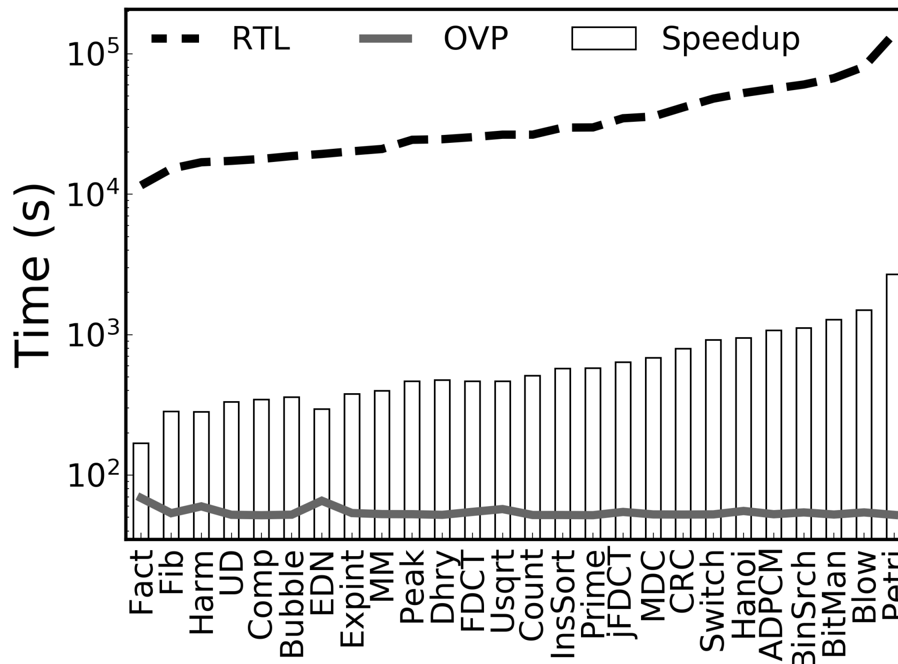
Results were performed on a Linux machine with a Quad-core Intel® Core™i7-7700K CPU and 32 GB DDR4 RAM memory. Fault analyses are obtained by injecting faults (i.e., bit-flips) into the processor's registers (i.e., R0-R15) in a random and uniformly distributed manner, which is widely accepted that circuits are affected (MUKHERJEE et al., 2003). The purpose is to analyze the parameters and Arm processors according to the AVF (MUKHERJEE et al., 2003), i.e., a percentage estimate of errors that are not masked, considering the different applications.

5.1.2 FI Simulation Performance of SOFIA w.r.t. RTL

Although electronic hardware engineers usually describe their circuit designs using Hardware Description Languages (HDL) and fault injection at that level seems to be more straightforward due to the accuracy of the results, there are two main reasons for not using HDL models. First, commercial processors are rarely available to Universities and general users in HDL descriptions (DEVOE, 2015). Second, the simulation time at this level is exceptionally high. This particularity made the simulation speedup of fault injection campaigns one of the leading motivations found in the literature to use virtual platforms (KALIORAKIS et al., 2015; ROSA et al., 2017). In this sense, our first experiment enable to quantify the simulation speed for each of the adopted applications, as shown in

Figure 5.1.

Figure 5.1 – Speedup of SOFIA over the RTL-FIM, considering the Cortex-M0 executing 26 benchmarks in bare-metal.



Source : The authors

Figure 5.1 depicts a comparison between the simulation time required to execute a complete fault injection campaign in the RTL and SOFIA approaches. To make straightforward and trustworthy comparisons, the simulation time was extracted considering the Arm Cortex-M0 and bare-metal applications. Results show that SOFIA achieves a remarkable simulation performance, reaching more than three orders of magnitude speedup when compared to the detailed fault injection approach conducted at RTL. Thus, if RTL is considered, thousands of simulations may take several months, which is not suitable to assess the soft error resilience of electronic computing systems. In this context, the utilization of a fault injection JIT-based virtual platform is promising since it can also be used to compare different processor models, ISAs, and benchmarks considering complex Operating Systems (OS) and large scenarios, as shown in Section 5.1.3.

5.1.3 Soft Error Reliability Mismatch

This Section presents the accuracy results of the soft error resilience assessment considering single-core processors, comparing SOFIA and RTL-FIM. The following

subsections detail each proposed experiment to provide a reasonable and consistency analysis.

5.1.3.1 Mismatch Analysis Considering Different Processor Architectures

The first discussion considers Arm processor architectures, their ISAs and different software stacks to provide us with a solid number of results. In this regard, Equation (5.1) is used to provide the mismatch between the reference approach (i.e., RTL) and the SOFIA. The mismatch here is defined as the difference between results obtained in RTL and SOFIA, for each application (*app*) and the same fault class (*class*) divided by the number of injected faults in the campaign.

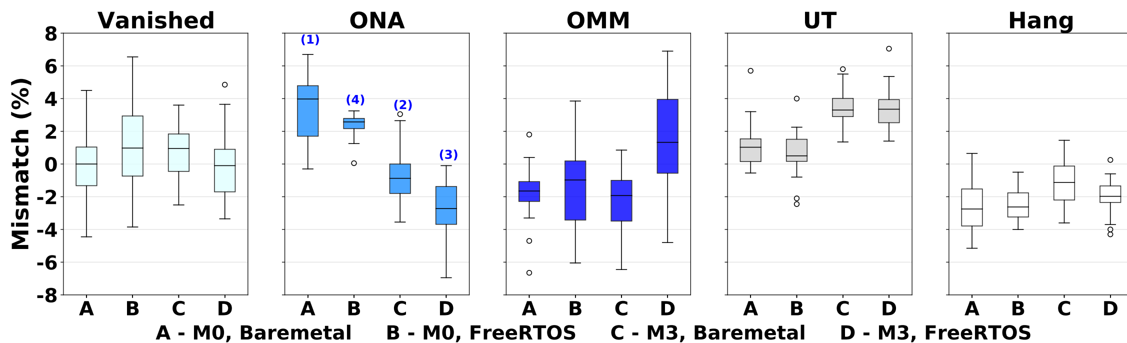
$$Mismatch = \frac{(RTL_{[\ app,\ class\ ]} - SOFIA_{[\ app,\ class\ ]})}{Number\ of\ Injected\ Faults} \tag{5.1}$$

The first results highlight the effects of the Arm Cortex-M0 and Cortex-M3 architectures and their respective ISAs on soft error resilience. Although both processors are from the Cortex-M Family, they differ in terms of computer architectures (i.e., Von Neumann and Harvard, respectively). While the Cortex-M3 has a larger ISA (i.e., entire thumb and thumb-2, 32-bit and 64-bit result multiplication, and 32-bit quotient division), the Cortex-M0 has a reduced ISA (i.e., most thumb, some thumb-2, and 32-bit result multiplication).

Different from CHO et al. (2013) and SCHIRMEIER; BREDDEMANN (September 2019), our evaluation considers an empirical data distribution that shows the minimum, maximum, and mean mismatch values with the interquartile ranges. Such data distribution enables us to show a detailed inter-benchmark mismatch variation considering all fault classifications and not only the EAFC (SCHIRMEIER; BREDDEMANN, September 2019) nor only the overall accuracy (CHO et al., 2013). Figure 5.2 shows the mismatch distribution for each fault class, considering the impact of using different processors and software stacks. This means that each bar in Figure 5.2 represents 26,000 fault injections, which means a confidence level of 99.8% and a error margin of 1%, according to the Equation (4.1). In addition, the experiments used the *GCC 4.9.3* compiler with *O0* optimization flag, which facilitates the reproducibility of the experiments by other researchers.

First, results show that the architectural difference between the two processors affects the soft error system's reliability, producing different fault class behaviors, as shown in Figure 5.2. A significant architectural difference is that the Arm Cortex-M0 has

Figure 5.2 – Boxplot of fault mismatch between SOFIA and RTL-FIM considering different architectures and software stacks. Outlier cases (represented by circles) are at least two standard deviations from the mean.



Source : The authors

a reduced instruction set, which demands the execution of a broader set of instructions to complete more complex operations than the Cortex-M3. For instance, while a 32-bit multiplication operation may vary from 1 up to 32 cycles in the Cortex-M0, the Cortex-M3 multiplier instruction needs a single cycle to complete the same operation.

From a bird's eye view, Figure 5.2 shows that results obtained from SOFIA differ more prominently, from the reference, in the occurrence of *ONA* and *UT* faults, whereas the mean is out of ±5% (i.e., a low mismatch reference). In this regard, the RTL approach is more likely to either mask an injected fault targeting a general-purpose register (i.e., *Vanished*) or propagate it to other registers (e.g., *ONA* and *UT*). In addition, the results differ due to the simulation nature of each fault injection approach. In SOFIA, faults are injected between instructions, which can restrict their propagation and, consequently, reduce the cumulative effects of soft errors. In turn, the nature of discrete event simulators allows the injection into any clock cycle, which can affect the execution of multi-cycle instructions (e.g., a division execution can take 2 to 12 cycles on the Cortex-M3). In this case, the value of a register can change a *mid* instruction, which may lead to either a masked fault; or an undefined processor state; or even wrong application execution.

On the other hand, looking at specific points, it is possible to infer that the mismatch distribution between bare-metal scenarios revealed a higher occurrence of *ONA* (i.e., mean value indicated by (1) in Figure 5.2) for the Cortex-M0 on RTL approach, and a lower mismatch with Cortex-M3 (i.e., mean value highlighted by (2) in Figure 5.2). Also, the results show that SOFIA presents lower mismatches of *Hang*, *OMM*, and *Vanish* faults for both processors, i.e., most mean values are in the ±5% range.

Results in Figure 5.2 (highlighted by (3) and (4)) show that the mismatch dis-

tribution from *ONA* faults is more disperse when using SOFIA, which affects the other fault types proportionally, i.e., the sum of the variations in the average mismatch between bare-metal and FreeRTOS is equal to zero. The collected results show that the mean values remain between ±4%, confirming that the inclusion of FreeRTOS did not affect the SOFIA soft error assessment accuracy.

5.1.3.2 Mismatch Analysis Considering Cross-compilers

Compilers play an essential role due to their direct impact on applications performance, power-efficiency, and reliability (HOSTE; EECKHOUT, 2008), as they provide software engineers with a wide variety of optimization settings (i.e., flags), which can be used to either configure debugging and warning messages or to achieve code optimization. In addition, industrial leaders employ different compilers in their projects, so assessing the impact of these compilers on soft error reliability is vital to guarantee the success of their products. On the one hand, most of the work on compilation flags in the literature focuses on performance optimization (MACHADO et al., 2017), and on the memory usage and code size reduction (SONG et al., 2014). Few are those that assess the soft error reliability provided by compilers (LINS et al., 2017; MEDEIROS et al., 2018; SERRANO-CASES et al., 2019). In this scenario, this Section investigates the impact of widely adopted compilers and their optimization flags on the soft error reliability to find the most reliable set for Arm processors.

The long simulation time inherent to RTL approach restricts its use for the soft error assessment of multiple scenarios. For that reason, the evaluation considers only the Arm Cortex-M3 to investigate whether the nature of cross-compilers and optimization flags affect the accuracy of soft error results obtained with SOFIA. In this sense, five cross-compilers are considered:

- ***GCC 4.9.3***: free-software, still widely used by legacy systems and applications;
- ***GCC 7.2.1***: free-software, currently shipped with popular Linux distributions;
- ***Clang 6.0.1***: free-software, which is an LLVM-based compiler;
- ***Arm 5.06***: proprietary-compiler developed based on the GCC compiler;
- ***Arm 6.10***: proprietary-compiler developed based on the LLVM compiler.

These compilers consider six optimization flags (i.e., *O0*, *O1*, *O2*, *O3*, *Os*, and *Ofast*), with the exception of the Arm 5.06 compiler, which has a different approach for *Os* and *Ofast*. In this case, the traditional flags *O0*, *O1*, *O2*, and *O3* are combined with

other flags (i.e., *Ospace* and *Otime*), which are equivalent to either *Os* and *Ofast*, resulting in eight flag combinations.

Table 5.2 presents a mismatch percentage summary of the compiler sets (i.e., compiler with optimization flag) between the RTL and SOFIA FI modules. Considering only bare-metal applications, each compiler set comprises 26,000 fault injection campaigns, which means that our results have a confidence level of 99.8% and a margin of error of 1%, according to Equation (4.1). The results show that compilers have an impact on soft error resilience. However, the mismatch between the two approaches is very low, with means close to 2 for all compiler sets.

Regarding the mismatch distribution, worst-case scenarios maintain the absolute mismatch below 8%. Considering that the calculated margin of error is at 1%, even the outlier values are very close between the two FI approaches, which means that SOFIA provides good reliability results with orders of magnitude faster than RTL. In turn, comparing the compilers, the results of the two GCCs show that they have the highest number of outliers in worst-cases (see red values in Table 5.2). These outliers may not yield an apparent correlation, but they present a difference in terms of executed instructions. For example, the GCC versions of the `Dhrystone` application execute 1.7× more instructions than the version generated by the Arm 5.06 compiler, which affects the application vulnerability window.

After presenting an overview of the compilers and optimization flags showing the accuracy of the results in relation to RTL, we increased fault injection campaigns to 10,000 for each compiler set and compared them using SOFIA. The goal is to find the most reliable compiler set, so the FI campaigns were increased to produce a more significant result in terms of confidence level and lower error margin. In this scenario, each compiler set has 260,000 FI campaigns, which leads to a confidence level of 98% and a 0.2% of margin of error.

Figure 5.3 shows the soft errors related to the fault injection campaigns for the five compilers and their optimization flags using SOFIA. This evaluation considers AVF to be the percentage of the sum of all faults that could be analyzed (i.e., *ONA*, *OMM*, *UT*, and *Hang*). Thus, the reliability is related to the percentage of *Vanishes* found. Among the evaluated compilers, the commercial *Arm 6.10* presents more occurrences of *Vanish*, indicating that it has a greater soft error resilience. From the open-source alternatives, *Clang* appears to be the best option due to two main reasons: (*i*) a higher number of

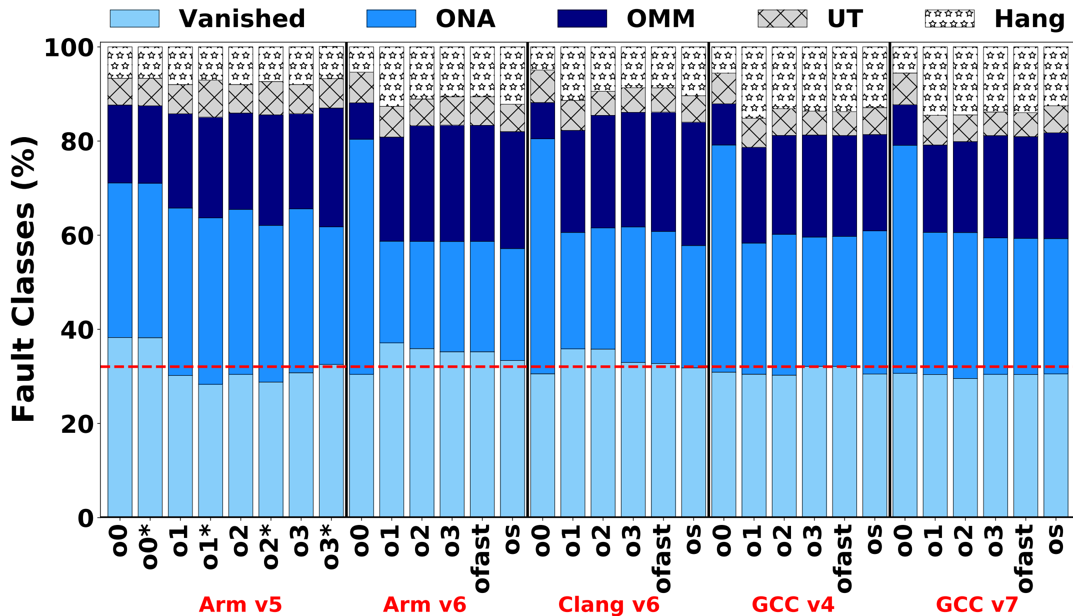
Table 5.2 – Mean and worst-case absolute mismatch percentages for different cross-compilers and their optimization flags in Arm Cortex-M3. The * means a combined flag equivalent to *Ofast*.

		Arm 5.06								Arm 6.10					
		O0	O0*	O1	O1*	O2	O2*	O3	O3*	O0	O1	O2	O3	Os	Ofast
Vanished	Mean	1.5	1.6	1.5	1.6	1.6	1.8	1.6	1.8	1.4	1.7	2.0	1.9	1.8	2.1
	Worst	4.2	**6.1**	4.7	4.2	5.6	5.1	4.9	5.9	**7.5**	4.9	5.4	5.9	5.2	5.1
ONA	Mean	1.1	1.1	1.1	1.4	1.5	1.6	1.3	1.6	1.8	2.9	1.6	1.2	1.7	1.3
	Worst	3.6	5.3	4.6	4.1	4.9	4.9	5.0	**6.0**	**6.8**	**7.1**	5.6	**6.2**	**6.0**	5.4
OMM	Mean	1.2	1.5	1.1	1.8	1.3	2.0	1.2	1.6	1.2	1.9	1.4	1.3	1.4	1.3
	Worst	4.0	4.5	3.8	5.5	3.1	5.2	3.2	5.4	4.8	5.5	**6.4**	3.5	3.3	5.1
UT	Mean	1.6	2.6	1.2	2.4	1.2	2.0	1.4	1.4	1.3	1.7	1.5	1.7	1.7	1.9
	Worst	**6.1**	5.9	5.2	**6.6**	5.7	5.6	5.2	3.1	3.4	4.3	4.2	3.9	4.5	4.5
Hang	Mean	1.3	1.6	1.4	1.6	1.4	2.2	1.7	1.9	2.0	1.5	1.7	1.5	1.7	1.6
	Worst	4.1	3.8	4.4	4.4	4.6	4.4	5.8	4.7	**6.0**	3.6	4.4	3.4	4.3	3.5

		Clang 6.0.1						GCC 4.9.3						GCC 7.2.1					
		O0	O1	O2	O3	Os	Ofast	O0	O1	O2	O3	Os	Ofast	O0	O1	O2	O3	Os	Ofast
Vanished	Mean	0.9	1.7	1.5	1.5	1.9	1.6	1.8	2.3	2.0	2.1	1.7	1.9	1.4	1.4	1.3	1.8	1.8	1.5
	Worst	2.6	4.6	4.1	3.9	5.5	3.8	4.3	**6.7**	**7.5**	**6.8**	5.6	3.9	3.2	4.4	**8.0**	**6.5**	**6.8**	5.3
ONA	Mean	1.8	2.0	1.5	1.5	1.7	1.7	1.4	2.4	1.8	1.7	1.9	2.2	1.4	1.6	1.1	1.7	1.7	1.8
	Worst	**6.2**	5.2	**6.3**	4.4	3.8	5.9	3.6	**6.3**	5.9	5.8	5.8	**6.9**	4.5	4.0	**6.8**	**6.2**	4.9	5.3
OMM	Mean	1.2	2.0	2.2	2.0	1.8	2.4	2.4	2.2	1.7	1.8	1.9	1.7	2.2	2.0	1.2	1.6	1.9	1.8
	Worst	**7.3**	**6.9**	5.1	4.9	4.7	5.0	**7.2**	**6.8**	**6.5**	**6.3**	5.0	4.0	5.6	**6.1**	4.5	**6.1**	**6.2**	**6.0**
UT	Mean	1.9	1.4	2.0	1.6	1.6	1.7	3.1	1.7	2.4	2.2	2.6	1.7	2.6	2.3	1.6	1.8	2.1	1.9
	Worst	4.5	5.3	5.5	5.5	**6.5**	5.4	**7.1**	**6.0**	**6.3**	4.5	**7.4**	**6.3**	5.6	5.0	**6.0**	3.9	4.9	3.9
Hang	Mean	1.9	1.0	0.8	0.8	1.6	0.7	1.5	1.8	1.6	1.6	1.8	2.2	1.6	1.8	1.5	1.6	1.4	1.4
	Worst	5.4	2.8	2.5	2.4	5.4	2.3	3.2	4.8	4.5	4.1	**6.9**	4.5	5.0	4.7	4.9	4.3	4.8	4.1

Source : The authors

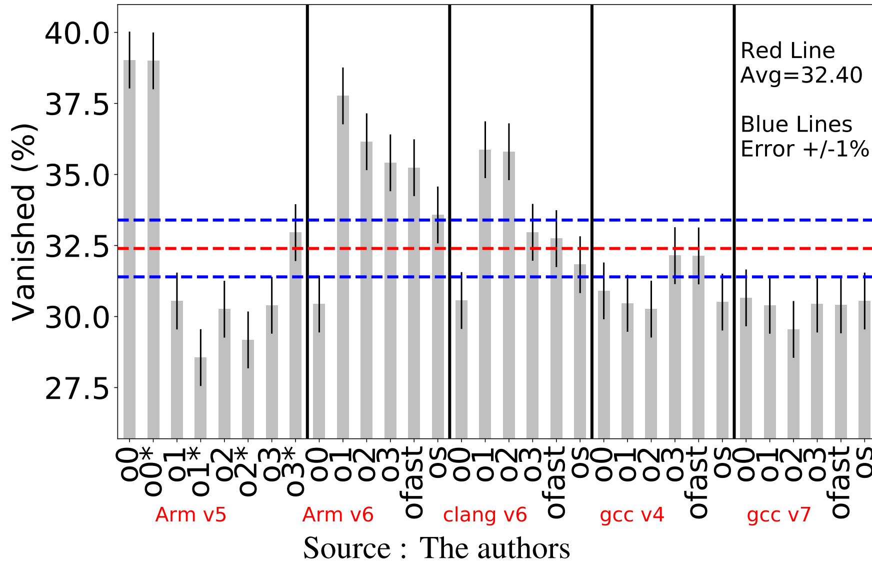
Figure 5.3 – Average faults for compilers and their optimization flags. The red line represents the average number of Vanished faults.



Source : The authors

vanishes were identified, and (*ii*) its adoption leads to a lower occurrence of *Hangs*. On the other hand, the GCC compilers followed the trend shown in Table 5.2 and continued to present the worst results in terms of soft error resilience, where no optimization flag passed

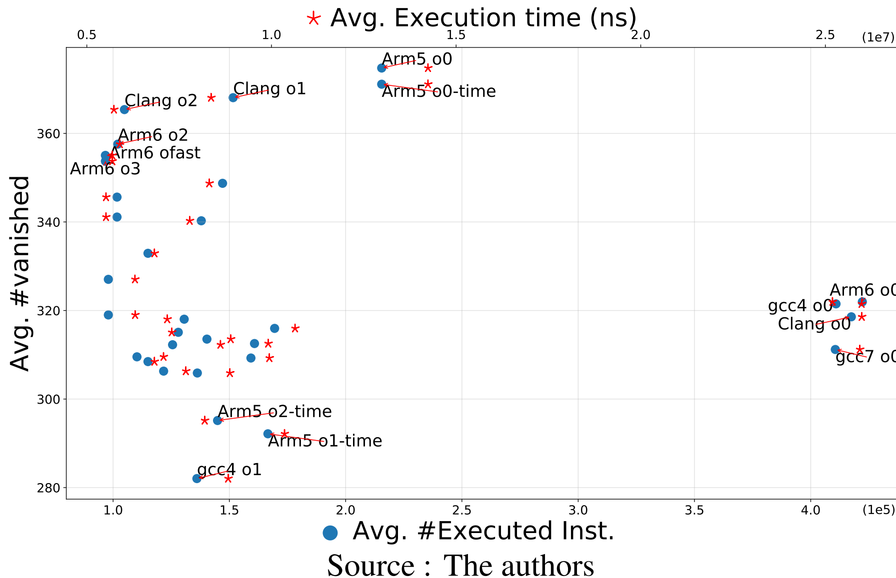
Figure 5.4 – Average vanished faults for compilers and their optimization flags.



Source : The authors

the dashed red line shown in Figure 5.3. To facilitate this view, Figure 5.4 is a zoom-in considering only the occurrence of *Vanishes*. Although the newer GCC version uses more specific Arm Cortex-M3 ISA instructions, the resulting improvement is negligible compared to the previous version. In short, compilers with better results regarding *Vanish* are LLVM-based. Although restricted to only two ISAs, these findings suggest a pattern for adopting LLVM-based compilers, but additional experiments considering other ISAs would be necessary to further explore the reliability benefits of this compiler.

Figure 5.5 – Average execution time and executed instructions *vs* number of Vanishes considering different compiler sets.



Source : The authors

Lastly, we believe that the industry will not adopt a compiler set just because it is more reliable, except for a niche such as critical-safety applications (e.g., autonomous vehicle applications). In this sense, we propose to evaluate which compiler set is more reliable and also offers the best performance. Figure 5.5 shows a trade-off between performance (i.e., execution time from RTL and executed instructions from SOFIA) and reliability (i.e., the occurrence of *Vanishes*), considering the aforementioned cross-compilers. The upper-left corner presents the best of both performance and reliability; conversely, the lower-right corner has the worst performance and reliability. In turn, the lower-left corner assemblies are the configurations that show reasonable performance but low soft error reliability.

Results show that the applications compiled with the *O0* flag presented a higher susceptibility to soft errors and a low performance, except for the codes generated with the *Arm 5.06* compiler, which showed a reliability improvement but still below the other sets of compilers. At the other end are the *Arm 6.10* and *Clang 6.0.1* compilers (i.e., both LLVM-based), the two have the best set of performance and reliability. Among them, the *O2* optimization flag stands out, presenting superior results for the two compilers. However, due to the significance of our results (high confidence level and low margin of error), *Clang* has a certain advantage, and it would be our compiler suggestion. Finally, the remaining compiler sets maintain similar results to the mean of this trade-off.

5.1.4 Closing Remarks

This Chapter demonstrated that the use of virtual platforms brings orders of magnitude speedups to FI campaigns. This also showed that the architectural difference between the two Arm processors affects the system's reliability, producing different fault class behaviors. However, the inclusion of an operating system did not affect the soft error assessment accuracy. Next, after a solid and extensive soft error reliability assessment considering a variety of off-the-shelf compilers, it is possible to conclude that the best compilers are LLVM-based and that the best set in terms of performance and reliability would be the *Clang 6.0.1* compiler using the *02* optimization flag.

6 SOFT ERROR RELIABILITY ASSESSMENT OF ML INFERENCE MODELS EXECUTING ON RESOURCE-CONSTRAINED IOT EDGE DEVICES

This Chapter provides in-depth insights and results related to the second main contribution of this Thesis: the early soft error assessment and mitigation of ML inference models executing on resource-constrained IoT edge devices. The underlying contribution has been published in several international conferences (ABICH et al., 2020; ABICH; REIS; OST, 2020; ABICH et al., 2022) and high quality journals (ABICH et al., 2021; ABICH et al., 2022). In this sense, different soft error reliability assessments of ML models build from industrial libraries and APIs targeting resource-constrained IoT edge systems are detailed discussed. The discussion is based on reasonable number of results obtained from a vast number of FI campaigns, taking into account different reliability aspects of a NN model, such as topology, layers, target ISA optimizations, memory parameters, precision bitwitdh, mitigation techniques, and parallelization.

Table 6.1 – Contributions organized by Case study.

Target Evaluation	ML Model CIFAR-10 CNN	 MobileNet CNN
Topology	✓	✓
Layers	✓	✓
ISA	✓	
Memory	✓	✓
Precision Bitwidth		✓
Soft Error Mitigation		✓
Parallelism	✓	

Source : from Authors

Table 6.1 details the target evaluation according to each executed CNN model. Section 6.1 presents the early soft error assessment considering the CIFAR-10 CNN execution in resource-constrained IoT edge devices. Furthermore, Section 6.2 comprise the soft error reliability results considering the MobileNet CNN with different bitwidth configurations.

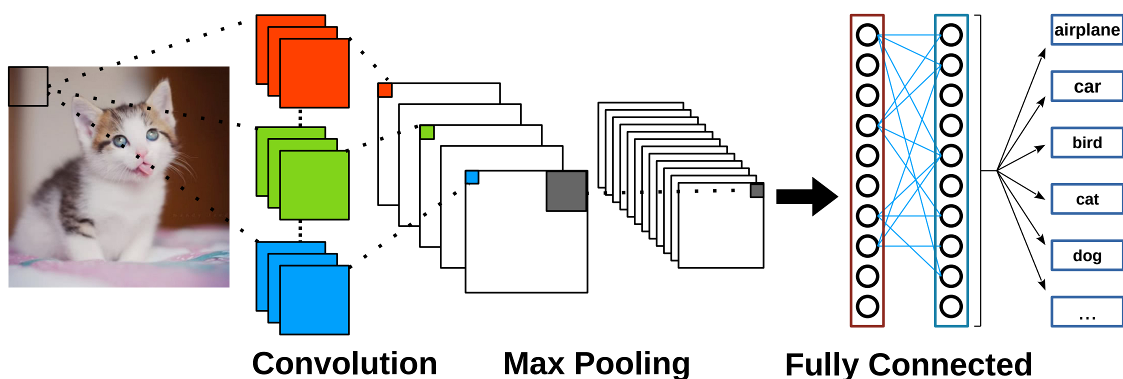
6.1 Soft Error Reliability Assessment of the CIFAR-10 CNN

This section presents the soft error reliability assessment results for the CIFAR-10 CNN case study. First, Section 6.1.1 presents a detailed description of the CIFAR-10 CNN developed with the CMSIS-NN library. Next, Section 6.1.2, Section 6.1.3, and Section 6.1.4 comprise the soft error reliability assessment considering the CIFAR-10 CNN execution, topology, isolated layers, target ISA optimizations, memory (code, parameters and data), and thread parallelism.

6.1.1 CIFAR-10 CNN Developed with CMSIS-NN

This Section presents the CIFAR-10 case-study, which consists of a 7-layer CNN developed with the CMSIS-NN kernels (LAI; SUDA; CHANDRA, 2018) to run in Common Microcontroller Software Interface Standard (CMSIS) supported processors (e.g., Arm Cortex-M).

Figure 6.1 – Brief illustration of the adopted CIFAR-10 CNN topology.



Source : Adapted from LAI; SUDA; CHANDRA (2018)

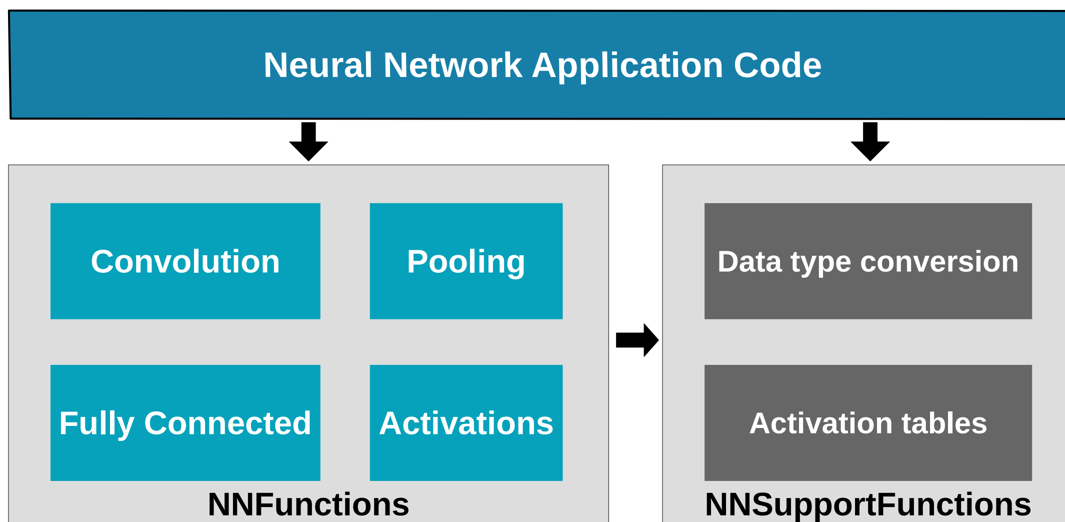
The CNN is trained with the CIFAR-10 dataset (KRIZHEVSKY; HINTON et al., 2009), consisting of 60,000 32x32 color images divided into ten output classes. Figure 6.1 shows an overview of a typical CNN topology based on the built-in example provided in CMSIS-NN (LAI; SUDA; CHANDRA, 2018). Table 6.2 details the CNN layers with the kernel shapes (i.e., filter and output) used in each layer. Such CNN consists of multiple layers composed of convolution layers, interspersed by non-linear activation layers, pooling layers, and a fully-connected layer at the end of CNN.

Table 6.2 – Layer parameters for the CIFAR-10 CNN.

	Layer Type	Filter Shape	Output Shape
1	Convolution	5x5x3x32	32x32x32
2	Max Pooling	3x3	16x16x32
3	Convolution	5x5x32x32	16x16x32
4	Max Pooling	3x3	8x8x32
5	Convolution	5x5x32x64	8x8x64
6	Max Pooling	3x3	4x4x64
7	Fully-connected	4x4x64x10	1x10

Source : Adapted from LAI; SUDA; CHANDRA (2018)

Figure 6.2 – Overview of the CMSIS-NN kernel structure.



Source : Adapted from LAI; SUDA; CHANDRA (2018)

The overview of CMSIS-NN kernel is shown in Figure 6.2. The kernel code consists of two parts: *NNFunctions* and *NNSupportFunctions*. As described in (LAI; SUDA; CHANDRA, 2018), the NNFunctions includes the functions that implement popular neural network layer types while the NNSupportFunctions includes the utility functions.

Different from the traditional NN models trained using 32-bit floating-point data representation, the CMSIS-NN uses low-precision fixed-point representation as high precision is generally not required during the inference. The use of such quantization avoids the need for floating-point de-quantization between layers, as some Arm Cortex-M processors do not have a dedicated Floating-Point Unit (FPU). The CMSIS-NN kernels deploy optimizations that aim to boost performance while reducing the memory-footprint when executing complex NN inferences in resource-constrained processors. Such optimizations focus on enabling neural networks on Cortex-M based systems that support SIMD instructions, especially 16-bit Multiply-And-Accumulate (MAC) instructions, such as Signed

Multiply Accumulate Long Dual (SMLAD), which are very useful for NN computation. In this sense, the CMSIS-NN kernel implements matrix multiplications with a 2x2 dimension to enable some data reuse and reduce the number of load instructions. Further, it performs accumulation with dedicated SIMD MAC instruction, and data transformation occurs without reordering, achieving reasonable performance (LAI; SUDA; CHANDRA, 2018).

To achieve such a performance, the convolution layer implements a partial image-to-column data transformation considering the Height-Width-Channel (HWC) format. Rectified Linear activation Unit (ReLU) layer implements a loop over all elements and makes them 0 if they are negative using a similar concept as SIMD Within a Register (SWAR). After the activation layer, the Max Pooling layer reduces the feature dimensions, the number of parameters, and computations in the network using split x-y pooling, contributing to performance improvements while reducing the need for additional memory. The fully-connected layer has a matrix-vector multiplication that can also be implemented with a 1×2 kernel size to improve even more the execution time. This layer has full connections to all activations in the previous layer to define the output classification probabilities.

6.1.2 Soft Error Reliability Assessment of CIFAR-10 CNN Execution on Resource-constrained IoT Devices

This Section aims to explore the erroneous behavior of the adopted CNN when exposing its layers to faults (i.e., flipped bits). Section 6.1.2.1 presents the early results when considering the execution of the CIFAR-10 CNN in two Arm Cortex-M processors: Cortex-M3 and Cortex-M4.

6.1.2.1 Experimental Setup

In order to estimate the percentage of errors from the CNN case study, the results are obtained by injecting bit-flips in the general-purpose registers (i.e., r0-r15) of both Cortex-M3 and Cortex-M4 Arm processors considering *random register file* and *function lifespan* FI techniques. Table 6.3 shows the experimental setup used to perform the proposed evaluation.

Each fault injection considers a single input image for one CNN execution, thus

Table 6.3 – Experimental Setup

ML Model	Cifar-10 CNN
API/Libraries	CMSIS-NN
Dataset	Cifar-10
Arm Processors (executed instructions)	Cortex-M3 (47 millions) Cortex-M4 (15 millions)
Target FI	Register File, Function Lifespan
CNN Topology	3 Convolution (3 ReLU Activations), 3 Max Pooling, 1 Fully-Connected
Number of FI Campaigns	22
Injections per Campaign	17k
Total Fault Injections	374k

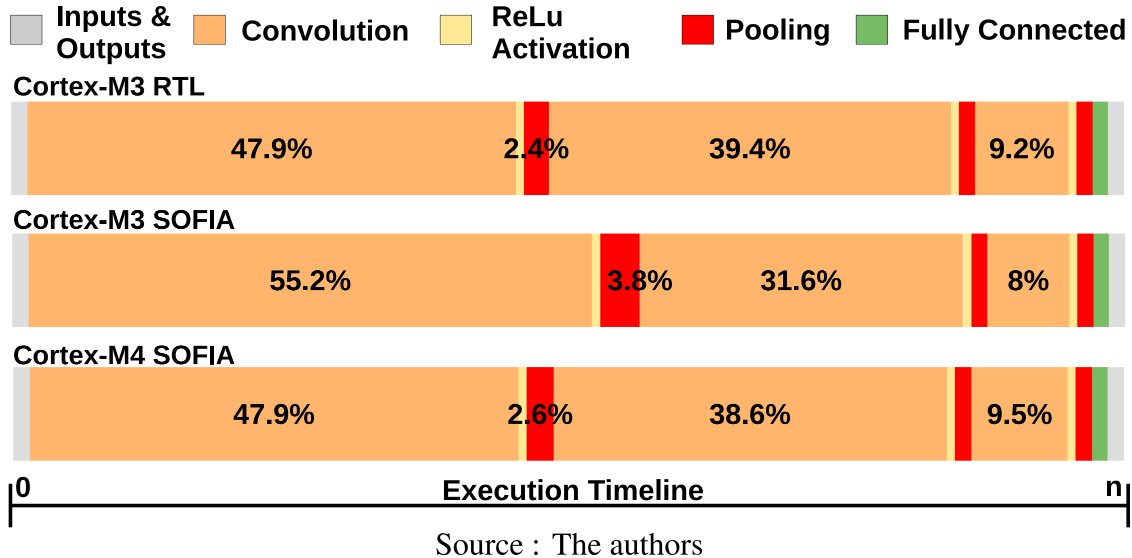
Source : The authors

not considering fault propagation to the subsequent executions. One of the main concerns when assessing the soft error reliability of a system is to develop a precise, well-covered, and realistic approach. In this sense, this work sought to ensure that the number of fault injections has a statistical significance by applying the equations developed by (LEVEUGLE et al., 2009). These results comprise 17k fault injections per campaign, thus generating a 1% error margin with a 99% confidence level. Although both processors share the same architecture (ARMv7-M) and instruction set (Thumb-2), the Cortex-M4 has an additional Digital Signal Processor (DSP) extension with a range of saturating and SIMD instructions. For that reason, Cortex-M3 requires 3× as many instructions to execute the same CNN (Table 6.3). In addition, these results consider the same compilation environment: GCC 9.3 and optimization flag -O2.

6.1.2.2 CIFAR-10 CNN Execution Lifetime

Asserting the experimental setup accuracy is paramount to obtain meaningful and useful results. In this regard, the CIFAR-10 CNN was executed in the absence of faults over a RTL description of the Arm Cortex-M3, comparing gathered outputs with those obtained with the SOFIA execution. Results show that the CIFAR-10 CNN execution in SOFIA presents no difference in the output probabilities for the selected image w.r.t. the RTL execution (4.5 hours), thus validating the utility of the adopted framework, which needs only 1 second to execute the same inference CNN model. Figure 6.3 shows the CNN execution detailing the lifespan of each layer, considering both the Arm Cortex-M3 and the Cortex-M4.

Figure 6.3 – CNN timeline with CMSIS-NN Kernel executing on Cortex-M3 and Cortex-M4 processors.



Source : The authors

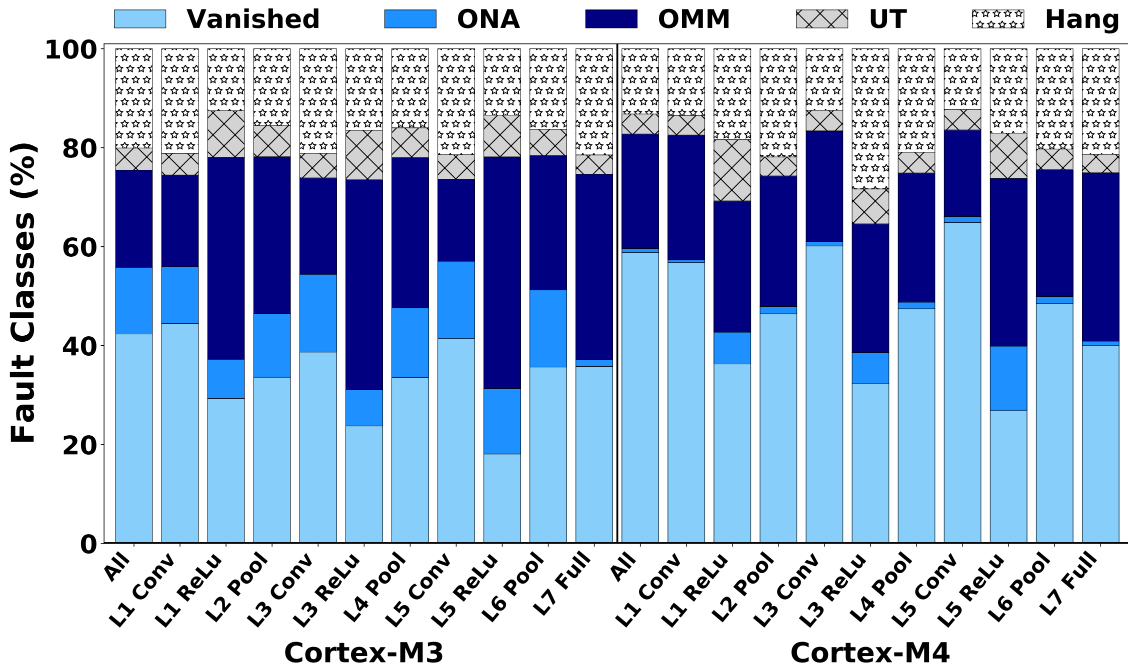
Observing the functions' lifespan in Figure 6.3, it is notable that convolution layers execute at most of the time while the remaining layers execute in a small part of the CNN execution. In this sense, it is expected that faults injected randomly are more likely to reach the convolution layers, having a considerable impact on the CNN execution. The execution percentages for the RTL model are based on clock cycles, while in SOFIA, the percentages are based on the number of executed instructions. Such percents differ due to the number of required clock cycles to execute some instructions (e.g., division requires 2 to 12 cycles in Cortex-M3 RTL description). Note that the RTL description of the Cortex-M4 is not available at the Arm University program. These findings assist in the decision to properly evaluate the fault impact on the CNN layers considering both processor architectures.

6.1.2.3 CIFAR-10 CNN Soft Error Reliability Assessment

Figure 6.4 shows the results for the FI campaigns targeting all CNN layers. Such results are compared with those obtained from fault injection campaigns that target individual CNN layers and activation functions.

The CNN functions use several loops to process data, which requires many control instructions, consequently generating high UT and Hang occurrence (i.e., 22% on average) in both architectures. Even with a reduced number of crashes (UT and Hangs), ReLU and pooling layers show high soft error vulnerability in the Cortex-M3 when compared to Cortex-M4 (i.e., high occurrence of OMM and ONA). The results also show a

Figure 6.4 – Fault classification when injecting faults in the CNN Layers.



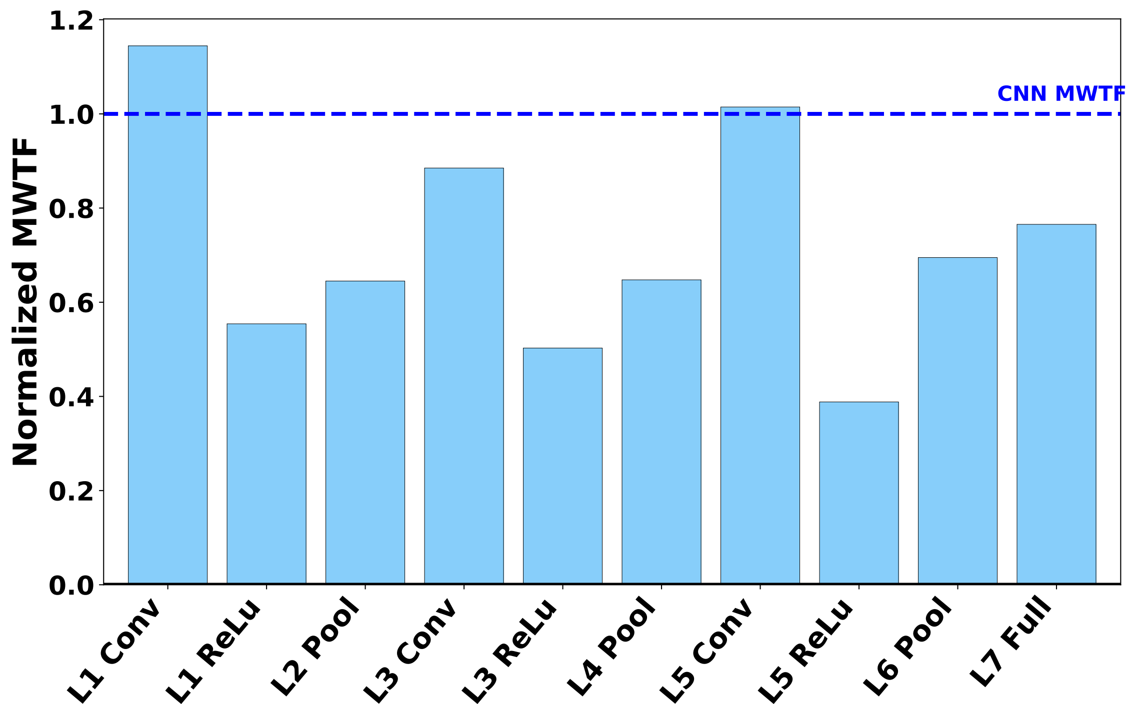
Source : The authors

higher masking effect on Cortex-M4 due to the CMSIS-NN optimizations targeting SIMD instructions (i.e., ∼20% more vanished when comparing to Cortex-M3), reflecting such masking factor when injecting faults on the CNN Convolution layers. In turn, the injections in the ReLU activation layers produced similar results in both processors. The ReLU layers perform a reduced data set (i.e., less execution time), thus reducing the masking rate even without SIMD optimizations.

The MWTF (REIS; CHANG et al., 2005) metric is used to evaluate the CNN soft error reliability, since it shows the average amount of work an application can perform until reaching a failure (i.e., higher values are better). Figure 6.5 shows the MWTF of each layer normalized with the CNN MWTF for fault injections on the Cortex-M3. The results show a reasonable MWTF reduction regarding the occurrence of soft errors in the different layers. While soft errors from pooling layers reduce the MWTF in 35% on average, the faults from ReLU activation layers achieve an MWTF reduction of 62%. In this manner, the activation layers are critically affected in both architectures once such functions present high fault occurrence. Our findings denote that the CNN reliability is related to the layer types and its topology since the soft error occurrence increases from input to output layers.

Table 6.4 compares the output data from the fault injection campaigns along with the fault-free execution. The effective faults are very similar in both processors once

Figure 6.5 – Normalized MWTF for fault injections in different layers of CNN in Cortex-M3.



Source : The authors

Table 6.4 – Percentages of tolerable and critical faults in the CNN.

		Cortex-M3				Cortex-M4			
	Layers	Correct	Tolerable	Critical	Crash	Correct	Tolerable	Critical	Crash
1	Convlution	56.0	10.3	8.2	25.5	57.3	13.0	12.2	17.4
	ReLU	37.2	**3.6**	**37.3**	21.9	42.7	**5.4**	**21.2**	30.8
2	Max Pooling	46.5	21.7	10.1	21.8	47.9	17.8	8.6	25.7
3	Convlution	54.4	12.8	6.7	26.1	61.1	10.1	12.3	16.6
	ReLU	31.1	**2.7**	**39.8**	26.4	38.6	**5.0**	**21.1**	35.4
4	Max Pooling	47.6	21.0	9.4	22.0	48.8	17.6	8.5	25.1
5	Convlution	57.1	**7.8**	8.8	26.4	66.1	**7.5**	10.0	16.4
	ReLU	31.3	**5.0**	**41.8**	21.9	39.9	**6.5**	**27.4**	26.2
6	Max Pooling	51.2	18.8	8.4	21.6	49.9	18.7	7.0	24.4
7	Fully-Connected	37.2	15.4	**22.1**	25.4	40.9	11.4	**22.6**	25.1
all	All	55.8	11.8	7.8	24.5	59.6	11.1	12.0	17.2

Source : The authors

the ONA fault types do not affect the output probabilities, thus not affecting the CNN accuracy. The effective and the critical faults achieve higher percentages while tolerable faults have lower ones. Note that highlighted results represent the worst-case scenarios. While pooling and convolution layers have high tolerable faults, the ReLU and the fully-connected functions are critically affected by the injected faults in both architectures. Even with tolerable faults, critical faults reach from 13% up to 60,85% of the output data, which reveals the vulnerability on the accuracy of the adopted CNN. Such results demonstrate that the occurrence of soft errors has a significant impact on the CNN execution and its

accuracy when considering the individual analysis of distinct layers and processors.

6.1.3 The Impact of Soft Errors in Memory Units of Edge Devices Executing CIFAR-10 CNN

This Section explores the CIFAR-10 CNN's behaviour in the presence of soft errors by applying fault injections to the register file and isolated memory sections.

6.1.3.1 Experimental Setup

Table 6.5 shows the experimental setup, where 102k bit-flip are injected in the general-purpose register file (i.e., r0-r15) and Flash and RAM memory sections. This section applies the equations developed by Leveugle *et al.* (LEVEUGLE et al., 2009) to ensure the statistical relevance of the fault injection results (KRZYWINSKI; ALTMAN, 2013). In this sense, each FI campaign includes 17k experiments, providing results with an 1% error margin and 99% confidence level. Note that these experiments do not consider the propagation of faults to subsequent executions, as each experiment consists of one bit-flip injection and a single CNN execution with a single input image.

Table 6.5 – Experimental Setup

ML Model	Cifar-10 CNN
API/Libraries	CMSIS-NN
Dataset	Cifar-10
Arm Processor (executed instructions)	Cortex-M3 (47 millions) Cortex-M4 (15 millions)
Target FI	Register Files, Flash (code and parameters), RAM
Number of FI campaigns	6
Injections per campaign	17k
Total Fault Injections	102k

Source : The authors

The FI campaigns are conducted considering both Arm Cortex-M3 and M4 processors. Although both Arm Cortex-M3 and M4 rely on the same instruction set architecture (ARMv7-M), results in Table 6.5 show that the Cortex-M3 requires 3× as many instructions w.r.t. to the M4. The reason for that is the DSP capabilities of Arm Cortex-M4, which includes saturating arithmetic and SIMD instructions. Table 6.6 shows that such ISA specific optimizations reflects on the memory occupation of the target evaluation boards.

Table 6.6 – On-chip Memory Occupation (MO) for the inference CNN models, considering code, parameters and data

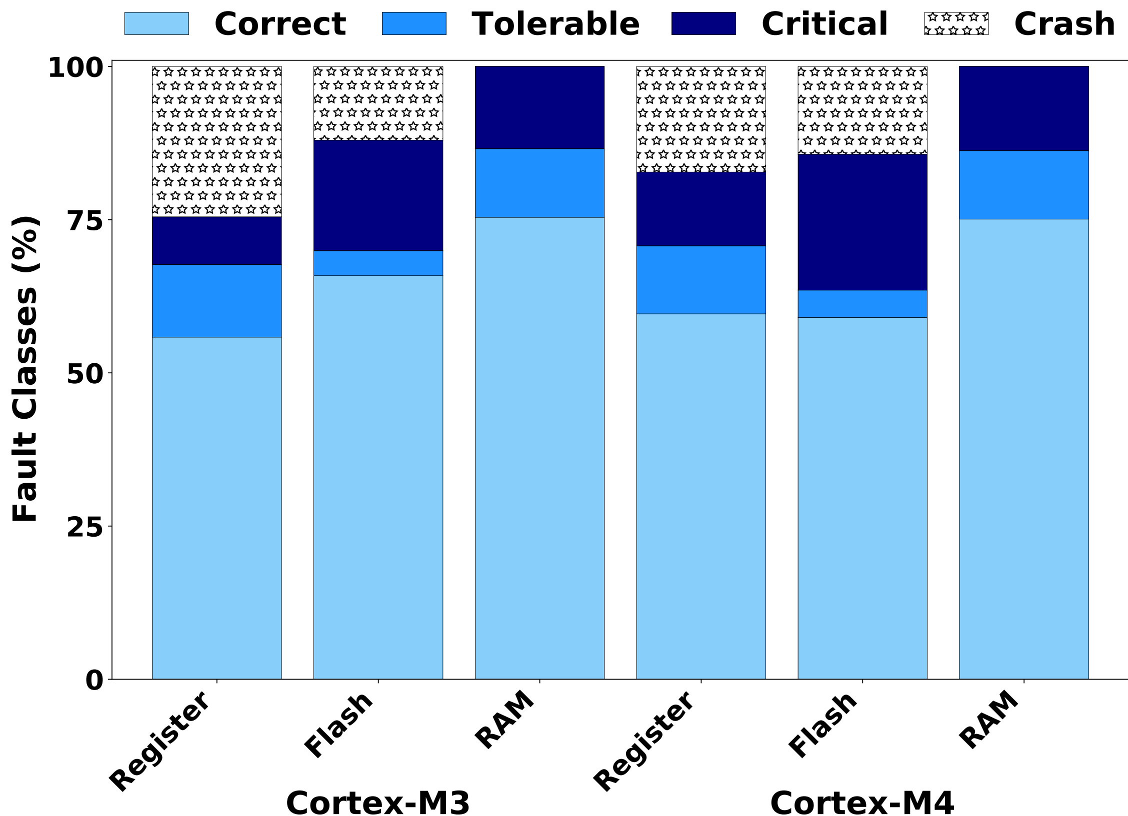
Arm Processor	Reference Chip	On-chip Memory Flash$_{(kB)}$	SRAM$_{(kB)}$	CNN Inference Model Code$_{(kB)}$	Parameters$_{(kB)}$	Data$_{(kB)}$	MO$_{(\%)}$
Cortrex-M3	STM32F103RCT6	256	48	4.3	36.5	43.2	27.63
Cortrex-M4	STM32F407ZGT6	1024	192	7.6	36.5	43.2	7.15

Source : The authors

6.1.3.2 Soft Error Reliability Analysis

Figure 6.6 shows the CIFAR-10 CNN soft error results gathered from the injection of bit-flips in the register file, Flash and RAM memory sections considering the Arm Cortex-M3 and M4. The results show that the RAM section has no *crash* occurrence. This is because this type of fault does not affect the application's object code and NN parameters, but rather the CNN application data. In turn, the Flash section presents less *crash* occurrence than the register file. In regard to the memory sections, Flash and RAM show many *critical faults* because these faults are not overwritten at runtime (i.e., up to 2.3× more *critical faults* than the register file).

Figure 6.6 – Results showing fault classifications for Register File, Flash, and RAM memory sections for the CIFAR-10 CNN considering two Arm processors.



Source : The authors

To understand the nature of the memory section faults, the CNN code, parameters and data are evaluated separately considering both Flash and RAM sections. Table 6.7 details the fault impact of the Flash code section for both Arm Cortex-M3 and M4 processors. Note that processor architecture influences the resulting CNN object code and therefore its vulnerability to bit-flips. The main difference relies on the use of SIMD instructions, i.e., the split of the convolution layers into convolution and matrix multiplication and the use of standard *memset* and *memmove* functions to fill out buffers in the Cortex-M4. These features directly impact on the occurrence of soft errors, since the variation in memory footprint and execution lifespan alter the vulnerability window (i.e., probability of a fault strike).

Table 6.7 – Percentages of memory occupation (MO), execution lifespan (ELS), and classification of faults occurring on CIFAR-10 CNN functions object code stored in Flash memory section

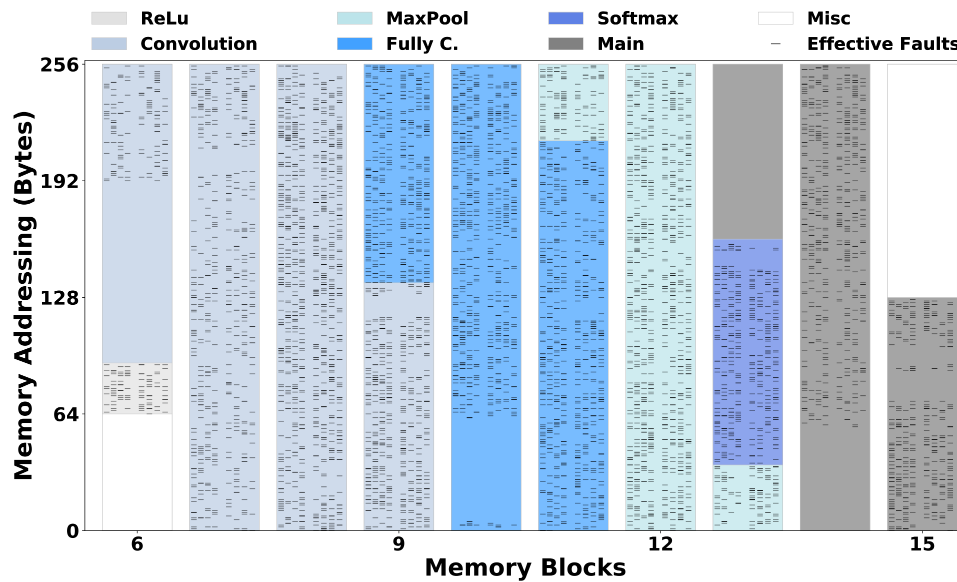
Sections	**MO**		**ELS**		**Correct**		**Tolerable**		**Critical**		**Crash**	
	M3	M4	M3	M4	M3	M4	M3	M4	M3	M4	M3	M4
Convolution	34.3	39.6	94.8	9.7	35.1	39.3	5.4	4.8	35.0	37.3	25.5	18.6
Matrix M.	-	21.2	-	85.6	-	52.0	-	5.4	-	25.1	-	17.5
ReLu	1.2	1.4	0.4	0.6	13.2	41.7	4.7	3.0	43.4	32.7	38.7	22.6
MaxPool	14.1	8.9	4.6	3.0	10.3	17.4	4.4	12.2	50.3	35.5	35.0	34.9
Fully C.	24.9	10.8	<0.1	0.1	24.1	28.2	14.7	9.3	33.6	37.2	27.5	25.3
Softmax	5.2	2.1	<0.1	<0.1	16.8	12.2	21.9	19.7	41.7	48.4	19.6	19.7
Main	20.3	9.7	0.2	0.2	51.5	52.0	4.8	4.3	29.1	28.0	14.6	15.7
MemFunc	-	6.3	-	0.8	-	58.3	-	1.9	-	15.6	-	24.2

Source : The authors

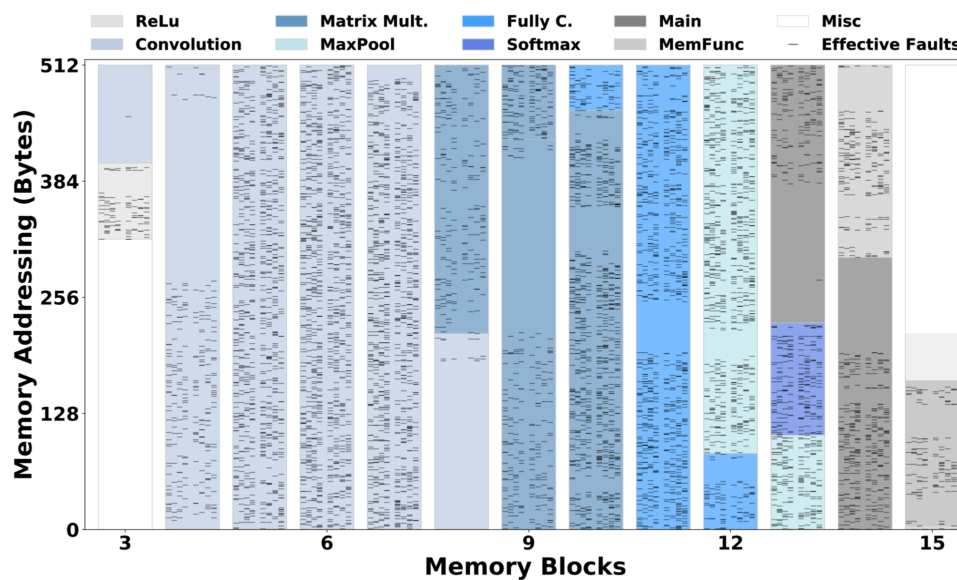
Table 6.7 also shows the fault percentages for each function object code section. Percentages are normalised according to the number of faults that affect each memory section, disregarding the faults that fall into portions of the code that do not influence the CNN execution (i.e., initialization and architecture-specific functions). These results show that the Arm Cortex-M4 is more reliable than Arm Cortex-M3. On the Arm Cortex-M3 side, activation and pooling are the most affected functions. In turn, pooling, fully-connected, and softmax present more soft errors when executed on the Arm Cortex-M4, because they have less SIMD instructions.

To understand the nature of the errors shown in Figure 6.6, Figure 6.7 presents the fault impact of the Flash memory addressing (i.e., CNN object code) for the two Arm processor architectures. The main difference between the object codes in these architectures is the use of SIMD instructions and the split of the convolution layers into convolution and matrix multiplication in Arm Cortex-M4. Considering these differences, the faults that occur in Arm Cortex-M3 have unaffected sections that comprise saturation instructions.

Figure 6.7 – Fault effects on Flash memory sections considering the CNN object code.



(a) Cortex-M3



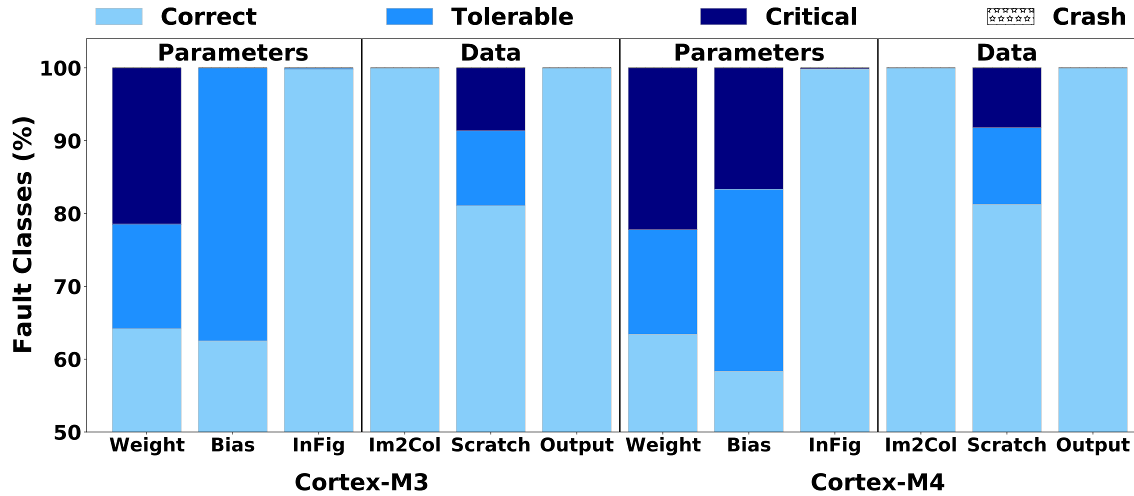
(b) Cortex-M4

Source : The authors

In the Arm Cortex-M4, the non-affected sections also include these instructions along with code portions with SIMD instructions. In addition, the first portion of the convolution code object is flawless due to the understanding of the first layer and, consequently, it has a smaller execution window than the other CNN functions.

To analyse the soft error impact between Flash and RAM sections, Figure 6.8 details the faults on the CNN parameters and data for the two Arm processor architectures. The most affected CNN parameters (i.e., Flash) are weights and bias, while the most affected data (i.e., RAM) is the scratch buffer. Note that the input figure (*InFig*) is not

Figure 6.8 – Fault classifications for CIFAR-10 CNN parameters and data stored in Flash and RAM memory sections for Arm Cortex-M3 and M4 processors.



Source : The authors

affected since the faults occur during the execution of the CNN. Image-to-column (*Im2Col*) is an auxiliary buffer used for matrix to column conversion, thus presenting either, high masking effect and low probability to propagate faults to the outputs. Weights and bias are key parameters used in CNN inferences, and therefore faults that occur in these areas tend to slightly impact on the output classification probabilities. In turn, the scratch buffer stores the inferences' results between layers, which present more faults. This is because the buffer is used as input and output of the convolution layers in a streaming fashion, consequently affecting the output probabilities.

In regard to the faults of the two processor architectures, the most significant difference was bias, which comprises essential data used in inferences. In this case, the number of instructions executed on Arm Cortex-M4 is 3× less than on Arm Cortex-M3, opening a vulnerability gap where the fault can occur before or during the inference, which might affect the output classification probabilities.

6.1.4 Impact of Thread Parallelism on the Soft Error Reliability of CIFAR-10 CNN

This Section proposes a multi-threaded version of the convolution neural network based on the CMSIS-NN kernel (LAI; SUDA; CHANDRA, 2018) applying to the Arm Cortex-A processor (i.e., ARMv7-A).

The first experiment analyses the CNN profile to understand the CNN application

Table 6.8 – CNN runtime percentages according to layer type

Layer Type	Thumb Runtime (%)	SIMD Runtime (%)
Convolution	**97.15**	**97.28**
MaxPooling	2.59	2.22
ReLU	0.23	0.40
Fully-Connected	0.02	0.10

Source : The authors

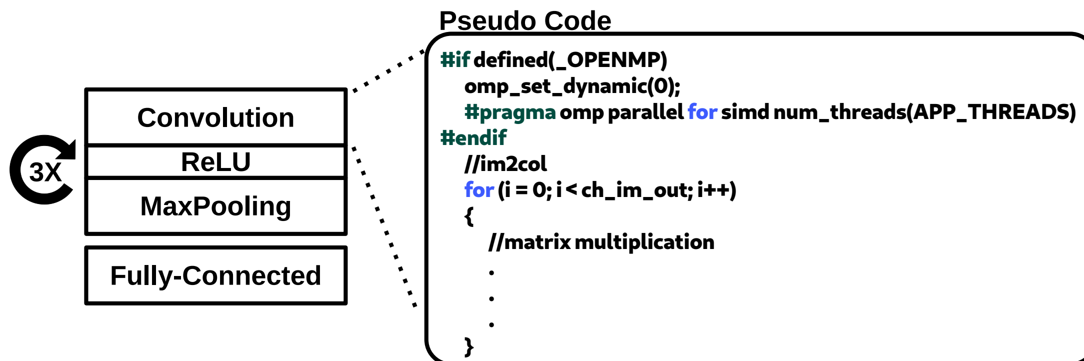
structure and which functions are most used. Table 6.8 shows the execution time percentages for a single thread running on the Arm Cortex-A7 processor according to each CNN layer kernel. These values demonstrate that the convolution layers execute almost all the time (i.e., 97%) in both the Thumb and SIMD instruction sets. Therefore, these layers are the best candidates to be parallelized to achieve such a performance improvement. In this regard, the following Section describes changes made to the CMSIS-NN kernel to allow parallel execution of convolution layers.

6.1.4.1 Parallel Convolution Kernels

To improve the performance of computational and memory limited edge devices, current approaches have been applying fixed-point arithmetic and quantization of weights and activations on 8-bit (LAI; SUDA; CHANDRA, 2018) or smaller data types (CAPOTONDI et al., 2020). On the other hand, this work extends the CMSIS-NN convolution kernel to support multiple threads considering the SIMD and Thumb instruction sets when executed on an Arm Cortex-A7 processor. The proposed extension relies on the OpenMP library (DAGUM; MENON, 1998) to support threads according to the number of cores available in the adopted processor. The main feature of this extension is that it does not change the goals of the CMSIS-NN kernel, i.e., it keeps the performance and memory optimizations with a low memory footprint overhead ($< 1\%$). Figure 6.9 demonstrates an example of pseudo-code where the proposed extension instantiates the threads in the convolution kernel.

The convolution kernel usually comprises two operations. First, the input reordering and expansion. Then, matrix multiplication operations are applied. In this regard, threads are defined between the two operations not to change the CMSIS-NN kernel's characteristics. Furthermore, the *APP_THREADS* environment variable specifies the maximum number of threads to run in parallel, and the *omp_set_dynamic* method guaran-

Figure 6.9 – CMSIS-NN extension to provide a parallel convolution kernel.



Source : The authors

tees that the number of threads will be the same as defined by the user in the *num_threads* method, avoiding bottlenecks, as shown in Figure 6.9. In addition, the *SIMD* and *ordered* directives indicate that the loop can be turned into a SIMD loop (i.e., multiple iterations of the loop can be executed concurrently using SIMD instructions), and those operations must be ordered to ensure they do not affect the inference precision.

6.1.4.2 Multi-thread Speedup Analysis Experimental Setup

The first experiment aims to reveal the speedup gains when applying thread parallelism to the CMSIS-NN convolution kernel. Table 6.9 shows the experimental setup used to evaluate the relative speedup considering the Thumb and SIMD instruction sets. To ensure results consistency, we consider the GCC compiler version 9.3 along with optimization flag $-O2$ and OpenMP version 4.5 for all experiments. The optimizations from the CMSIS-NN library are disable in Thumb versions by the architecture specific flag $-D__ARM_FEATURE_DSP = 0$, ensuring that the object code is generated entirely by the compiler. To provide accurate results regarding application runtime, we validate the proposed CMSIS-NN extension on a Raspberry Pi 2 (RPi2) Model b board (PI, 2021). In this sense, the number of threads is defined ranging from 1 to 4, since the RPi2 board is composed of a quad-core Arm Cortex-A7 processor.

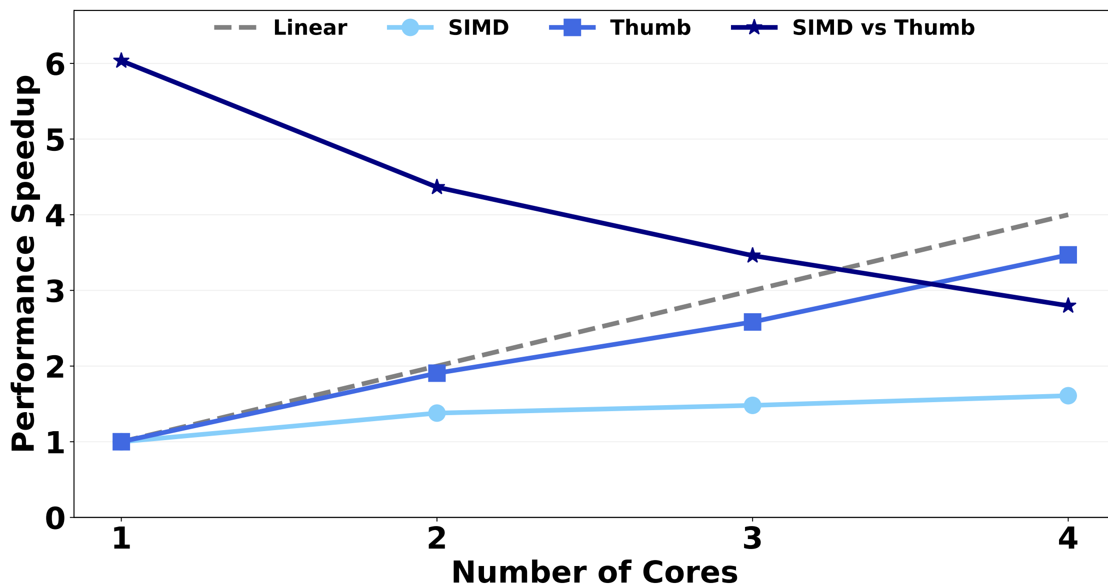
Figure 6.10 shows the speedup improvement applying multi-threaded parallelism in the adopted CNN application. Such results are calculated from the execution time obtained from the multi-thread execution over the single thread version. The SIMD-based CNN application presents a low speedup improvement in multi-threaded versions. On the other hand, the Thumb-based CNN application shows an almost linear speedup improvement (i.e., up to 3.5× improvement w.r.t. single-core version). This performance difference

Table 6.9 – Validation Experimental Setup

Evaluation Board	Raspberry Pi 2 Model b
Processor	Arm Cortex-A7
ML Model	Cifar-10 CNN
API/Libraries	CMSIS-NN
Dataset	CIFAR-10
Compiler and optimization flag	GCC 9.3 with $-O2$
OpenMP version	4.5
Instruction Sets	Thumb and SIMD
Parallel Threads	1, 2, 3, and 4

Source : The authors

Figure 6.10 – Speedup results for 1, 2, 3 and 4 thread implementations for Thumb and SIMD instruction sets.



Source : The authors

is because the Thumb implementation (i.e., generic convolution) has no optimization, whereas the SIMD version promotes architecture-specific CMSIS-NN optimizations. In this sense, the compiler can optimize the object code of the Thumb instruction set more than SIMD. However, the SIMD vs Thumb curve shows that SIMD still has a performance 2.8× higher than Thumb even with the multi-core speedup.

6.1.4.3 Soft Error Reliability Analysis Experimental Setup

This Section explores the system soft error reliability by injecting faults (i.e., flipped bits) on processor registers during the execution of the CNN application with the

proposed CMSIS-NN extension.

Table 6.10 – FI Experimental Setup

Processor	Arm Cortex-A7
ML Model	Cifar-10 CNN
API/Libraries	CMSIS-NN
Dataset	CIFAR-10
Compiler and optimization flag	GCC 9.3 with $-O2$
OpenMP version	4.5
Instruction Sets	Thumb and SIMD
Target FI	General-purpose registers
Number of FI campaigns	6
Injections per campaign	17k
Total Fault Injections	102k

Source : The authors

To assess the reliability of the proposed CMSIS-NN extension, we have used the SOFIA framework to inject faults in the general-purpose registers of the Arm Cortex-A7 processor (i.e., r0-r15). Note that all CMSIS-NN optimizations, including SIMD instructions and OpenMP extension, only require general-purpose registers. Table 6.10 shows the FI experimental setup, which includes experiments with Thumb and SIMD instruction sets considering 1, 2, and 4 cores/threads running on the Arm Cortex-A7 model supported by the SOFIA framework. Such experimental setup considers the same version of compiler and optimization flags that was used for speedup analysis, keeping the consistency of the FI campaign results.

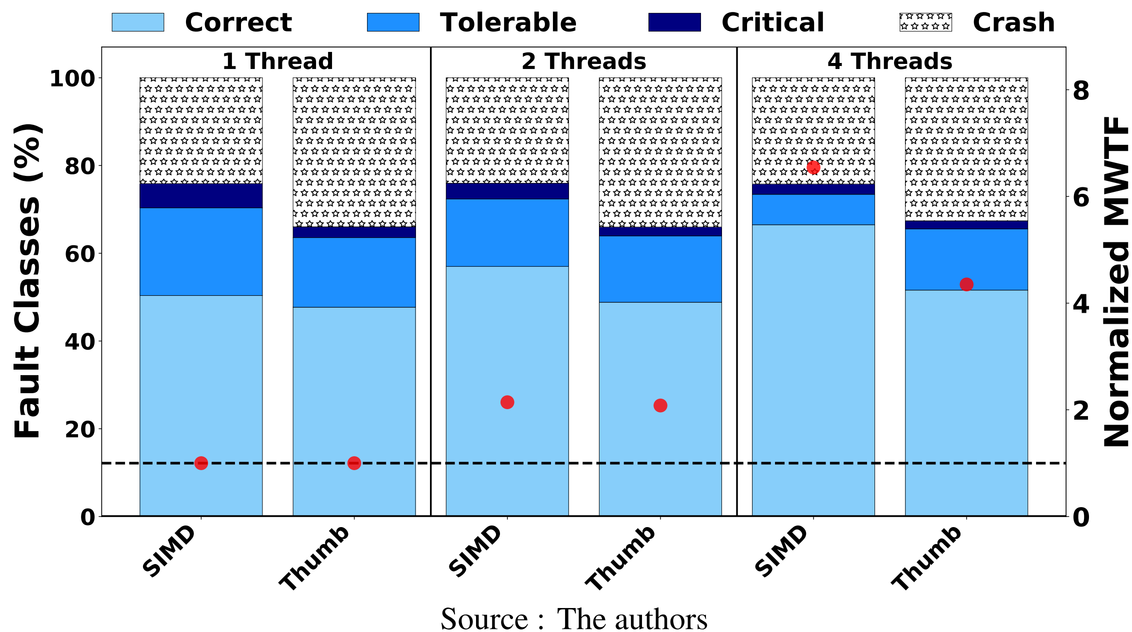
Note that each FI campaign contains N experiments, where each one contains a single input image and one CNN execution. Therefore, faults cannot be propagated to subsequent CNN executions. In addition, developing a realistic and precise approach is one of the main concerns for assessing the soft error reliability of a system. In this sense, this work applies the equations developed by LEVEUGLE et al. 2009. These equations ensure the statistical significance of the fault injection results. Thus, each FI campaign has 17k experiments to generate results with a 1% error margin and 99% confidence level.

6.1.4.4 Soft Error Reliability Assessment

Figure 6.11 presents the fault injection campaign results executing the adopted CNN application considering Thumb and SIMD instruction sets. The x-axis shows the Arm Cortex-A7 configurations for single, dual, and quad-core systems. The left y-axis shows the fault classes, while the red dots on the right y-axis show the MWTF gain

compared to the single thread execution version.

Figure 6.11 – Results from FI campaigns evaluating the thread parallelism with single, dual, and quad-core configurations.



Source : The authors

The results show that soft errors can affect the Thumb version more than the optimised SIMD version in terms of crashes. These faults are mostly unexpected terminations (e.g., handling exceptions) and are faults detected by the architecture. Even when the number of threads is compared, the same behaviour is observed (i.e., similar percentages in single, dual, and quad-core). This is mainly due to a large number of control and memory instructions between the calculations. Furthermore, the CNN application runs on a Linux kernel; thus, the longer the execution time, the more kernel control instructions are executed.

On the other hand, the SIMD version exploits MAC instructions to provide optimizations in the inference phase, reducing the fault impact. In addition, CNN's performance improvement by increasing the number of threads also increases correct outputs and tolerable faults. However, the SIMD instruction set presented a slight increase in critical faults. This behaviour is caused by the more significant number of SIMD MAC instructions executed between load/store operations, which are more sensitive to soft errors affecting the registers that contain the inference data.

Finally, Figure 6.11 also shows the impact of soft error reliability on parallel applications. Regarding the execution on dual and quad-core Arm Cortex-A7 processors, we can see that increasing the number of threads positively impacts the soft error reliability, reducing affected outputs (i.e., tolerable and critical faults) and increasing correct

outputs for both SIMD and Thumb versions. Although the SIMD version shows a higher percentage of critical faults, it has the best trade-off between reliability and performance when comparing the normalised MWTF. This is due to the architecture-specific optimizations of CMSIS-NN kernels aimed at SIMD instructions. According to the results shown in Figure 6.10, even with an almost linear speedup in Thumb parallel execution, the SIMD parallel version still provides 3× more performance than the Thumb, consequently impacting the resulting MWTF.

6.2 Soft Error Reliability Assessment of the MobileNet CNN

This section presents the soft error reliability assessment results for the MobileNet CNN case study. First, Section 6.2.1 presents a detailed description of the MobileNet CNN developed with the CMix-NN library. Next, Section 6.2.2, Section 6.2.3, and Section 6.2.4 comprise the soft error reliability assessment considering the MobileNet CNN execution, topology, isolated layers, precision-bitwidth, memory (code, parameters and data), and soft error mitigation techniques.

6.2.1 MobileNet CNN Developed with CMix-NN

This Section presents the MobileNet case-study (HOWARD et al., 2017), which consists of a 29-layer CNN developed with the CMix-NN library (CAPOTONDI et al., 2020) to run in SIMD featured Arm Cortex-M processors. The CNN is trained with the ImageNet dataset (DENG et al., 2009), consisting of 10 million labeled images depicting 1000 object categories.

Table 6.11 shows the MobileNet layers that include multiple convolution layers interspersed by depthwise separable convolutions, with an average pooling and a fully-connected layer at the end of CNN. Figure 6.12 shows an overview of a typical CNN topology and the depthwise separable convolutions, which factorize a standard convolution into a depthwise convolution and a 1×1 pointwise convolution. In this factorization, depthwise separable convolution splits filtering and combining processes into two layers, drastically reducing computation and memory footprint.

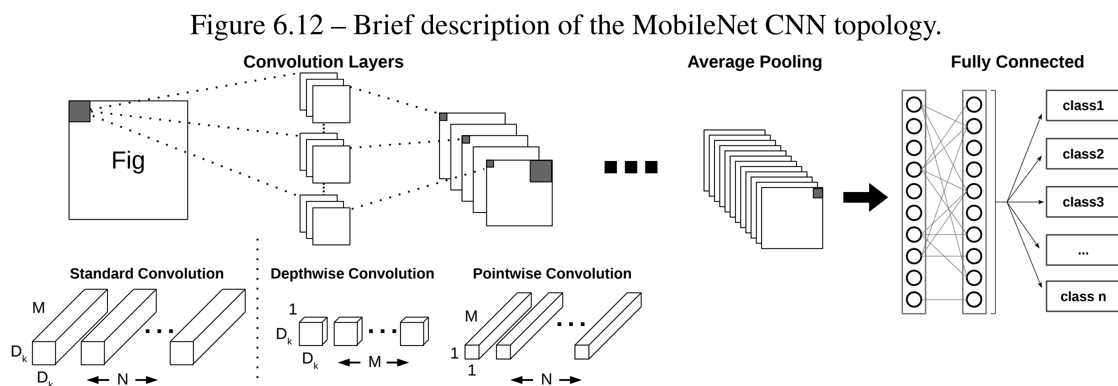
The MobileNet CNN (HOWARD et al., 2017) is a streamlined architecture that aims to build lightweight deep inference networks. Different from standard convolution

Table 6.11 – Layer parameters for the Mobilenet CNN.

	Layer Type	Stride	Filter Shape	Input Shape
1	Standard Convolution	2	3x3x3x8	160x160x3
2	Depthwise Convolution	1	3x3x8	80x80x8
3	Poitwise Convolution	1	1x1x8x16	80x80x8
4	Depthwise Convolution	2	3x3x16	80x80x16
5	Poitwise Convolution	1	1x1x16x32	40x40x16
6	Depthwise Convolution	1	3x3x32	40x40x32
7	Poitwise Convolution	1	1x1x32x32	40x40x32
8	Depthwise Convolution	2	3x3x32	40x40x32
9	Poitwise Convolution	1	1x1x32x64	40x40x32
10	Depthwise Convolution	1	3x3x64	20x20x64
11	Poitwise Convolution	1	1x1x64x64	20x20x64
12	Depthwise Convolution	2	3x3x64	20x20x64
13	Poitwise Convolution	1	1x1x64x128	10x10x64
14-23	5x Depthwise Convolution	1	3x3x128	10x10x128
	Poitwise Convolution	1	1x1x128x128	10x10x128
24	Depthwise Convolution	2	3x3x128	10x10x128
25	Poitwise Convolution	1	1x1x128x256	5x5x128
26	Depthwise Convolution	2	3x3x256	5x5x256
27	Poitwise Convolution	1	1x1x256x256	5x5x256
28	Average Pool	1	5x5	5x5x256
29	Fully-connected	1	256x1000	1x1x256
30	Softmax	1	Classifier	1x1x1000

Source : Adapted from HOWARD et al. (2017)

Figure 6.12 – Brief description of the MobileNet CNN topology.



Source : Adapted from HOWARD et al. (2017)

layers that filter and combine inputs into a new set of outputs in one step, MobileNet CNN is based on depthwise separable convolutions, which split this single step into two layers: depthwise convolutions and pointwise convolutions. First, depthwise convolutions apply a single filter to each input channel. Then, the pointwise convolution uses a 1×1 convolution to combine the depthwise convolution outputs. The depthwise separable convolution splits this into two layers, one separate layer for filtering and another layer for combining data, which drastically reduces the computation and model size. In this case study, the adopted MobileNet CNN was configured with a 3×3 depthwise separable

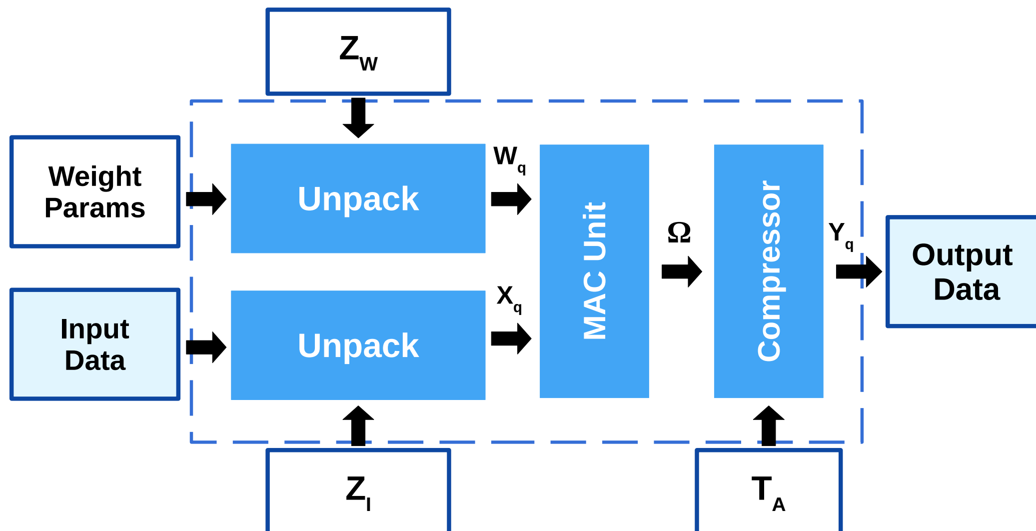
convolution, representing savings of up to 9 times in computational cost compared to standard convolutions. In addition to the convolution layers, an average pooling layer is employed to reduce the spatial resolution to 1 before the fully connected layer. In this sense, MobileNet has 29 layers, considering depthwise and pointwise convolutions as separate layers, except for the first layer that is a full standard convolution.

The adopted MobileNet CNN uses the CMix-NN library (CAPOTONDI et al., 2020) to implement mixed low-precision standard convolution and depthwise separable convolution layers functions. Unlike traditional CNN models that are trained using 32-bit floating-point data representation, the CMix-NN uses mixed low-precision unsigned integer representation. The CMix-NN kernels deploy optimizations focusing on enabling the execution of CNNs on Cortex-M based systems that support SIMD instructions, especially 16-bit *MAC* instructions (e.g., SMLAD). The CMix-NN library provides a complete set of convolutional kernels featuring a mixed low-bitwidth for the weights, input, and output activations that supports 8, 4, and 2 bitwidth combinations and different quantization techniques.

A typical mixed-precision *Quantized Convolutional Layer (QCL)* workload splits the convolution between quantized image-to-column and a matrix multiplication loop. The quantized functions load *Q*-bits input data in temporary buffers casting from the original *Q*-bits format to execute through vectorized SIMD 2×16 MAC instructions. Figure 6.13 illustrates the QCL internal components with memory requirements (e.g., inputs, weights, and structures) and the computational dataflow, which implement the mixed low-precision convolutional functions. The low-precision *MAC unit* accumulates the convolution result over a temporary 32-bit precision variable through vectorized MAC operations. In asymmetric quantizations, Z_w and Z_i apply the offset to the loaded parameter values to transpose them into the custom asymmetric domain. While the *Unpack* operation loads the convolution operands, the *Compressor* unit operates the final compression on the high-precision accumulation, considering a set of parameters T_A, which varies depending on the applied quantization technique.

In addition, CMix-NN library supports *Per-Layer (PL)* and *Per-Channel (PC)* compression techniques for any combination of bitwidth between input, output, and weights. While a PL quantization exploits a single *min/max* value for the entire layer, the PC computes a *min/max* value for any output channel. This latter approach is most beneficial when the weight distribution varies widely between channels. Furthermore, the CMix-NN library also supports the *Inter-Channel Normalization (ICN)* activation (RUSCI; CAPO-

Figure 6.13 – CMix-NN quantized convolutional layer.



Source : Adapted from CAPOTONDI et al. (2020)

TONDI; BENINI, 2019). This technique allows the introduction of lower bitwidth models with negligible inference loss, opening opportunities to exploit the soft error reliability of convolutional layers with precision bitwidth. The MobileNet mixed precision parameters have been acquired through the training scripts[1] provided by RUSCI; CAPOTONDI; BENINI (2019).

6.2.2 The Impact of Precision Bitwidth on the Soft Error Reliability of the MobileNet CNN Execution on Resource-constrained IoT Devices

This Section explores the soft error reliability of the MobileNet CNN considering different aspects: precision bitwidth and layer vulnerability analysis. In this sense, Section 6.2.2.1 details the experimental setup used to perform the fault injection campaigns. Section 6.2.2.2 rely on to analyze the execution lifetime of the adopted CNN. Furthermore, Section 6.2.2.3 exploits the MobileNet soft error reliability considering different precision bitwidth configurations.

6.2.2.1 Experimental Setup

To estimate the percentage of errors from the adopted case study, the two FI techniques mentioned in Section 4.1.2 are used to obtain the results, by injecting bit-flips in the general-purpose registers (i.e., r0-r15) of the Arm Cortex-M7 processor. Note that

[1]https://github.com/marco-fariselli/training-mixed-precision-quantized-networks

all optimizations from CMix-NN, including SIMD instructions, requires only the general-purpose registers. Table 6.12 shows the experimental setup to perform the proposed evaluation. For this experimental setup we consider 8 and 4 bit precision quantizations in weights (i.e., *w* in Table 6.12) and input/output activations (i.e., *a* in Table 6.12).

Table 6.12 – Experimental Setup.

Arm Processors	Cortex-M7
ML model	MobileNet CNN
API/Libraries	CMix-NN
Dataset	ImageNet
CNN Topology	1 Standard Convolution, 13 Depthwise, 13 Pointwise, 1 Average Pooling, 1 Fully-Connected
Target FI	Register File, Function Lifespan
Evaluated Precisions (w: weights, a: activations)	w4a4, w8a4, w4a8, w8a8
Number of FI Campaigns	40
Injections per Campaign	17k
Total Fault Injections	680k

Source : The authors

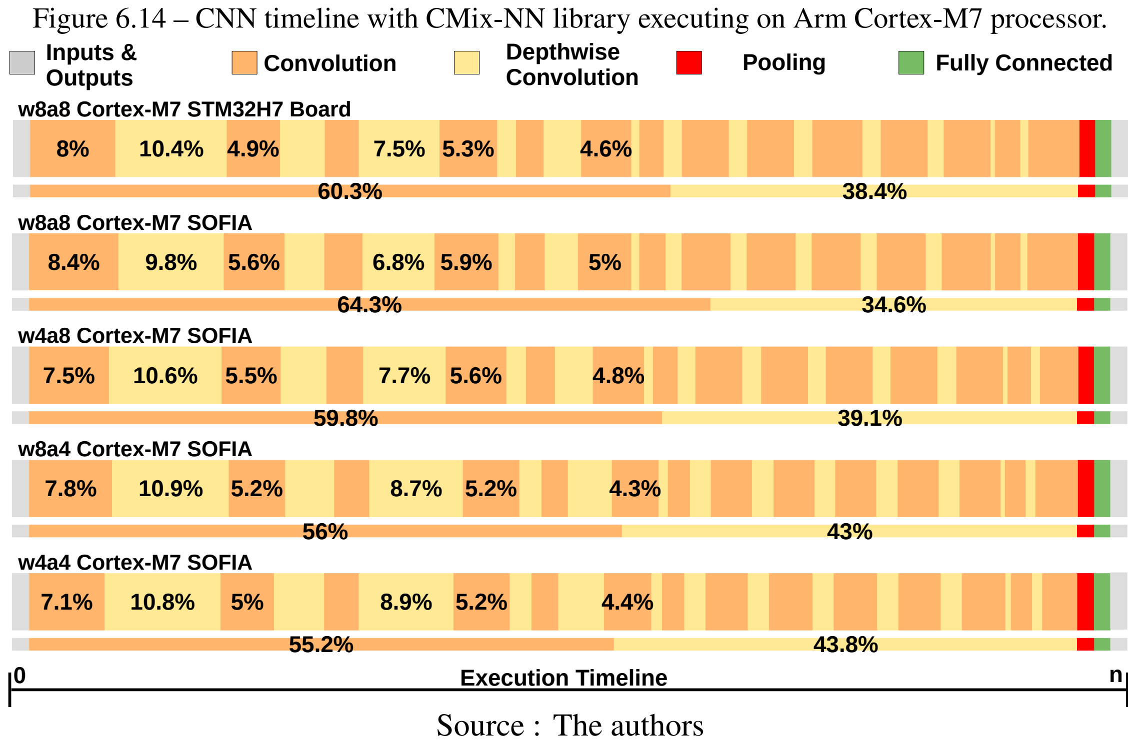
In order to maintain the coherence, these results consider the same compilation environment: GCC 9.3 and optimization flag -O2. Each fault injection scenario considers a single input image for one CNN execution, thus not considering fault propagation to the subsequent executions. Developing a precise, well-covered, and realistic approach is one of the main concerns when assessing a system's soft error reliability. Aiming to ensure that the number of fault injections has a statistical significance, this work applies the equations developed by (LEVEUGLE et al., 2009). In this sense, the fault injection campaigns consider 17k faults per scenario, thus generating a 1% error margin with a 99% confidence level.

6.2.2.2 MobileNet CNN Execution Lifetime

Asserting the experimental setup accuracy is paramount to obtain meaningful and useful results. To validate the adopted framework's utility, we compare the outputs reported in MobileNet[2] execution on the STM32H7 board with those gathered from the

[2]https://github.com/EEESlab/mobilenet_v1_stm32_cmsis_nn.git

Figure 6.14 – CNN timeline with CMix-NN library executing on Arm Cortex-M7 processor.

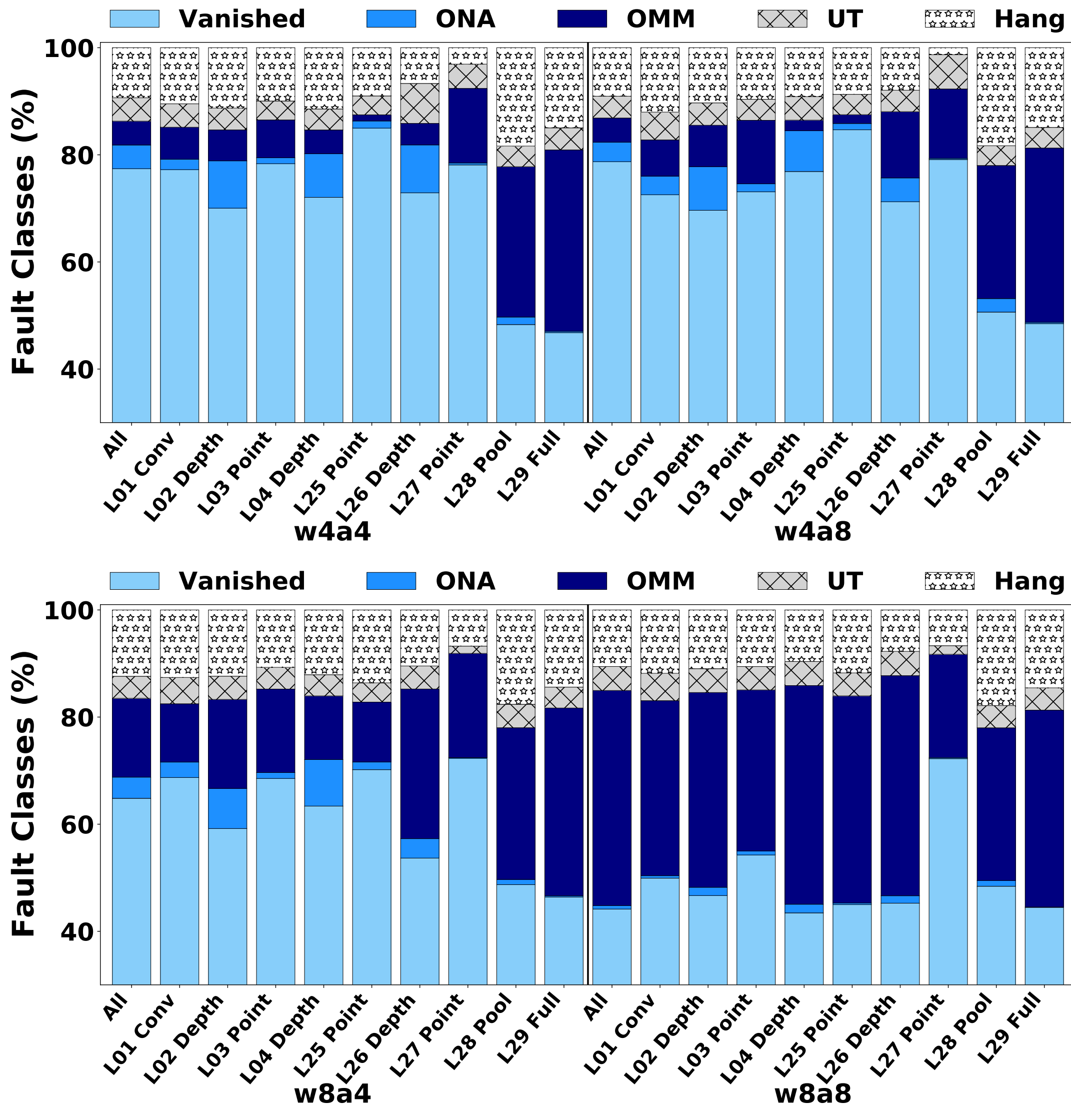


Source : The authors

execution in the absence of faults over the SOFIA. The results from CNN execution in SOFIA presented no difference in the output probabilities for the selected image w.r.t. the on-chip execution. Figure 6.14 shows the MobileNet CNN execution detailing the lifespan of each layer, considering the execution on Arm Cortex-M7 through the STM32H7 board and SOFIA.

Observing the functions' lifespan in Figure 6.14, it is notable that both standard convolution and depthwise separable convolution layers execute at most of the time while the remaining layers execute in a small part of the CNN execution. The variation of precision bitwidth also changes the function lifespans increasing the execution percent of depthwise separable convolutional layers. In this sense, it is expected that faults injected randomly are more likely to reach the convolution layers, having a considerable impact on the CNN reliability according to the precision bitwidth variations. Moreover, the execution percentages for the STM32H7 board are based on clock cycles, while in SOFIA, the percentages are based on the number of executed instructions. Such percents differ due to the number of required clock cycles to execute some instructions (e.g., a division might require from 2 to 12 cycles).

Figure 6.15 – Results from fault injection campaigns when injecting faults in the CNN layers with different precision bitwidth configurations.



Source : The authors

6.2.2.3 Soft Error Reliability Analysis

Aiming to evaluate the reliability considering the different aspects of the CNN execution, Figure 6.15 shows the results for the FI campaigns targeting all CNN layers when considering different precision bitwidth configurations. Such results are compared with those obtained from fault injection campaigns that target distinct layers of the CNN topology considering different volumes of data processed. The CNN functions use several loops to process data, which requires many control instructions, consequently generating high UT and Hang occurrence (i.e., 15% on average) in all precision configurations. Fully-connected and pooling layers show high soft error vulnerability in all configura-

tions since such layers do not vary in terms of quantized precision bitwidth. Observing the effect of quantizations on the CNN reliability, the quantization of inference weights affects reliability more than the quantization of activations since there is an increase in the masking rate when compared to *w8a8* (i.e., ~21% more vanished on *w4a4* and *w4a8* scenarios). According to the differences between the convolution layers, depthwise separable convolution layers are more susceptible to soft errors (i.e., high occurrence of ONA). Furthermore, the precision bitwidth can be linked to the increased vulnerability due to the greater probability of a failure spreading to the next layers (on average, OMM increases from 5% in *w4a4* to 35% in *w8a8* configurations).

Table 6.13 – Percentages of tolerable and critical faults in the CNN considering different precision bitwidth configurations.

		w4a4				w4a8			
	Layers	Correct	Tolerable	Critical	Crash	Correct	Tolerable	Critical	Crash
1	Std. Convolution	79,16	**0**	5,99	14,84	76,03	**0**	6,76	17,21
2	Depthwise	78,87	**0**	5,79	15,34	77,79	**0**	7,74	14,47
3	Pointwise	79,46	**0,03**	7,03	13,48	74,58	**0**	**11,86**	13,55
4	Depthwise	80,21	**0**	4,44	15,35	84,51	**0**	1,97	13,52
25	Pointwise	86,32	**0,83**	0,32	12,53	85,85	**1,10**	0,55	12,49
26	Depthwise	81,85	**3,83**	0,19	14,13	75,68	11,48	0,91	11,93
27	Pointwise	78,48	10,36	3,56	7,59	79,34	11,26	1,70	7,70
28	Av. Pool	49,71	16,35	**11,74**	22,21	53,16	**9,40**	**15,45**	21,99
29	Fully-Connected	47,05	29,62	4,24	19,08	48,73	26,37	6,17	18,73
all	All	81,83	**1,29**	3,18	13,71	82,36	**0,80**	3,72	13,12

		w8a4				w8a8			
	Layers	Correct	Tolerable	Critical	Crash	Correct	Tolerable	Critical	Crash
1	Std. Convolution	71,59	**0,04**	**10,85**	17,52	50,38	24,50	8,19	16,94
2	Depthwise	66,68	**0**	**16,61**	16,71	48,21	31,28	5,09	15,41
3	Pointwise	69,64	**0,04**	**15,56**	14,77	55,01	24,13	5,93	14,93
4	Depthwise	72,06	**0**	**11,90**	16,04	45,05	35,57	5,26	14,12
25	Pointwise	71,61	10,64	0,55	17,21	45,26	32,93	5,78	16,04
26	Depthwise	57,29	26,36	1,58	14,76	46,64	32,44	8,66	12,26
27	Pointwise	72,42	14,90	4,56	8,11	72,42	15,32	3,98	8,28
28	Av. Pool	49,68	17,56	**10,76**	21,99	49,48	14,58	13,94	22,01
29	Fully-Connected	46,62	31,15	3,92	18,31	44,54	31,48	5,28	18,71
all	All	68,78	**2,43**	**12,29**	16,50	44,79	35,49	4,69	15,04

Source : The authors

Table 6.13 shows the results comparing the output data from the fault injection campaigns with the fault-free execution. These results allow us to evaluate the soft error impact on the CNN accuracy according to the classification presented in Section 4.1.2. Considering the results of fault injections in all layers of the CNN, most of the effective faults tend to become critical, or system crashes in lower precision configurations (i.e., lower percents of tolerable faults in *w4a4*, *w4a8* and *w8a4*). Such an effect can be noted in

the convolutional layers with a high volume of data processing where the tolerable faults tend to 0% (i.e., layers 1 to 4). In turn, the number of effective faults increases when reducing the volume of data processed in the convolution layers (i.e., layers 25 and 26), where the number of effective faults increases when incrementing the precision bitwidth. Such results demonstrate that soft errors have a significant impact on the CNN execution and its accuracy, when considering the individual analysis of distinct layers and precision bitwidths.

6.2.3 The Impact of Soft Errors in Memory Units of Edge Devices Executing the MobileNet CNN

This Section explores the MobileNet CNN's behaviour in the presence of soft errors by applying fault injections to the register file and isolated memory sections.

6.2.3.1 Experimental Setup

Table 6.14 shows the experimental setup, where 459k bit-flip are injected in the general-purpose register file (i.e., r0-r15) and Flash and RAM memory sections. This work applies the equations developed by Leveugle *et al.* (LEVEUGLE et al., 2009) to ensure the statistical relevance of the fault injection results (KRZYWINSKI; ALTMAN, 2013). In this sense, each FI campaign includes 17k experiments, providing results with an 1% error margin and 99% confidence level. Note that these experiments do not consider the propagation of faults to subsequent executions, as each experiment consists of one bit-flip injection and a single CNN execution with a single input image.

The FI campaigns are conducted considering the Arm Cortex-M7. As mentioned before, the MobileNet was developed with the CMix-NN library, which only supports SIMD instructions. However, due to the large volume of data inherent to weights and bias of MobileNet and the limited amount of memory available in STM32F407ZGT6 (Table 6.15), this model was deployed in the STM32H743VIT6 that includes a Cortex-M7.

6.2.3.2 Soft Error Reliability Analysis

This case study assesses the soft error reliability of the MobileNet CNN considering its precision bitwidth variation defined according to the CMix-NN library. Figure 6.16

Table 6.14 – Experimental Setup

Arm Processor (executed instructions)	Cortex-M7 (400 millions)
ML Model	MobileNet CNN
API/Library	CMix-NN
Dataset	ImageNet
Precision Bitwidths	8, 4, and 2
API/Libraries	CMix-NN
Target FI	Register Files, Flash (code and parameters), RAM
Number of FI campaigns	27
Injections per campaign	17k
Total Fault Injections	459k

Source : The authors

Table 6.15 – On-chip Memory Occupation (MO) for 8-bit precision inference CNN models, considering code, parameters and data

Arm Processor	**Reference Chip**	**On-chip Memory**		**CNN Inference Model**			**MO$_{(\%)}$**
		Flash$_{(kB)}$	**SRAM$_{(kB)}$**	**Code$_{(kB)}$**	**Parameters$_{(kB)}$**	**Data$_{(kB)}$**	
Cortex-M7	STM32H743VIT6	2048	1024	20.2	1492.3	450.1	63.87

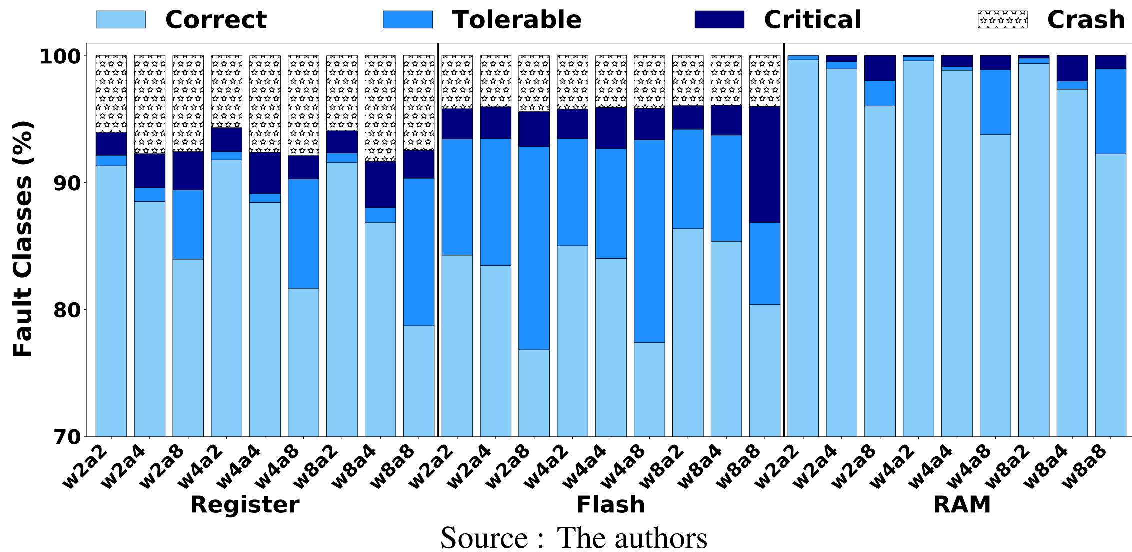
Source : The authors

shows soft error results obtained from the injection of bit-flips in the register file, Flash and RAM sections considering the Arm Cortex-M7. The Flash results present a large number of faults - most of them are tolerable. In addition, the percentage of critical faults remains similar for most precision bitwidths, the exception is the *w8a8* configuration. The results for the RAM section show that the higher precision (i.e., weights w and activations a) the more critical errors are expected.

Figure 6.17 presents the relative trade-off between the normalised Mean Work To Failure (MWTF) and memory-saving, considering different precision bitwidth configurations. The MWTF shows the average amount of work an application can perform until reaching a failure (i.e., higher values are better) (REIS; CHANG et al., 2005). Here, the memory-saving represents the amount of Flash/RAM not required by an implementation with regard to the highest and lowest requirements. This measure is based on the *.text* (Flash area for CNN code), *.data* (Flash area for CNN parameters) and *.bss* (RAM area for uninitialised variables) values reported by the compiler.

Figure 6.17 (a) shows that lower MWTF values are obtained at reduced precision bitwidths for code (CM), parameters (PM) and data (DM). This occurs because the incidence of a bit-flip in a reduced precision configuration is more likely to modify the code or weights, which might ultimately impact on the CNN's functional behaviour or output

Figure 6.16 – Results showing fault classifications for Register File, Flash, and RAM memory sections for the MobileNet CNN according to the precision bitwidth.



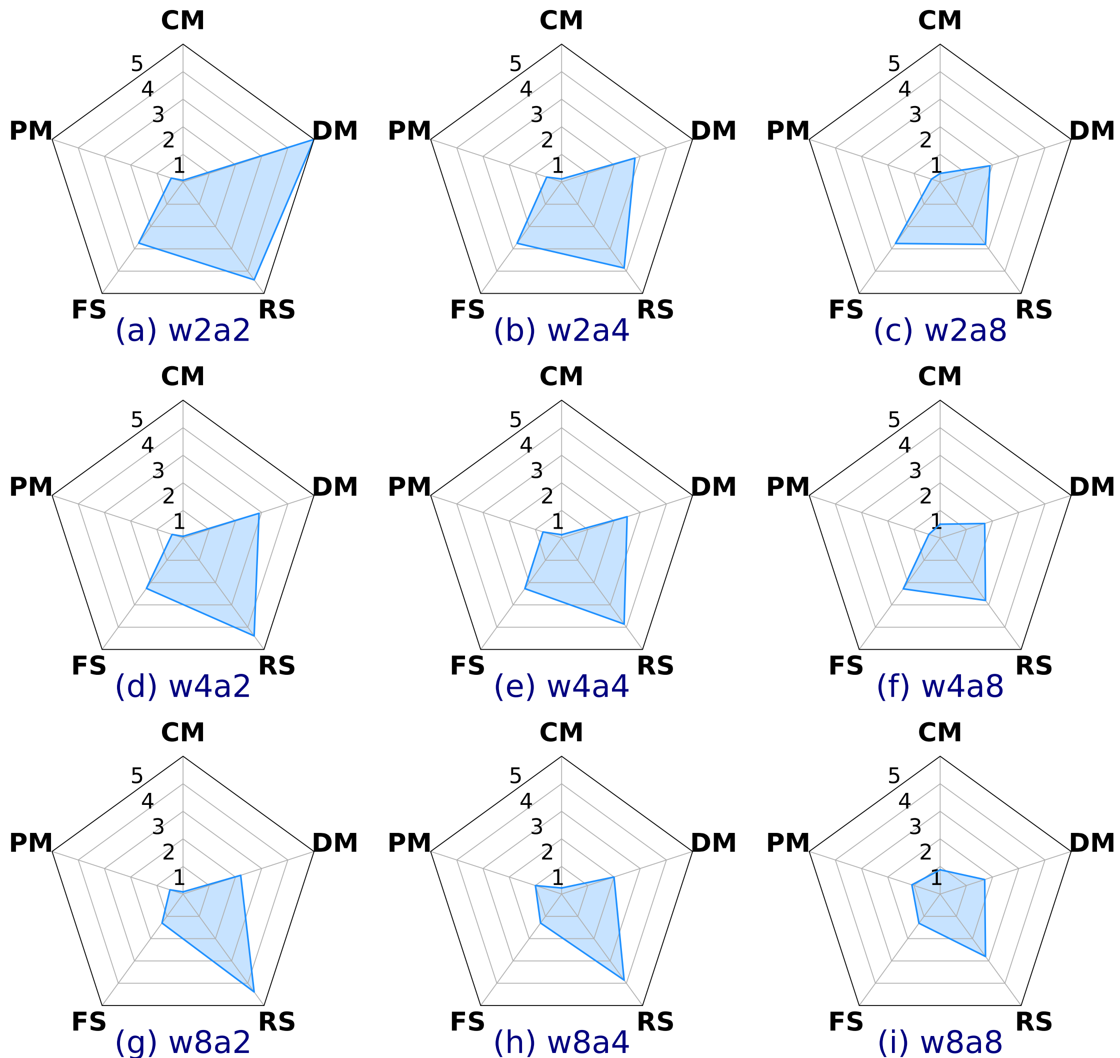
Source : The authors

classification. It is important to mention that the opposed happens in register files whereas faults are more often masked during the inference.

In regard to the memory-saving, results show that higher precision bitwidths (e.g., Figure 6.17 (i)) incur a larger amount of parameters and data, which lead to a larger vulnerability window. In this sense, the occurrence of a soft error might have a greater impact on the precision bitwidth variation of weights (from *w2* to *w8*), with an increase in the occurrence of critical failures when increasing the precision of activations (from *a2* to *a8*). This behavior is intensified in the *w8a8* configuration because, in this case, there is no precision offset between weights and activations and thus the fault tend to propagate through the CNN execution.

The results demonstrated that the soft error reliability of edge devices executing CNNs varies according to the affected memory section. If on the one hand the fault effects on register files are related to executed instructions and execution time. On the other hand, the fault effects on memory addressing are closely associated with the size of function object code, CNN parameters, and data. Furthermore, even in quantized solutions, a single bit-flip occurring on sensitive data stored in memory can compromise the entire CNN application.

Figure 6.17 – Relative trade-off between the normalised *Mean Work To Failure* (MWTF) and memory-saving, considering different precision bitwidth configurations, comparing *Code MWTF* (CM), *Parameters MWTF* (PM), *Data MWTF* (DM), *Flash-Saving* (FS), and *RAM-Saving* (RS).



Source : The authors

6.2.4 Applying Lightweight Soft Error Mitigation Techniques to Embedded Mixed Precision Deep Neural Networks

This Section explores the soft error reliability of the MobileNet CNN considering different aspects: precision bitwidth, layer vulnerability, mitigation techniques, and relative trade-off analysis. In this sense, Section 6.2.4.1 details the experimental setup used to perform the fault injection campaigns. Section 6.2.4.2 exploits the MobileNet soft error reliability considering different precision bitwidth configurations. Then, Section 6.2.4.3 presents the reduction of soft errors when applying two system-level mitigation techniques. Finally, Section 6.2.4.4 presents the relative performance and reliability trade-off for the adopted soft error mitigation techniques.

6.2.4.1 Experimental Setup

To provide trustworthy results, experiments consider more than 4.5 million fault injections to assess the soft error reliability of two mitigation techniques applied to the MobileNet on ImageNet. This work considers two FI techniques (i.e., *function lifespan* and *random register file*) to inject flipped bits in the general-purpose registers (i.e., r0-r15) of the Arm Cortex-M7 processor. This choice was motivated for two reasons. First, because the two FI techniques provide better coverage of the soft errors presented. Second, because all CMix-NN optimizations, including the SIMD instructions, require only the general-purpose registers. Note that this work focuses on the assessment and mitigation of soft errors originated from general-purpose registers, and hence it is assumed that the memory is protected by some type of error correction, such as Error Correcting Code (ECC) or parity bit.

Table 6.16 – Experimental Setup

Processor	Arm Cortex-M7
ML Model	MobileNet CNN
API/Library	CMix-NN
Dataset	ImageNet
Topology	1 Standard Convolution, 13 Depthwise, 13 Pointwise, 1 Average Pooling, 1 Fully-Connected
Target Layers	All, L01, L02, L03, L04, L25, L26, L27, L28, L29
Evaluated Precisions (w: weights, a: activations)	w2a2, w2a4, w2a8, w4a2, w4a4, w4a8, w8a2, w8a4, w8a8
Mitigation Techniques	P-TMR and RAT
Number of FI campaigns	270
Injections per campaign	17k
Total Fault Injections	4.59 millions

Source : The authors

Table 6.16 shows the experimental setup. The adopted MobileNet CNN has per-channel quantization with ICN layers (PC+ICN)(RUSCI; CAPOTONDI; BENINI, 2019), configured with the width multiplier of 0.5 and input sizes of 192, since this configuration has the minimum channel width required by 2-bit configurations.

In addition, this work uses the same compilation environment (i.e., *Clang 6.0.1*

and optimization flag *-O2*) to set the bitwidth configurations and the mitigation techniques. Each fault injection campaign considers a single input image for each MobileNet CNN execution, thus not considering the fault propagation to the subsequent executions. Also, each campaign is a particular configuration scenario and the 270 campaigns comprise: 10 target layers * 9 precision bitwidths * 3 mitigation cases (Code unprotected, P-TMR, and RAT). Furthermore, conducting a precise, well-covered, and realistic approach is key when assessing a system's soft error reliability. In this sense, to ensure the results' statistical significance, this work injects 17k faults per campaign, which according to LEVEUGLE et al. 2009, generates a margin of error of 1% with a 99% confidence level.

6.2.4.2 Soft Error Reliability Assessment of the MobileNet CNN Considering the Precision Bitwidth

Initially, we validated the SOFIA framework's faultless reference against the outputs reported by the MobileNet repository[3] and its on-chip execution. In this regard, we executed MobileNet CNN on an STM32H743[4] device, and then compared it with those collected from its execution on SOFIA, where no difference in the output probabilities was shown. This experiment is of paramount importance to guarantee the reproducibility and meaningfulness of the results.

After validating the reference flow, we generate fault injection campaigns considering the variation of precision bitwidth with weights raging from *w2* to *w8* and input/output activations from *a2* to *a8*. Table 6.17 shows the MobileNet CNN soft error results detailing the fault classification for each bitwidth configuration.

In general, results show a similar soft error reliability behaviour between the different quantization configurations. For example, the results from fault injections in *All* layers in Table 6.17, crash occurrences handle 6.92% on average across all precision bitwidth configurations; this is because MobileNet CNN functions use multiple loops to process data that require many control instructions. Note that CNNs are known to have a lot of redundancy built-in, due to which they present a reasonable masking rate, which justifies the average of 86.7% of correct outputs. However, a single critical fault occurrence in safety-critical systems running the underlying trained models can lead to fatal consequences (i.e., life-threatening).

Table 6.17 also shows how the quantization of inference activations affects the

[3]https://github.com/EEESlab/mobilenet_v1_stm32_cmsis_nn.git
[4]https://www.st.com/en/microcontrollers-microprocessors/stm32h743vi.html

Table 6.17 – Percentages of tolerable and critical faults on MobileNet CNN considering different precision bitwidth configurations.

	Layers	w2a2 Correct	w2a2 Tolerable	w2a2 Critical	w2a2 Crash	w2a4 Correct	w2a4 Tolerable	w2a4 Critical	w2a4 Crash	w2a8 Correct	w2a8 Tolerable	w2a8 Critical	w2a8 Crash
1	Std Convolution	91.65	**0.01**	**2.03**	6.31	89.98	**0.01**	**1.42**	8.58	89.81	**0.41**	0.93	8.85
2	Depthwise	92.66	**0.08**	0.15	7.11	92.77	**0.01**	0.24	6.98	85.62	5.02	**2.95**	6.41
3	Pointwise	90.58	**0.07**	**3.76**	5.58	88.32	**0.12**	**3.44**	8.12	83.31	**2.06**	**6.23**	8.40
4	Depthwise	91.61	**0.06**	**1.55**	6.77	88.55	**0.00**	**4.37**	7.08	79.93	6.59	**7.11**	6.36
25	Pointwise	92.84	**1.31**	0.47	5.39	89.75	**1.85**	0.58	7.82	87.53	**3.66**	0.78	8.03
26	Depthwise	84.49	7.10	**1.29**	7.12	84.44	7.13	**1.26**	7.17	82.22	11.27	0.55	5.96
27	Pointwise	80.04	9.84	**2.13**	8.00	79.94	11.15	**1.21**	7.70	79.28	12.19	0.71	7.81
28	Average Pooling	65.28	20.11	**3.86**	10.74	65.19	20.02	**4.04**	10.75	65.55	20.19	**3.47**	10.78
29	Fully-Connected	75.08	17.57	**1.89**	5.46	74.30	18.15	**1.93**	5.62	74.19	18.17	**1.97**	5.67
all	All	91.32	**0.84**	**1.80**	6.04	88.51	**1.11**	**2.66**	7.72	83.96	5.45	**3.02**	7.56

	Layers	w4a2 Correct	w4a2 Tolerable	w4a2 Critical	w4a2 Crash	w4a4 Correct	w4a4 Tolerable	w4a4 Critical	w4a4 Crash	w4a8 Correct	w4a8 Tolerable	w4a8 Critical	w4a8 Crash
1	Std Convolution	91.81	**0.00**	**1.75**	6.45	90.18	**0.01**	**1.08**	8.74	89.60	**0.60**	0.87	8.93
2	Depthwise	92.84	**0.00**	0.15	7.01	92.79	**0.05**	0.18	6.98	84.63	8.72	0.65	6.00
3	Pointwise	90.64	**0.03**	**3.78**	5.55	88.51	**0.01**	**3.47**	8.01	81.47	6.35	**4.31**	7.86
4	Depthwise	91.60	**0.00**	**1.54**	6.86	88.04	**0.04**	**5.03**	6.89	78.94	13.78	**1.19**	6.08
25	Pointwise	92.11	**2.04**	0.25	5.60	89.95	**1.92**	0.42	7.71	85.76	5.78	0.54	7.93
26	Depthwise	85.49	6.77	**1.21**	6.53	84.36	7.09	**1.50**	7.05	80.51	13.05	0.46	5.98
27	Pointwise	81.51	8.35	**2.32**	7.83	82.10	8.89	**1.43**	7.58	80.22	10.81	0.94	8.04
28	Average Pooling	65.66	19.55	**3.85**	10.94	65.41	20.21	**3.79**	10.59	66.04	19.54	**3.49**	10.94
29	Fully-Connected	75.19	17.18	**1.95**	5.67	74.21	18.16	**1.99**	5.64	74.42	17.91	**2.16**	5.51
all	All	91.79	**0.66**	**1.88**	5.67	88.43	**0.72**	**3.24**	7.61	81.68	8.61	**1.86**	7.85

	Layers	w8a2 Correct	w8a2 Tolerable	w8a2 Critical	w8a2 Crash	w8a4 Correct	w8a4 Tolerable	w8a4 Critical	w8a4 Crash	w8a8 Correct	w8a8 Tolerable	w8a8 Critical	w8a8 Crash
1	Std Convolution	92.88	**0.01**	0.70	6.41	90.19	**0.01**	0.87	8.93	89.78	**0.64**	0.66	8.92
2	Depthwise	92.85	**0.06**	0.19	6.89	92.06	**0.03**	0.32	7.58	83.90	8.90	0.54	6.66
3	Pointwise	90.56	**0.04**	**3.58**	5.82	87.73	**0.02**	**3.87**	8.38	78.65	11.58	**1.49**	8.28
4	Depthwise	91.60	**0.06**	**1.39**	6.95	86.15	**0.02**	**6.35**	7.48	75.77	17.18	0.68	6.36
25	Pointwise	89.99	4.14	0.20	5.68	88.72	**3.23**	0.28	7.77	79.27	9.22	**3.54**	7.97
26	Depthwise	84.19	7.03	**1.62**	7.16	82.93	7.94	**1.92**	7.21	79.98	8.45	**4.94**	6.64
27	Pointwise	81.24	8.51	**2.49**	7.76	81.39	8.86	**1.64**	8.11	84.01	6.58	**1.64**	7.78
28	Average Pooling	65.80	19.73	**3.55**	10.92	65.65	19.56	**3.69**	11.10	66.73	6.89	**15.44**	10.94
29	Fully-Connected	74.52	17.67	**2.13**	5.68	73.56	18.39	**2.11**	5.94	75.79	16.56	**2.23**	5.42
all	All	91.60	**0.74**	**1.78**	5.88	86.84	**1.22**	**3.61**	8.34	78.71	11.64	**2.21**	7.45

Source : The authors

MobileNet CNN soft error reliability by reducing the correct outputs in up to 10.12% when varying from *a2* to *a8*. Although it affects less, increasing the weight bitwidth also reduces the soft error reliability by up to 2.21% on the correct outputs when varying the configurations from *w2* to *w8*. This occurs because a SIMD MAC instruction splits a 32-bits register into different segments, which are set according to the bitwidth configuration. This leads to a higher probability to overwrite the faulty bit, thus reducing the probability to propagate the fault to the inference phases. Results show that both higher precision bitwidths and the unpack/compress process can led to an increase number of faults. The unpack/compress process is related to load and store instructions that are executed before and after SIMD MAC operations, which increases the probability of a fault impacts on the output probabilities. Note that the increase in the precision bitwitdh can also reduce the fault criticality since a soft error in a less significant bit is less likely to generate a critical fault. Such behaviour can be seen in Table 6.17 *All* when comparing *w8a4* and *w8a8* configurations, where the number of critical faults decreases and tolerable faults increases

significantly.

To understand the layers' vulnerability to the soft errors, Table 6.17 shows the results obtained from FI campaigns that target distinct layers of the MobileNet CNN topology considering different data volumes. Results show that the most effective faults tend to become either critical or system crashes in low-precision activation configurations (i.e., *a2* and *a4*). Such an effect can be seen in convolutional layers with a high volume of input data processing, where tolerable faults tend to 0% (i.e., layers 1 to 4). In turn, the number of tolerable faults increases as the input data volume reduces in the convolution layers (i.e., layers 25 and 27). This is because the dimensions of input activations are reduced during MobileNet CNN execution, while the channel width increases, thus making the data volume larger in weights w.r.t. input activations. Consequently, faults occurring in these layers are more likely to propagate to the output, i.e., the appearance of a high number of tolerable and critical faults in layers 26 and 27. Note that the occurrence of faults (i.e., Tolerable + Critical + Crash) also increases alongside the precision bitwidth.

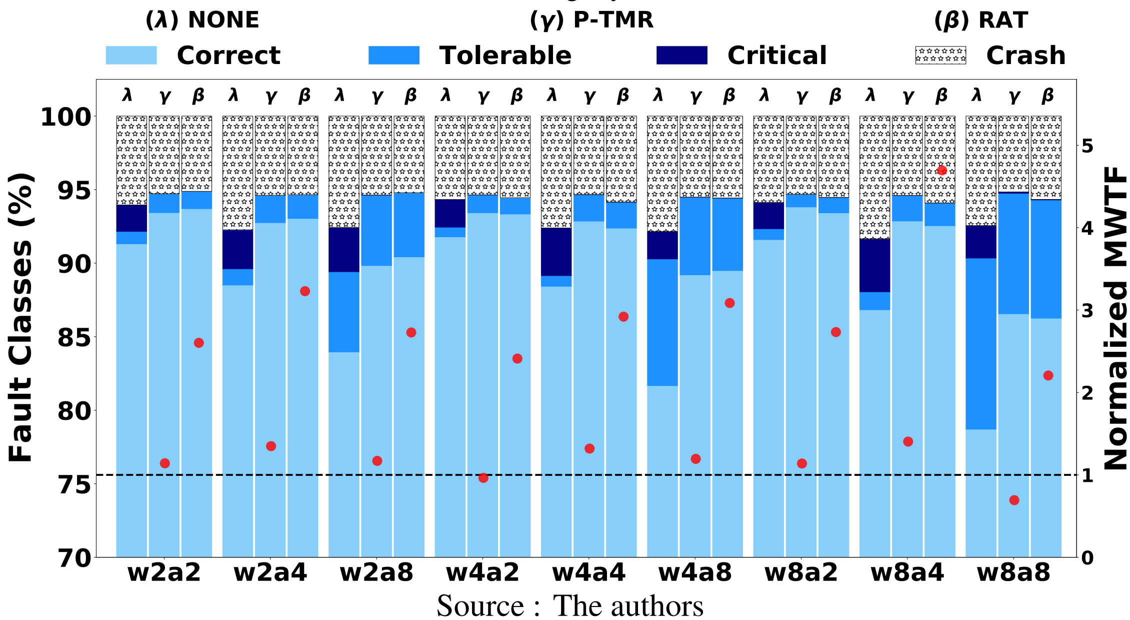
6.2.4.3 Applying mitigation techniques to MobileNet CNN

Aiming to reduce the MobileNet CNN susceptibility to soft errors, this Section considers the use of two software-based mitigation techniques: P-TMR and RAT. Both techniques are applied to the matrix multiplication function, which is considered here the most critical one due to its higher active period within the MobileNet execution time.

Figure 6.18 shows the reliability improvement of MobileNet CNN by applying the two mitigation techniques. The x-axis has three bars for each adopted precision bitwidth configuration. The first bar represents MobileNet CNN with unprotected code (λ), and the other two the mitigation techniques P-TMR (γ) and RAT (β). The left-hand y-axis shows the soft error percentage obtained from the fault injection campaigns, and the right-hand y-axis shows the MWTF normalized by the unprotected version. While the left-hand metric shows an overview of the generated faults, the one at the right-hand side relies on a well-accepted reliability metric to compare the two mitigation techniques.

As expected, Figure 6.18 shows that both mitigation techniques significantly increase the number of correct outputs while reducing the number of critical faults. In general, a lower precision bitwidth configuration (i.e., 2 and 4-bits to *w* and *a*) lead to a reduction in fault occurrences. On the other hand, 8-bit configurations present a higher occurrence of tolerable faults. This is due to larger bitwidth operations that reduce the fault masking rate. Even under these conditions, both mitigation techniques protect the

Figure 6.18 – Results showing fault classifications comparing MobileNet CNN without protection, with P-TMR, and RAT mitigation techniques. The red dots indicate the normalized MWTF (right y-axis).



Source : The authors

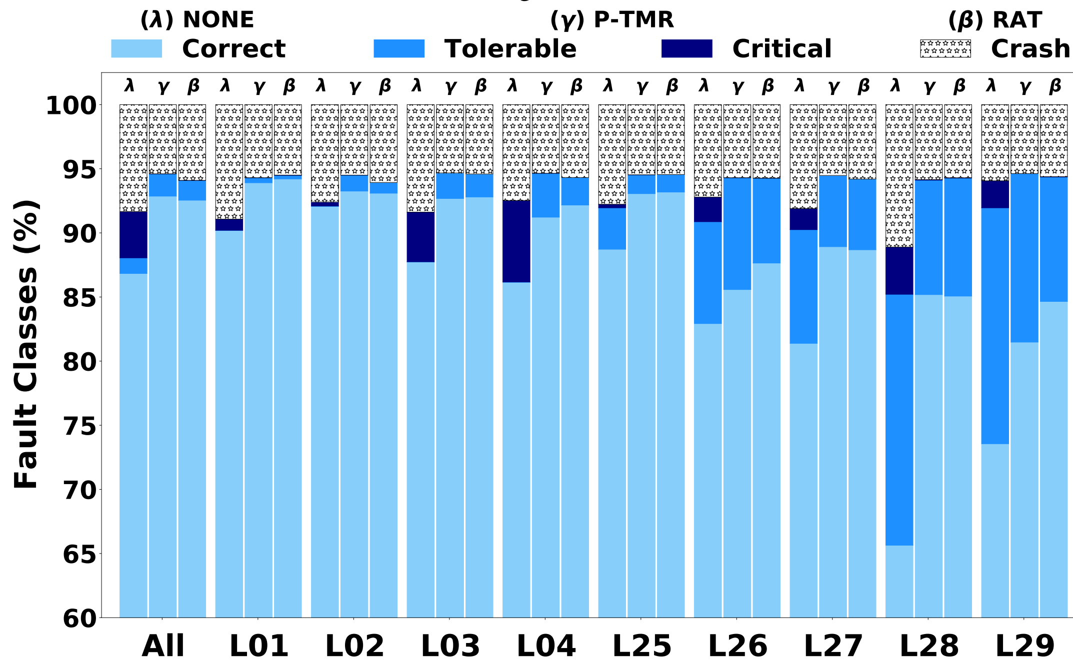
code and turn critical faults into correct or tolerable ones. Figure 6.18 shows that P-TMR has a significant AVF improvement in all scenarios, but the performance penalty does not compensate for w4a2 and w8a8 configurations (i.e., normalized $MWTF < 1$). In turn, the RAT improved the MWTF in all configurations, raising up to 4.7× in the *w8a4* precision bitwidth.

Figure 6.19 shows the reliability improvement per layer for the most affected precision bitwidth configurations (i.e., *w8a4* and *w8a8*). Compared to the unprotected version, both mitigation techniques show soft error reliability improvements. On the one hand, Figure 6.19.a shows a reduction of up to 8% in critical faults and system crashes in the 4-bit precision activation. On the other hand, Figure 6.19.b illustrates that some tolerable, critical, and crash faults become correct outputs for 8-bit precision activations. In the P-TMR perspective, this effect occurs mainly due to faults striking registers used by redundant instructions. Unlike, RAT reduces the number of vulnerable registers during the critical function's execution, taking advantage of the inherent high resilience of MobileNet.

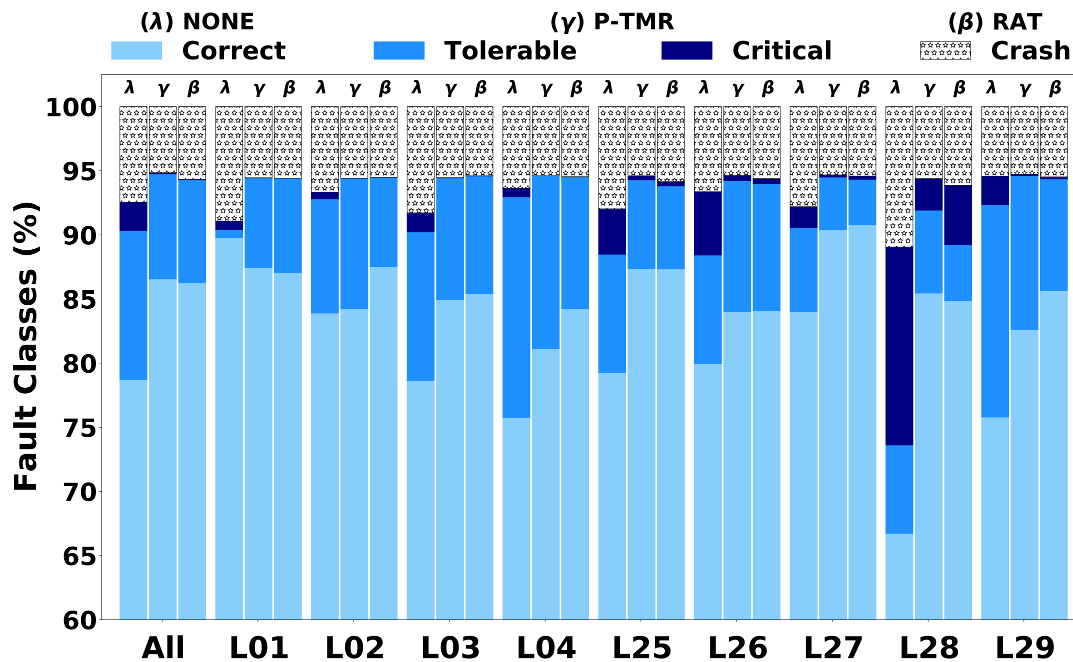
6.2.4.4 Trade-off Between Performance and Reliability

This Section details the drawbacks introduced by the two mitigation techniques and discusses the trade-off between increased protection and performance penalty. Figure 6.20 shows the performance overhead of the P-TMR and RAT compared to no protection

Figure 6.19 – Results of fault classification by layer of the two most affected precision bitwidth configurations.



(a) w8a4



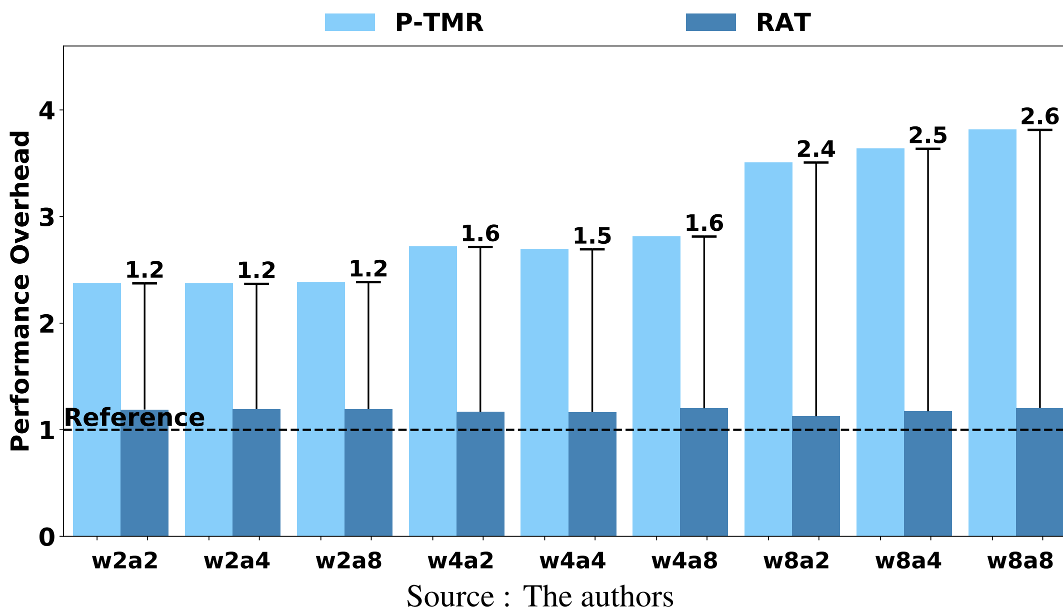
(b) w8a8

Source : The authors

execution. The execution times were extracted by running the MobileNet CNN on an STM32H743 board. Results show a performance degradation of up to 1.2× for RAT and 3.8× for P-TMR, depending on the precision bitwidth configuration. In this regard, the original MobileNet achieves ∼1.9 inferences per second. In turn, when applying

the P-TMR mitigation technique, the number of inferences per second reduces to ~0.5 while the RAT ~1.5 in worst-case scenarios. Note that the most remarkable performance overhead occurs because P-TMR is applied to the application's intermediate code without further optimization, i.e., the application is compiled with *-O2* and the mitigation technique is applied without architecture-specific optimizations. This approach is required to avoid code removals made by the compiler's backend.

Figure 6.20 – MobileNet execution time overhead considering P-TMR and RAT.



Source : The authors

Figure 6.21 compares the relative trade-off between reliability, accuracy, performance and memory footprint overhead for two precision bitwidth configurations (*w8a4* and *w8a8*). Table 6.18 shows the footprint overhead, which is calculated based on the additional hardened application code size resulting from both P-TMR and RAT mitigation techniques w.r.t. the original application code (i.e., Flash memory).

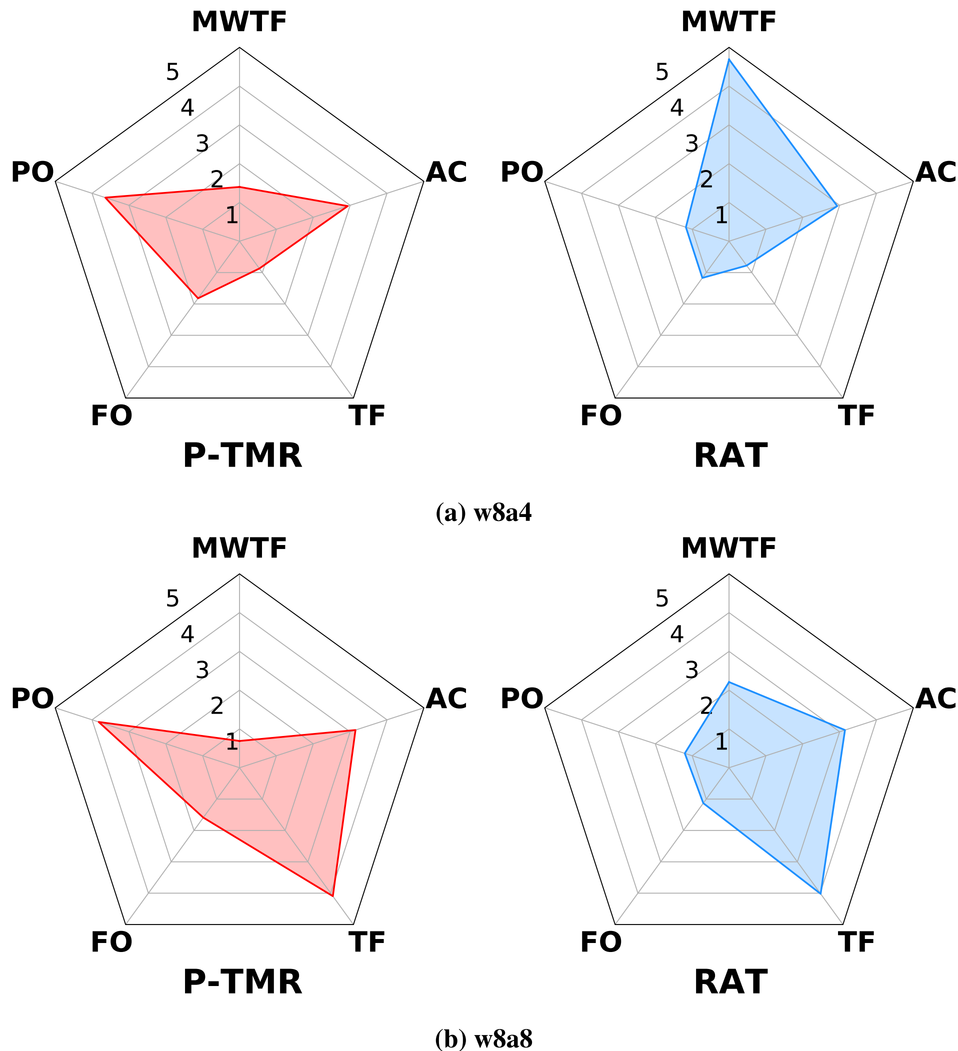
Table 6.18 – Normalized MobileNet footprint overhead when applying soft error mitigation techniques.

Mitigation	w2a2	w2a4	w2a8	w4a2	w4a4	w4a8	w8a2	w8a4	w8a8
P-TMR	1.78	1.80	1.59	1.80	1.82	1.60	1.81	1.82	1.59
RAT	1.16	1.16	1.13	1.16	1.17	1.13	1.16	1.17	1.13

Source : The authors

This comparison provides an overview of the advantages and disadvantages of both mitigation techniques when applied to the MobileNet CNN. Gathered values are normalized between 1 and 5, and the top axis represents the MWTF improvement, the two

Figure 6.21 – Relative trade-off between P-TMR and RAT mitigation techniques considering *w8a4* and *w8a8* precision bitwidth configurations, comparing *Mean Work To Failure* (MWTF), *Performance Overhead* (PO), *Footprint Overhead* (FO), *Accuracy* (AC), and *Tolerable Faults* (TF).



(a) w8a4

(b) w8a8

Source : The authors

left axes represent the performance and memory overheads, and the two right axes show the precision and the tolerable percentages. Figure 6.21 clearly shows that RAT presents a significant lower performance overhead, which directly led to an improved MWTF w.r.t. the P-TMR.

The resulting performance overhead can be explained not only by the increased number of instructions but also the instruction set employed by each mitigation technique. Table 6.19 shows that P-TMR consists of almost 5× more *Thumb* instructions, which correspond to near 90% of the entire hardened code. This highly increase is mainly due to the register spilling forced by the high register pressure and the impact of replicated instructions. In turn, RAT slightly increases the number of executed *Thumb* and *SIMD*,

Table 6.19 – Percentage of use and relative increase in executed instructions considering different instruction sets.

Type	w8a4 None	w8a4 P-TMR	w8a4 RAT	w8a8 None	w8a8 P-TMR	w8a8 RAT
SIMD	34.3%	9.8%	32.2%	36.7%	10.1%	30.5%
Thumb	65.7%	90.2%	67.8%	63.3%	89.9%	69.5%
Type	**None**	**P-TMR**	**RAT**	**None**	**P-TMR**	**RAT**
SIMD	1×	1.1×	1.05×	1×	1.1×	1.06×
Thumb	1×	4.8×	1.15×	1×	4.9×	1.23×

Source : The authors

maintaining the percentage of both instruction sets compared to the reference MobileNet execution. Aforementioned results demonstrate that RAT provides the best relative performance, reliability and memory footprint utilisation trade-off.

These results are of paramount importance for safety-critical applications because, in addition to reliability, these applications have real-time requirements. For example, in self-driving cars, a late reaction can lead to a fatal accident (BANERJEE et al., 2018). In this context, traditional soft error mitigation solutions involving time redundancy, such as TMR, may not be suitable for such kind of applications. Even when partially applied, this technique might inflict a significant response time penalty that is not tolerable in real-time applications, thus justifying the need for lightweight techniques, such as RAT, especially for resource constraint systems.

7 CONCLUSIONS AND FUTURE WORK

The first goal of this Thesis comprises to make an extensive and statistical significance soft error consistency assessment of a JIT-based fault injection framework. This work has investigated the soft error assessment consistency of a JIT virtual platform simulator (SOFIA) with more than 12 million fault injections considering single and multi-core Arm processor architectures. The fault injection campaigns considered different cross-compilers, software stacks, programming models, and 52 applications. Results demonstrated that the architectural difference, such as the ISA, between the two Arm processors, affects the reliability of single-core systems. However, the addition of tiny operating systems (e.g., FreeRTOS) in the software stack did not affect the consistency of soft error assessment. Regarding cross-compilers, those based on LLVM appeared to be more reliable ones, with the best compiler set being the *Clang 6.0.1* using the *02* optimization flag.

Furthermore, we showed a worsened of the mismatch while increasing the number of cores that can be mitigated a little by improving the workload balance and context switching in parallel applications, such as in the MPI programming model. Finally, we demonstrated that by tuning the SOFIA for a more detailed simulation by the quantum size parameter (i.e., 44-instruction block), we obtained the best cost-benefit in terms of soft error assessment accuracy, with a worst-case mismatch of 8.76% and high simulation performance, reaching up to 345 MIPS. We conclude that achieved mismatches are acceptable and are not a hindrance to evaluate soft errors at early design phases. Further, given the remarkably achieved speedup, SOFIA's utilization appears promising since it can also be used to compare different processor models, ISAs, kernel, and complex benchmarks with billion of instructions. Finally, authors also believe that the high statistical significance presented gives to this work the potential to be a reference for other studies with concerns about soft error resilience of Arm processors. In this sense, the obtained results validates *the first hypothesis of this Thesis*. Considering the consistency of these results, it is possible to state that SOFIA provides a consistent soft error reliability assessment while achieving some performance w.r.t. RTL and gem5 simulators. Also, these findings allow us to proceed to assess the reliability of more complex benchmarks such as ML applications targeting resource-constrained IoT devices.

In the second goal of this Thesis, we use the SOFIA framework to assess the soft error reliability of CNNs developed based on CMSIS-NN and CMix-NN specialized

libraries, which enable the execution of complex CNNs in Arm Cortex-M processor architectures featuring SIMD instructions. SOFIA's flexibility allows us to cover different aspects of a given software stack while adopting different fault injection techniques. This work uses two fault injection techniques (*Random register file* and *Function Lifespan*) to isolate specific moments of CNN's execution, making it possible to perform fault injections on each CNN layer. The first case study comprises the soft error reliability evaluation of the quantized CIFAR-10 CNN (i.e., 8-bit precision) developed with CMSIS-NN library considering two Arm Cortex-M processor architectures and trained with the CIFAR-10. The evaluated results demonstrate that the adopted CNN has a high susceptibility to soft errors since, in most cases, the effective faults exceed 50%. The activation layers are more susceptible to soft errors since critical failures affect both the MWTF and the accuracy of CNN. Furthermore, the results from Cortex-M4 show that SIMD based optimizations reduce the CNN susceptibility to soft errors when comparing to Cortex-M3. The second case study comprise the MobileNet CNN, which is developed based on the mixed low-precision CMix-NN library considering the Arm Cortex-M7 processor architecture and trained on the ImageNet dataset. The evaluated results demonstrate that the adopted CNN has a high susceptibility to soft errors in higher precision bitwidth configurations since, in most cases, the effective faults exceed 50%. Also, the precision bitwidth of the weights is more susceptible to soft errors than the activations, since critical failures affect both the reliability and the accuracy of CNN. Moreover, reducing the precision bitwidth of weights and activations further affects CNN's soft error reliability by increasing the masking rate by 21%. Considering such early evaluations, our third case study relies on apply lightweight soft error mitigation techniques to mixed precision MobileNet CNN. The evaluated results demonstrate that the adopted CNN has a high susceptibility to soft errors in higher precision bitwidth configurations, since the fault occurrence achieves up to 20%. Results also demonstrate that the variation of precision bitwidth of the activations is more susceptible to soft errors than the weights, since critical failures affect both the reliability and the accuracy of CNN. Moreover, the reduction of weights and activations precision bitwidth increases the fault-masking capability of up to 10%, thus reducing the MobileNet CNN susceptibility to the occurrence of soft errors. However, such precision bitwidth reduction does not eliminate the occurrence of critical failures, requiring the use of fault mitigation techniques. Finally, gathered results show that RAT provides significant soft error reliability improvement at a lower performance penalty (i.e., ~1.2× on average) when compared to the P-TMR. Such conducted studies validates *the second*

hypothesis of this Thesis, which early investigates and identifies the correlation between FI results, NN optimized kernels, and reduced precision parameters of CNNs executing on resource-constrained IoT edge devices.

The third goal of this Thesis relies on investigate the soft error reliability of two CNN inference models deployed in three low-power Arm microprocessors. The results obtained from more than 500k fault injection campaigns show that CNN's code and parameters stored in Flash memory are more susceptible to soft errors than related data in RAM memory. While boosting the performance, SIMD instructions supported by Arm Cortex-M4 and M7 processors increase the memory footprint and the CNN susceptibility to soft errors. In regard to the precision bitwidth variation on MobileNet CNN, the occurrence of soft errors might have a greater impact on weights, with an increase in the occurrence of critical errors when increasing the precision of activations. Finally, this confirms *the third hypothesis of the Thesis*, that the reduced precision optimizations impacts not only in the CNN execution, but also in the CNN code, parameters, and data stored in memory.

Although many works in the literature address dedicated to ultra-low-power integrated circuits (e.g., Arm Cortex-M family), an alternative to achieve better performance requirements is the execution of underlying ML models in more powerful processors (e.g., Arm Cortex-A family). As an effort to cover the constraints of this approach, the late results of this thesis comprises the assessment of the soft error reliability of a CIFAR-10 trained CNN model, which was developed based on the CMSIS-NN kernel to support multi-threaded execution. The proposed extension showed that threaded parallelism could increase the performance of the adopted CNN application with a low memory footprint overhead (i.e., less than 1%). Furthermore, the results showed that the evaluated CNN application is highly susceptible to failures as it presents critical faults in both configurations. In turn, multi-threaded versions positively impact CNN reliability while increasing the number of correct outputs, performance and, consequently, the MWTF of the adopted CNN.

Future works will focus on two main directions. The first direction comprises the soft error assessment of accelerator-based IoT devices, which are more suitable for highly computing-intensive AI applications. The second direction relies on explore the faults occuring on memory sections. Although protection mechanisms implemented in Flash and RAM memories can be employed to prevent data loss, identifying the most vulnerable CNN's code, parameters or data gives the opportunity to promote bespoke software-based

solutions such as the replication of a specific function or set of parameters.

Regarding the contributions of this work, Table 7.1 summarizes the papers submitted for publication during the Ph.D. program. It is important to mention that the first two publications will be used in future works that aim to assess the reliability of ML models executing on NoC-based multiprocessor systems. Also, the most recent submissions refer to the results presented in the Chapter 5 and Chapter 6.

Table 7.1 – Publications Summary.

Year	Target	Title	Status	Thesis Section
2018	ISCAS	Exploring the impact of soft errors on NoC-based multiprocessor systems	Published	—
2018	SBCCI	A design patterns-based middleware for multiprocessor systems-on-chip	Published	—
2020	ICECS	Soft Error Reliability Assessment of Neural Networks on Resource-constrained IoT Devices	Published	Section 6.1.2
2021	IET-CDT	Evaluation of the Soft Error Assessment Consistency of a JIT-based Virtual Platform Simulator	Published	Chapter 5
2021	LASCAS	The Impact of Precision Bitwidth on the Soft Error Reliability of the MobileNet Network	Published	Section 6.2.2
2021	TCAS-1	Applying Lightweight Soft Error Mitigation Techniques to Embedded Mixed Precision Deep Neural Networks	Published	Section 6.2.4
2022	TCAS-2	The Impact of Soft Errors in Memory Units of Edge Devices Executing Convolutional Neural Networks	Published	Section 6.1.3 and Section 6.2.3
2022	LASCAS	Impact of Thread Parallelism on the Soft Error Reliability of Convolution Neural Networks	Published	Section 6.1.4

Source : The authors

REFERENCES

ABADI, M. et al. Tensorflow: A system for large-scale machine learning. In: **Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation**. USA: USENIX Association, 2016. (OSDI'16), p. 265–283. ISBN 9781931971331.

ABBASITABAR, H.; ZARANDI, H. R.; SALAMAT, R. Susceptibility analysis of LEON3 embedded processor against multiple event transients and upsets. In: **International Conference on Computational Science and Engineering (CSE)**. [S.l.: s.n.], 2012. p. 548–553.

ABICH, G. et al. Evaluation of the soft error assessment consistency of a JIT-based virtual platform simulator. **IET Computers & Digital Techniques**, v. 15, n. 2, p. 125–142, 2021.

ABICH, G. et al. The impact of soft errors in memory units of edge devices executing convolutional neural networks. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 69, n. 3, p. 679–683, 2022.

ABICH, G. et al. Applying lightweight soft error mitigation techniques to embedded mixed precision deep neural networks. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 68, n. 11, p. 4772–4782, 2021.

ABICH, G. et al. Impact of thread parallelism on the soft error reliability of convolution neural networks. In: **IEEE Latin American Symposium on Circuits and Systems (LASCAS)**. [S.l.: s.n.], 2022. p. 1–4.

ABICH, G. et al. Soft error reliability assessment of neural networks on resource-constrained IoT devices. In: **IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. [S.l.: s.n.], 2020. p. 1–4.

ABICH, G.; REIS, R.; OST, L. The impact of precision bitwidth on the soft error reliability of the mobilenet network. In: **IEEE Latin American Symposium on Circuits and Systems (LASCAS)**. [S.l.: s.n.], 2020. p. 1–4.

ADAM, K.; MOHD, I. I.; YOUNIS, Y. M. The impact of the soft errors in convolutional neural network on gpus: Alexnet as case study. **Procedia Computer Science**, v. 182, p. 89–94, 2021. ISSN 1877-0509.

ADI, E. et al. Machine learning and data analytics for the iot. **Neural Computing and Applications**, Springer, v. 32, n. 20, p. 16205–16233, 2020.

AGUIAR, Y. de et al. Evaluation of radiation-induced soft error in majority voters designed in 7 nm finfet technology. **Microelectronics Reliability**, Elsevier, v. 76, p. 660–664, 2017.

AKOPYAN, F. et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 34, n. 10, p. 1537–1557, 2015.

AL-RFOU, R. et al. Theano: A Python framework for fast computation of mathematical expressions. **arXiv e-prints**, abs/1605.02688, may 2016.

AMOH, J.; ODAME, K. M. An optimized recurrent unit for ultra-low-power keyword spotting. **ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies**, v. 3, n. 2, 2019.

ARM. **Arm University Program**. 2020. Available from Internet: <http://www.arm.com/support/university>.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer Networks**, v. 54, n. 15, p. 2787 – 2805, 2010. ISSN 1389-1286.

AVIRNENI, N. D. P.; SOMANI, A. Low overhead soft error mitigation techniques for high-performance and aggressive designs. **IEEE Transactions on Computers**, IEEE, v. 61, n. 4, p. 488–501, 2011.

AVIŽIENIS, A.; LAPRIE, J.-C.; RANDELL, B. Dependability and its threats: a taxonomy. In: **Building the Information Society**. [S.l.: s.n.], 2004. p. 91–120.

AYODELE, T. O. Types of machine learning algorithms. **New advances in machine learning**, InTech, v. 3, p. 19–48, 2010.

AZIZIMAZREAH, A. et al. Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs. In: **2018 IEEE International Conference on Networking, Architecture and Storage (NAS)**. [S.l.: s.n.], 2018. p. 1–10.

BAILEY, D. et al. The NAS parallel benchmarks summary and preliminary results. In: **Conference on Supercomputing (SC)**. [S.l.: s.n.], 1991. p. 158–165.

BANDEIRA, V. et al. Non-intrusive fault injection techniques for efficient soft error vulnerability analysis. In: **2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)**. [S.l.: s.n.], 2019. p. 123–128.

BANERJEE, S. S. et al. Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data. In: **IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)**. [S.l.: s.n.], 2018. p. 586–597.

BARAK, J. et al. A double-power-mosfet circuit for protection from single event burnout. **IEEE Transactions on nuclear science**, IEEE, v. 55, n. 6, p. 3467–3472, 2008.

BARLOW, H. B. Unsupervised learning. **Neural computation**, MIT Press, v. 1, n. 3, p. 295–311, 1989.

BARTH, J.; DYER, C.; STASSINOPOULOS, E. Space, atmospheric, and terrestrial radiation environments. **IEEE Transactions on Nuclear Science**, v. 50, n. 3, p. 466–482, 2003.

BAUMANN, R. Soft errors in advanced computer systems. **IEEE Design & Test of Computers**, v. 22, n. 3, p. 258–266, 2005.

BELKIN, M. et al. Reconciling modern machine-learning practice and the classical bias–variance trade-off. **Proceedings of the National Academy of Sciences**, National Acad Sciences, v. 116, n. 32, p. 15849–15854, 2019.

BELLARD, F. QEMU, a fast and portable dynamic translator. **Proceedings of the Annual Conference on USENIX Annual Technical Conference**, v. 41, n. 46, p. 10–5555, 2005.

BENSO, A. et al. A C/C++ source-to-source compiler for dependable applications. In: **IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)**. [S.l.: s.n.], 2000. p. 71–78.

BINKERT, N. et al. The gem5 simulator. **SIGARCH Computer Architecture News**, v. 39, n. 2, p. 1–7, 2011.

BODMANN, P. et al. Soft error effects on arm microprocessors: Early estimations vs. chip measurements. **IEEE Transactions on Computers**, p. 1–1, 2021.

BORTOLON, F. T. et al. Exploring the impact of soft errors on noc-based multiprocessor systems. In: **International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2018. p. 1–5.

BOSIO, A. et al. A reliability analysis of a deep neural network. In: **2019 IEEE Latin American Test Symposium (LATS)**. [S.l.: s.n.], 2019. p. 1–6.

BREWER, R. M. et al. The impact of proton-induced single events on image classification in a neuromorphic computing architecture. **IEEE Transactions on Nuclear Science**, v. 67, n. 1, p. 108 – 115, 2019.

BRUGUIER, G.; PALAU, J.-M. Single particle-induced latchup. **IEEE transactions on nuclear science**, IEEE, v. 43, n. 2, p. 522–532, 1996.

BUTKO, A. et al. Accuracy evaluation of gem5 simulator system. In: **7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)**. [S.l.: s.n.], 2012. p. 1–7.

CAPOTONDI, A. et al. Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 67, n. 5, p. 871–875, 2020.

CARUANA, R.; NICULESCU-MIZIL, A. An empirical comparison of supervised learning algorithms. In: **Proceedings of the 23rd International Conference on Machine Learning**. New York, NY, USA: Association for Computing Machinery, 2006. (ICML '06), p. 161–168. ISBN 1595933832.

CECCHINEL, C. et al. An architecture to support the collection of big data in the internet of things. In: **2014 IEEE World Congress on Services**. [S.l.: s.n.], 2014. p. 442–449.

CELEBI, M. E.; AYDIN, K. **Unsupervised learning algorithms**. [S.l.]: Springer, 2016.

CHATZIDIMITRIOU, A. et al. Demystifying soft error assessment strategies on ARM CPUs: Microarchitectural fault injection vs. neutron beam experiments. In: **International Conference on Dependable Systems and Networks (DSN)**. [S.l.: s.n.], 2019. p. 26–38.

CHE, S. et al. Rodinia: A benchmark suite for heterogeneous computing. In: **International Symposium on Workload Characterization (IISWC)**. [S.l.: s.n.], 2009. p. 44–54.

CHEN, J.; RAN, X. Deep learning with edge computing: A review. **Proceedings of the IEEE**, v. 107, n. 8, p. 1655–1674, 2019.

CHEN, Y.-H.; EMER, J.; SZE, V. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. **ACM SIGARCH Computer Architecture News**, v. 44, n. 3, p. 367 – 379, 2016.

CHEN, Z.; LI, G.; PATTABIRAMAN, K. A low-cost fault corrector for deep neural networks through range restriction. In: **2021 51th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)**. [S.l.: s.n.], 2021.

CHEN, Z. et al. Binfi: An efficient fault injector for safety-critical machine learning systems. In: **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)**. New York, NY, USA: Association for Computing Machinery, 2019. (SC '19), p. 1–23.

CHO, H. et al. Quantitative evaluation of soft error injection techniques for robust system design. In: **Design Automation Conference (DAC)**. [S.l.: s.n.], 2013. p. 1–10.

CHOLLET, F. et al. Keras: The python deep learning library. **Astrophysics source code library**, p. ascl–1806, 2018.

CIANI, L.; CATELANI, M.; VELTRONI, L. Fault tolerant techniques to diagnose and mitigate single event upset (seu) effects on electronic programmable devices. In: CITESEER. **Proc. of 16th ImEKo TC4 symposium**. [S.l.], 2008.

CISCO. **Cisco Annual Internet Report White Paper**. 2021. <https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report>.

CORNELIOU, P. et al. Fine-grained vulnerability analysis of resource constrained neural inference accelerators. In: IEEE. **2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)**. [S.l.], 2021. p. 1–6.

DAGUM, L.; MENON, R. OpenMP: an industry standard API for shared-memory programming. **IEEE Computational Science and Engineering**, v. 5, n. 1, p. 46–55, 1998.

DAWIT, M.; FRISK, F. Edge machine learning for energy efficiency of resource constrained iot devices. In: **SPWID 2019: The Fifth International Conference on Smart Portable, Wearable, Implantable and Disabilityoriented Devices and Systems**. [S.l.: s.n.], 2019. p. 9–14.

DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: IEEE. **2009 IEEE conference on computer vision and pattern recognition**. [S.l.], 2009. p. 248–255.

DENG, L. The mnist database of handwritten digit images for machine learning research [best of the web]. **IEEE Signal Processing Magazine**, v. 29, n. 6, p. 141–142, 2012.

DEVOE, M. **An Industry-University Development Tools Collaboration**. 2015. 5–5 p. Available from Internet: <https://issuu.com/opensystemsmedia/docs/ecd_november_2015_issuu>.

DIDEHBAN, M.; LOKAM, S. R. D.; SHRIVASTAVA, A. InCheck: An in-application recovery scheme for soft errors. In: **ACM/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2017. p. 1–6.

DIDEHBAN, M.; SHRIVASTAVA, A. nZDC: A compiler technique for near zero silent data corruption. In: **Proceedings of the 53rd Annual Design Automation Conference (DAC)**. [S.l.]: ACM, 2016. (DAC '16), p. 48:1 – 48:6. ISBN 978-1-4503-4236-0.

DIDEHBAN, M.; SHRIVASTAVA, A.; LOKAM, S. R. D. NEMESIS: A software approach for computing in presence of soft errors. In: **IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2017. p. 297–304.

DINH, L.; SOHL-DICKSTEIN, J.; BENGIO, S. Density estimation using real nvp. **arXiv preprint arXiv:1605.08803**, 2016.

DODD, P. et al. Current and future challenges in radiation effects on cmos electronics. **IEEE Transactions on Nuclear Science**, IEEE, v. 57, n. 4, p. 1747–1763, 2010.

DY, J. G.; BRODLEY, C. E. Feature selection for unsupervised learning. **Journal of machine learning research**, v. 5, n. Aug, p. 845–889, 2004.

FENG, S. et al. Shoestring: probabilistic soft error reliability on the cheap. **ACM SIGARCH Computer Architecture News**, v. 38, n. 1, p. 385–396, 2010.

FENG, S. et al. Encore: low-cost, fine-grained transient fault recovery. In: **IEEE/ACM International Symposium on Microarchitecture (MICRO)**. [S.l.: s.n.], 2011. p. 398–409.

FERLET-CAVROIS, V.; MASSENGILL, L. W.; GOUKER, P. Single event transients in digital cmos - a review. **IEEE Transactions on Nuclear Science**, IEEE, v. 60, n. 3, p. 1767–1790, 2013.

FIGUEIREDO, M. A. T.; JAIN, A. K. Unsupervised learning of finite mixture models. **IEEE Transactions on pattern analysis and machine intelligence**, IEEE, v. 24, n. 3, p. 381–396, 2002.

GAROFALO, A. et al. Pulp-nn: A computing library for quantized neural network inference at the edge on risc-v based parallel ultra low power clusters. In: **2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. [S.l.: s.n.], 2019. p. 33–36.

GAROFALO, A. et al. PULP-NN: Accelerating Quantized Neural Networks on Parallel Ultra-Low-Power RISC-V Processors. **Philosophical Transactions of the Royal Society A**, v. 378, n. 2164, p. 1–21, 2020.

GAVA, J.; REIS, R.; OST, L. RAT: A lightweight system-level soft error mitigation technique. In: **IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SOC)**. [S.l.: s.n.], 2020. p. 165–170.

GEISSLER, F. de A.; KASTENSMIDT, F. L.; SOUZA, J. E. P. Soft error injection methodology based on qemu software platform. In: **2014 15th Latin American Test Workshop - LATW**. [S.l.: s.n.], 2014. p. 1–5.

GHOSH, S.; DUBEY, S. K. Comparative analysis of k-means and fuzzy c-means algorithms. **International Journal of Advanced Computer Science and Applications**, Citeseer, v. 4, n. 4, 2013.

GRANAT, R. et al. Simulating and detecting radiation-induced errors for onboard machine learning. In: **2009 Third IEEE International Conference on Space Mission Challenges for Information Technology**. [S.l.: s.n.], 2009. p. 125–131.

GRANLUND, T.; GRANBOM, B.; OLSSON, N. Soft error rate increase for new generations of srams. **IEEE Transactions on Nuclear Science**, IEEE, v. 50, n. 6, p. 2065–2068, 2003.

GUAN, H. et al. In-place zero-space memory protection for cnn. In: **Proceedings of the 33rd International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2019.

GUAN, Q. et al. Design, use and evaluation of P-FSEFI: A parallel soft error fault injection framework for emulating soft errors in parallel applications. In: **Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques**. [S.l.]: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016. (SIMUTOOLS'16), p. 9 – 17. ISBN 978-1-63190-120-1.

GUDIVADA, V.; APON, A.; DING, J. Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. **International Journal on Advances in Software**, v. 10, n. 1, p. 1–20, 2017.

GUPTA, S. et al. Deep learning with limited numerical precision. In: BACH, F.; BLEI, D. (Ed.). **Proceedings of the 32nd International Conference on Machine Learning**. Lille, France: [s.n.], 2015. (Proceedings of Machine Learning Research, v. 37), p. 1737–1746.

GUSTAFSSON, J. et al. The mälardalen WCET benchmarks: Past, present and future. In: **International Workshop on Worst-Case Execution Time Analysis (WCET)**. [S.l.: s.n.], 2010. p. 136–146.

HAN, S.; MAO, H.; DALLY, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. **arXiv preprint arXiv:1510.00149**, 2015.

HANDS, A. et al. Single event effects in power mosfets due to atmospheric and thermal neutrons. **IEEE Transactions on Nuclear Science**, IEEE, v. 58, n. 6, p. 2687–2694, 2011.

HAO, C. et al. Enabling design methodologies and future trends for edge ai: Specialization and codesign. **IEEE Design & Test**, v. 38, n. 4, p. 7–26, 2021.

HARI, S. K. S. et al. Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults. In: **Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems**. [S.l.]: Association for Computing Machinery, 2012. p. 123–134. ISBN 9781450307598.

HARI, S. K. S. et al. Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation. In: **2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)**. [S.l.: s.n.], 2017. p. 249–258.

HARI, S. K. S. et al. Ganges: Gang error simulation for hardware resiliency evaluation. In: **2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)**. [S.l.: s.n.], 2014. p. 61–72.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. Unsupervised learning. In: **The elements of statistical learning**. [S.l.]: Springer, 2009. p. 485–585.

HATHAWAY, R. J.; BEZDEK, J. C. Extending fuzzy and probabilistic clustering to very large data sets. **Computational Statistics & Data Analysis**, v. 51, n. 1, p. 215 – 234, 2006. ISSN 0167-9473. The Fuzzy Approach to Statistical Analysis.

HE, Y.; ZHANG, X.; SUN, J. Channel pruning for accelerating very deep neural networks. In: **Proceedings of the IEEE international conference on computer vision**. [S.l.: s.n.], 2017. p. 1389–1397.

HILD, K. E. et al. Feature extraction using information-theoretic learning. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 28, n. 9, p. 1385–1392, 2006.

HOSTE, K.; EECKHOUT, L. COLE: Compiler optimization level exploration. In: **International Symposium on Code Generation and Optimization (CGO)**. [S.l.: s.n.], 2008. p. 165–174.

HOWARD, A. G. et al. MobileNets: Efficient convolutional neural networks for mobile vision applications. **arXiv preprint arXiv:1704.04861**, 2017.

HUTSON, J. et al. Evidence for lateral angle effect on single-event latchup in 65 nm srams. **IEEE Transactions on Nuclear Science**, IEEE, v. 56, n. 1, p. 208–213, 2009.

IBRAHIM, Y. et al. Soft error resilience of deep residual networks for object recognition. **IEEE Access**, v. 8, p. 19490–19503, 2020.

IMPERAS. **DEV - Virtual Platform Development and Simulation**. 2021. Available from Internet: <https://www.imperas.com/dev-virtual-platform-development-and-simulation/>.

IMPERAS. **Open Virtual Platforms (OVP)**. 2021. Available from Internet: <http://www.ovpworld.org/>.

ISO. **Road vehicles – Functional safety**. 2011. Available from Internet: <https://www.iso.org/standard/68383.html>.

JACOB, B. et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], June 2018. p. 2704–2713.

JAIN, V. et al. Supervised learning of image restoration with convolutional networks. In: **2007 IEEE 11th International Conference on Computer Vision**. [S.l.: s.n.], 2007. p. 1–8.

JASEMI, M.; HESSABI, S.; BAGHERZADEH, N. Enhancing Reliability of Emerging MemoryTechnology for Machine Learning Accelerators. **IEEE Transactions on Emerging Topics in Computing**, p. 1–7, 2020.

JOHNSON, G. et al. A review of the techniques used for modeling single-event effects in power mosfets. **IEEE Transactions on Nuclear Science**, IEEE, v. 43, n. 2, p. 546–560, 1996.

JOHNSTON, A. The influence of vlsi technology evolution on radiation-induced latchup in space systems. **IEEE Transactions on Nuclear Science**, IEEE, v. 43, n. 2, p. 505–521, 1996.

JUDD, P. et al. Reduced-precision strategies for bounded memory in deep neural nets. **arXiv preprint arXiv:1511.05236**, 2015.

KALIORAKIS, M. et al. Differential fault injection on microarchitectural simulators. In: **International Symposium on Workload Characterization (IISWC)**. [S.l.: s.n.], 2015. p. 172–182.

KANUNGO, T. et al. An efficient k-means clustering algorithm: Analysis and implementation. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 24, n. 7, p. 881–892, 2002.

KASTENSMIDT, F.; RECH, P. Radiation effects and fault tolerance techniques for fpgas and gpus. In: **FPGAs and parallel architectures for aerospace applications**. [S.l.]: Springer, 2016. p. 3–17.

KASTENSMIDT, F. L.; CARRO, L.; REIS, R. A. da L. **Fault-tolerance techniques for SRAM-based FPGAs**. [S.l.]: Springer, 2006.

KATO, N. et al. The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective. **IEEE Wireless Communications**, v. 24, n. 3, p. 146–153, 2017.

KHALID, S.; KHALIL, T.; NASREEN, S. A survey of feature selection and feature extraction techniques in machine learning. In: IEEE. **2014 Science and Information Conference**. [S.l.], 2014. p. 372–378.

KHAN, S. et al. Energy-efficient deep cnn for smoke detection in foggy iot environment. **IEEE Internet of Things Journal**, v. 6, n. 6, p. 9237–9245, 2019.

KHOSHAVI, N.; BROYLES, C.; BI, Y. A survey on impact of transient faults on bnn inference accelerators. **arXiv preprint arXiv:2004.05915**, 2020.

KRIEBEL, F. et al. Robustness for smart cyber physical systems and internet-of-things: From adaptive robustness methods to reliability and security for machine learning. In: **2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [S.l.: s.n.], 2018. p. 581–586.

KRIZHEVSKY, A.; HINTON, G. et al. CIFAR-10/100 - Learning multiple layers of features from tiny images. Citeseer, 2009.

KRZYWINSKI, M.; ALTMAN, N. Points of significance: Significance, p values and t-tests. **Nature Methods**, v. 10, n. 11, p. 1041–1042, 2013.

KUNDU, S. et al. Reliability analysis for ML/AI hardware. **arXiv preprint arXiv:2103.12166**, 2021.

KUVAISKII, D. et al. Elzar: Triple modular redundancy using intel avx (practical experience report). In: **IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)**. [S.l.: s.n.], 2016. p. 646–653.

LABRINIDIS, A.; JAGADISH, H. V. Challenges and opportunities with big data. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 5, n. 12, p. 2032 – 2033, 2012. ISSN 2150-8097.

LAI, L.; SUDA, N.; CHANDRA, V. CMSIS-NN: Efficient neural network kernels for arm Cortex-M cpus. **arXiv preprint arXiv:1801.06601**, 2018.

LATTNER, C.; ADVE, V. LLVM: a compilation framework for lifelong program analysis transformation. In: **International Symposium on Code Generation and Optimization (CGO)**. [S.l.: s.n.], 2004. p. 75–86.

LEE, J. et al. Integrating machine learning in embedded sensor systems for internet-of-things applications. In: **2016 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)**. [S.l.: s.n.], 2016. p. 290–294.

LEVEUGLE, R. et al. Statistical fault injection: Quantified error and confidence. In: **Design, Automation and Test in Europe Conference (DATE)**. [S.l.: s.n.], 2009. p. 502–506.

LI, G. et al. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In: **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**. [S.l.: s.n.], 2017. p. 1 – 12.

LI, G.; PATTABIRAMAN, K.; DEBARDELEBEN, N. Tensorfi: A configurable fault injector for tensorflow applications. In: **2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)**. [S.l.: s.n.], 2018. p. 313–320.

LI, H.; OTA, K.; DONG, M. Learning iot in edge: Deep learning for the internet of things with edge computing. **IEEE Network**, v. 32, n. 1, p. 96–101, 2018.

LI, X. et al. A memory soft error measurement on production systems. In: **USENIX Annual Technical Conference**. [S.l.: s.n.], 2007. p. 275–280.

LIBANO, F. et al. On the reliability of linear regression and pattern recognition feedforward artificial neural networks in fpgas. **IEEE Transactions on Nuclear Science**, v. 65, n. 1, p. 288 – 295, 2017.

LIBANO, F. et al. Selective hardening for neural networks in FPGAs. **IEEE Transactions on Nuclear Science**, v. 66, n. 1, p. 216–222, 2019.

LINS, F. M. et al. Register file criticality and compiler optimization effects on embedded microprocessor reliability. **IEEE Transactions on Nuclear Science**, v. 64, n. 8, p. 2179–2187, 2017.

LIU, R. et al. Single event transient and tid study in 28 nm utbb fdsoi technology. **IEEE Transactions on Nuclear Science**, IEEE, v. 64, n. 1, p. 113–118, 2016.

LOVELESS, T. et al. On-chip measurement of single-event transients in a 45 nm silicon-on-insulator technology. **IEEE Transactions on Nuclear Science**, IEEE, v. 59, n. 6, p. 2748–2755, 2012.

LUZA, L. M. et al. Investigating the impact of radiation-induced soft errors on the reliability of approximate computing systems. In: **2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)**. [S.l.: s.n.], 2020. p. 1–6.

MAATEN, L. V. D.; POSTMA, E.; HERIK, J. Van den. Dimensionality reduction: a comparative review. **Journal of Machine Learning Research - JMLR**, v. 10, n. 66-71, p. 13, 2009.

MACHADO, R. S. et al. Comparing performance of C compilers optimizations on different multicore architectures. In: **International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)**. [S.l.: s.n.], 2017. p. 25–30.

MAGNUSSON, P. S. et al. Simics: A full system simulation platform. **Computer**, v. 35, n. 2, p. 50 – 58, 2002. ISSN 0018-9162.

MAHDAVINEJAD, M. S. et al. Machine learning for internet of things data analysis: A survey. **Digital Communications and Networks**, Elsevier, v. 4, n. 3, p. 161 – 175, 2018. ISSN 2352-8648.

MANSOUR, W.; VELAZCO, R. SEU fault-injection in vhdl-based processors: A case study. **Journal of Electronic Testing**, v. 29, n. 1, p. 87–94, 2013.

MARCHISIO, A. et al. Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges. In: **2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [S.l.: s.n.], 2019. p. 553–559.

MARTIN, M. M. K. et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. **SIGARCH Comput. Archit. News**, v. 33, n. 4, p. 92 – 99, 2005. ISSN 0163-5964.

MAVIS, D. G.; EATON, P. H. Soft error rate mitigation techniques for modern microcircuits. In: IEEE. **2002 IEEE International Reliability Physics Symposium. Proceedings. 40th Annual (Cat. No. 02CH37320)**. [S.l.], 2002. p. 216–225.

MCKAY, A. Distribution of the coefficient of variation and the extended" t" distribution. **Journal of the Royal Statistical Society**, JSTOR, v. 95, n. 4, p. 695–698, 1932.

MEDEIROS, G. et al. Evaluation of compiler optimization flags effects on soft error resiliency. In: **Symposium on Integrated Circuits and Systems Design (SBCCI)**. [S.l.: s.n.], 2018. p. 1–6.

MITTAL, S. A survey on modeling and improving reliability of dnn algorithms and accelerators. **Journal of Systems Architecture**, Elsevier, v. 104, p. 101689, 2020. ISSN 1383-7621.

MORENTE-MOLINERA, J. A. et al. Improving supervised learning classification methods using multigranular linguistic modeling and fuzzy entropy. **IEEE Transactions on Fuzzy Systems**, v. 25, n. 5, p. 1078–1089, 2017.

MUKHERJEE, S.; EMER, J.; REINHARDT, S. The soft error problem: an architectural perspective. In: **11th International Symposium on High-Performance Computer Architecture**. [S.l.: s.n.], 2005. p. 243–247.

MUKHERJEE, S. S. et al. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In: **International Symposium on Microarchitecture (MICRO)**. [S.l.: s.n.], 2003. p. 29–40.

MUTUEL, L. H. Appreciating the effectiveness of single event effect mitigation techniques. In: IEEE. **2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC)**. [S.l.], 2014. p. 5B1–1.

NAJAFABADI, M. M. et al. Deep learning applications and challenges in big data analytics. **Journal of Big Data**, Springer, v. 2, n. 1, p. 1, 2015.

NETHERCOTE, N.; SEWARD, J. Valgrind: a framework for heavyweight dynamic binary instrumentation. **ACM Sigplan notices**, v. 42, n. 6, p. 89 – 100, 2007.

NICOLESCU, B.; VELAZCO, R. Detecting soft errors by a purely software approach: method, tools and experimental results. In: **Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2003. p. 57–62.

NORMAND, E. Single-event effects in avionics. **IEEE Transactions on Nuclear Science**, v. 43, n. 2, p. 461–474, 1996.

OH, N.; SHIRVANI, P. P.; MCCLUSKEY, E. J. Error detection by duplicated instructions in super-scalar processors. **IEEE Transactions on Reliability**, v. 51, n. 1, p. 63–75, 2002.

PARASYRIS, K. et al. GemFI: A fault injection tool for studying the behavior of applications on unreliable substrates. In: **International Conference on Dependable Systems and Networks (DSN)**. [S.l.: s.n.], 2014. p. 622–629.

PATEL, A. et al. MARSS: A full system simulator for multicore x86 CPUs. In: **Design Automation Conference (DAC)**. [S.l.: s.n.], 2011. p. 1050–1055.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. **the Journal of machine Learning research**, JMLR. org, v. 12, p. 2825–2830, 2011. Available from Internet: <https://scikit-learn.org/>.

PI, R. **Raspberry Pi 2 Model b**. 2021. Available from Internet: <https://www.raspberrypi.org>.

PING, L.; TAN, J.; YAN, K. SERN: Modeling and analyzing the soft error reliability of convolutional neural networks. In: **Proceedings of the 2020 on Great Lakes Symposium on VLSI**. New York, NY, USA: Association for Computing Machinery, 2020. p. 445–450. ISBN 9781450379441.

PRINZIE, J.; STEYAERT, M.; LEROUX, P. Radiation effects in cmos technology. In: **Radiation Hardened CMOS Integrated Circuits for Time-Based Signal Processing**. [S.l.]: Springer, 2018. p. 1–20.

PUNITHAVATHI, P. et al. A lightweight machine learning-based authentication framework for smart iot devices. **Information Sciences**, v. 484, p. 255 – 268, 2019. ISSN 0020-0255.

QI, X.; LIU, C. Enabling deep learning on iot edge: Approaches and evaluation. In: **2018 IEEE/ACM Symposium on Edge Computing (SEC)**. [S.l.: s.n.], 2018. p. 367–372.

REAGEN, B. et al. Ares: A framework for quantifying the resilience of deep neural networks. In: **2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2018. p. 1–6.

REIS, G. A.; CHANG, J.; AUGUST, D. I. Automatic instruction-level software-only recovery. **IEEE Micro**, v. 27, n. 1, p. 36–47, 2007.

REIS, G. A.; CHANG, J. et al. Software-controlled fault tolerance. **ACM Transactions on Architecture and Code Optimization (TACO)**, 2005.

REIS, G. A. et al. SWIFT: Software implemented fault tolerance. In: **International Symposium on Code Generation and Optimization (CGO)**. [S.l.: s.n.], 2005. p. 243–254.

REPPEN, A. M.; SONER, H. M. Bias-variance trade-off and overlearning in dynamic decision problems. **arXiv preprint arXiv:2011.09349**, 2020.

RODRIGUES, G. S. et al. Analyzing the impact of using pthreads versus OpenMP under fault injection in ARM Cortex-A9 dual-core. In: **European Conference on Radiation and Its Effects on Components and Systems (RADECS)**. [S.l.: s.n.], 2016. p. 1–6.

ROSA, F. et al. A fast and scalable fault injection framework to evaluate multi/many-core soft error reliability. In: **International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)**. [S.l.: s.n.], 2015. p. 211–214.

ROSA, F. et al. Evaluation of multicore systems soft error reliability using virtual platforms. In: **International New Circuits and Systems Conference (NEWCAS)**. [S.l.: s.n.], 2017. p. 85–88.

ROSA, F. da et al. Extensive evaluation of programming models and ISAs impact on multicore soft error reliability. In: **Design Automation Conference (DAC)**. [S.l.: s.n.], 2018. p. 1–6.

ROSA, F. R. da. **Early evaluation of multicore systems soft error reliability using virtual platforms**. Thesis (PhD) — PGMICRO - UFRGS, 6 2018.

ROSA, F. R. da et al. Using machine learning techniques to evaluate multicore soft error reliability. **IEEE Transactions on Circuits and Systems-I: Regular papers**, v. 66, n. 6, p. 2151–2164, 2019.

RUSCI, M.; CAPOTONDI, A.; BENINI, L. Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers. **arXiv preprint arXiv:1905.13082**, 2019.

SALAMI, B.; UNSAL, O. S.; KESTELMAN, A. C. On the resilience of rtl nn accelerators: Fault characterization and mitigation. In: **2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)**. [S.l.: s.n.], 2018. p. 322–329.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. **IBM Journal of Research and Development**, v. 3, n. 3, p. 210–229, 1959.

SANGCHOOLIE, B. et al. A study of the impact of bit-flip errors on programs compiled with different optimization levels. In: **European Dependable Computing Conference (EDCC)**. [S.l.: s.n.], 2014. p. 146–157.

SANTOS, F. F. dos et al. Impact of reduced precision in the reliability of deep neural networks for object detection. In: **2019 IEEE European Test Symposium (ETS)**. [S.l.: s.n.], 2019. p. 1–6.

SANTOS, F. F. dos et al. Analyzing and increasing the reliability of convolutional neural networks on gpus. **IEEE Transactions on Reliability**, v. 68, n. 2, p. 663–677, 2018.

SCHIRMEIER, H.; BORCHERT, C.; SPINCZYK, O. 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**. Rio de Janeiro, RJ, Brazil: [s.n.], September 2015. p. 319–330.

SCHIRMEIER, H.; BREDDEMANN, M. Quantitative cross-layer evaluation of transient-fault injection techniques for algorithm comparison. In: **European Dependable Computing Conference (EDCC)**. Naples, Italy: [s.n.], September 2019. p. 15–22.

SCHWANK, J. et al. Effects of particle energy on proton-induced single-event latchup. **IEEE Transactions on Nuclear Science**, IEEE, v. 52, n. 6, p. 2622–2629, 2005.

SCHWANK, J. R. et al. Effects of angle of incidence on proton and neutron-induced single-event latchup. **IEEE transactions on nuclear science**, IEEE, v. 53, n. 6, p. 3122–3131, 2006.

SEIFERT, N. et al. Soft error susceptibilities of 22 nm tri-gate devices. **IEEE Transactions on Nuclear Science**, v. 59, n. 6, p. 2666–2673, 2012.

SERRANO-CASES, A. et al. Non-intrusive automatic compiler-guided reliability improvement of embedded applications under proton irradiation. **IEEE Transactions on Nuclear Science**, v. 66, n. 7, p. 1500–1509, 2019.

SHANTHAMALLU, U. S. et al. A brief survey of machine learning methods and their sensor and iot applications. In: **2017 8th International Conference on Information, Intelligence, Systems Applications (IISA)**. [S.l.: s.n.], 2017. p. 1–8.

SMYTH, P.; WOLPERT, D. Stacked density estimation. **Advances in neural information processing systems**, v. 10, p. 668–674, 1997.

SOLIMAN, K.; NICHOLS, D. K. Latchup in cmos devices from heavy ions. **IEEE Transactions on Nuclear Science**, IEEE, v. 30, n. 6, p. 4514–4519, 1983.

SONAGARA, D.; BADHEKA, S. Comparison of basic clustering algorithms. **International Journal of Computer Science and Mobile Computing**, v. 3, n. 10, p. 58–61, 2014.

SONG, L. et al. COMP: Compiler optimizations for manycore processors. In: **International Symposium on Microarchitecture**. [S.l.: s.n.], 2014. p. 659–671.

STMICROELECTRONICS. **AI expansion pack for STM32CubeMX**. 2020. Available from Internet: <https://www.st.com/en/embedded-software/x-cube-ai.html>.

SUN, D.; LIU, S.; GAUDIOT, J.-L. Enabling embedded inference engine with arm compute library: A case study. **arXiv preprint arXiv:1704.03751**, 2017.

SZYDLO, T.; SENDOREK, J.; BRZOZA-WOCH, R. Enabling machine learning on resource constrained devices by source code generation of the learned models. In: **International Conference on Computational Science**. [S.l.]: Springer, 2018. p. 682–694.

TABANELLI, E.; TAGLIAVINI, G.; BENINI, L. **DNN is not all you need: Parallelizing Non-Neural ML Algorithms on Ultra-Low-Power IoT Processors**. 2021. Available from Internet: <https://arxiv.org/abs/2107.09448>.

TABER, A.; NORMAND, E. Single event upset in avionics. **IEEE Transactions on Nuclear Science**, IEEE, v. 40, n. 2, p. 120–126, 1993.

TANIKELLA, K. et al. gemV: A validated toolset for the early exploration of system reliability. In: **2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)**. [S.l.: s.n.], 2016. p. 159 – 163.

TINY-DNN. **Tiny-DNN Framework**. 2017. Available from Internet: <https://github.com/tiny-dnn/tiny-dnn>.

TITUS, J. et al. Effect of ion energy upon dielectric breakdown of the capacitor response in vertical power mosfets. **IEEE Transactions on Nuclear Science**, IEEE, v. 45, n. 6, p. 2492–2499, 1998.

TITUS, J. L. An updated perspective of single event gate rupture and single event burnout in power mosfets. **IEEE Transactions on nuclear science**, IEEE, v. 60, n. 3, p. 1912–1928, 2013.

TORO, D. G. et al. Study of a cosmic ray impact on combinatorial logic circuits of an 8bit sar adc in 65nm cmos technology. In: **2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)**. [S.l.: s.n.], 2013. p. 241–244.

TRENTIN, E.; FRENO, A. Unsupervised nonparametric density estimation: A neural network approach. In: IEEE. **2009 International Joint Conference on Neural Networks**. [S.l.], 2009. p. 3140–3147.

TRINDADE, M. G. et al. Assessment of machine learning algorithms for near-sensor computing under radiation soft errors. In: **IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. [S.l.: s.n.], 2020. p. 1–4.

TRINDADE, M. G. et al. Assessment of a hardware-implemented machine learning technique under neutron irradiation. **IEEE Transactions on Nuclear Science**, v. 66, n. 7, p. 1441 – 1448, 2019.

TSUGE, S. et al. Dimensionality reduction using non-negative matrix factorization for information retrieval. In: **2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat.No.01CH37236)**. [S.l.: s.n.], 2001. v. 2, p. 960–965 vol.2.

UMUROGLU, Y. et al. Finn: A framework for fast, scalable binarized neural network inference. In: **Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays**. [S.l.]: Association for Computing Machinery, 2017. (FPGA '17), p. 65 – 74. ISBN 9781450343541.

VALENTINI, G.; DIETTERICH, T. G. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. **Journal of Machine Learning Research**, v. 5, n. Jul, p. 725–775, 2004.

VARGAS, F.; NICOLAIDIS, M. Seu-tolerant sram design based on current monitoring. In: IEEE. **Proceedings of IEEE 24th International Symposium on Fault-Tolerant Computing**. [S.l.], 1994. p. 106–115.

WEINBERGER, K. Q.; SHA, F.; SAUL, L. K. Learning a kernel matrix for nonlinear dimensionality reduction. In: **Proceedings of the Twenty-First International Conference on Machine Learning**. New York, NY, USA: Association for Computing Machinery, 2004. (ICML '04), p. 106. ISBN 1581138385.

WIRTH, G.; KASTENSMIDT, F. L.; RIBEIRO, I. Single event transients in logic circuits—load and propagation induced pulse broadening. **IEEE Transactions on Nuclear Science**, IEEE, v. 55, n. 6, p. 2928–2935, 2008.

WROBEL, T. F. et al. Current induced avalanche in epitaxial structures. **IEEE Transactions on Nuclear Science**, v. 32, n. 6, p. 3991–3995, 1985.

WU, H. et al. Integer quantization for deep learning inference: Principles and empirical evaluation. **arXiv preprint arXiv:2004.09602**, 2020.

XIE, S.; LIU, Y. Improving supervised learning for meeting summarization using sampling and regression. **Computer Speech & Language**, v. 24, n. 3, p. 495 – 514, 2010. ISSN 0885-2308.

XU, H.; LI, G. Density-based probabilistic clustering of uncertain data. In: IEEE. **2008 International Conference on Computer Science and Software Engineering**. [S.l.], 2008. v. 4, p. 474–477.

ZHANG, Y. et al. Parallel DNN inference framework leveraging a compact risc-v isa-based multi-core system. In: **Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining**. [S.l.: s.n.], 2020. p. 627–635.

ZIEGLER, J. F. Terrestrial cosmic rays. **IBM Journal of Research and Development**, v. 40, n. 1, p. 19–39, 1996.

APPENDIX A — SOFT ERROR CONSISTENCY ASSESSMENT FOR MULTI-CORE PROCESSORS

Given the growing complexity of both application and software stacks, most computing systems consider multi-core processors. Differently from single-core, in multi-core processor architectures, the number of cores and the parallel programming model have a direct impact in terms of performance, power efficiency, and reliability. According to the previous section, the software stack complexity, as well as the instruction set, slightly affects the accuracy of the soft error evaluation of SOFIA w.r.t. an RTL approach. Unfortunately, RTL descriptions of multi-core processors are not freely available to the public, and for that reason, this section considers the gem5 simulator (BINKERT et al., 2011) as the reference model.

In this scenario, this section evaluates the soft error consistency assessment of SOFIA using gem5-FIM as a reference model. Appendix A.1 details the adopted experimental setup. Next, Appendix A.2 presents the simulation speedup improvement achieved by SOFIA, detailing some virtual platform parameters. Then, Appendix A.3 presents the assessment of soft error consistency considering three main aspects: number of cores (Appendix A.3.1), different parallel programming models (Appendix A.3.2), and different VP parameters that directly impacts on the soft error resilience (Appendix A.3.3). Additionally, it is important to mention that the results used in this section have been conducted in (ROSA et al., 2017) and (ROSA, 2018).

A.1 Experimental Setup

Table A.1 presents the proposed experimental setup used to measure the soft error analysis consistency between the two VPs. The target processor is an Arm Cortex-A9 processor (ARMv7-A architecture) because it can be configured to use one, two, or four cores.

The proposed experimental setup adopts two distinct workloads: the Rodinia benchmark suite (CHE et al., 2009) and the NAS Parallel Benchmarks (NPB) (BAILEY et al., 1991). The two benchmarks provide a set of applications that use different parallel programming models (i.e., OpenMP, MPI, CUDA, and OpenCL) designed to assess the performance of parallel supercomputers. This experimental setup considers 16 OpenMP

Table A.1 – gem5 vs. SOFIA Experimental Setup.

Processors	Arm Cortex-A9 with 1, 2, and 4 cores
Benchmarks	Rodinia (CHE et al., 2009) and NPB (BAILEY et al., 1991)
Number of Applications	27
Programming Models	Serial, MPI, and OpenMP
M*DEV Quantum Sizes	448,000, 4,480, 448, and 44
Injections per Campaign	8,000
Number of FI Campaigns	526
Total Fault Injections	1,248,000 (Appendix A.3.1) and 1,072,000 (Appendix A.3.2) and 1,888,000 (Appendix A.3.3)

Source : The Authors

Rondinia applications: (A) backprop, (B) BFS (Breadth-First Search) , (C) heartwall, (D) hotspot, (E) hotspot3d, (F) kmeans, (G) lavaMD, (H) lud, (I) myocyte, (J) nn (Nearest Neighbours), (K) nw (Needleman-Wunsch), (L) particle filter, (M) pathfinder, (N) sradv1, (O) sradv2, and (P) stream cluster. In addition, we also selected 11 NPB applications (from Serial, MPI, and OpenMP programming models): (BT) Block Tri-diagonal solver, (CG) Conjugate Gradient, (DC) Data Cube, (DT) Data Trafic, (EP) Embarrassingly Parallel, (FT) Discrete 3D fast Fourier Transform, (IS) Integer Sort, (LU) Lower-Upper Gauss-Seidel solver, (MG) Multi-Grid on a sequence of meshes, (SP) Scalar Penta-diagonal solver, and (UA) Unstructured Adaptive mesh.

To avoid external influences and ensure the most solid comparison between virtual platforms, the software stack of both uses the same compilation environment regarding compiler (*GCC 6.2.0*), optimization flag (*-O3*), libraries, and target an identical Linux kernel (*version 3.13.0-rc2*). Operating system reliability is not the main focus of this section and, therefore, fault injections only occur during the application lifespan (i.e., the OS startup is not subject to faults). Nevertheless, the operating system calls that arise during this period (i.e., application execution time) are susceptible to fault injections as part of the application's behavior.

A.2 FI Simulation Performance of SOFIA w.r.t. gem5-FIM

Simulation speedup is one of the main reasons for adopting VPs, however, their engines make them different from each other, providing more or less accuracy according to their performance. For example, gem5 is an event-based cycle-accurate simulator, i.e.,

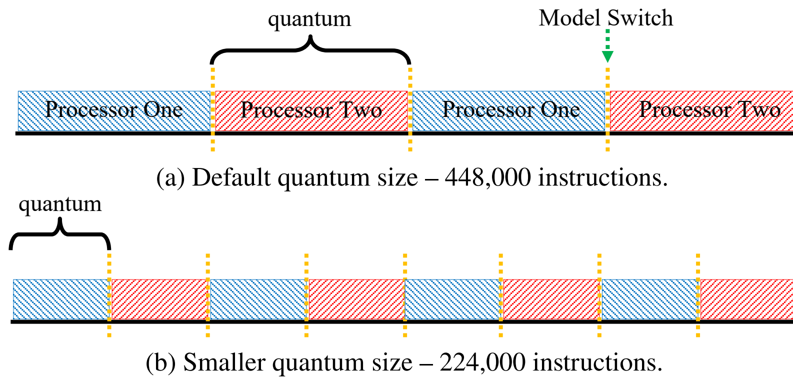
it describes the target microarchitecture as components (e.g., register-file, pipeline, and cache) interconnected by a series of events. A scheduler in the gem5 engine executes these events at each simulation tick, updating the whole system state. One tick corresponds to 1 picosecond; therefore, for a 2 GHz CPU clock, events are executed at a rate of 500 ticks per CPU cycle in the simulated system and, consequently, a complete instruction requires a few thousand ticks (i.e., the resulting clock cycle accuracy at the expense of higher computation and memory cost).

Due to the high simulation speed, typically at hundreds of MIPS, virtual platform simulators based on JIT dynamic binary translation, such as M*DEV, appear to have an advantage over event-driven simulators since it translates the target ISA (e.g., ARMv7-A) to host x86-64 instructions. Further, a complete instruction is the M*DEV minimal simulation granularity; in other words, the simulation always advances one instruction, which provides a higher simulation speed than gem5. Similar to an OS scheduler in which several processes share the same CPU time, the M*DEV engine simulates each model instance (i.e., processor, core, peripheral) for a fixed-length instruction block called *Quantum*. The quantum size is configurable using a variable *time-slice*, representing a time in seconds that refers to an internal configuration parameter and not to the simulation, host, or real-time. The quantum size is given by Equation (A.1), where, by default, the time-slice is 0.001 seconds (one millisecond) and the target processor's nominal MIPS rate is 448 MIPS, resulting in a quantum size of 448,000 instructions, being our reference size in Table A.1.

$$\textit{Quantum Size} = \textit{(processor nominal MIPS rate)} \times 1e^{6} \times \textit{(time slice duration)} \qquad \text{(A.1)}$$

The M*DEV also deploys a scheduling policy to manage the simulation of processors and other components. For example, the simulator selects the first processor, after it has been simulated for 448,000 instructions, it is suspended, and the next processor assumes. In the case of multi-core processors, such as Arm Cortex-A9x2, each processor core receives a separate quantum and is scheduling accordingly. Figure A.1 shows two simulation scenarios for a dual-core processor, one with the default quantum and the other using half of its size (i.e., 224,000 instructions). By reducing the quantum size, the number of model switches for an identical workload increases. This configuration choice delays inter-core communication (or synchronization events) and consequently reduces

Figure A.1 – M*DEV scheduling policy varying the quantum size for a dual-core processor executing the same workload.



(a) Default quantum size – 448,000 instructions.

(b) Smaller quantum size – 224,000 instructions.

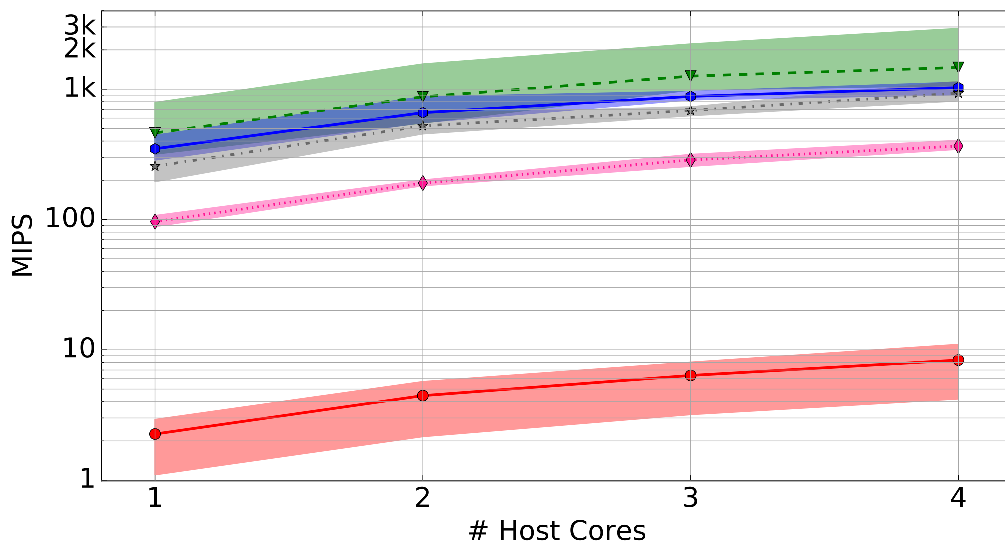
Source : Adapted from (ROSA, 2018)

their simulation speedup.

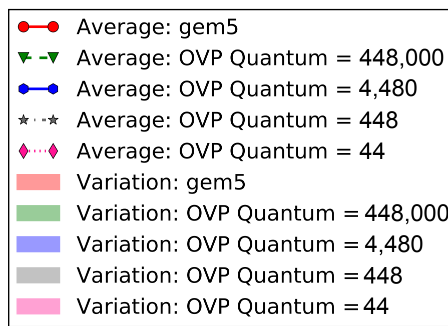
To quantify the simulation speedup difference between gem5-FIM and SOFIA, we analyzed multiple configurations and workloads on a Quad-core Intel Core I7-7700K 4.2 GHz with 16 GB DDR4 2400 MHz. Figure A.2 presents the results from 1 to 4 cores, showing the scalability and speedup of supported parallel simulations by the two VPs. Figure A.2.a displays the first set of simulations considering the Rodinia applications. In this experiment, using four cores, the gem5-FIM atomic simulation speedup ranges from 4.2 to 11 MIPS (Figure A.2.a), while the SOFIA ranges from 345 to 2,921 MIPS depending on the quantum size and application. Note that the reduction in quantum size decreases the number of instructions executed per block, directly affecting the simulator performance due to the increasing switching between the models (i.e., cores).

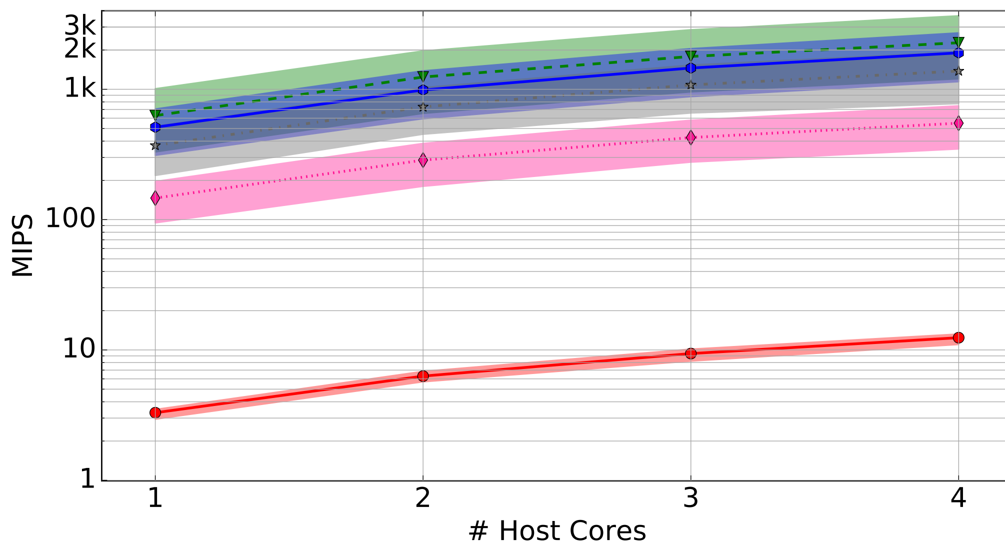
In addition, Figure A.2.b presents NPB applications, which are larger than Rodinia's, reaching up to 87 billion instructions. Figure A.2.b shows a better performance of SOFIA in all configurations while the gem5-FIM atomic remains stable. Looking at the scenario with four cores, the longest workload reaches 12.5 MIPS using atomic gem5. On the other hand, SOFIA reaches 3,910 MIPS, approximately 312 times faster. The SOFIA simulation speedup increases as the application grows due to the just-in-time engine algorithm, thus benefiting from larger applications. For example, comparing the larger and smaller applications, the simulation speedup ranges from 1,190 to 3,910 MIPS (i.e., an increase of 3.28 times) where the gem5 atomic difference is less than 20%, ranging from 10 to 12 MIPS.

Figure A.2 – Simulation speedup and scalability of the two virtual platforms, showing the performance gain achieved by using the SOFIA.



(a) Rodinia





(b) NPB

Source : Adapted from (ROSA, 2018)

A.3 Soft Error Mismatch Analysis

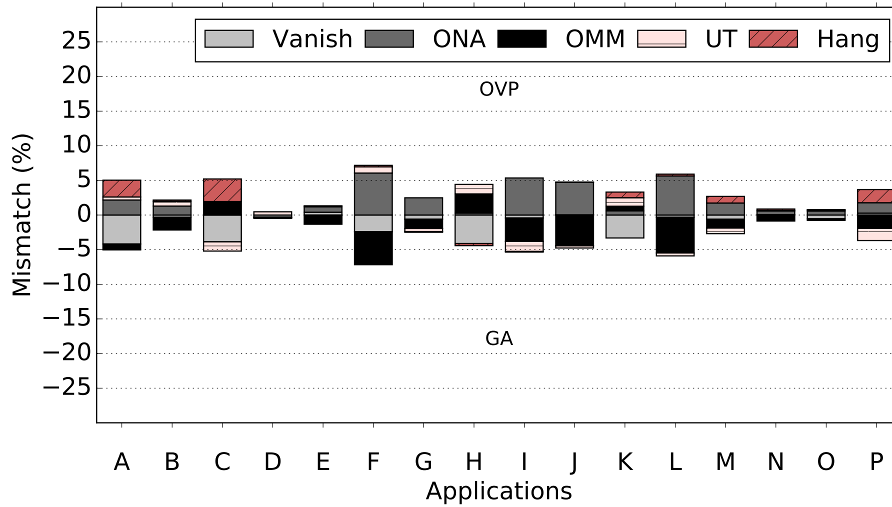
This section presents the assessment of soft error resilience in multi-core processors comparing SOFIA and gem5-FIM. First, we assess the impact due to the number of cores (Appendix A.3.1). Then, we investigate which programming model would be the most reliable (Appendix A.3.2). Finally, in Appendix A.3.3, we discuss how SOFIA configuration (i.e., quantum size) affects the soft error resilience.

A.3.1 Mismatch Analysis Considering the Number of Cores

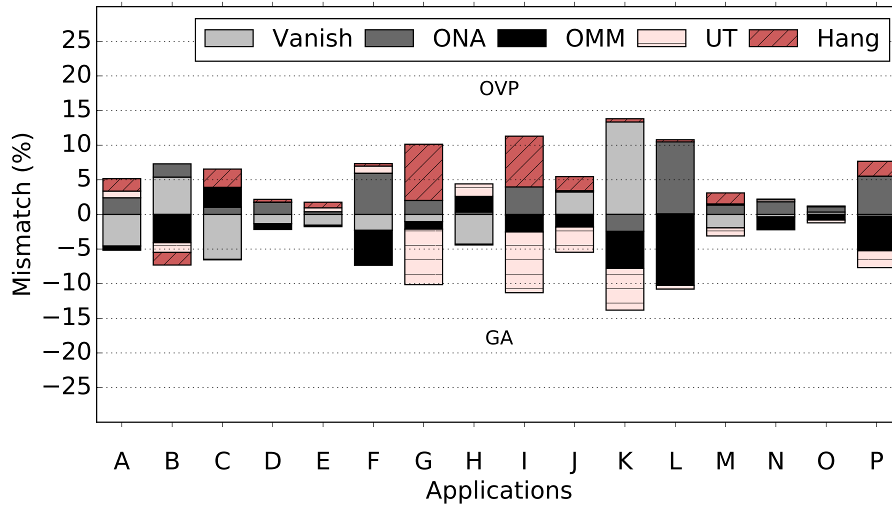
In the last decade, multi-core architectures have been gaining prominence in several semiconductor sectors, being found today in cars, medical, and consumer electronic devices. Due to its importance, engineers must understand how this architectural choice affects the system's reliability. With this intention, we analyzed the Arm Cortex-A9 for the Rodinia and NPB benchmark suites. For simplicity, this analysis considers only OpenMP-based applications; the difference between the programming models is analyzed in Appendix A.3.2.

Figure A.3 and Figure A.4 presents a detailed multi-core mismatch between the SOFIA and the gem5-FIM atomic (GA), considering Rodinia and NPB applications respectively. Concerning the two benchmarks, Rodinia applications execute on average 80 million instructions, while NPB applications execute on average 17 billion instructions (i.e., 212x larger). The longer NPB execution reduces the probability of *ONA* due to a higher likelihood of a bit masking when compared to Rodinia applications. This behavior is seen in Rodinia applications that reach more than 5% of *ONA* mismatch (F, I, L, and P). Further, Rodinia applications have a higher number of *Hangs* than NPB applications, increasing notably with the number of cores. The two possible causes are: (*i*) the fault affected a loop statement (e.g., while, for) where a longer execution translates to more significant recovery time; and (*ii*) kernel malfunctions: the fault injection leads to unrecoverable kernel perturbations (e.g., a thread scheduler error). A longer execution time reduces the Linux kernel exposure time (i.e., the probability of kernel function be stroke by a fault). In other words, the longer the applications, proportionally, the fewer kernel functions are executed. In short, longer workloads reduce overall mismatch. NPB applications' average mismatch varies from 1.32% to 1.68%, in contrast to Rodinia, which ranges from 1.39% and 2.63%. Further, the worst-case mismatch between the gem5-FIM

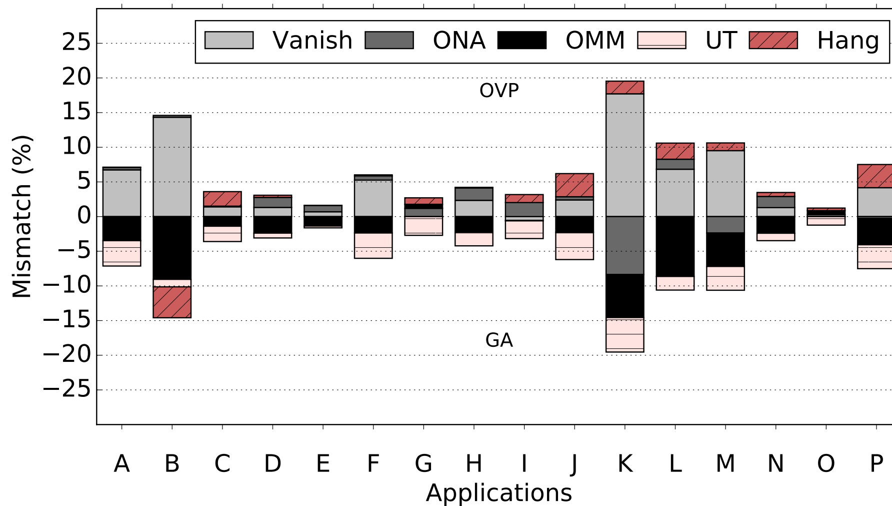
Figure A.3 – Mismatch between gem5-FIM and SOFIA varying the number of cores in Arm Cortex-A9 while executing the Rodinia Benchmarks.



(a) Single-core



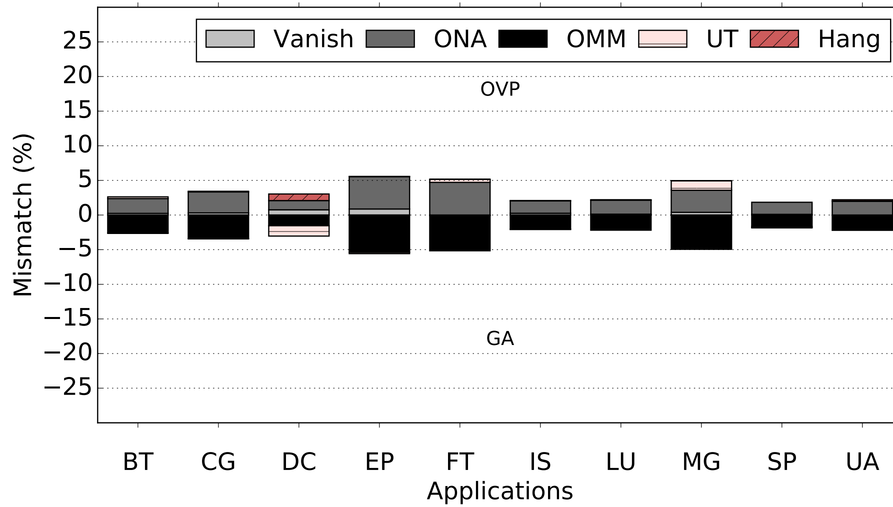
(b) Dual-core



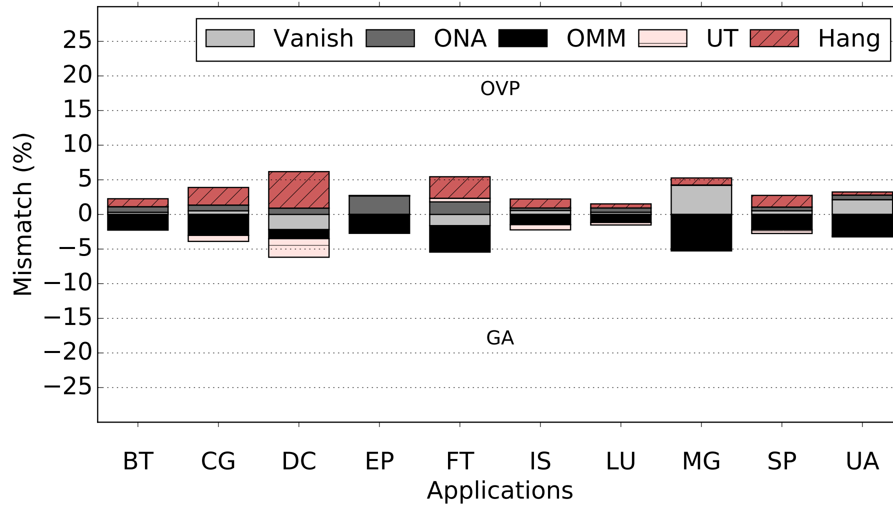
(c) Quad-core

Source : The Authors

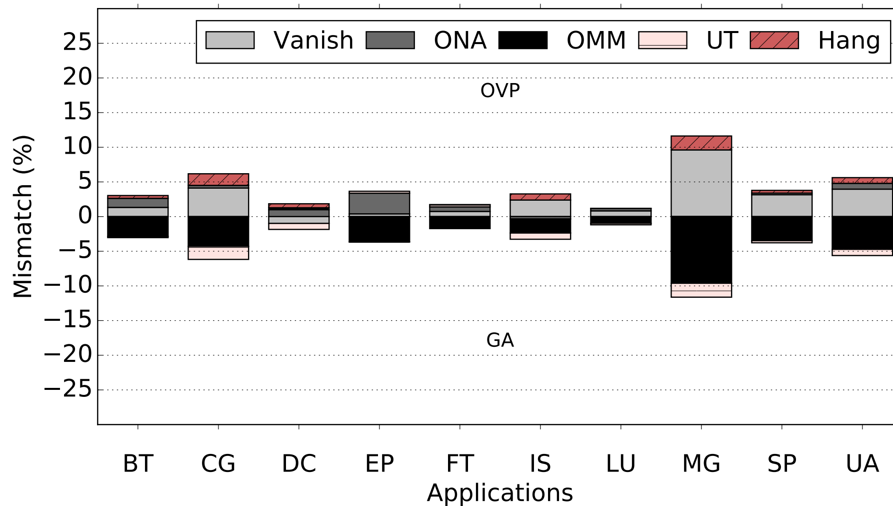
Figure A.4 – Mismatch between gem5-FIM and SOFIA varying the number of cores in Arm Cortex-A9 while executing NPB Benchmarks.



(a) Single-core



(b) Dual-core



(c) Quad-core

Source : The Authors

and SOFIA is reduced to up to 55% for NPB applications.

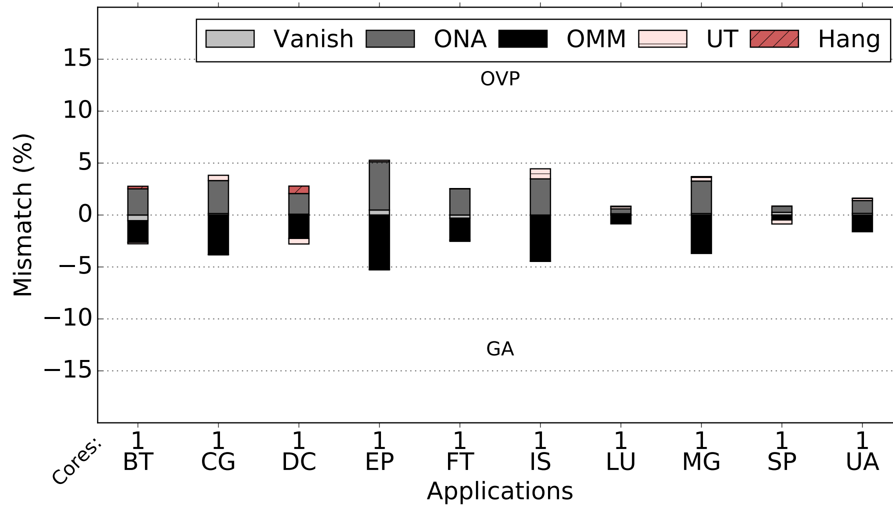
Analyzing the presence of multi-core architectures, applications show a worsening mismatch while increasing the number of cores, most notably in the Rodinia applications B, D, and K (Figure A.3) and in the NPB applications MG and CG (Figure A.4). In the worst-cases, Rodinia's mismatch increases to 17.71% using quad-core compared to 6.06% using single-core processors. In addition, the average error grows from 1.39% to 2.51% considering one and two cores and remains stable for four cores with a mismatch of 2.63%, increasing the core count results in more thread context switching. For Rodinia applications, this behavior combined with sub-linear scalability (i.e., underutilized cores) leads to further errors in the kernel. These kernel errors occur because, during CPU downtime, the OS executes the scheduler algorithm by running one application at a time (i.e., no other threads are running), and then moves to a sleep mode (i.e., *waiting for the interruption*). Furthermore, the number of *Vanish* increases in multi-core architectures (2 and 4 cores). This difference can be attributed to the inter-core communications. Due to its instruction-accurate engine, the SOFIA simulation time (i.e., the number of instructions executed) is affected by the running application characteristics (Appendix A.3.3 details this behavior by modifying parameters on the SOFIA framework).

A.3.2 Mismatch Analysis Considering Parallel Programming Models

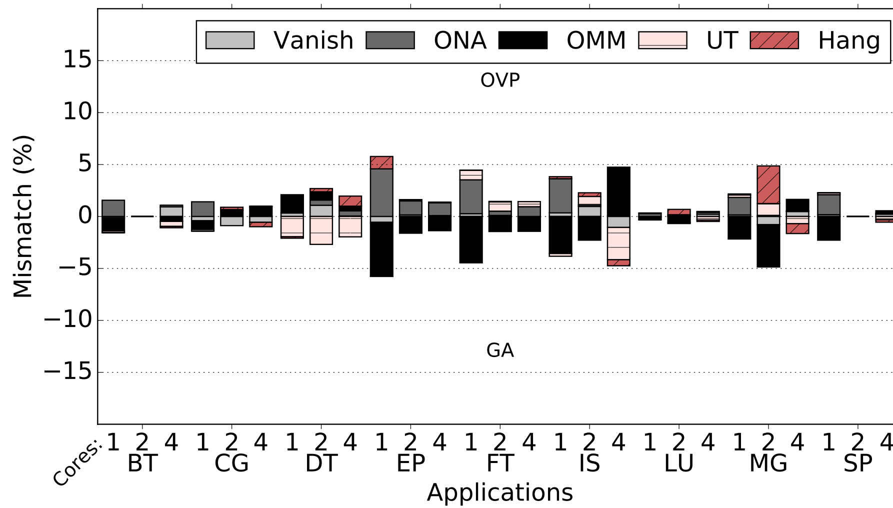
The emerging use of multi-core processors requires specialized libraries that include additional complexity in the soft error assessment. Regarding this distinction, this section considers NPB applications to assess the mismatch between our reference *Serial* programming model on a single-core processor with the *OpenMP* and *MPI* programming models based on multi-core architectures. The OpenMP library uses a series of forks and joins approaches to parallel loop statements, in which the API automatically creates children threads, being suitable for shared memory. On the other hand, the MPI standard is adequate for distributing systems due to the use of a message-oriented parallelization technique, which requires direct parallelization of the user in relation to the creation and communication of threads.

The introduction of parallel programming models increases the software stack, which makes some components more critical to the system's correct behavior. For example, injecting faults into a thread scheduling function has a potentially more hazardous effect on system reliability than a purely arithmetic code portion. By comparing the active

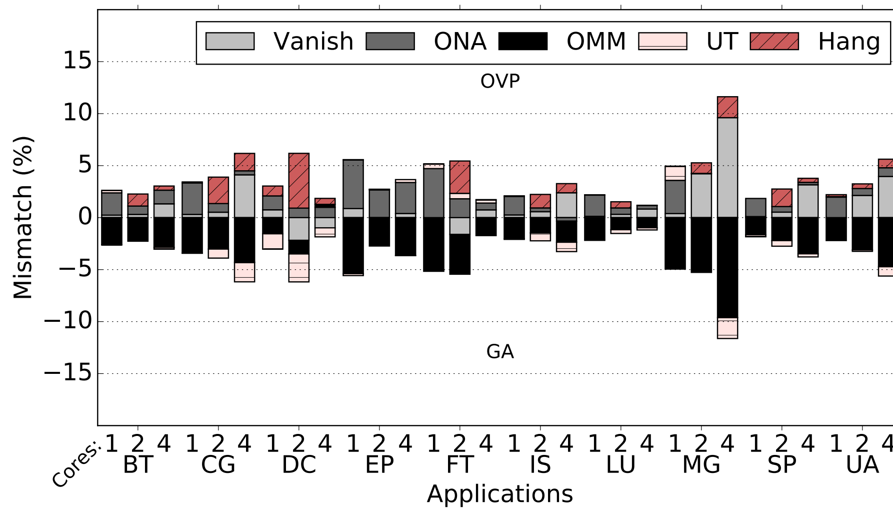
Figure A.5 – Mismatch between gem5-FIM and SOFIA considering different programming models.



(a) Serial



(b) MPI



(c) OpenMP

Source : The Authors

periods of these critical functions with the application's execution time, it is possible to define a time interval called the *vulnerability window*, which varies with the number of calls and executions of the function. In this sense, the use of the NBP benchmark suite provides a real high-performance workload, allowing a more accurate assessment of the impact of the OpenMP and MPI libraries on the system's reliability. Due to its reduced vulnerability window, the parallelization mechanism has a limited effect on the final reliability assessment, less than 23% in the worst-case. To assess this reliability, Figure A.5 shows FI campaigns and mismatches comparing MPI and OpenMP applications.

First, we compare the *Serial* implementation with the two parallelization libraries, considering a single-core processor. The purpose is to assess how each software stack affects the susceptibility to soft errors. For each application, we assess 8,000 FI campaigns, which means that our results have a confidence level of 95% and a margin of error of 1.1%. Comparing with both parallel programming models, no significant variation was found for single-core execution, i.e., the fault distribution is within the margin of error in most applications. However, some applications follow the same pattern, while BT, CG, and IS have fewer soft errors in the *Serial* implementation; EP, FT, and SP have fewer soft errors when increasing the software stack using either of the two parallel programming models.

On the other hand, when we compare the two parallel programming models in a multi-core system, we see that out of 27 possible scenarios between the MPI (Figure A.5.b) and OpenMP (Figure A.5.c), in 22 the MPI has a higher masking rate (i.e., executions without any errors). This is due to two main reasons: *First*, MPI applications have a better workload balance among the used cores, i.e., the number of executed instructions per core is very similar. For instance, the average difference concerning executed instructions per core is around 4% for both ARMv7-A and ARMv8-A considering MPI applications, while the OpenMP variation reaches up 16%. As the OpenMP does not fully utilize the available cores due to the fork/join parallelization approach where a loop statement executes in parallel and other code portions hastily, corroborating the results presented in (ROSA et al., 2018). By contrast, the MPI has individual and independent working threads for each running core providing a better workload balance during its execution. Whenever a core is sub-utilized, it executes a thread scheduling policy and, when no thread is suitable, the core waits in a sleep mode. Consequently, the kernel's relative exposure time and its probability of suffering a transient fault increase, as the scheduling is more often executed. *Second*, OpenMP benchmarks have a shorter execution time, 16% on average, compared against the MPI applications. By consequence, this reduces the

vulnerability window of the MPI inner-functions when comparing against the OpenMP. Further, the longer execution increases the chance of the injected fault being erased due to the software and microarchitectural masking mechanisms. These results show that MPI should be prioritized over OpenMP to improve the reliability of multi-core systems.

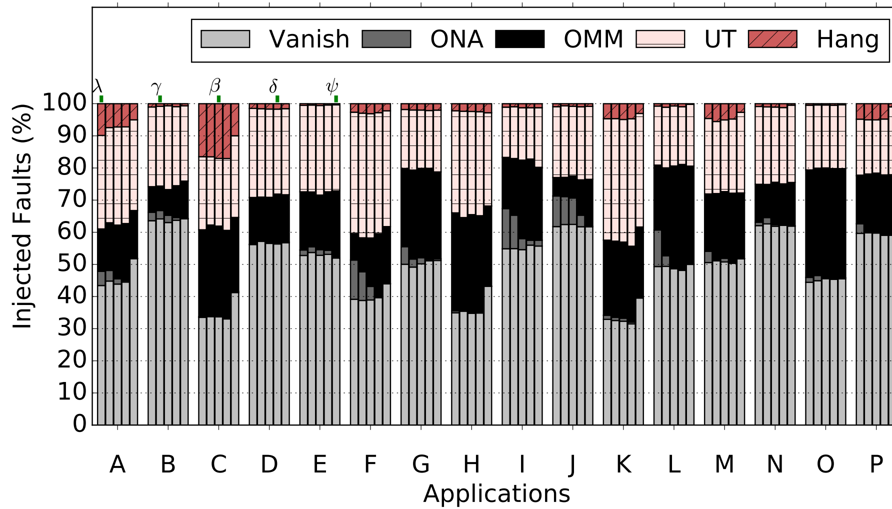
A.3.3 Mismatch Analysis Considering the SOFIA Quantum Parameter

This section explores the impact of using distinct quantum-sized configurations (i.e., 448,000, 4,480, 448, and 44 instructions per block) in the SOFIA to assess the soft error reliability of single and multi-core processors. This analysis provides the trade-off between the performance discussed in Section A.2 with the reliability accuracy brought by each configuration, making it easier for engineers to understand and choose the best configuration for their purposes.

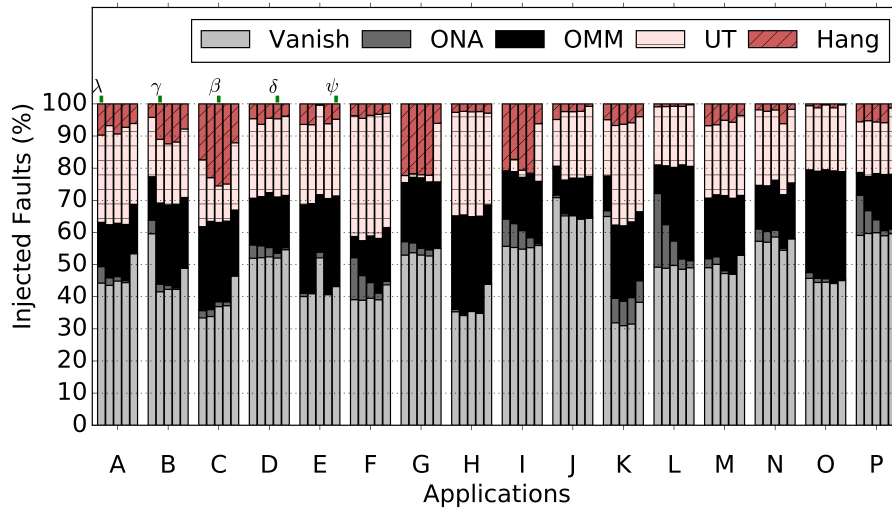
Figure A.6 shows the reference FI framework, the gem5-FIM atomic (ψ), compared to SOFIA using four quantum sizes 448,000 (λ), 4,480 (γ), 448 (β), and 44 (δ) instructions per block for single-, dual-, and quad-cores in an Arm Cortex-A9, respectively. Figure A.6 shows that the variation in quantum size affects the accuracy of the soft error results of SOFIA. As expected, the best performing configuration (i.e., λ – first column of each application shown in Figure A.6) is the one with the greatest mismatch with the results provided by the reference (i.e., ψ – last column of each application shown in Figure A.6). On the other hand, the one with the lowest performance of SOFIA (i.e., δ), which is at least 31× faster than the reference in quad-core simulations according to Section A.2, presents the closest values. For example, the quad-core processor model (Figure A.6.c) has an average improvement of 40% in the results consistency when using the smallest block (δ) instead of the largest block (λ) in the applications that have a large mismatch with the reference results (e.g., B and K). Note that reducing the quantum size decreases the communication cycles between cores, approaching the SOFIA and gem5-FIM behaviors. Another behavior is the migration from *ONA* to *OMM* by decreasing the quantum size, in other words, the incorrect content previously restricted to the register file migrates to the end of the memory.

The resulting mismatch shown in Figure A.6 can be traced back to its block-based simulation engine, as discussed in Section A.2, where each core executes a fixed amount of instructions before moving to the next one. Note that inter-core communications are completed during the core switch, leading to temporally unsynchronized cores. Inter-core

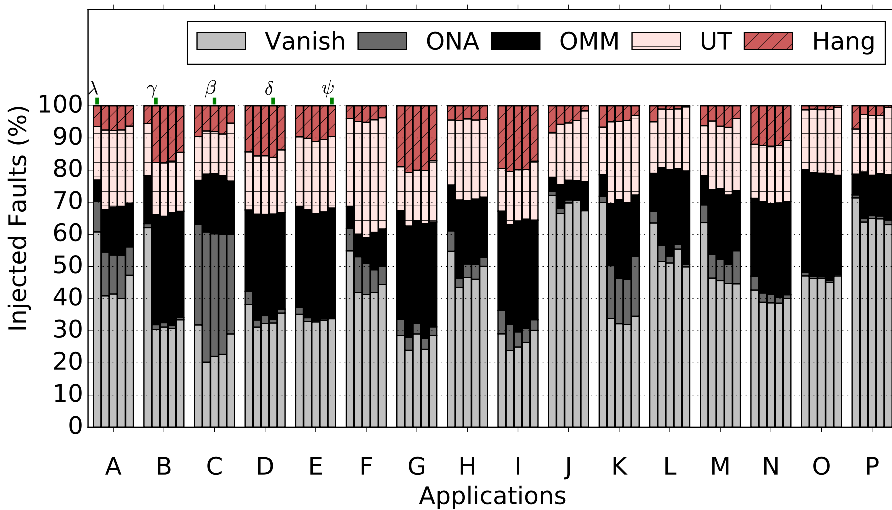
Figure A.6 – Mismatch between the gem5-FIM (ψ) and SOFIA quantum sizes: 448,000 (λ); 4,480 (γ); 448 (β); and 44 (δ).



(a) Single-core



(b) Dual-core



(c) Quad-core

Source : Adapted from (ROSA, 2018)

communication is necessary to synchronize events across multiple cores, for instance, in a parallelization library. Rodinia OpenMP-based applications use a fork-join parallelization paradigm, where synchronization barriers coordinate multiple children threads execution. One synchronization event that requires all cores to reach the same statement (i.e., a barrier) requires multiple quantum executions to complete. Delaying these communication events lead to some additional instructions executed by SOFIA due to other cores waiting. In this regard, we investigate how programming models behave with quantum size changes, as shown in Table A.2.

Table A.2 – Mismatch comparison of programming models of SOFIA with default (DF) and small quantum size (Q) in relation to gem5-FIM.

#	*Workload and Programming Models*	*Single-core (%)*		*Dual-core (%)*		*Quad-core (%)*	
		DF	*Q*	*DF*	*Q*	*DF*	*Q*
Worst Case	NPB Serial	5.24	3.51	*	*	*	*
	NPB MPI	5.17	1.19	4.08	1.64	4.69	**5.59**
	NPB OpenMP	5.39	2.06	5.27	1.67	9.61	2.25
	Rodinia OpenMP	6.06	4.16	13.35	8.76	17.71	3.64
Best Case	NPB Serial	0.01	0.01	*	*	*	*
	NPB MPI	0.01	0.00	0.01	0.00	0.01	0.01
	NPB OpenMP	0.01	0.01	0.01	0.00	0.01	0.01
	Rodinia OpenMP	0.03	0.00	0.04	0.01	0.01	0.01
Average	NPB Serial	1.15	0.55	*	*	*	*
	NPB MPI	1.06	0.26	0.83	0.35	0.63	**0.80**
	NPB OpenMP	1.32	0.52	1.42	0.41	1.68	0.42
	Rodinia OpenMP	1.39	0.82	2.51	1.53	2.63	1.08

Source : The authors

Table A.2 shows the mismatch between gem5-FIM and SOFIA using the default quantum size (DF) of 448,000 instructions and a smaller quantum size (Q) of 44 instructions, considering four distinct workloads: NPB Serial, NPB MPI, and NPB OpenMP, along with Rodinia OpenMP applications. Results show that the SOFIA(Q) decreases the mismatch whenever compared to the gem5-FIM atomic for the Rodinia suite, and is further accentuated in the quad-core system due to the instruction count reduction, as previously mentioned. For example, the average mismatch reduced from 2.63% to 1.08%, as shown in the last row of Table A.2. In the same scenario, the worst-case result decreased from 17.71% to only 3.64%. Interestingly, we have one case where the reduction of the quantum size causes a worsening in the soft error consistency, which is for NPB MPI in a quad-core system. In this scenario, the results derived from the Table A.2 show an average difference of 0.17% (i.e., 0.80% - 0.69%) and a worst-case of 0.9% (i.e., 5.59% - 4.69%).

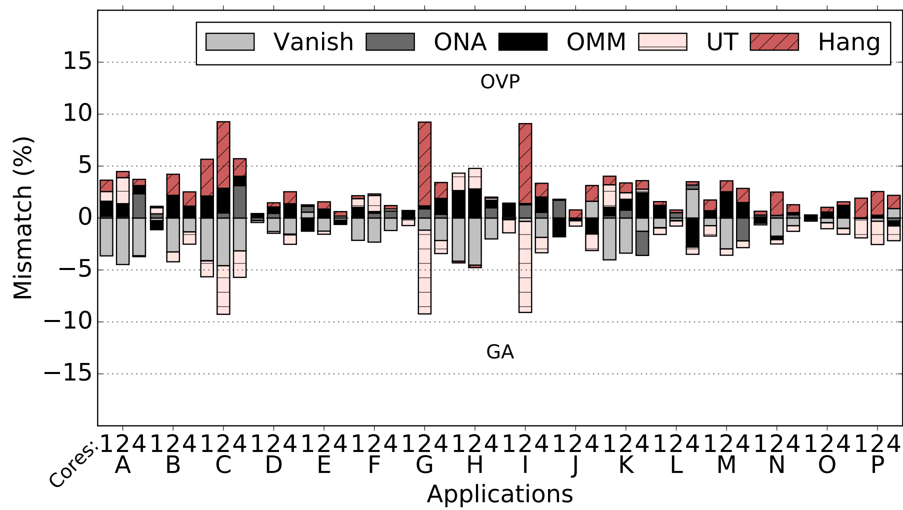
In order to understand where the mismatches shown in Table A.2 come from, Figure A.7 shows the accuracy difference presented by Rodinia (Figure A.7.a) and NPB applications (Figure A.7.b) when SOFIA is configured with the smallest quantum size (i.e., a 44-instruction block). Regarding the migration from *ONA* to *OMM*, this trend is evident when looking at Rodinia's applications A, F, I, L, and P, compared to Figure A.3 that is configured with the default quantum size. In addition, the FI campaigns simulated with the SOFIA(Q) presents a mismatch reduction in 24 out of 26 scenarios (except Rodinia's applications C and O) with a significant (5×) improvement in the worst-case of OpenMP-based benchmarks, which is justified by the impact of synchronization barriers between children threads. The most significant reductions are for Rodinia's application K, using a quad-core system, reducing from 20% to 4%, and MG application of the NPB benchmark, also in a quad-core system, reducing from 12% to less than 2.5%.

Regarding the benchmarks' difference, Rodinia includes applications with up to 220 *million*, while NPB applications vary from 16 to 87 *billion* instructions. By consequence, NPB benchmarks have more extended computations between synchronization points than the Rodinia, which impacts on the soft error assessment accuracy. NPB benchmark also has a better workload distribution and scalability, which means, in conjunction with the more prolonged execution, that children threads have enough instructions to complete one or more quantum blocks between OpenMP barriers. On the other hand, Rodinia applications have less computation between synchronization points, sometimes shorten then one quantum execution, leading OpenMP barriers to execute extra instructions while waiting for other threads. The Rodinia behavior magnifies the mismatch originated due to the SOFIA simulation policy using fixed-length instructions blocks, and, as a consequence, these applications benefit more by reducing the quantum size, achieving an accuracy gain up to 5×, as seen in Figure A.7.

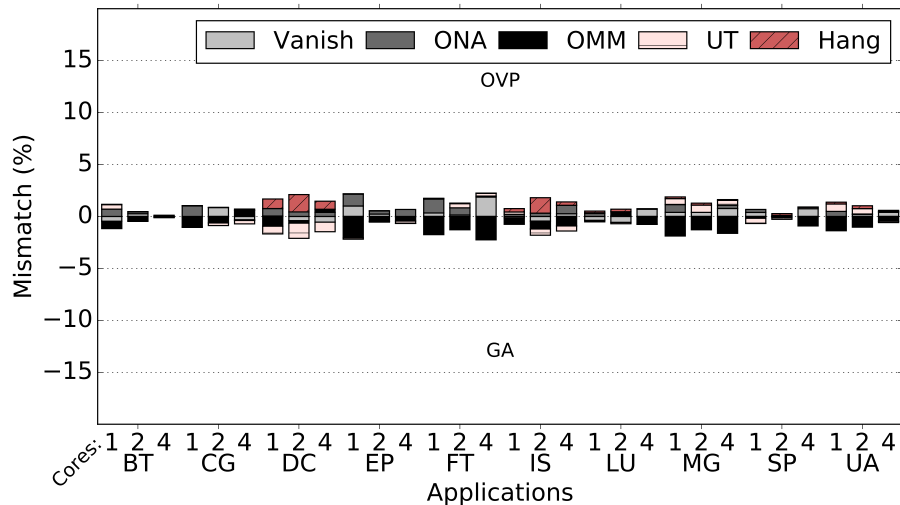
A.4 Closing Remarks

This section presents a complete multi-core soft error resilience assessment. As simulation performance is fundamental for early design space explorations, this section started by showing that the peak SOFIA simulation speed is around 4,000 MIPS, considering a quad-core system. This finding proves that JIT-based FI frameworks are efficient means to assess the soft error resilience early in the design phase. Next, results demonstrate that applications showed a worsened mismatch while increasing the number of cores.

Figure A.7 – Multi-core mismatch between gem5-FIM and SOFIA with smallest quantum size (a 44-instruction block).



(a) Rodinia Benchmarks



(b) NPB Benchmarks

Source : The authors

However, MPI-based applications have been shown to bring the best results in terms of reliability accuracy, even in multi-core scenarios. This mismatch can be mitigated by improving the workload balance and context switching in parallel applications, such as in the MPI programming model. Finally, we show that the SOFIA Quantum parameter affects the soft error reliability assessment and that a good trade-off between simulation performance and reliability accuracy would be using a small quantum size, such as a 44-instruction block.

APPENDIX B — LIST OF WORKS PUBLISHED BY THE AUTHOR

This section presents a list of publications related to the author's Ph.D.:

International Journals:

(I) **G. Abich**, R. Garibotti, V, Bandeira, F. da Rosa, J. Gava, F. Bortolon, G. Medeiros, F. G. Moraes, R. Reis, L. Ost, Evaluation of the soft error assessment consistency of a JIT-based virtual platform simulator. IET Computers & Digital Techniques Journal, 2021.

(II) **G. Abich**, J. Gava, R. Garibotti, R. Reis, and L. Ost, Applying Lightweight Soft Error Mitigation Techniques to Embedded Mixed Precision Deep Neural Networks. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I-REGULAR PAPERS, 2021.

(III) **G. Abich**, R. Garibotti, R. Reis, and L. Ost, The Impact of Soft Errors in Memory Units of Edge Devices Executing Convolutional Neural Networks. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: Express Briefs, 2022. *Indicated to Best Paper Awards in LASCAS 2022 and extended to a Special Issue of TCAS II.*

International Conferences:

(I) **G. Abich**, M. G. Mandelli, F. R. da Rosa, F. G. Moraes, L. Ost, R. Reis, Extending FreeRTOS to support dynamic and distributed mapping in multiprocessor systems. IEEE International Conference on Electronics, Circuits, and Systems (ICECS), 2016.

(II) J. C. Hamerski, **G. Abich**, R. Reis, L. Ost, A. Amory, Publish-subscribe programming for a NoC-based multiprocessor system-on-chip. IEEE International Symposium on Circuits and Systems (ISCAS), 2017.

(III) F. T. Bortolon, **G. Abich**, S. Bampi, R. Reis, F. G. Moraes, and L. Ost, Exploring the impact of soft errors on NoC-based multiprocessor systems. IEEE International Symposium on Circuits and Systems (ISCAS), 2018.

(IV) J. C. Hamerski, **G. Abich**, R. Reis, L. Ost, A. Amory, A design patterns-based middleware for multiprocessor systems-on-chip. Symposium on Integrated Circuits and Systems Design (SBCCI), 2018.

(V) **G. Abich**, J. Gava, R. Reis, and L. Ost, Soft Error Reliability Assessment of Neural Networks on Resource-constrained IoT Devices. IEEE International Conference on Electronics Circuits and Systems (ICECS), 2020.

(VI) **G. Abich**, R. Reis, and L. Ost, The Impact of Precision Bitwidth on the Soft Error Reliability of the MobileNet Network. IEEE Latin America Symposium on Circuits and System (LASCAS), 2021. *Indicated to Best Paper Awards and extended to TCAS I.*

(VII) **G. Abich**, J. Gava, R. Garibotti, R. Reis, and L. Ost, Impact of Thread Parallelism on the Soft Error Reliability of Convolution Neural Networks. IEEE Latin America Symposium on Circuits and System (LASCAS), 2022.

Workshops and Poster Presentations:

(I) F. T. Bortolon, **G. Abich**, S. Bampi, R. Reis, F. G. Moraes, and L. Ost, Exploring the impact of soft errors on NoC-based multiprocessor systems. A Richard Newton Young Student Fellow Program at 55th Design Automation Conference (DAC) 2018.

(II) **G. Abich**, R. Reis, and L. Ost, Exploring the Impact of Soft Errors in Neural Networks Executing on Resource-constrained IoT Devices. IEEE CASS Rio Grande do Sul Workshop (CASSW-RS) 2020.

(III) **G. Abich**, R. Reis, and L. Ost, Applying Lightweight Soft Error Mitigation Techniques to Embedded Mixed Precision Deep Neural Networks. IEEE CASS Rio Grande do Sul Workshop (CASSW-RS) 2021.