TECHNISCHE UNIVERSITÄT WIEN

Low Overhead SW-based HW Fault Tolerance for tinyML and Security Applications

D. Mueller-Gritschneder*, Johannes Kappes*, Johannes Geier**

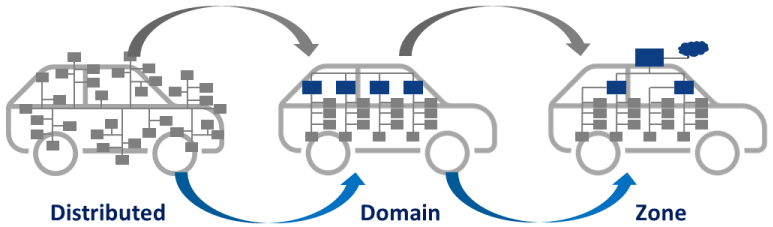
* TU Wien

** TU Munich

Computing Trends: Advanced Driving Functions

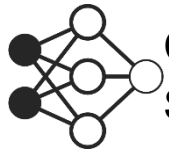
Hardware view:

- Migration from Distributed, Domain to Zone.
- Few very powerful compute platforms.
- Embedded HPC



Software View

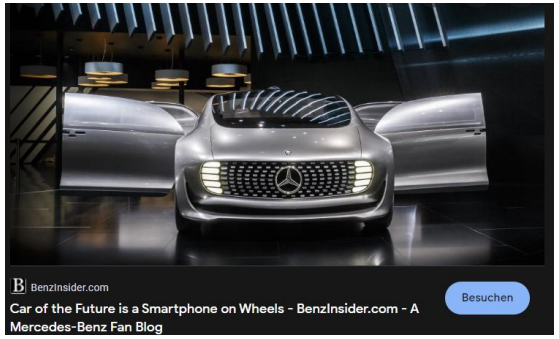
- Rising number of SW functions on same platform
- Rising number of AI-based workloads.



CNNs, Transformers,
SSM (MAMBA)

Safety-critical real-time system

- Real-time constraints / deadlines.
- Functional safe and secure.



https://www.benzinsider.com/2015/01/car-future-smartphone-wheels/

Computing Trend tinyML

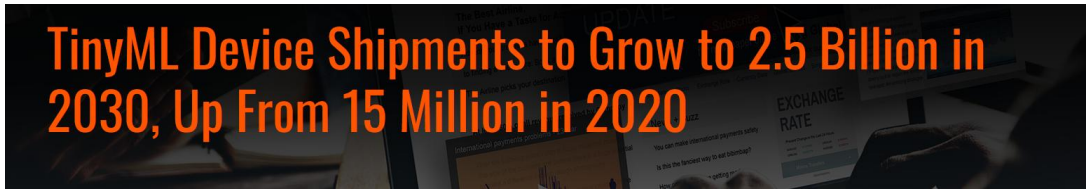
The Future of AI Is Tiny

Tiny AI reduces carbon footprints, brings deep learning at an affordable cost, creates context-aware consumer devices, cuts down data infrastructure, bolsters security, and more.

https://www.informationweek.com/data-management/the-future-of-ai-is-tiny/

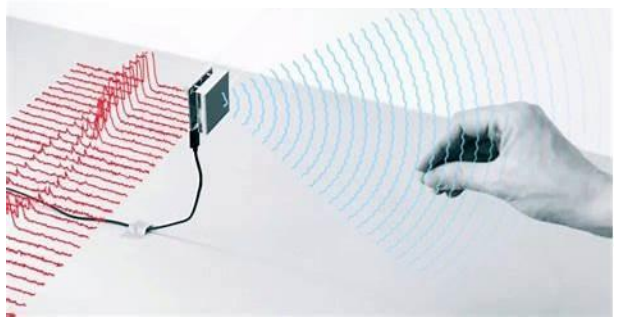
Running NN inference on low-power micro-controllers / IoT Devices

- **Audio:** Keyword Spotting (KWS) / Audio Wakeup
- **Vision:** Video Wakeup (Face Detection)
- **Radar:** Gesture Recognition
- **Accelerometer:** Activity Detection

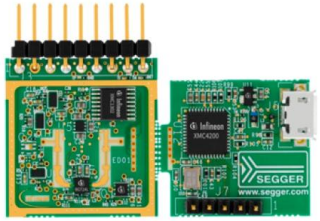


https://www.abiresearch.com/press/tinyml-device-shipments-grow-25-billion-2030-15-million-2020/

Gesture Recognition with Radar Sensor



tinyML App



Platform: Infineon XMC1302 MCU

32 MHz Micro-controller CPU

32 kB Flash, 16 kB RAM

https://philab.esa.int/world-breakthrough-in-onboard-ai-model-training-presented-by-%CF%86-lab-at-igarss/

Soft Errors got new Attention in Recent Years

https://www.nytimes.com/2022/02/07/technology/computer-chips-errors.html


NewScientist

Sign in | Enter search keywords

News Features Newsletters Podcasts Video Comment Culture Crosswords | This week's magazine

Health Space Physics Technology Environment Mind Humans Life Mathematics Chemistry Earth Society

Technology

Google and Facebook hit by faulty chips that can silently corrupt data

By Matthew Sparkes

17 June 2021


The New York Times

Tiny Chips, Big Headaches

As the largest computer networks continue to grow, some engineers fear that their smallest components could prove to be an Achilles' heel.

- Two studies by META and Google:
  - *Dixit et al. "Silent Data Corruptions at Scale.", ArXiv abs/2102.11245 (2021)*
  - *Hochschild et. al. „Cores that don't count", HotOS '21, https://sigops.org/s/conferences/hotos/2021/papers/hotos21-s01-hochschild.pdf*

- Higher-than-expected SDC rates observed in datacenter CPUs (test escapes, T>0 defects)

- Possible upcoming challenge as Embedded AI chips move into advanced technology nodes for driving autonomous functions

Functional Safety Aspects for AI

- **ISO/PAS 21448**
  - Safety of the Intended Functionality (SOTIF) -
    Situational awareness: Outlier objects, undiscovered scenes,…
  - ➢ Algorithmic Safety Techniques:
    Plausibility, Fallback, etc.

- **ISO 26262**
  - Systematic faults during development („bugs“)
    - ➢ Safety Design Flow, Verification
  - **<u>Safety in the presence of random HW faults</u>**
    - ➢ Fault Tolerance (ASIL level A-D)
      Error detection, handling, recovery, correction

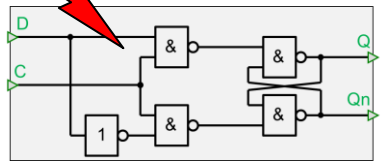
If undetected:
Silent data corruption (SDC)

sum=64 → sum=65600

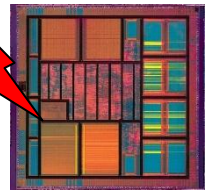
```
for(i=0;i<n;i++){
    sum+=c[i]*b[i];
}
WriteREG(sum,Addr);
```

Corruption of computation

‘0’ → ‘1’



Bit Flip at Circuit Level



Radiation-induced Soft Errors

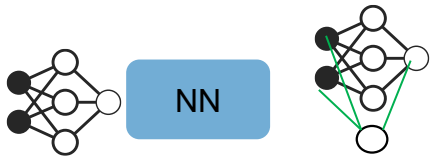
Defect-based Soft Errors

Exploration of Fault Tolerance for AI Workloads

Overheads

Neural Architecture Search

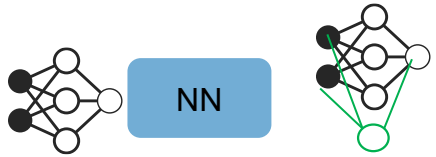
Bigger NN



?

Training

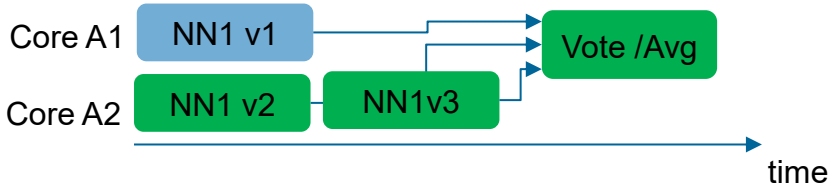
Fault-Tolerant NN Training



?

Task Planning

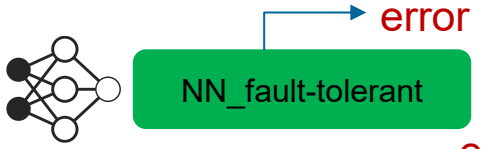
Redundant Execution (DMR/TMR)



>2x

Deployment ML Compiler

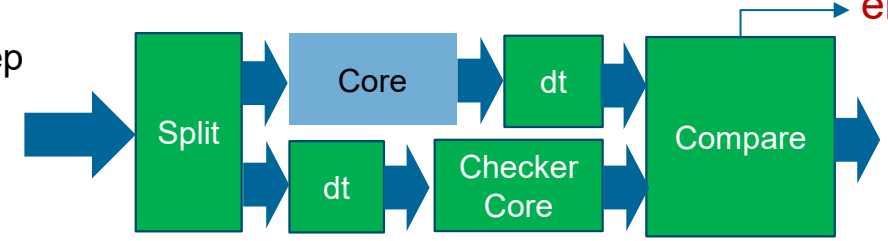
Algorithm-based Fault Tolerance (ABFT)
SW-implemented HW Fault Tolerance (SIHFT)



?

HW Architecture

Lock-step, dual-lock-step
Rad-hard
DMR,TMR
ECC, Parity

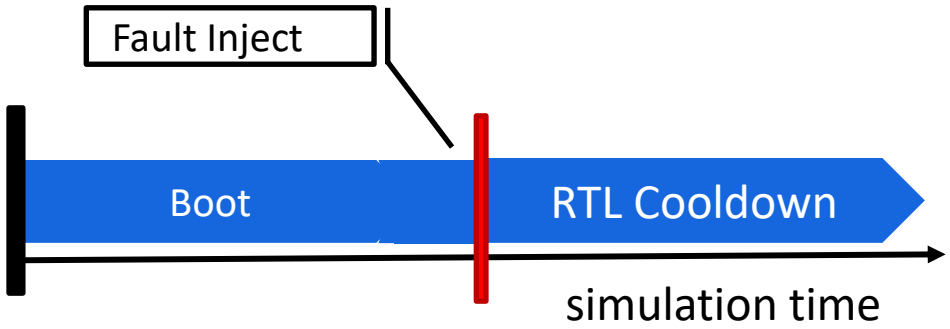


>2x

Works

- **Differential FEEC** : Speed up Fault Injection at RTL
- **SIHFT: REPAIR, COMPASEC**: Compiler-assisted Fault Tolerance for Safety and Security Apps
- **ABFT**: Fault Tolerance for tinyML Apps at Kernel Level and Graph Level

Accelerating Fault Injection Simulations



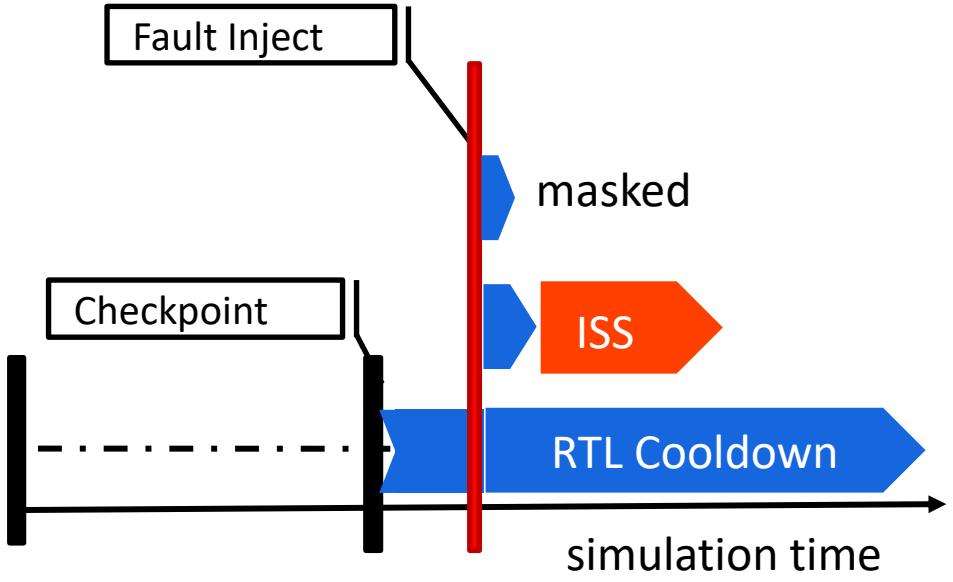
Target: Run FI as fast as possible to cover many error scenarios:

- **Warm-up:** Accelerate pre-FI
  - Checkpoint Restore Boot (CRB) [17,18,19]
  - High level warm-up (switch-down) [11,19]

2x

- **(Dynamic) Cooldown:** Accelerate post-FI
  - Early Exit, e.g., masking checks
  - Mixed-level cool-down [11,19] (switch-up abstraction)
  - Requires switching simulation

?x



Fault Effect Equivalence Check (FEEC)

Idea:

"If the states of two experiments are equal, their outcome will be, too"

- **Similar to Fault Pruning:**
  → Omit experiments with known outcome
  → Without prior analysis (use database)

- μ_t^a Simulation State of experiment a at time t

- o^a Outcome of experiment a

- **FEEC**: $\mu_t^a = \mu_t^b \Rightarrow o^a = o^b$

Challenges:

Storage/Trace:

- Simulation states (traces)
- Many experiments

Equivalence Checks:

- **Costly** (bit-wise comparison)

Solution: Hash States for FEEC

$$H(\mu_t^a) = H(\mu_t^b) \Rightarrow o^a = o^b$$

- **Reduces Storage to 64-bit per experiment state**
- **Hash not reversible (state info is lost)**

FEEC: Differential Checks

Idea:

"If the states of two experiments are equal, so are their deviations to another common experiment – at the same point in time"

- **Similar to Masking Checks:**
  - Compute Diff vs. Reference $\delta_t^a = \mu_t^a \oplus \mu_t^R$
  - If Diff is 0 fault masked $\delta_t^a = 0 \Rightarrow \mu_t^a = \mu_t^R$
- (Differential) **FEEC**: $\delta_t^a = \delta_t^b$

$$\Leftrightarrow \mu_t^a \oplus \mu_t^R = \mu_t^b \oplus \mu_t^R$$

$$\Leftrightarrow \mu_t^a = \mu_t^b \Rightarrow o^a = o^b$$

Challenges:

Storage:

- Diffs are quite cheap to store (sparse matrix)
- Are reversible with Reference states

Equivalence Checks:

- **Costly** (bit-wise comparison)
- Many Experiments

Solution: Hash Diff **FEEC**

$$H(t, \delta_t^a) = H\left(t, \delta_t^b\right) \Rightarrow o^a = o^b$$

- Reduces **experiment storage to 64-bit**
- Re-use **checkpoints for reference**

Checkpoint Differentials

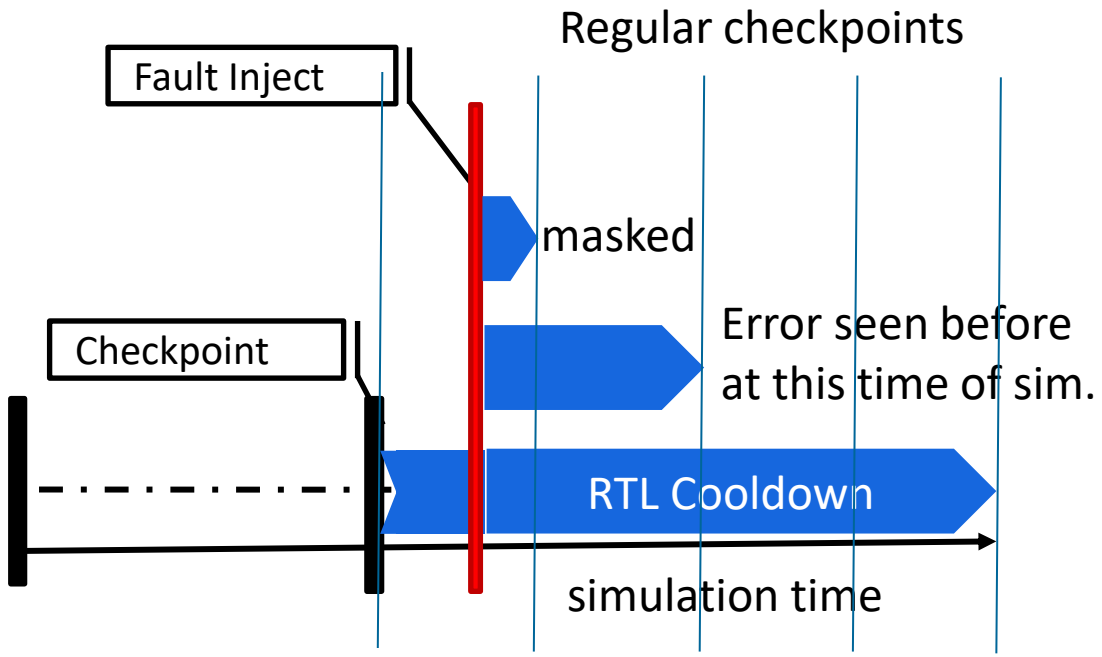
Leveraging existing Checkpoints:

1) Prepare FI Campaign

- Record Checkpoints

1) Execute FI Campaign

- Use Checkpoints to boot as close as possible to FI
- Compute Diffs at next checkpoint
- Perform **FEEC** with Diff-Hashes
- Store Diffs if no FEEC match



Evaluating Performance and Accuracy

1) Framework:

- Open-source RTL FI Simulator (Verilator based: **vrtlmod**)
  - Automatically add Differential FI API
  - "Black-box" DUT principle
- DUT: CV32E40P RISC-V CPU [28,29]

2) Campaign:

- Single-bit transient FI in CPU micro-architecture

Performance: “Accelerated vs. RTL”

1) Checkpoint Restore Boot (CRB):
- ✓ ~2x → as expected

2) Checkpoint Masking (CMSK):
- ✓ ~3-4x → Masking Rate ~50% (another 2x)

3) Mixed-level Simulation (MLS):
- ✓ ~4x → **Similar Performance to Masking**

4) Checkpoint Diff. FEEC (CDIF+FEEC)
- ✓ Up to ~25x
- ✓ Scales with sample size

No accuracy loss, except for MLS due to different timing of the ISS

simulator	BM	|N| : 10,000 $\overline{t_s}$[s]	×	100,000 $\overline{t_s}$[s]	×	1,000,000 $\overline{t_s}$[s]	×
RTL	aha-mont	0.74	-	0.73	-		
CRB		0.55	1.3	0.55	1.3		
CMSK		0.47	1.6	0.49	1.5		
MLS		0.60	1.2	0.60	1.2	0.60	~1.2
CDIF+FEEC		0.47	1.6	0.39	1.9	0.40	~1.8
RTL	huffbench	11.7	-	11.6	-		
CRB		6.1	1.9	6.1	1.9		
CMSK		3.7	3.2	3.8	3.0		
MLS		3.0	3.9	3.0	3.9		
CDIF+FEEC		3.1	3.8	1.5	7.6	1.0	~11
RTL	picojpeg	29.9	-	27.4	-		
CRB		15.0	2.0	15.0	1.8		
CMSK		7.7	3.9	8.1	3.4		
MLS		6.6	4.5	6.8	4.0		
CDIF+FEEC		5.9	5.1	2.4	11	1.3	~21
RTL	wikisort	65.6	-	63.9	-		
CRB		33.2	2.0	33.4	1.9		
CMSK		16.8	3.9	17.3	3.7		
MLS		14.2	4.6	14.2	4.5		
CDIF+FEEC		12.8	5.1	5.0	12.8	2.53	~25

Tab.: Performance in average experiment simulation time in seconds and speed-up factor vs. baseline RTL (×).

Agenda

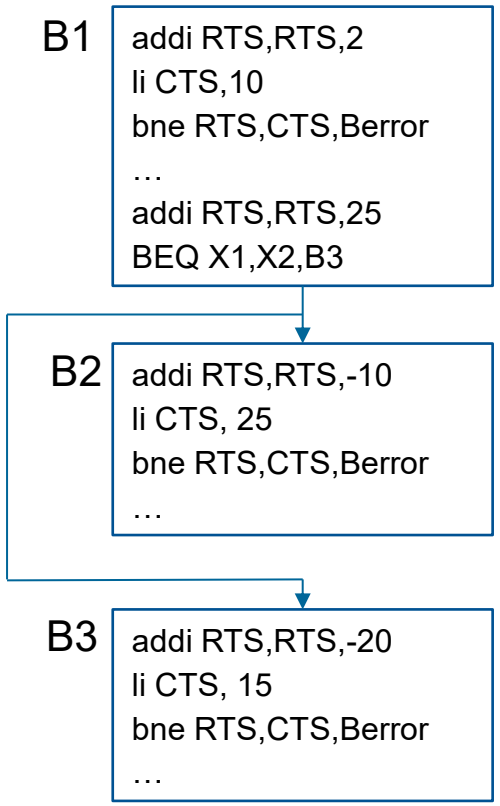
- **Differential FEEC** : Speed up Fault Injection at RTL
- **REPAIR, COMPASEC**: Compiler-assisted Fault Tolerance for Safety and Security Apps
- **ABFT for tinyML**: Fault Tolerance for tinyML Apps at Kernel Level and Graph Level

Software Implemented HW Fault Tolerance

- Modify given program by inserting logic for detection / handling runtime errors
- Attractive for embedded systems:
  - Fault Tolerance on Commercial Off-the-shelf HW
  - Flexibility (Selective hardening)

- At source-code level or at **compiler level**

SIHFT Methods: Runtime Signature Moitoring



- To protect control flow integrity
  - Enumerate basic blocks with unique signature
  - Update and check run time signature

Method	Covered Errors
CFCSS [Oh, 2002]	Illegal inter block jumps
RASM [Vankeirsbilck, 2017]	Illegal inter block jumps; Wrong branch taken

Intra block CFE: Instr. 1 -> Instr. N in B1
Inter block CFE: Instr. 1 in B1 -> Instr. 2 in B3

SIHFT Methods: Instruction Duplication

- Data-flow Integrity
- Duplicate instructions for computational redundancy, Checks placed strategically to ensure consistency

EDDI [Oh2002]

```
# BB1:
(i1)      ADD r4, r2, r3
(i1^d)    ADD r20, r18, r19
(i2)      XOR r6, r4, r7
(i2^d)    XOR r22, r20, r23
          BNE r6, r22, BBerr
          BNE r0, r16, BBerr
(i3)      BNE r6, r0, BB3
# BB2:
(i4)      AND r7, r6, r8
(i4^d)    AND r23, r22, r24
(i5)      OR  r7, r7, r1
(i5^d)    OR  r23, r23, r17
          BNE r4, r20, BBerr
          BNE r7, r23, BBerr
(i6)      SW  0(r4), r7
(i6^d)    SW  0(r20), r23
```

SWIFT [Reis2005]

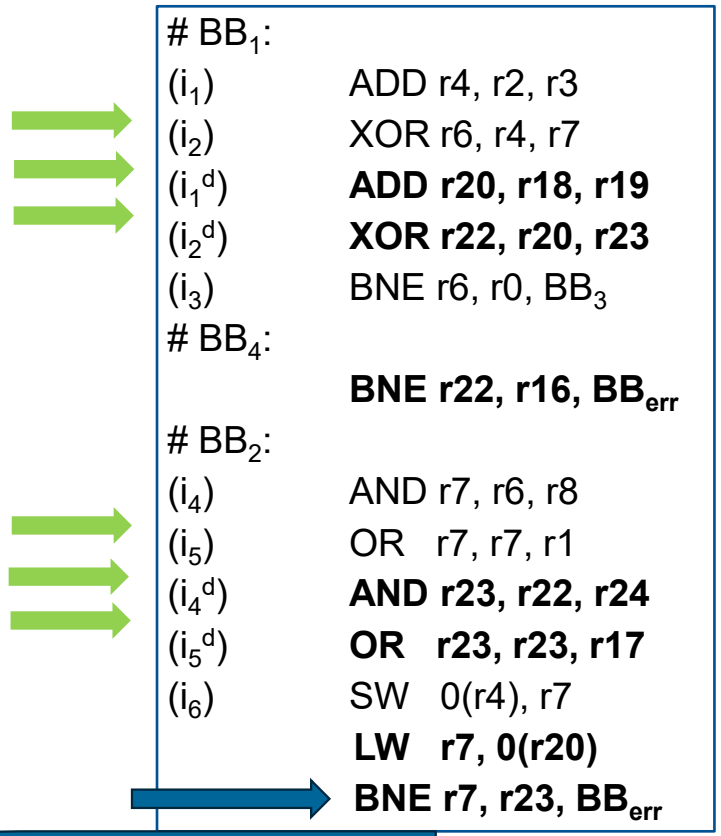
```
# BB1:
(i1)      ADD r4, r2, r3
(i1^d)    ADD r20, r18, r19
(i2)      XOR r6, r4, r7
(i2^d)    XOR r22, r20, r23
          BNE r6, r22, BBerr
          BNE r0, r16, BBerr
(i3)      BNE r6, r0, BB3
# BB2:
(i4)      AND r7, r6, r8
(i4^d)    AND r23, r22, r24
(i5)      OR  r7, r7, r1
(i5^d)    OR  r23, r23, r17
          BNE r4, r20, BBerr
          BNE r7, r23, BBerr
(i6)      SW  0(r4), r7
```

NZDC [Didehban2016]

```
# BB1:
(i1)      ADD r4, r2, r3
(i1^d)    ADD r20, r18, r19
(i2)      XOR r6, r4, r7
(i2^d)    XOR r22, r20, r23
(i3)      BNE r6, r0, BB3
# BB4:
          BNE r22, r16, BBerr
# BB2:
(i4)      AND r7, r6, r8
(i4^d)    AND r23, r22, r24
(i5)      OR  r7, r7, r1
(i5^d)    OR  r23, r23, r17
(i6)      SW  0(r4), r7
          LW  r7, 0(r20)
          BNE r7, r23, BBerr
```

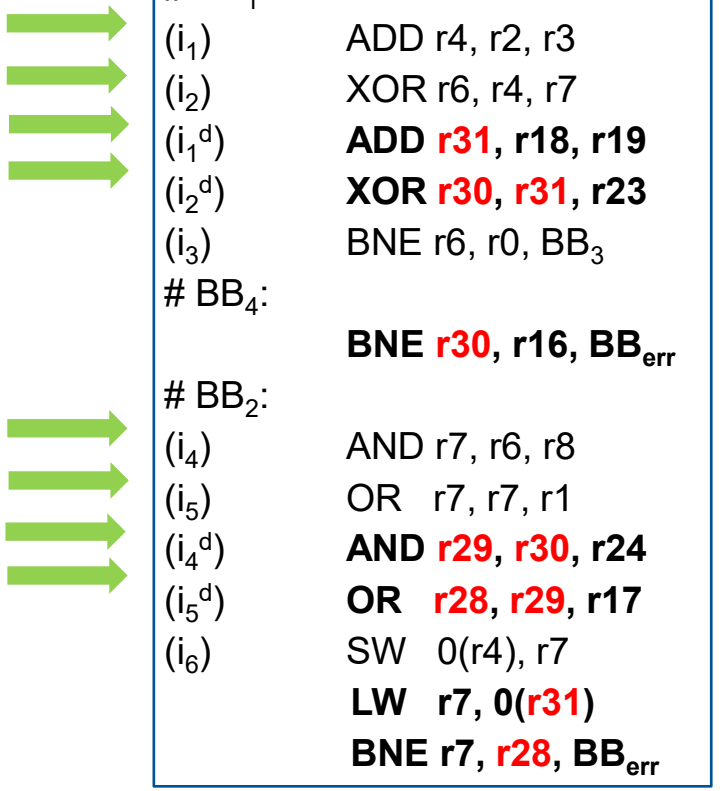
REPAIR

NZDC (CR: Coarse grained)



This check will detect illegal jumps to green instructions due to Imbalance between primary and secondary computation

REPAIR [Sharif21]



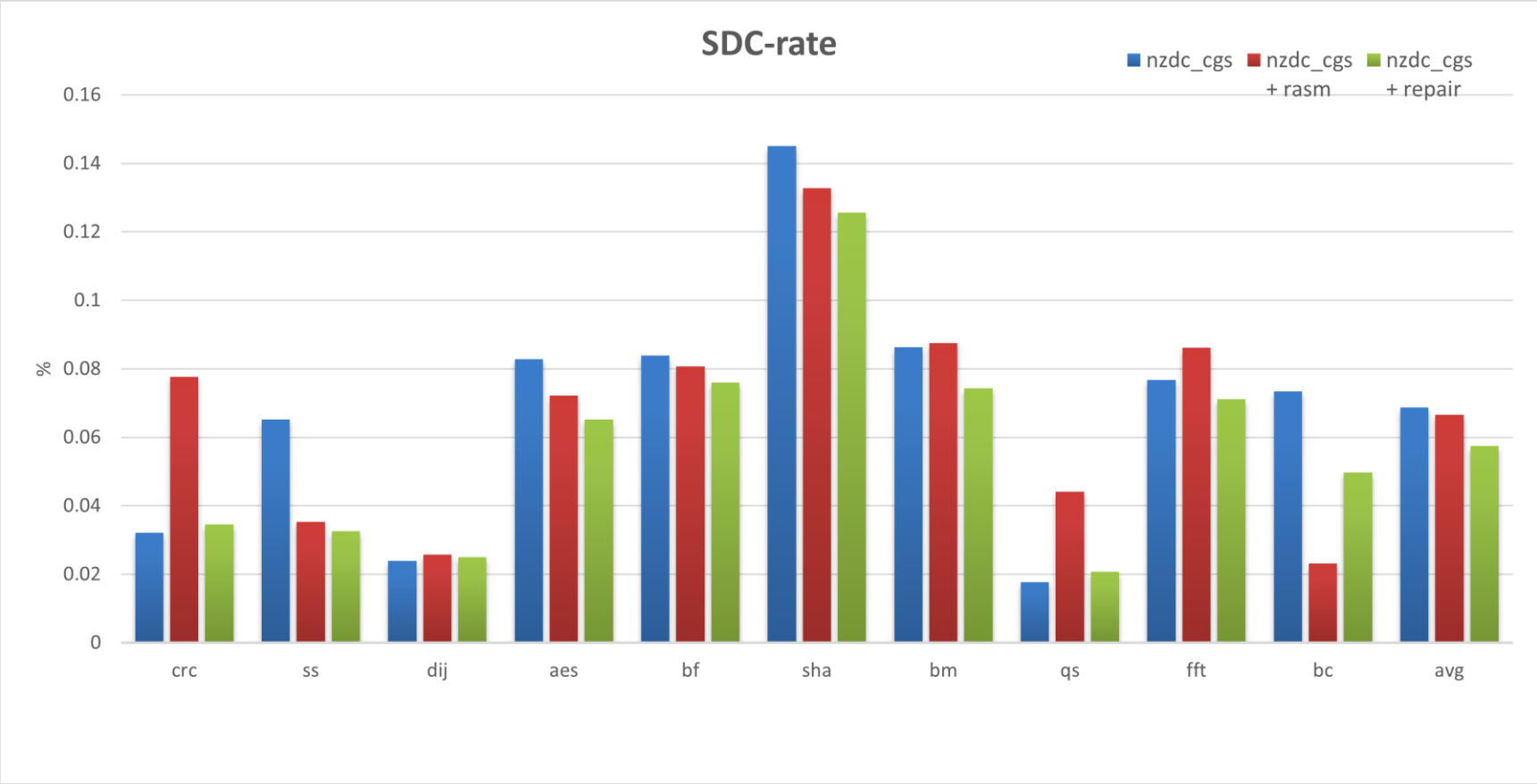
Dynamic register pairing between primary and shadow computation

More illegal jumps will be detected by the instruction duplication

Runtime monitoring can be removed for illegal jump detection.

REPAIR vs. SIHFT Evaluation using MiBench

$$SDCrate = \frac{no.of\ SDCs}{total\ no.of\ trials} * \gamma$$

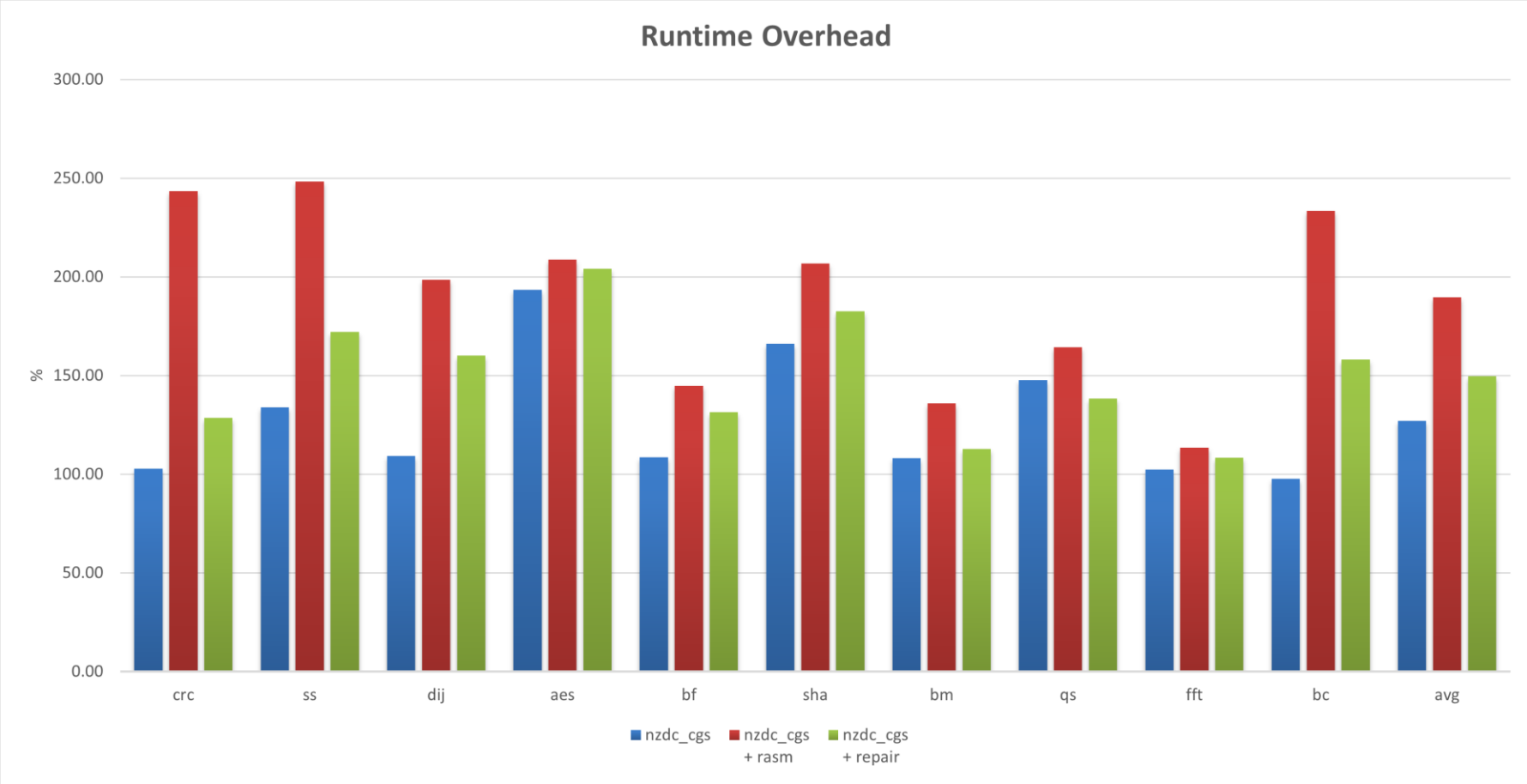


On average, REPAIR performs competitively with state-of-the-art methods

U Sharif, D Mueller-Gritschneder, U Schlichtmann: Repair: Control flow protection based on register pairing updates for sw-implemented hw fault tolerance, TECS 21

REPAIR vs. SIHFT Evaluation using MiBench

$$overhead = \frac{(runtime\ -\ baseline)}{baseline}$$



REPAIR shows better overhead performance than NZDC+RASM on all programs

CompaSEC

Security: Instruction Skip Fault Model

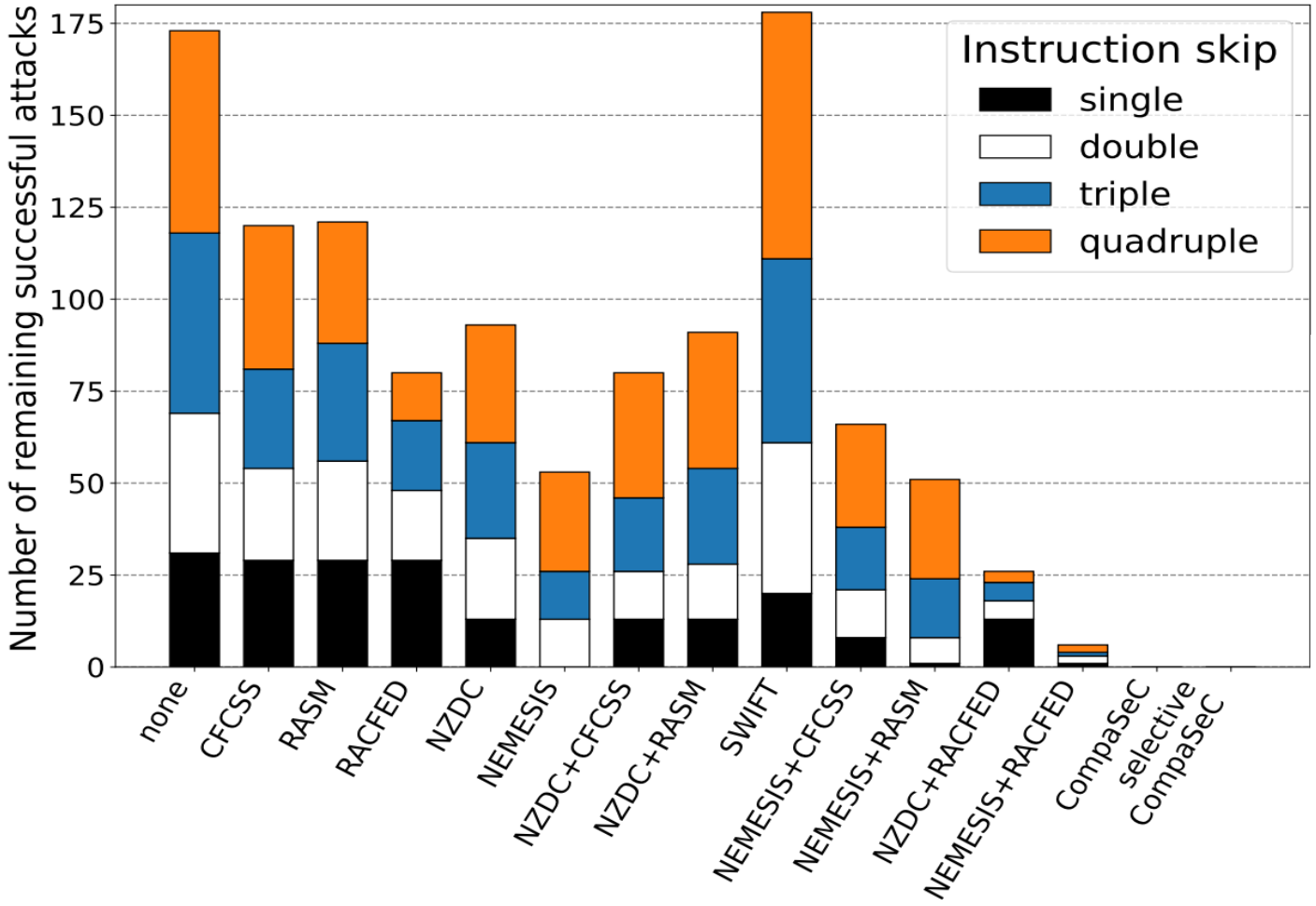
- E.g. caused by flushes of instruction cache (several instructions skipped in program flow)
- **Observation: Standard SIHFT techniques are not very efficient to detect instruction skips**

CompaS(eC) **Comp**iler-**a**ssisted **S**afety (**e**curity **C**ountermeasure)

https://github.com/tum-ei-eda/compas-ft-riscv

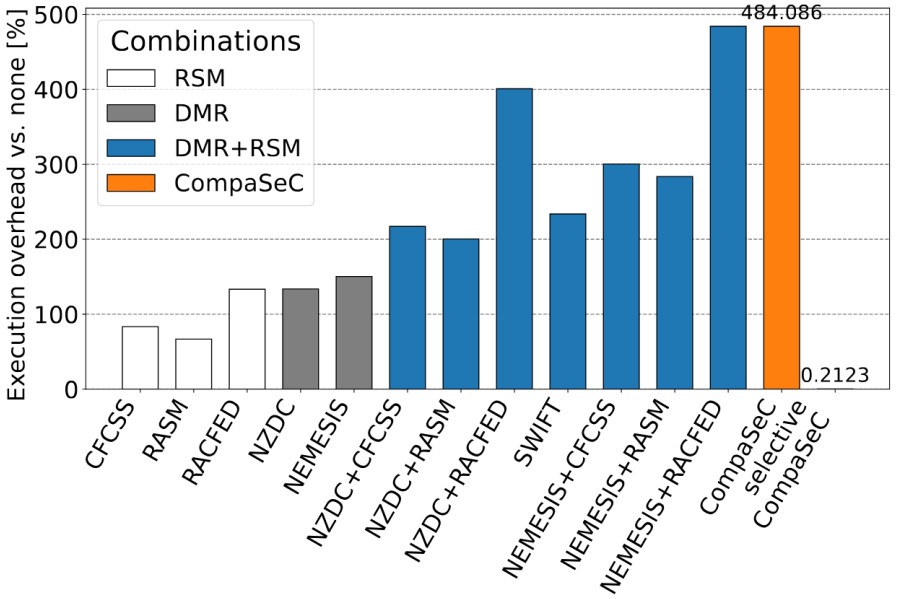
- Re-combination of several (sub)techniques RTM and Instr. duplication:
  “what works best against instruction skip model”

COMPASEC Results



Goal: Boot a malicious software image bypassing security checks

Tool: ARCHIE [9]: QEMU-based fault injection simulation



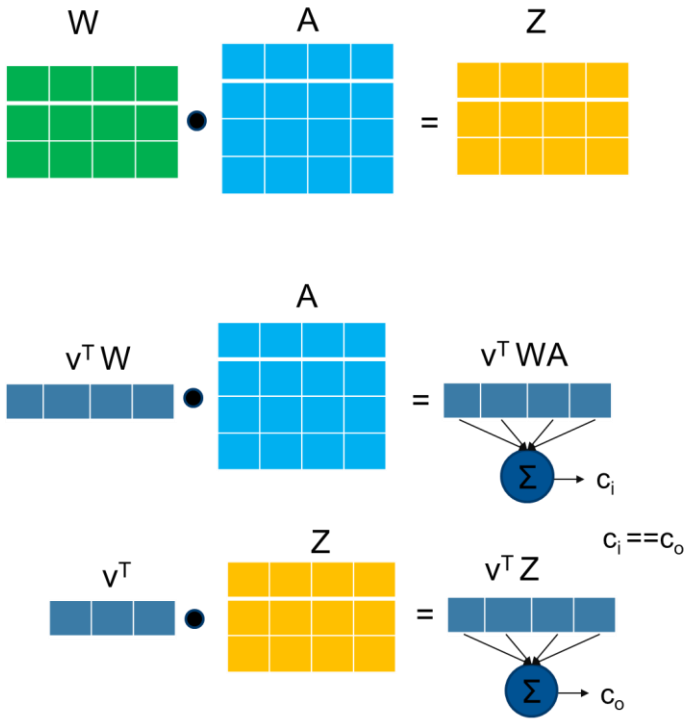
Johannes Geier, Lukas Auer, Daniel Mueller-Gritschneder, Uzair Sharif, and Ulf Schlichtmann. 2023. CompaSeC: A Compiler-Assisted Security Countermeasure to Address Instruction Skip Fault Attacks on RISC-V. ASP-DAC 23

Agenda

- **Differential FEEC** : Speed up Fault Injection at RTL
- **REPAIR, COMPASEC**: Compiler-assisted Fault Tolerance for Safety and Security Apps
- **ABFT for tinyML**: Fault Tolerance for tinyML Apps at Kernel Level and Graph Level

Algorithm-based Error Detection (ABFT)

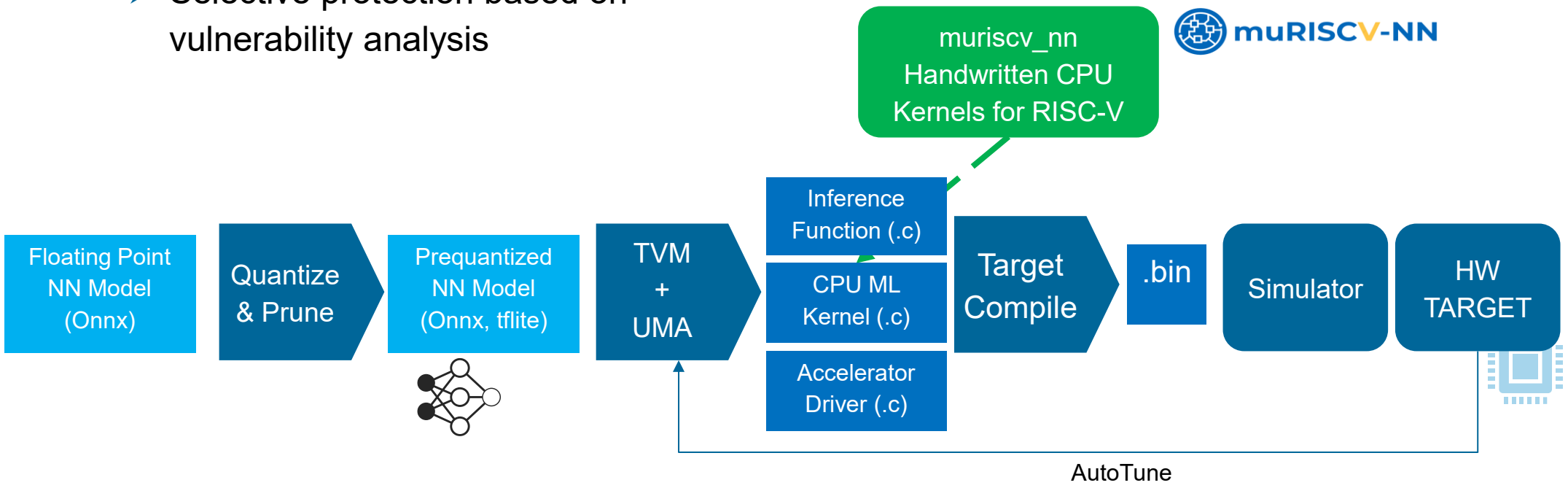
- ABFT uses checksums to find random errors in linear algebra operations (e.g. matrix-matrix-multiply) [1]
- NVIDIA: Filter and input fmap checksum (FIC) [2]
  - Efficient implementation for convolutions
- Less than 2x runtime / energy overhead



- [1] Huang et al. “Algorithm-based fault tolerance for matrix operations,” IEEE Transactions on Computers, 1984.
- [2] Hari et al. “Making convolutions resilient via algorithm-based error detection techniques,” IEEE Transactions on Dependable and Secure Computing, 2022.

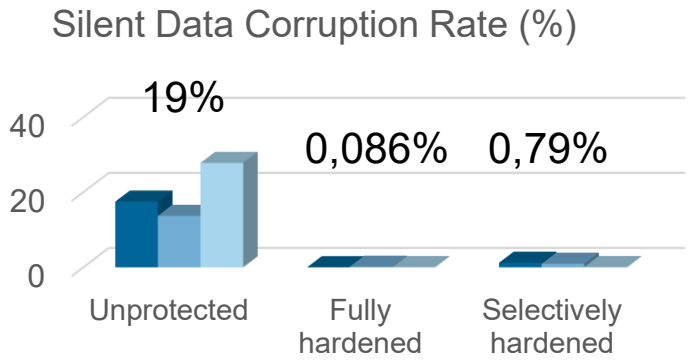
tinyML Flow with ABFT (Kernel-level)

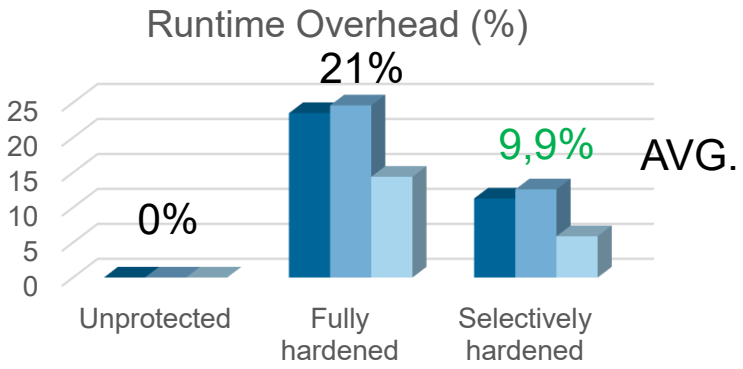
- Hand-coded **ABFT** and **SIHFT** in ML Kernel for dense, conv, depthwise-conv
- Integrated in TVM (ML Compiler) for tinyML workloads
  - Full protection
  - Selective protection based on vulnerability analysis

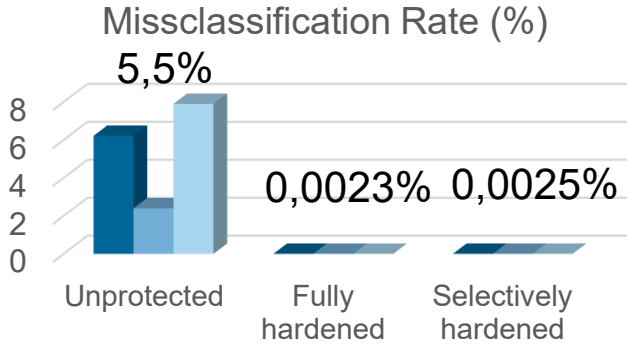


Exploration of Selective Hardening for AI Workloads

- Three tinyML neural networks : AWW, VWW, ResNET
- Mixture of instruction duplication and NVIDIA FIC (ABFT method)
- **ISS-level** Fault Injection to obtain SDC rates (RV32 CPU Model)





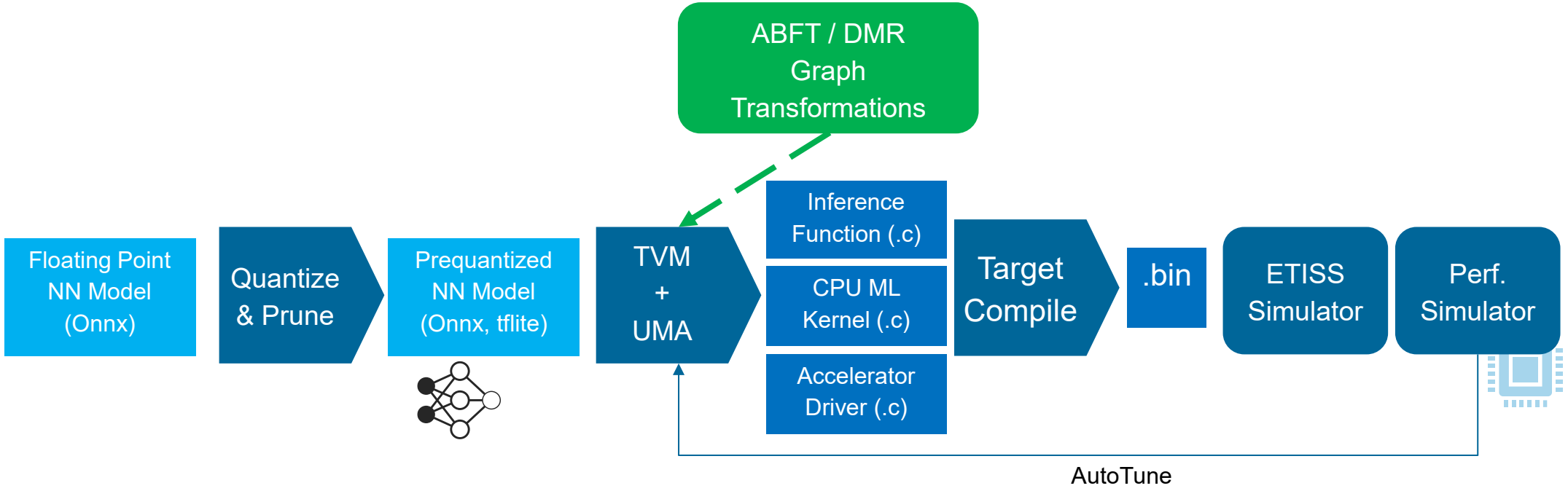


2200x Improvement

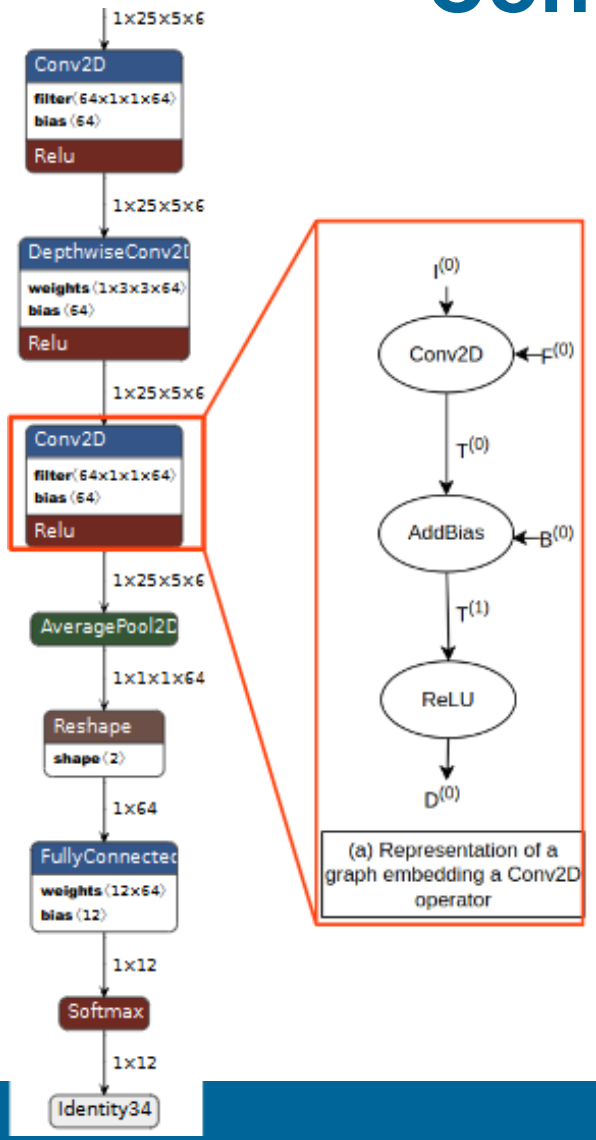
U Sharif, D Mueller-Gritschneder, R Stahl, U Schlichtmann Efficient software-implemented hw fault tolerance for tinyml inference in safety-critical applications, DATE23

tinyML Flow with ABFT (Graph Level)

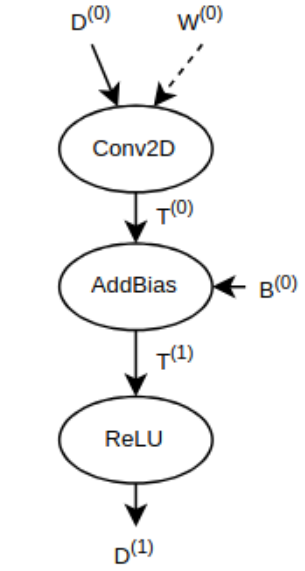
- ABFT and DMR as Graph Transformations
- Integrated in TVM (ML Compiler) for tinyML workloads



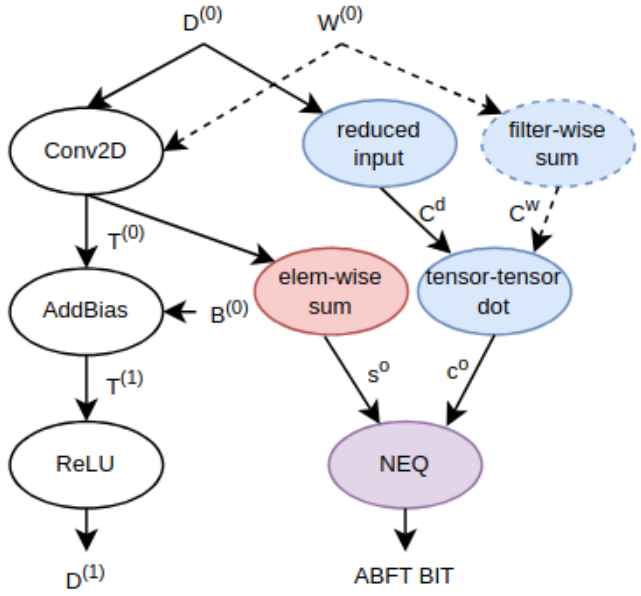
Graph-level Transformation for Conv2D



(a) Representation of a graph embedding a Conv2D operator



(a) Representation of a graph embedding a Conv2D operator



(b) Graph (a) after applied FIC method

Fig. 2. ABFT transformation for FIC(dw)

Fault Injection Campaign

Configuration	BM	sample size	FDR*[%]	EDC[%]	SDC[%]
BASE	AD	36,730	-	-	33.515 ±0.483
ABFT		66,764	~ 98.3	-	0.267 ±0.039
ABFT+DMRland		69,038	~ 99.5	-	0.085 ±0.022
BASE	AWW	70,703	-	12.898 ±0.247	9.896 ±0.220
ABFT		64,066	~ 85.9	1.553 ±0.096	1.594 ±0.097
ABFT+DMRland		111,924	~ 97.1	0.449 ±0.039	0.067 ±0.015
BASE	ResNET	79,195	-	12.633 ±0.231	14.435 ±0.245
ABFT		42,912	~ 93.8	0.680 ±0.078	0.944 ±0.091
ABFT+DMRland		49,727	~ 99.7	0.056 ±0.021	0.028 ±0.015

- Three tinyML neural networks : AWW, VWW, ResNET
- **ISS-level** Fault Injection to obtain SDC rates (RV32 CPU Model)

UPDATED RESULTS: J. Kappes to appear in IJCNN 2025

Performance Impact

TABLE I
PERFORMANCE MEASUREMENT OF DIFFERENT FAULT TOLERANCE DFG PASSES AND TINYML BENCHMARKS (BMS).

Metrics	BM[b]	BASE	FC	proposed graph-based fault tolerance[a] (*ours*) *overhead*	+FIC(dw)	*overhead*	+DMRland	*overhead*	kernel-implemented (from [14]) FC+FIC+ID *overhead*
I_{BM} $[1 \cdot 10^6]$	AD	1.60	3.37	*(+110%)*	-		3.42	*(+114%)*	*(+135%)*
ROM [kB]		302	584	*(+93%)*	-		600	*(+99%)*	*N/A*
RAM [kB]		3.30	9.43	*(+185%)*	-		9.56	*(+189%)*	*(+2.694%)*
I_{BM} $[1 \cdot 10^6]$	AWW	24.0	24.0	*(+0%)*	26	*(+8.3%)*	30.7	*(+28%)*	*(+45%)*
ROM [kB]		60.1	62.6	*(+4.2%)*	79.2	*(+32%)*	101	*(+68%)*	*N/A*
RAM [kB]		45.7	45.7	*(+0%)*	65.74	*(+44%)*	66.12	*(+45%)*	*(+3.533%)*
I_{BM} $[1 \cdot 10^6]$	ResNET	89.8	89.9	*(+0.11%)*	92.7	*(+3.2%)*	98.3	*(+9.5%)*	*(+17.07%)*
ROM [kB]		116	118	*(+1.7%)*	137	*(+18%)*	156	*(+34%)*	*N/A*
RAM [kB]		68.7	68.7	*(+0%)*	105	*(+53%)*	189	*(+175%)*	*(+4.874%)*

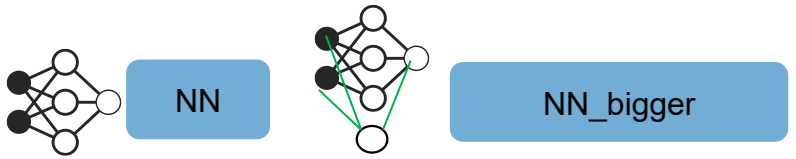
[a] ABFT methods FC and FIC(dw), and DMRland method are added on-top from each other from left to right.

[b] FC-AutoEncoder on ToyADMOS dataset for Anomaly Detection (AD), DS-CNN on Speech Commands dataset for Audio Wakeup Word (AWW), and ResNET on CIFAR10 dataset for Image Classification (ResNET). All BMs from MLPerf™Tiny [28].

Exploration of Fault Tolerance for AI Workloads

Neural Architecture Search

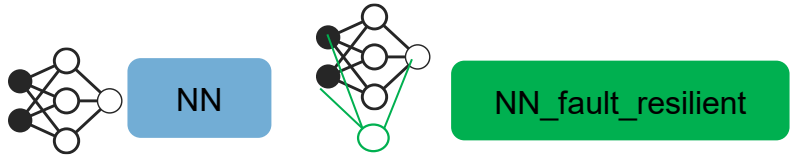
Bigger NN



5%-12% EDC rate

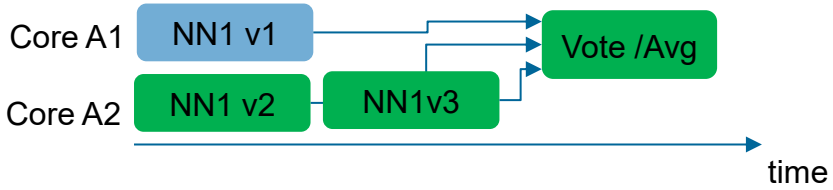
Training

Fault-Tolerant NN Training



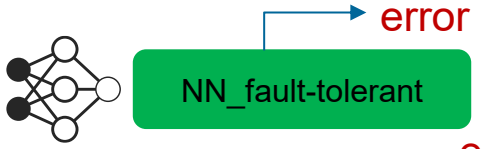
Task Planning

Redundant Execution (DMR/TMR)



Deployment ML Compiler

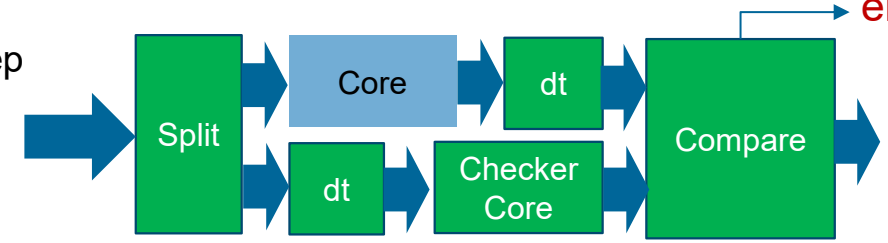
Algorithm-based Fault Tolerance (ABFT)
SW-implemented HW Fault Tolerance (SIHFT)



0.0025%-0.045% EDC rate
10-30 % Overhead

HW Architecture

Lock-step, dual-lock-step
Rad-hard
DMR,TMR
ECC, Parity



Do we want to protect ML workloads?

- Base accuracy a,
- Accuracy with Soft Errors: $a^* \sim= a - a_{loss} = a - (\text{EDCrate} * \text{SoftErrors / Inference})$
- Accuracy drop can be minimal (PI TIMES THUMB estimation):

  "10 Gbits of SRAM and *an* SER of 600 FITs *per* megabit can experience *an error* every 170 *hours*"*

  *10 inferences / sec, 10% EDCrate: soft error / (170h * 360sec/h * 10 inferences/sec)*

  a_{loss} *= 10% * 1/612 000 = 1,63e-5% (1 in 6 mio inference runs)*

Using the 10-30% making the model bigger might give you more accuracy than you loose by EDC
But: No detection (monitoring of SDC rates)

*https://pld.ttu.ee/IAF0030/454636.pdf

Thanks

- Contributors
  - Marc Greim
  - Uzair Sharif
  - Rafael Stahl
  - Philipp van Kempen
  - Karsten Emrich
  - Conrad Foik
  - Johannes Geier
  - Leonidas Kontopoulos
  - Jefferson Parker Jones
  - Johannes Kappes
  - …

Thank you for your attention.