AccHASHTAG: Accelerated Hashing for Detecting Fault-Injection Attacks on Embedded Neural Networks

Nojan Sheybani[1], **Jung-Woo Chang**[1], **Mojan Javaheripi**[1], **and Farinaz Koushanfar**[1]
{nsheyban, juc023, mojavahe, farinaz}@ucsd.edu
[1]University of California San Diego

Abstract

❖ Presenting AccHASHTAG, a highly-accurate **real time fault detection** methodology for DNNs deployed in embedded applications

❖ Leveraging **Algorithm/Software/Hardware co-design** approach to develop AccHASHTAG. AccHASHTAG incorporates a lightweight methodology that ensures low-overhead fault detection

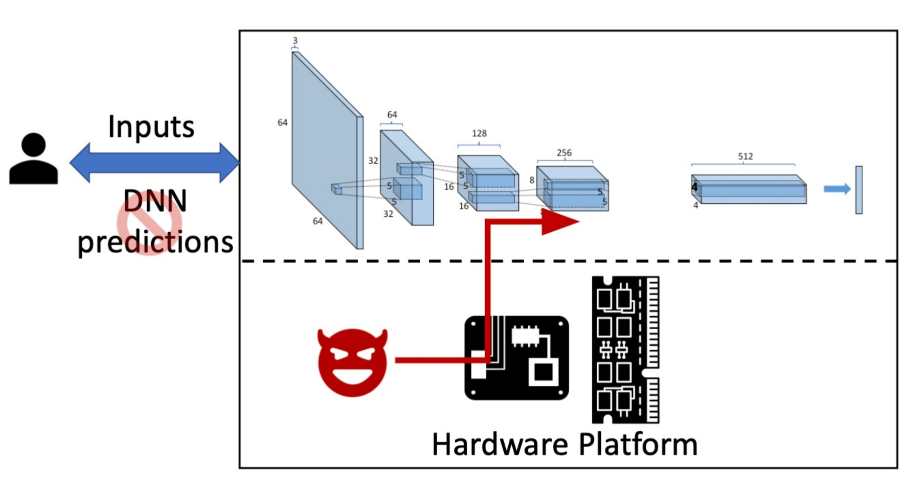
❖ Delivers **0% false positive** and has **no effect on inference accuracy**

❖ Presenting AccHASHTAG's **effectiveness, reliability**, and **efficiency** on various DNN benchmarks

Motivation

❖ Changing **a few bits** of the victim DNN's weights can: (1) Reduce the classification accuracy below a random guess or (2) Cause misprediction to attacker's desired output class

❖ There are many defenses of bitflip attacks, but no prior works with 0% false positive rate and provable performance bounds



Methodology

❖ AccHASHTAG consists of two phases:

1 Pre-Processing: **One-time process** where mechanism is calibrate for victim's DNN. Sensitivity analysis is performed to find top-k most vulnerable layers, called **checkpoint layers**. **Parameter sensitivity** is defined as the effect of per-layer weight change (P) on DNN loss ($\mathcal{L}$):

$$S(p_n) = (\mathcal{L}(P) - \mathcal{L}(\tilde{P}|\widetilde{p_n} = -p_n))^2 \approx 2p_n \frac{\partial \mathcal{L}}{\partial p_n} \; Taylor$$

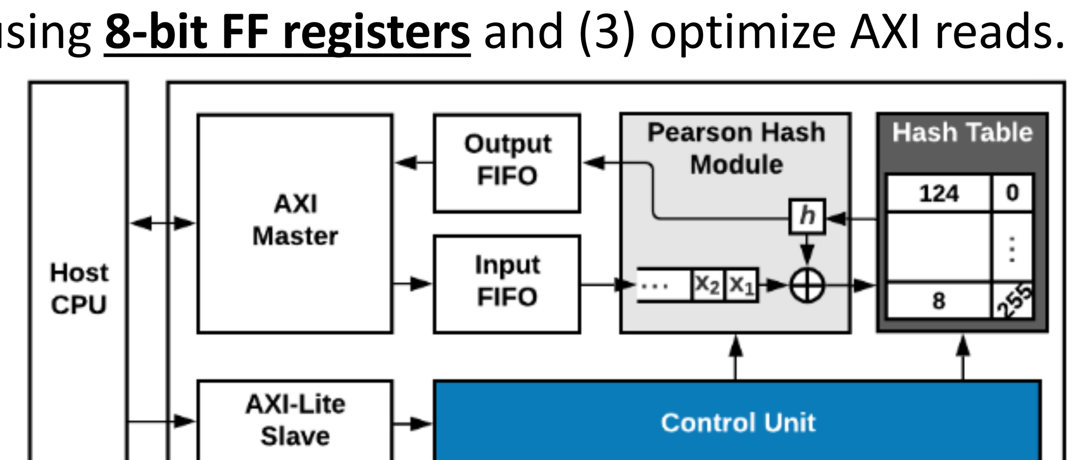
$$S(p_n) = p_n \frac{\partial \mathcal{L}}{\partial p_n}$$

Layer sensitivity is defined as the average top-5 sensitivity of the enclosed parameters.

2 Online Execution: New **Pearson hashes** are extracted from checkpoint layers **in parallel** to each DNN inference. Hashes are **validated against ground-truth hashes** gathered in pre-processing phase. Upon hash-mismatch, an alarm is raised and ground-truth weights are reloaded. Operations in this phase are **done on a customized FPGA.**

Hardware Optimization

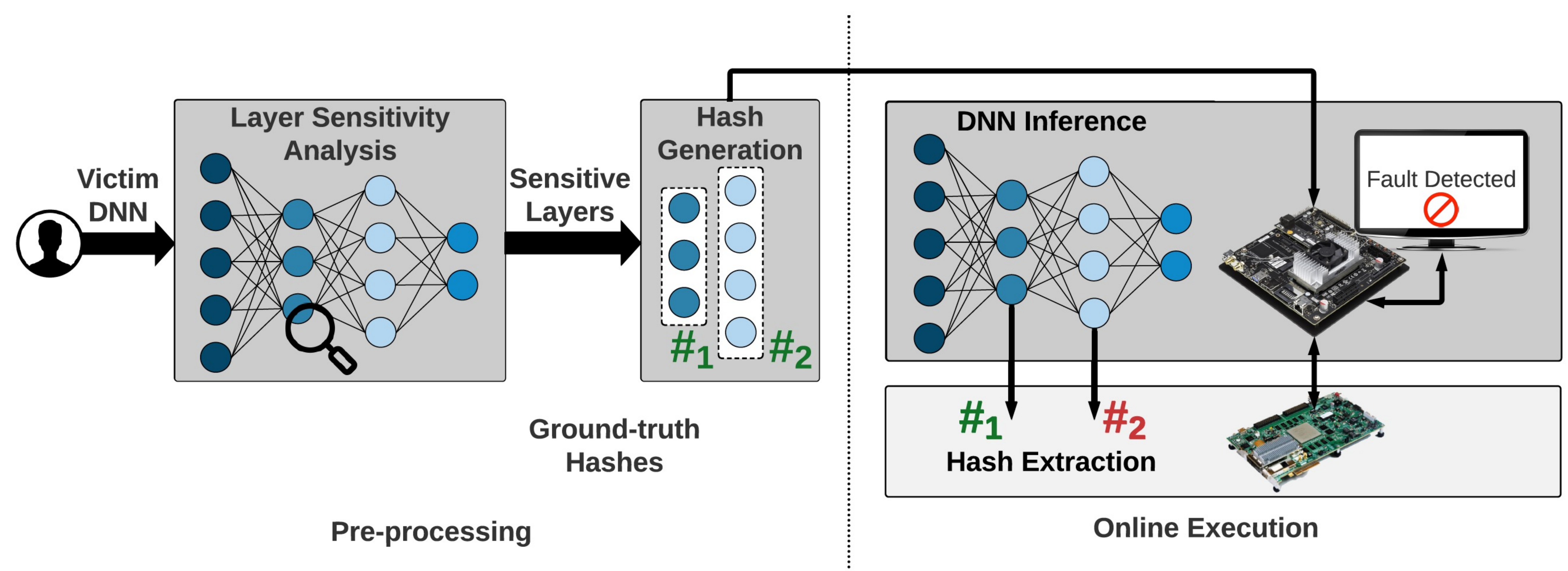
❖ To increase throughput of the online execution phase, we (1) **parallelize** Pearson Hash module with **deep pipelining** (2) implement hash tables entirely using **8-bit FF registers** and (3) optimize AXI reads.



AccHASHTAG's Global Flow

❖ **Threat Model**
- The attacker has access to (1) victim DNN architecture and parameters (2) physical address of model parameters and (3) subset of data used for training DNN
- The attacker performs a Row-Hammer Attack on DRAM locations corresponding to vulnerable parameters, to ensure stealthiness and reduce RHA overhead

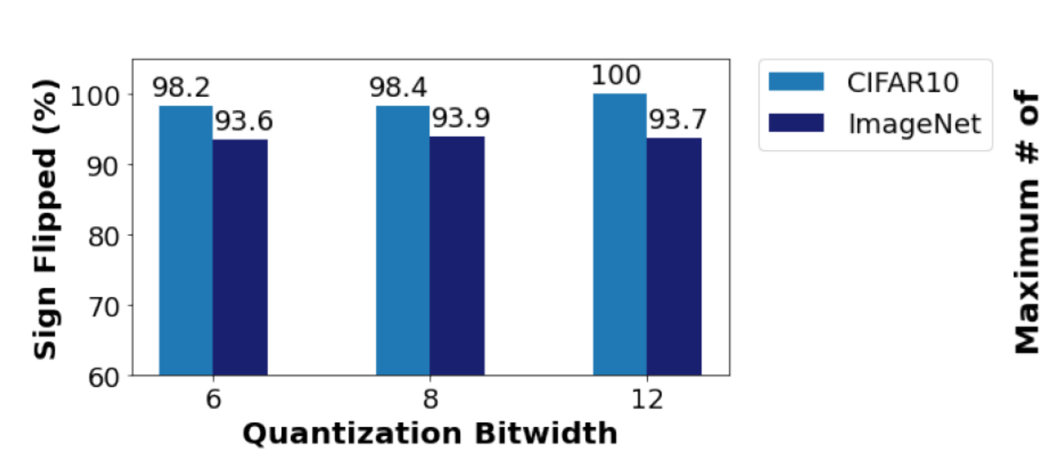
❖ AccHASHTAG computes ground-truth hashes and detects fault-injection attacks in **real-time** using hashes computed on an FPGA in communication with the host CPU.

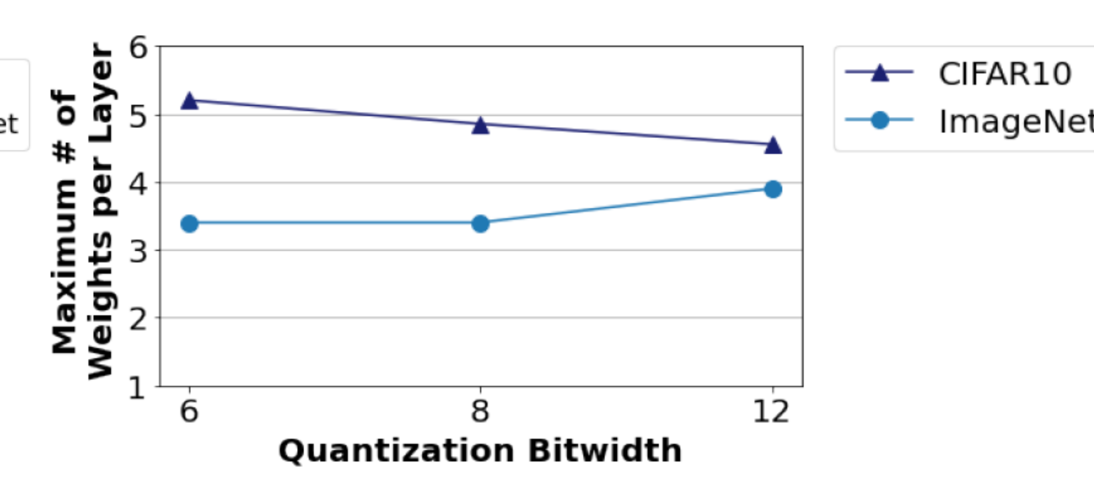


Experimental Results

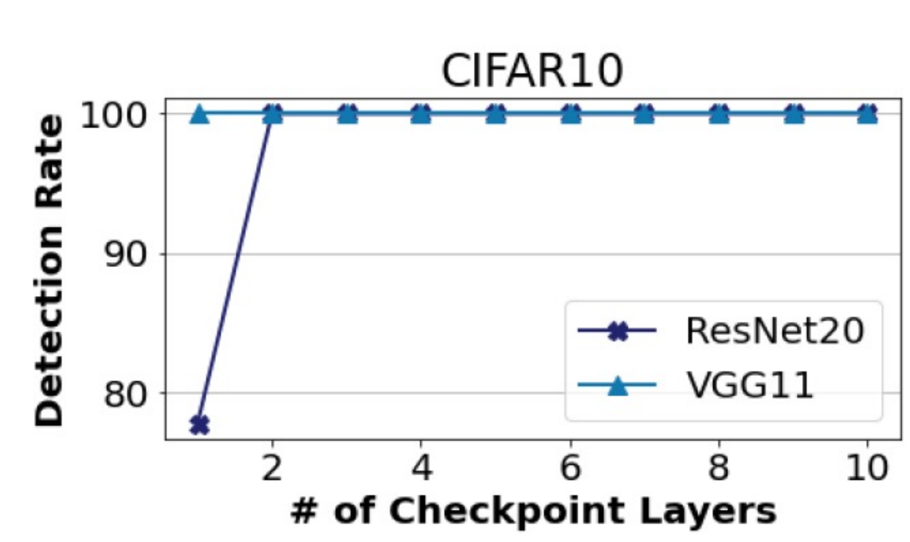
❖ We evaluate AccHASHTAG on various benchmarks to corroborate its properties:

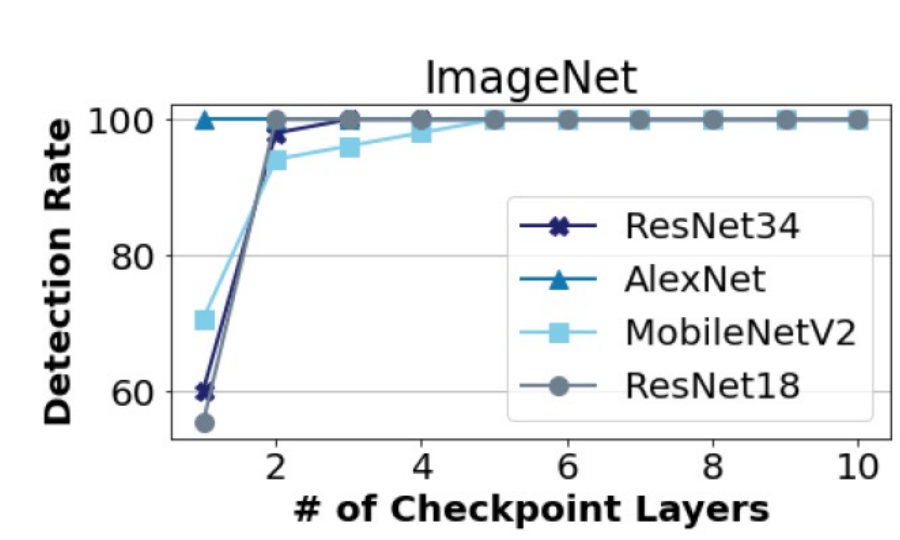
- **Attack Analysis:** We show that oftentimes the attacks are focused on the **sign bits** of the weights, and layers are not targeted **more than around 5 times** by an attacker.





- **Detection Performance:** AccHASHTAG achieves **100% detection rate** with very few checkpoints. **2 checkpoint layers** for CIFAR10 and **5 checkpoint layers** for ImageNet.





- **Overhead Analysis**: AccHASHTAG delivers perfect detection performance while incurring a negligible storage and computation cost, making it suitable for **embedded DNN applications**. The FPGA modules enable **1.5-2.6x faster hash generation** compared to CPU execution

Dataset	Model	Layers	Top-1 Acc (%)	Bit Flips
CIFAR10	VGG11	8 CONV, 3 FC	89.3	90
	ResNet20	19 CONV, 1 FC	91.9	18
ImageNet	AlexNet	5 CONV, 3 FC	55.5	25
	ResNet18	20 CONV, 1 FC	68.8	8
	ResNet34	36 CONV, 1FC	72.8	9
	MobileNet	52 CONV, 1 FC	70.3	3

Benchmark	#	DNN Inference (ms)		CPU Detection		FPGA Detection
		CPU	GPU	Storage (%)	Time (ms)	Time (ms)
VGG11	1	1698.4	110.7	3e-3	0.009	0.003
ResNet20	2	654.8	59.4	2e-2	0.012	0.005
AlexNet	1	7957.9	240.7	4e-4	0.928	0.614
ResNet18	2	20938.8	198.5	4e-3	0.066	0.035
ResNet34	3	40870.6	229.7	3e-3	1.889	1.059
MobileNet	5	2313.6	182.2	4e-2	0.020	0.007

Conclusion

❖ Presenting AccHASHTAG, a fault-injection detection methodology that incurs negligible storage and runtime overhead on resource-constrained embedded devices.

❖ Leveraging Algorithm/Software/Hardware **co-design** principle to achieve **100% detection rate** with **0% false alarms** and guaranteed detection performance with **provable statistical bounds**