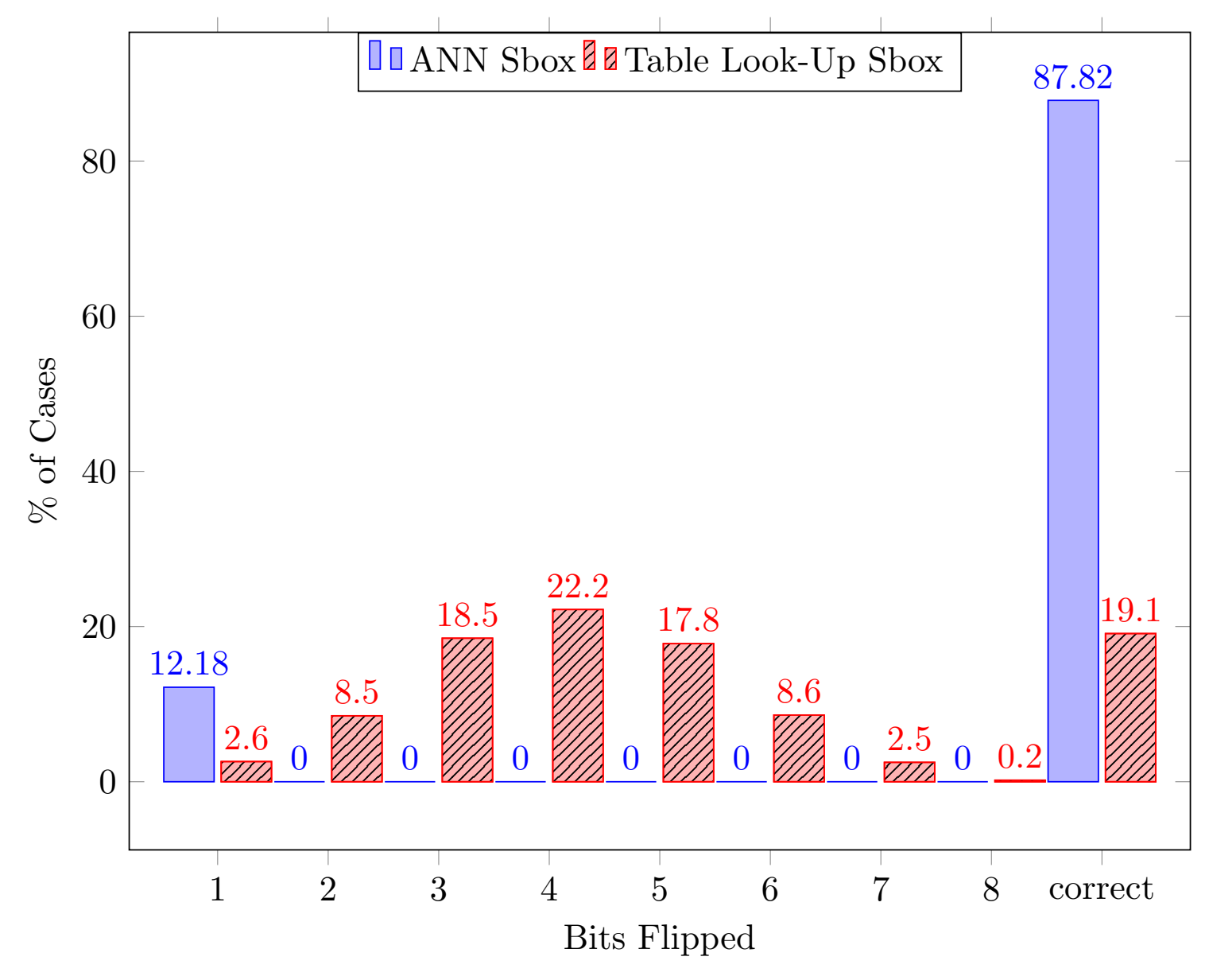
Enhancing Fault Tolerance of Neural Networks for Security-Critical Applications

Manaar Alam[1], Arnab Bag[1], Debapriya Basu Roy[1], Dirmanto Jap[2], Jakub Breier[3], Shivam Bhasin[2], and Debdeep Mukhopadhyay[1]

[1]Department of Computer Science and Engineering, IIT Kharagpur,[2]Temasek Laboratories, NTU, Singapore, [3]Underwriters Laboratories, Singapore

1. Introduction

- Cryptographic applications suffer from targeted faults from powerful adversaries.
- NNs exhibit "some degree" of robustness functioning almost correctly even after the faults in any of its parameters.
- $f = SBox(x \oplus k)$ is learned using NN and faults are injected in the parameters.



- We develop a highly fault tolerant cryptographic primitive like AES SBox using NN having higher degree of fault tolerance than the standard implementation.
- We implement the fault tolerant NN architecture in an FPGA with tailored implementation strategies.

2. Fault Model and Assumption

- Learning phase is fault-free. Faults can be injected during the classification phase.
- We consider *single location* fault model.
- An adversary can employ single-bit flip, multiple-bit flips or zero/random values.

3. Design Idea vs. Fault Tolerance

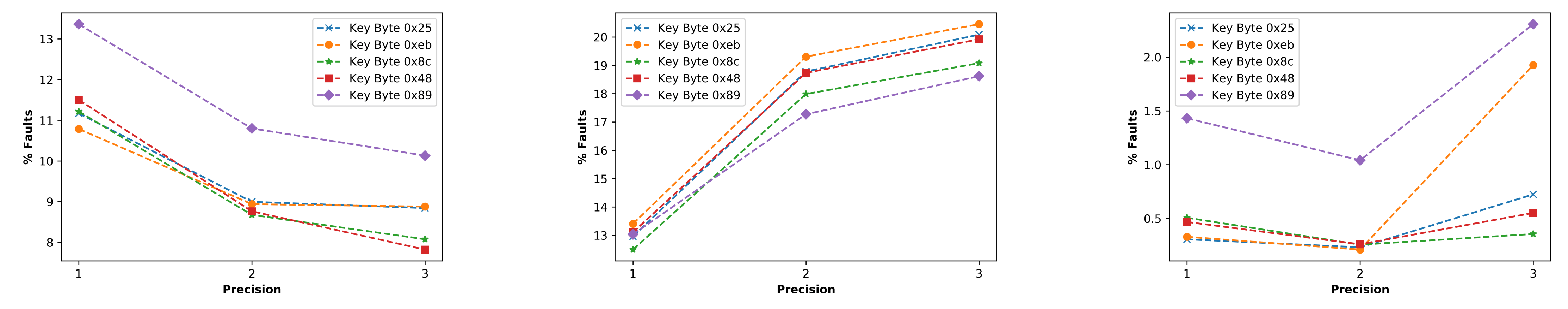


Figure 1: Effect of Selecting Precision on Fault Tolerance

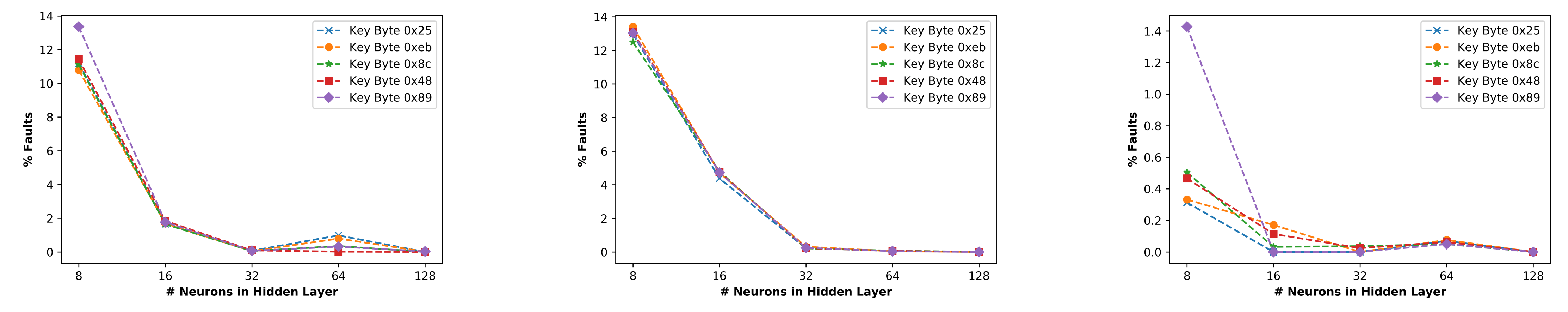


Figure 2: Effect of Number of Neurons in Hidden Layer on Fault Tolerance

4. Conditions for Implementing Fully Fault-Tolerant Architecture

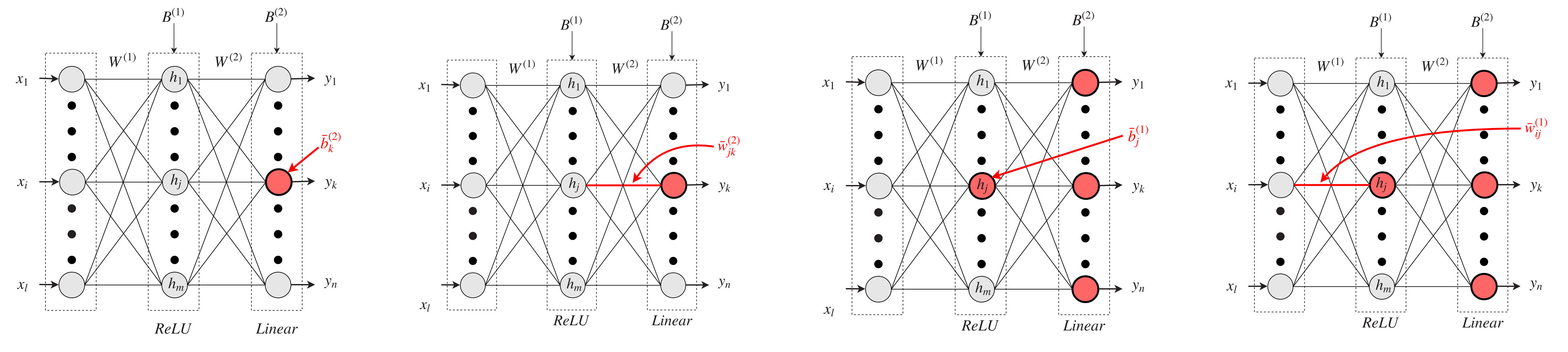


Figure 3: Effect of Fault at Different Locations

The constraints obtained for different weight parameters.

$$\delta < \begin{cases} y_c - y_{f_2}, & \text{if } c \neq f_2 \\ y_{f_2} - y_r, & \text{otherwise, for all } r \end{cases}$$

$$\delta < \begin{cases} \frac{y_c - y_{f_2}}{h_{f_1}}, & \text{if } c \neq f_2 \\ \frac{y_{f_2} - y_r}{h_{f_1}}, & \text{otherwise, for all } r \end{cases}$$

$$\delta < \pm \frac{y_c - y_r}{w^{(2)}_{f_1 r} - w^{(2)}_{f_1 c}}$$

$$\delta < \pm \frac{y_c - y_r}{w^{(1)}_{f_0 f_1} w^{(2)}_{f_1 r} - w^{(1)}_{f_0 f_1} w^{(2)}_{f_1 c}}$$

These constraints are used while training the NN.

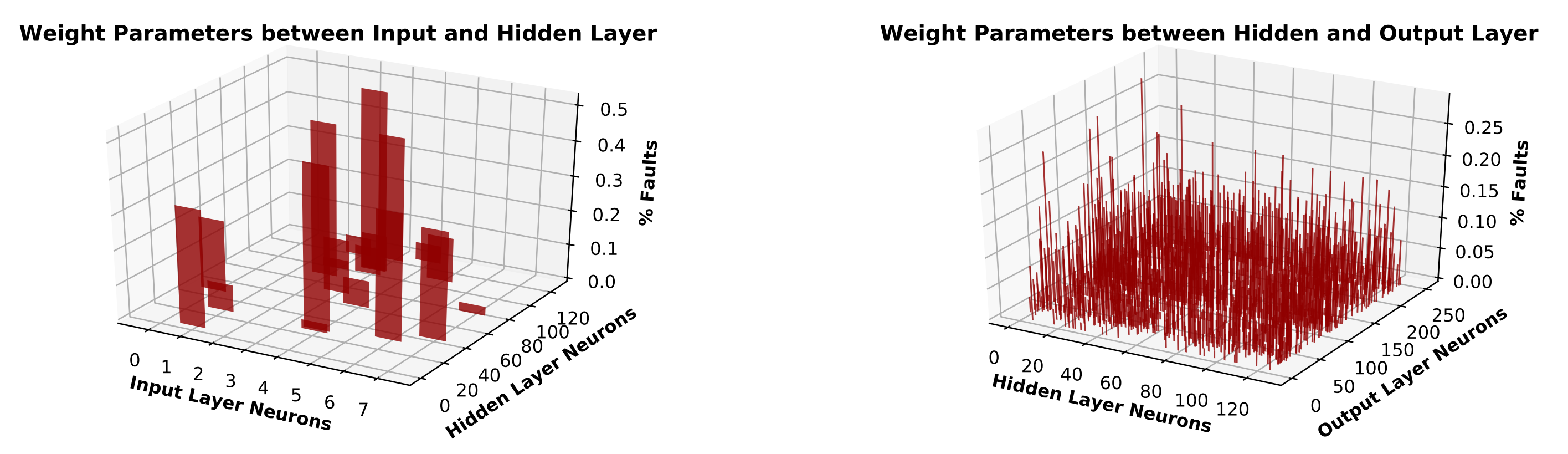


Figure 4: Effect of weight parameters on fault for Standard NN

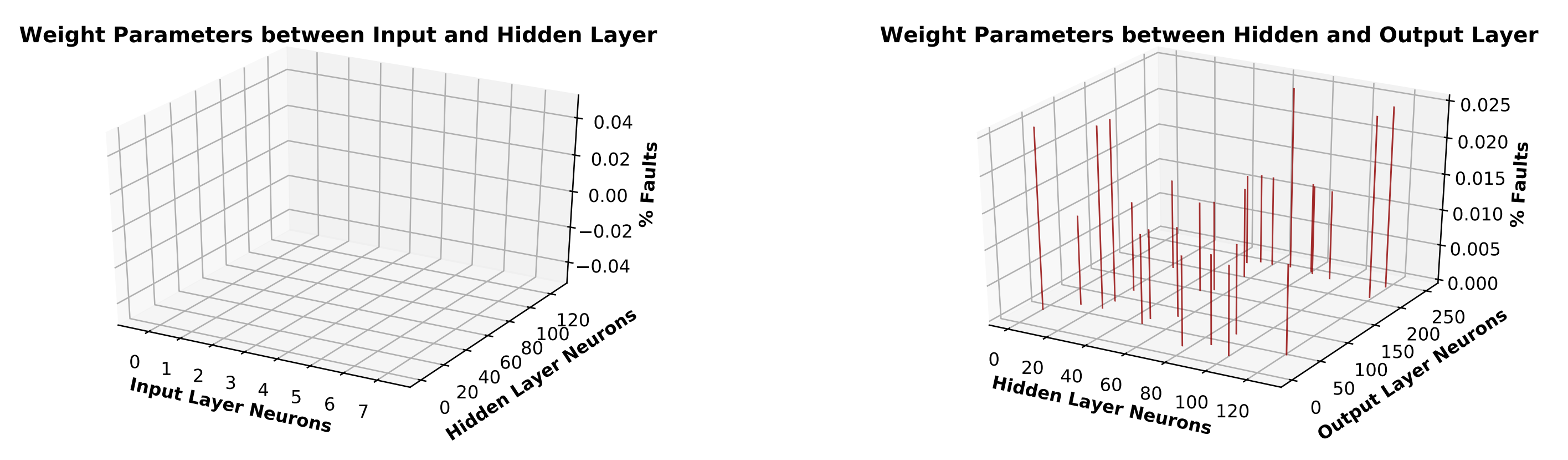


Figure 5: Effect of weight parameters on fault for Modified NN

5. Implementation on FPGA

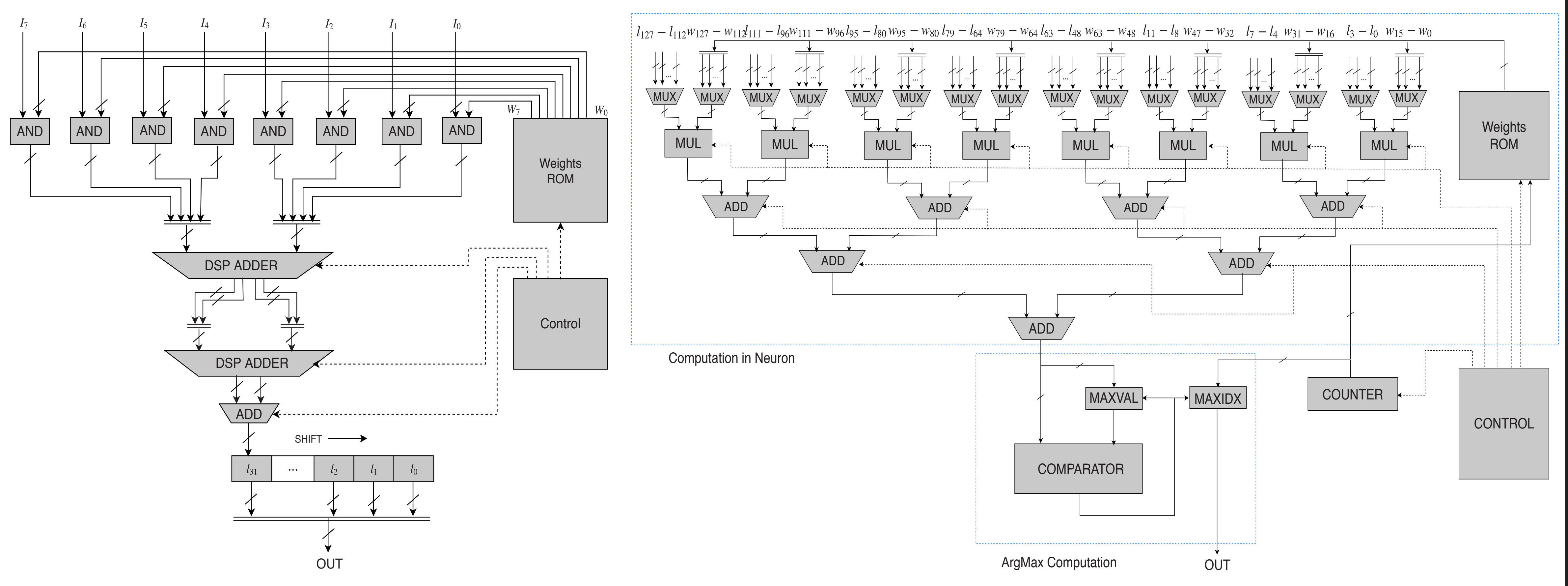


Figure 6: Top Level Architecture of the Hidden Layer and Output Layer

6. Initial Results

Table 1: Post Place & Route Resource Utilisation for Artrix-7 FPGA for Different NNs

Design	#Slice (%)	#LUT (%)	#Register (%)	#DSP (%)	#BRAM (%)	Freq. (MHz)	#Clock Cycle	Delay (us)	% Faults
8-8-256	127 (1.55)	324 (1.56)	199 (0.47)	33 (36.67)	5 (10)	151.95	1350	8.88	0.16
8-32-256	341 (4.18)	934 (4.49)	951 (2.29)	33 (36.67)	4 (8)	149.43	25576	171.15	0.04
8-64-256	427 (5.24)	978 (4.70)	1007 (2.43)	33 (36.67)	6 (12)	141.40	49352	349.01	2.7×10^{-3}
8-128-256	601 (7.37)	1442 (6.93)	1657 (3.98)	33 (36.67)	9 (18)	128.66	96910	753.18	1.2×10^{-5}
LUT-based	24 (0.29)	80 (0.40)	17 (0.40)	0 (00)	0 (00)	259.80	1	3.85 $\times 10^{-3}$	100

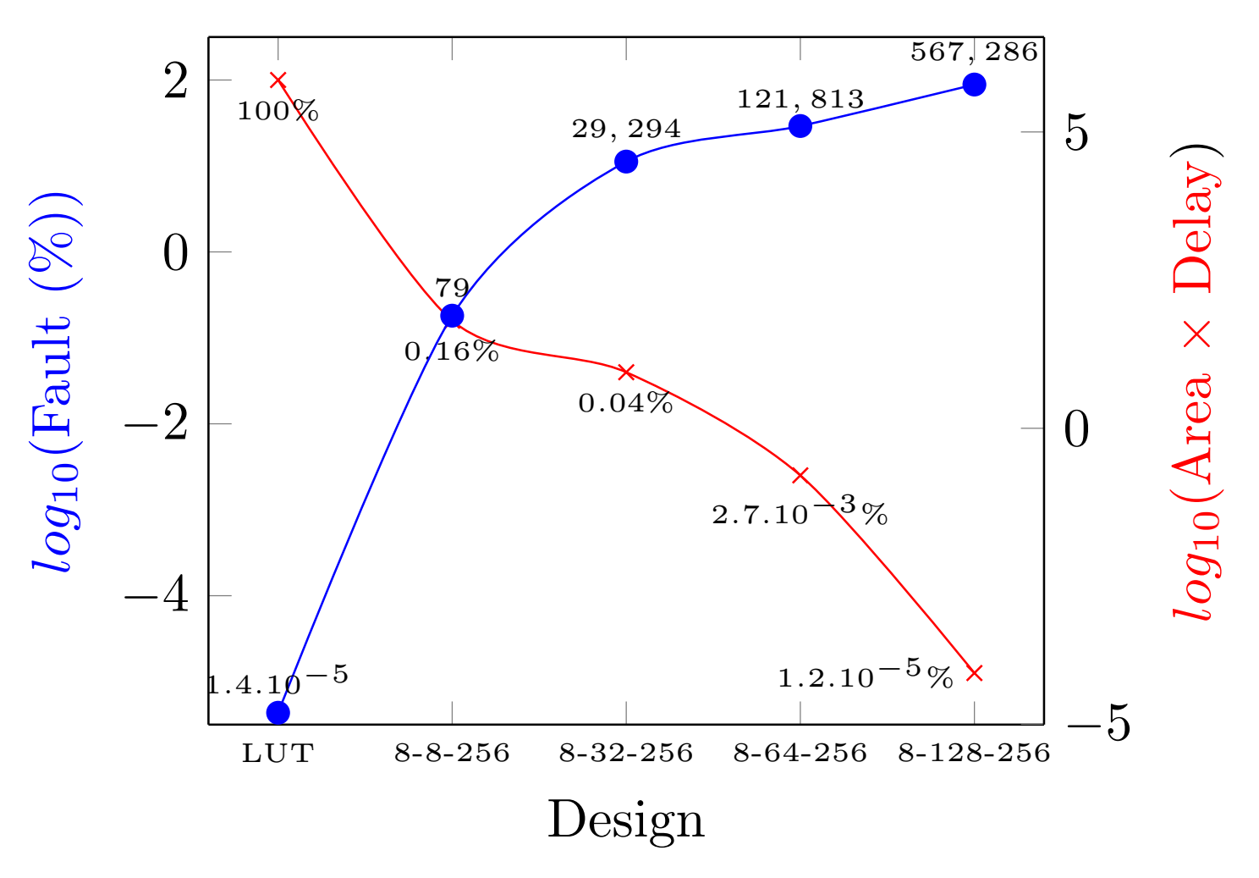


Figure 7: %Faults Tolerance vs. Overhead

7. Conclusion

- Proposed method is able to generate highly fault tolerant design of AES SBox (with key addition).
- Implementation overhead increases with the increase in fault tolerance.