UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Institute ITACA
Information and Communication Technologies

On Improving the Robustness Of Convolutional Neural Networks

Juan-Carlos Ruiz-García

ITACA-UPV (**Spain**)

jcruizg@disca.upv.es

IFIP WG 10.4 SUMMER MEETING

27th-30th JUNE 2024, GOLD COAST (AUSTRALIA)

CNNs in Cyber-Physical Systems

- Use of Convolutional neuronal networks (CNN) for environmental sensing

Transportation

Space exploration

- Real-time constrainsts in decisions → need of local inference → use of HW accelerators to support totally or partially the CNN inference process
- In edge devices, robustness must be considered attending to PPA properties (performance, power consumption and area)

Prototyping HW-based CNN acceleration solutions for edge devices

Graphics Processing (**GPU**)

- ✓ Performance
- × Energy consumption

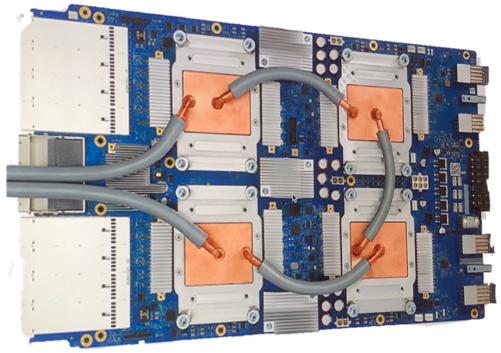
Prototyping HW-based CNN acceleration solutions for edge devices

Graphics Processing (**GPU**)

- ✓ Performance
- × Energy consumption

Tensor processing (**TPU**)

- ✓ Performance
- ✓ Energy consumption
- × Flexibility



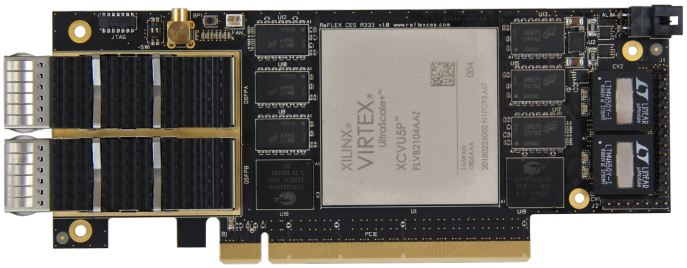
Prototyping HW-based CNN acceleration solutions for edge devices

Graphics Processing (**GPU**)

- ✓ Performance
- × Energy consumption

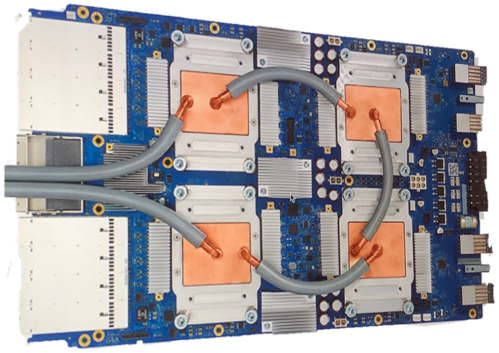
Field-Programmable Gate Array (**FPGA**)

- ✓ Performance per watt
- ✓ Flexibility and adaptability
- × Design



Tensor processing (**TPU**)

- ✓ Performance
- ✓ Energy consumption
- × Flexibility



FPGA-based designs ...

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DualPortRegisterFile is
    Generic (ADDRESS_SIZE : POSITIVE;
             REGISTER_SIZE : POSITIVE);
    Port ( rst_i : in STD_LOGIC;
           clk_i : in STD_LOGIC;
           en_i : in STD_LOGIC;
           write_en_i : in STD_LOGIC;
           readReg1_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           readReg2_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           writeReg_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           writeData_i : in STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0);
           readData1_o : out STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0);
           readData2_o : out STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0));
end DualPortRegisterFile;

architecture Behavioral of DualPortRegisterFile is
    type RegFile is array (0 to (2**ADDRESS_SIZE)-1) of STD_LOGIC_VECTOR(REGISTER_SIZE-1 downto 0);
    signal registers : RegFile := (others => (others => '0'));
begin

    process(rst_i, clk_i)
    begin
        if rst_i = '1' then
            registers <= (others => (others => '0'));
        elsif rising_edge(clk_i) then
            if en_i = '1' then
                if write_en_i = '1' then
                    registers(to_integer(unsigned(writeReg_i))) <= writeData_i;
                end if;
            end if;
        end if;
    end process;

    readData1_o <= registers(to_integer(unsigned(readReg1_i)));
    readData2_o <= registers(to_integer(unsigned(readReg2_i)));

end Behavioral;
```

RTL design

FPGA-based designs ...


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DualPortRegisterFile is
    Generic (ADDRESS_SIZE : POSITIVE;
             REGISTER_SIZE : POSITIVE);
    Port ( rst_i : in STD_LOGIC;
           clk_i : in STD_LOGIC;
           en_i : in STD_LOGIC;
           write_en_i : in STD_LOGIC;
           readReg1_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           readReg2_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           writeReg_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           writeData_i : in STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0);
           readData1_o : out STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0);
           readData2_o : out STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0));
end DualPortRegisterFile;

architecture Behavioral of DualPortRegisterFile is
    type RegFile is array (0 to (2**ADDRESS_SIZE)-1) of STD_LOGIC_VECTOR(REGISTER_SIZE-1 downto 0);
    signal registers : RegFile := (others => (others => '0'));
begin

    process(rst_i, clk_i)
    begin
        if rst_i = '1' then
            registers <= (others => (others => '0'));
        elsif rising_edge(clk_i) then
            if en_i = '1' then
                if write_en_i = '1' then
                    registers(to_integer(unsigned(writeReg_i))) <= writeData_i;
                end if;
            end if;
        end if;
    end process;

    readData1_o <= registers(to_integer(unsigned(readReg1_i)));
    readData2_o <= registers(to_integer(unsigned(readReg2_i)));

end Behavioral;
```

RTL design

Synthesis

FPGA-based designs ...


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DualPortRegisterFile is
    Generic (ADDRESS_SIZE : POSITIVE;
             REGISTER_SIZE : POSITIVE);
    Port ( rst_i : in STD_LOGIC;
           clk_i : in STD_LOGIC;
           en_i : in STD_LOGIC;
           write_en_i : in STD_LOGIC;
           readReg1_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           readReg2_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           writeReg_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           writeData_i : in STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0);
           readData1_o : out STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0);
           readData2_o : out STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0));
end DualPortRegisterFile;

architecture Behavioral of DualPortRegisterFile is
    type RegFile is array (0 to (2**ADDRESS_SIZE)-1) of STD_LOGIC_VECTOR(REGISTER_SIZE-1 downto 0);
    signal registers : RegFile := (others => (others => '0'));
begin

    process(rst_i, clk_i)
    begin
        if rst_i = '1' then
            registers <= (others => (others => '0'));
        elsif rising_edge(clk_i) then
            if en_i = '1' then
                if write_en_i = '1' then
                    registers(to_integer(unsigned(writeReg_i))) <= writeData_i;
                end if;
            end if;
        end if;
    end process;

    readData1_o <= registers(to_integer(unsigned(readReg1_i)));
    readData2_o <= registers(to_integer(unsigned(readReg2_i)));

end Behavioral;
```

RTL design

Synthesis

Placement and routing

FPGA-based designs ...


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DualPortRegisterFile is
    Generic (ADDRESS_SIZE : POSITIVE;
             REGISTER_SIZE : POSITIVE);
    Port ( rst_i : in STD_LOGIC;
           clk_i : in STD_LOGIC;
           en_i : in STD_LOGIC;
           write_en_i : in STD_LOGIC;
           readReg1_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           readReg2_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           writeReg_i : in STD_LOGIC_VECTOR (ADDRESS_SIZE-1 downto 0);
           writeData_i : in STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0);
           readData1_o : out STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0);
           readData2_o : out STD_LOGIC_VECTOR (REGISTER_SIZE-1 downto 0));
end DualPortRegisterFile;

architecture Behavioral of DualPortRegisterFile is
    type RegFile is array (0 to (2**ADDRESS_SIZE)-1) of STD_LOGIC_VECTOR(REGISTER_SIZE-1 downto 0);
    signal registers : RegFile := (others => (others => '0'));
begin

    process(rst_i, clk_i)
    begin
        if rst_i = '1' then
            registers <= (others => (others => '0'));
        elsif rising_edge(clk_i) then
            if en_i = '1' then
                if write_en_i = '1' then
                    registers(to_integer(unsigned(writeReg_i))) <= writeData_i;
                end if;
            end if;
        end if;
    end process;

    readData1_o <= registers(to_integer(unsigned(readReg1_i)));
    readData2_o <= registers(to_integer(unsigned(readReg2_i)));

end Behavioral;
```

Electronic Design Automation (**EDA**) toolkit

- RTL design
- Synthesis
- Placement and routing
- Bitstream generation

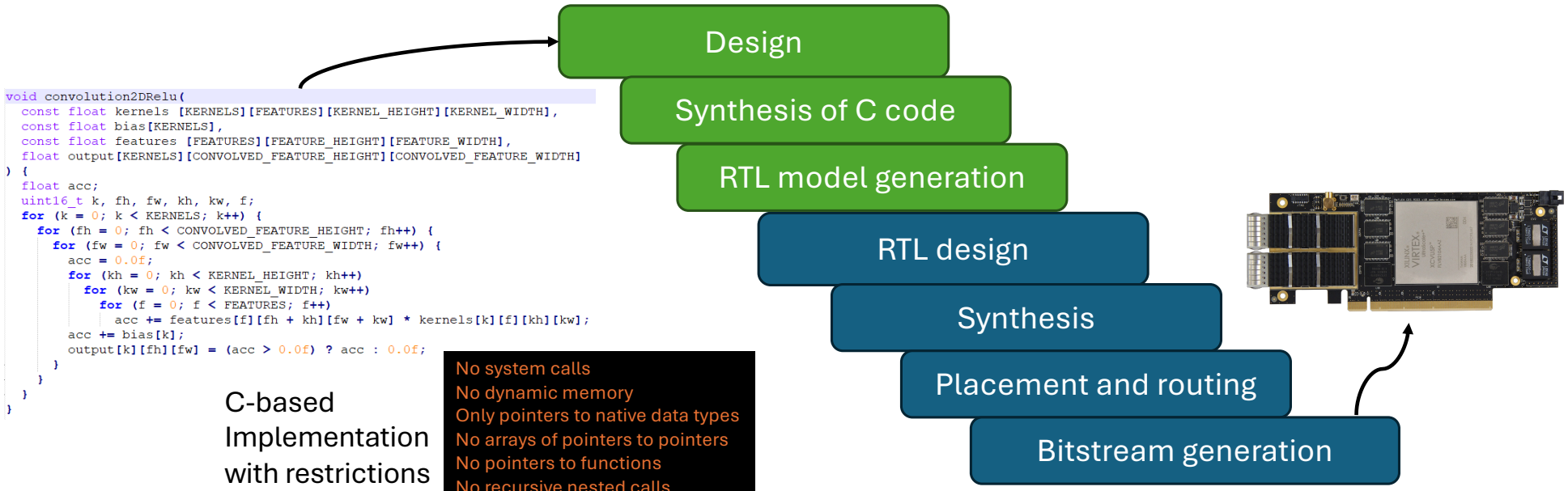
FPGA-based designs ... using HLS

- Use of High-Level Synthesis (**HLS**) tools to prototype CNNs on FPGAs that have been designed using high-level programming languages

```
void convolution2DRelu(
  const float kernels [KERNELS][FEATURES][KERNEL_HEIGHT][KERNEL_WIDTH],
  const float bias[KERNELS],
  const float features [FEATURES][FEATURE_HEIGHT][FEATURE_WIDTH],
  float output[KERNELS][CONVOLVED_FEATURE_HEIGHT][CONVOLVED_FEATURE_WIDTH]
) {
  float acc;
  uint16_t k, fh, fw, kh, kw, f;
  for (k = 0; k < KERNELS; k++) {
    for (fh = 0; fh < CONVOLVED_FEATURE_HEIGHT; fh++) {
      for (fw = 0; fw < CONVOLVED_FEATURE_WIDTH; fw++) {
        acc = 0.0f;
        for (kh = 0; kh < KERNEL_HEIGHT; kh++)
          for (kw = 0; kw < KERNEL_WIDTH; kw++)
            for (f = 0; f < FEATURES; f++)
              acc += features[f][fh + kh][fw + kw] * kernels[k][f][kh][kw];
        acc += bias[k];
        output[k][fh][fw] = (acc > 0.0f) ? acc : 0.0f;
      }
    }
  }
}
```

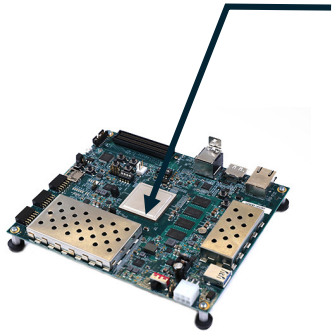
C-based Implementation with restrictions

- No system calls
- No dynamic memory
- Only pointers to native data types
- No arrays of pointers to pointers
- No pointers to functions
- No recursive nested calls

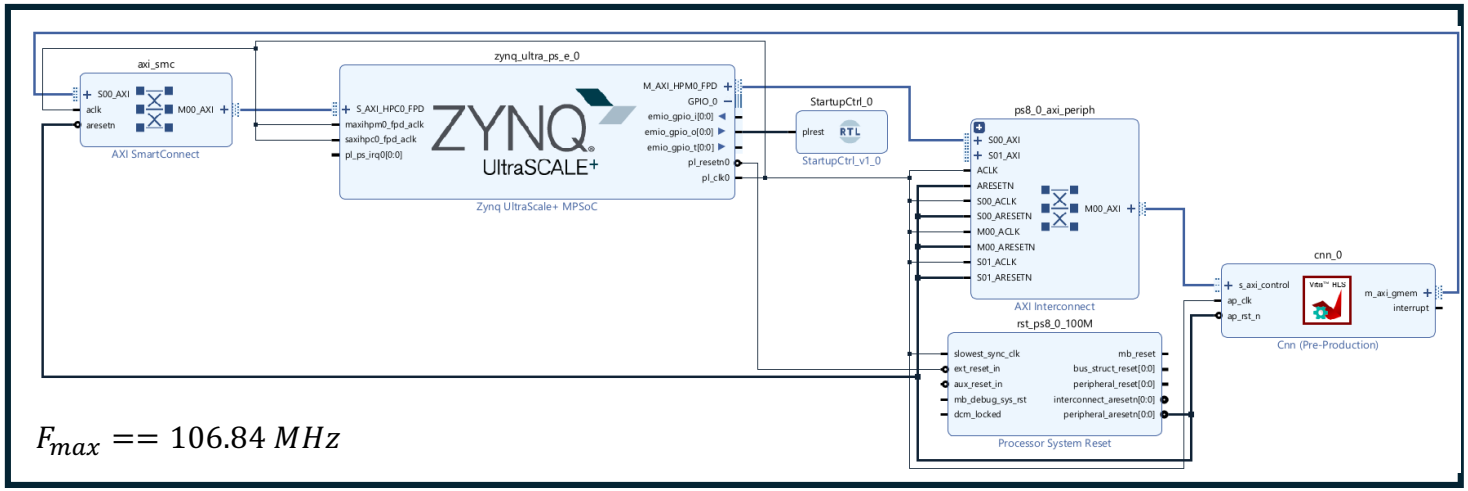


Prototyping workflow of Lenet-5

C-based Lenet-5 (Lenet5FloatHLS) → AMD Vitis: Simulation, synthesis, co-simulation and RTL generation → RTL Lenet-5 model → AMD Vivado Design Suite: Synthesis, placement and routing → Zynq Ultrascale+ BitStream

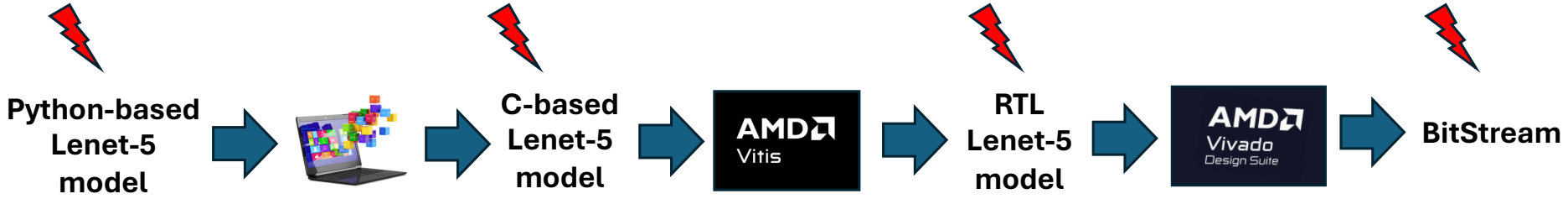


ZCU104 prototyping board



$F_{max} == 106.84\ MHz$

Robustness evaluation process



- Multiplicity and location of faults
  (single vs multiple faults, accidental vs malicious faults)
- Duration of the fault injection campaigns: use of statistical fault injection
- Fault injection process (reproducibility, representativity, CNN optimizations)
  - Which low-level faults can be emulated using high-level models? How?
  - How do CNN optimizations impact the fault injection process?

Use of evaluation results for protection without CNN retraining

- **[EDCC 2024]**[1] Idenfication of non-significant bits + use of those bits to hold ECC parity errors

* Note: The concrete división between red and green bits will vary from one CNN to another

- **[SAFECOMP 2024]**[2] Use of non-significant and invariant bits for BF16 CNN protection with ECCs

* Note: The concrete división between blue, red and green bits will vary from one CNN to another

[1] **[EDCC 2024]** Juan Carlos Ruiz, David de Andrés, Luis J. Saiz-Adalid, Joaquin Gracia-Moran: Zero-Space In-Weight and In-Bias Protection for Floating-Point-based CNNs. EDCC 2024: 89-96, Lovaina (Bélgica), Abril 2024.

[2] **[SAFECOMP 2024]** Juan Carlos Ruiz, David de Andrés, Luis J. Saiz-Adalid, Joaquin Gracia-Moran: In-Memory Zero-Space Floating-Point-based CNN Protection Using Non-Signicant and Invariant Bits, SAFECOMP 2024, Florencia (Italia), Septiembre 2024.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Institute ITACA
Information and Communication Technologies

On Improving the Robustness Of Convolutional Neural Networks

Juan-Carlos Ruiz-García

ITACA-UPV (**Spain**)

jcruizg@disca.upv.es

IFIP WG 10.4 SUMMER MEETING

27th-30th JUNE 2024, GOLD COAST (AUSTRALIA)

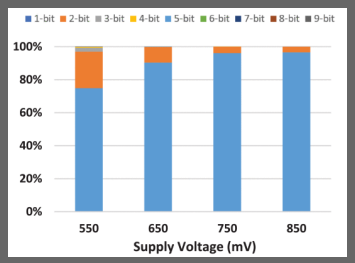
Already known facts

- Intrinsic robustness of CNNs to bitflips and stuck-at faults
- INT8 CNNs are more robust than FP32/BF16 CNNs to single bitflips
  - FP32: 22,84 → 01000001101101101011100001010001
    → 01**1**00001101101101011100001010001 (**4213236374582759000000!!**)
  - INT8: 68 → 01000100
    → 01**1**00100 (100)
- This may not be true in the case of multiple faults
  - FP32: 22,84 → 01000001101101101011100001010001
    → 01100001101101101011100**1010**0001 (no effect on CNN accuracy)
  - INT8: 68 → 01000100
    → **1010**0100 → potential effect on CNN accuracy

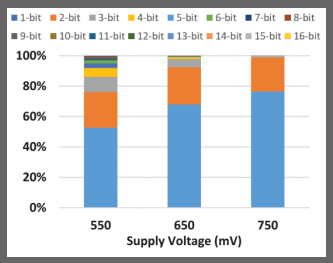
Importance of multi-bitflips

- Accidental faults: The number of bits altered by a single ionizing particle increases as CMOS integration does and voltaje is reduced

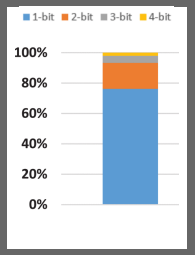
N. J. Pieper et al., "Study of Multicell Upsets in SRAM at a 5-nm Bulk FinFET Node," in *IEEE Transactions on Nuclear Science*, vol. 70, no. 4, pp. 401-409, April 2023



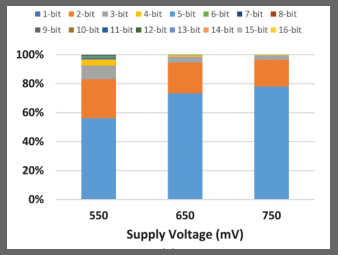
alpha particles



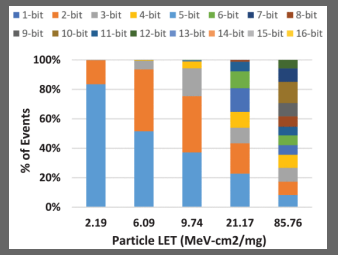
14-MeV neutrons



terrestrial neutrons



thermal neutrons

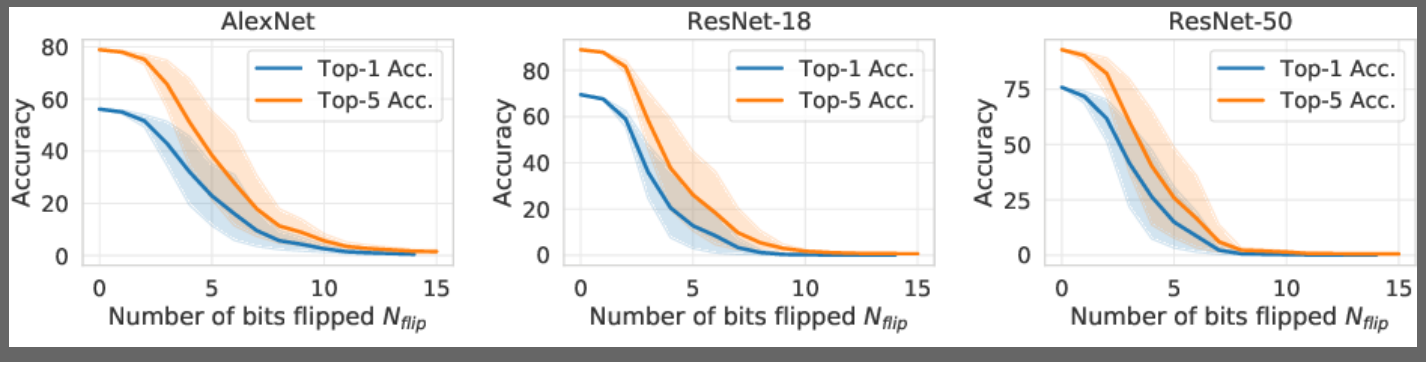


heavy ions

Importance of multi-bitflips

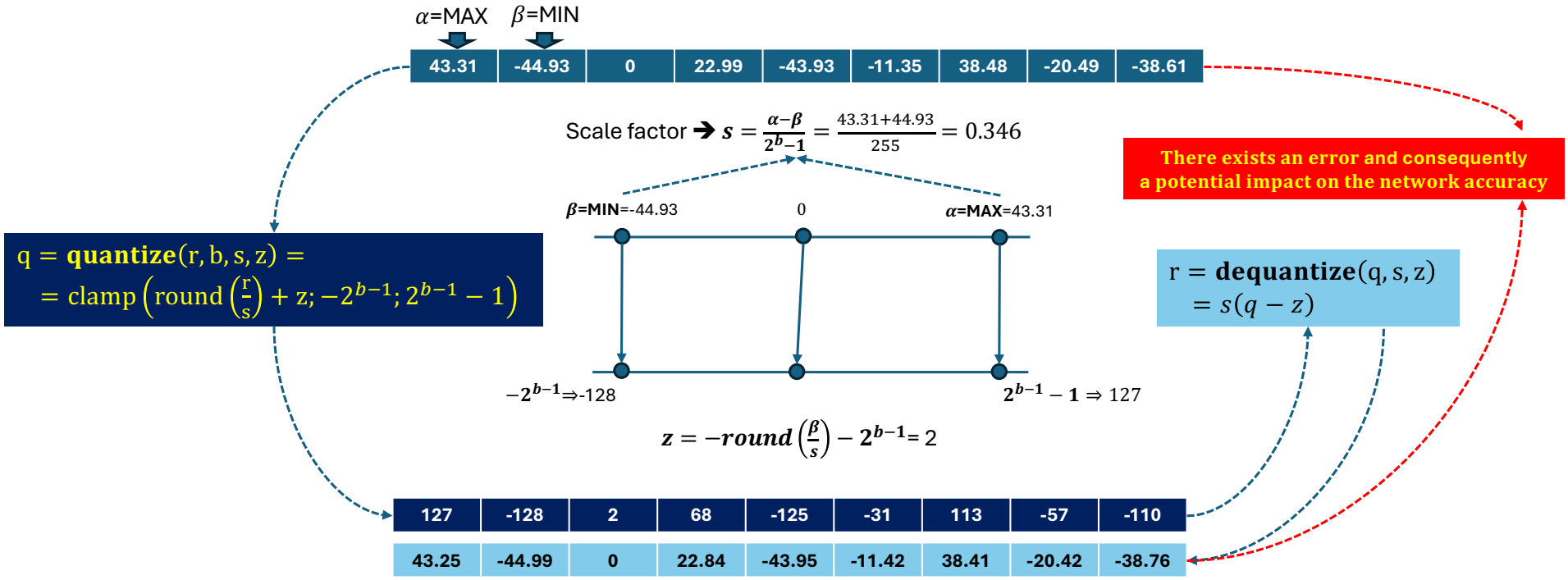
- Malicious faults: A reduced number (5-10) of flipped bits in parameters can lead a CNN to crush

Adnan Siraj Rakin, Zhezhi He, and Deliang Fan, "Bitflip attack: Crushing neural network with progressive bit search" in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 1211–1220, 2019.



Using Integers

- Example using affine (asymmetric) quantization and FP32/BF16→INT8



Operations with quantized numbers*

$$y(j) = B(j) + \sum_{i=0}^{I} x(i) \times w(i,j)$$

biases are adjusted so that → $z_b = 0$ and $S_b = S_x \times S_w$
and remember the dequantization formula → $r_? = s_?(q_?\text{-}z_?)$

$$s_y(q_y\text{-}z_y) = S_b \times q_b + \sum_{i=0}^{I} S_x(q_x - zx) \times S_w(q_w - zw)$$

$$q_y = \frac{S_x \times S_w}{S_y} q_b + \frac{S_x \times S_w}{S_y} [\sum_{i=0}^{I} (q_x - zx) \times (q_w - zw)] + z_y, \text{ where } M_0 = \frac{S_x \times S_w}{S_y} \in [0.1[$$

Tip: $M_0 = 0.111$ → $M'_0 = 2^3 \times M_0$ [shift left]$= 111$ → $M_0 = M'_0 / 2^3$ [shift right]

So $M'_0 = 2^{32} M_0$

Quantized output computation →

$$q_y = \frac{M'_0}{2^{32}} (q_b + [\sum_{i=0}^{I} (q_x - zx) \times (q_w - zw)]) + z_y$$

* [CoRR 2021] Markus Nagel et al. "A White Paper on Neural Network Quantization", CoRR abs/2106.08295 (2021)

C-based implementation

$$q_y = \frac{M'0}{2^{32}} \left(q_b + \left[\sum_{i=0}^{I}(q_x - zx)\times(q_w - zw)\right]\right) + z_y$$

```c
// Applies a linear transformation
// https://pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear
void fullyConnected_1(
                const float input_features [FC1_INPUT_FEATURES],
                const float input_weights [FC1_FEATURES][FC1_INPUT_FEATURES],
                const float bias[FC1_FEATURES],
                float output_features[FC1_FEATURES]) {

        float accumulated;

        uint16_t f;
        uint16_t nif;
        uint16_t nv;

        // Go through all the values of that feature
        fc1_F: for (f = 0; f < FC1_FEATURES; f++) {

                accumulated = 0.0f;

                // For each feature
                fc1_NIF: for (nif = 0; nif < FC1_INPUT_FEATURES; nif++) {

                        accumulated += input_features[nif] * input_weights[f][nif];
                }

                output_features[f] = accumulated + bias[f];
        }

}
```

FP-based computation

precomputed values

Result of previous layer

Output

```c
void fullyConnected(
    const uint8_t q_x[FEATURES],
    const int8_t z_x,
    const int8_t q_w[OUTPUTS][FEATURES],
    const int8_t z_w[OUTPUTS],
    const uint32_t M[FEATURES],
    const int32_t q_b[FEATURES],
    uint8_t q_y [OUTPUTS],
    const int8_t z_y ) {

  int32_t acc;
  int64_t mXacc, y;
  uint16_t j, i;

  for (j = 0; j < OUTPUTS; j++) {
    acc = 0;
    for (i = 0; i < FEATURES; i++)
      acc += (q_x[i] - z_x) * (q_w[j][i] - z_w[j]);
    acc += q_b[j];
    mXacc = (int64_t)m[j] * acc;
    y = (mXacc>>31 & 0x1) ? (mXacc>>32) + 1 : mXacc>>32;
    y += z_y;
    if (y < 0) q_y[j] = (uint8_t)0;
    else if (y > 255) q_y[j] = (uint8_t)255;
    else q_y[j] = (uint8_t)y;
  }
}
```

INT-based computation