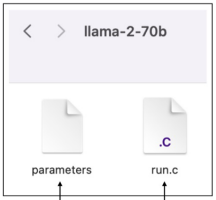
Quantification of Neural Networks on Fixed-Point Architectures with Formal Guarantees

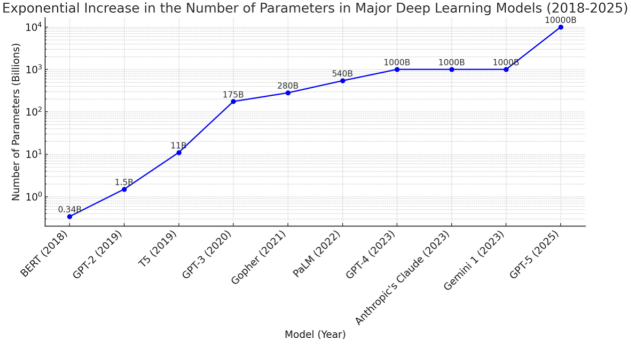
Défi Embarquabilité de l'IA

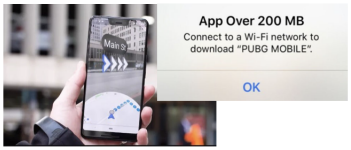
Dorra Ben Khalifa

April 7, 2026

DNNs are significantly over-parametrized!







DNNs are significantly over-parametrized!

- **LLaMA 2-70B: training at scale**
  - ◇ 6,000 NVIDIA A100 GPUs running for 12 days non-stop computation
  - ◇ 1.7 million GPU-hours to fine-tune the model's intelligence
  - ◇ Final trained model compressed into a 140GB file

What Can Be Reduced?

- ☞ Number of weights (pruning), knowledge distillation, ...
- ☞ Size of weights, i.e. **precision** ... but how?
  - ◇ **Reduced floating-point precision** (16 bits, 32 bits) for speed or scale
  - ◇ **Mixed floating-point precisions** offers the best of both worlds: **performance** (speed, energy, ...) and **accuracy**
  - ◇ **Fixed-Point arithmetic:** relatively easy to implement in hardware, no floating point units, faster than floating point arithmetic, less memory & energy

Objective:
Synthesize **fixed point code** of a NN (from floating point NN) while guaranteeing an **error threshold** on the results

What is Quantization?

Why does it work?

- DNNs are robust to perturbations
- Small losses in accuracy are recovered by retraining the quantized models
- ...

Advantages

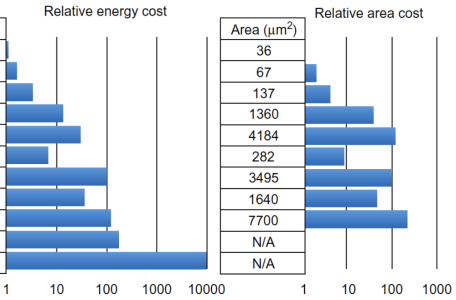
- $4\times$ memory reduction (FP32 $\rightarrow$ INT8)
- $\searrow$ RAM accesses $\rightarrow$ $\searrow$ power and time
- FP arithmetic is not always available on embedded devices

Advantages of Fixed-Point Numbers

- Energy, delay, and area vary a lot between numeric formats and word-length[1]

	Addition	Multiplication
8-bit integer	0.03 pJ/36μm^2	0.2 pJ/282μm^2
32-bit float	0.9 pJ/4184μm^2	3.7 pJ/7700μm^2

Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FB Add	0.4	1360
32b FB Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FB Mult	1.1	1640
32b FB Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A



[1] Energy numbers are from the talk **"An Optimization Playground for Precision and Number Representation Tuning"** of **Olivier Sentieys**, Univ. Rennes, Inria

Quantization Ensuring Error Bounds on Outputs[2]

[2] Dorra Ben Khalifa, Matthieu Martel: *Efficient Implementation of Neural Networks Usual Layers on Fixed-Point Architectures.* LCTES 2024: 12-22

Quantization Ensuring Error Bounds on Outputs

y: original NN output **ŷ**: output of quantized NN

✏ **Input:** *NN* + error threshold on $|y - \hat{y}|$

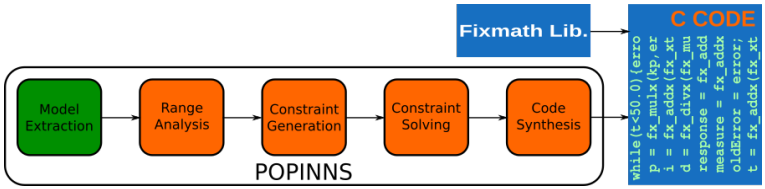
✏ **Output:** a fixed-point code implementing *NN* and satisfying threshold

```
num_classes = 4
input_shape = (8, 8, 1)
model = keras.Sequential([
    keras.Input(shape=input_shape),
    layers.Conv2D(1,kernel_size=(3, 3),activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(num_classes,activation="relu")])

threshold = 6
popinns(model,input_shape,imgs,threshold)
```

```
...
fixed_t W_3[36] = { -416, -1392, -2547, 1925, ... };
tmp = 0;
for (int j=0;j<9;j++) {
  tmp=fx_addx(tmp,fx_mulx(W_3[0*9+j], x[3][0][j][0],12));
};
x[4][0][0][0] = RELU(fx_xtox(tmp,12,8));
tmp = 0;
for (int j=0;j<9;j++) {
  tmp=fx_addx(tmp,fx_mulx(W_3[1*9+j], x[3][0][j][0],12));
};
x[4][0][1][0] = RELU(fx_xtox(tmp,12,8));
...
```

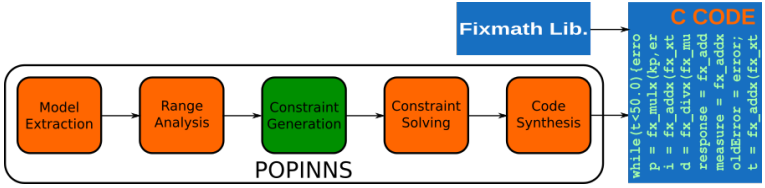
✏ **Approach:** generate a set of constraints modeling the accuracy of the computations across the NN

Overview of the Tool



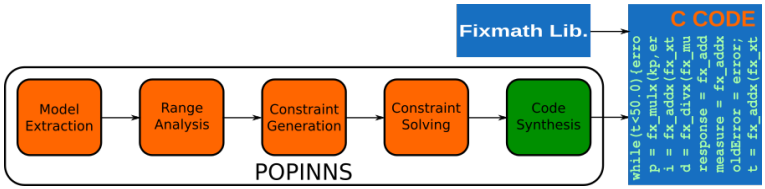

```
num_classes = 4
input_shape = (8, 8, 1)
model = keras.Sequential([
    keras.Input(shape=input_shape),
    layers.Conv2D(1,kernel_size=(3, 3),activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(num_classes,activation="relu")])

threshold = 6
popinns(model,input_shape,imgs,threshold)
```

Overview of the Tool




```
...
(assert (>= fy_5_0_2 8))
(assert (>= fy_5_0_3 8))
(assert (>= f_0_0 (+ fy_0_0_0 0 5)))
(assert (>= fx_0_0_0 (+ fy_0_0_0 (- 1) 5)))
(assert (>= fx_0_0_0 (+ (- 1 f_0_0) fy_0_0_0 5)))
(assert (>= fx_0_0_1 (+ fy_0_0_0 (- 1) 5)))
(assert (>= fx_0_0_1 (+ (- 1 f_0_0) fy_0_0_0 5)))
(assert (>= fx_0_0_2 (+ fy_0_0_0 (- 1) 5)))
(assert (>= fx_0_0_2 (+ (- 1 f_0_0) fy_0_0_0 5)))
(assert (>= f_0_0 (+ fy_0_0_1 0 5)))
(assert (>= fx_0_0_1 (+ fy_0_0_1 (- 1) 5)))
(assert (>= fx_0_0_1 (+ (- 1 f_0_0) fy_0_0_1 5)))
(assert (>= fx_0_0_2 (+ fy_0_0_1 (- 1) 5)))
(assert (>= fx_0_0_2 (+ (- 1 f_0_0) fy_0_0_1 5)))
(assert (>= fx_0_0_3 (+ fy_0_0_1 (- 1) 5)))
(assert (>= fx_0_0_3 (+ (- 1 f_0_0) fy_0_0_1 5)))
(assert (= fx_1_0_0 fy_0_0_0))
(assert (= fx_1_0_1 fy_0_0_1))
...
```

Overview of the Tool

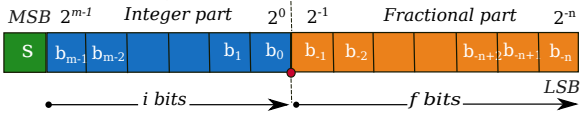



```
...
fixed_t W_3[36] = { -416, -1392, -2547, 1925, ... };
tmp = 0;
for (int j=0;j<9;j++) {
  tmp=fx_addx(tmp,fx_mulx(W_3[0*9+j], x[3][0][j][0],12));
};
x[4][0][0][0] = RELU(fx_xtox(tmp,12,8));
tmp = 0;
for (int j=0;j<9;j++) {
  tmp=fx_addx(tmp,fx_mulx(W_3[1*9+j], x[3][0][j][0],12));
};
x[4][0][1][0] = RELU(fx_xtox(tmp,12,8));
...
```

Constraints Generation Based Error Propagation through DNN Layers

Errors of Fixed-Point Arithmetic

- A fixed-point number in format $Q_{i,f}$ (leftmost bit used for the sign)



- **Error on** x in format $Q_{i,f}$: $\varepsilon(x) \leq 2^{-f}$

- **Errors of addition/subtraction:** In absence of overflow

$$z = x \pm y \qquad \varepsilon(z) = \varepsilon(x) \pm \varepsilon(y) \quad \text{where} \quad \pm \in \{+, -\}$$

- **Errors of multiplication:**

$$z = x \times y \qquad \varepsilon(z) = \varepsilon(x) \cdot y + \varepsilon(y) \cdot x + \varepsilon(x) \cdot \varepsilon(y) + \varepsilon_{\times}$$

$\varepsilon_{\times}$: the error due to the multiplication given by $\varepsilon_{\times} \leq 2^{-(\mathbf{f}_x + \mathbf{f}_y)} - 2^{-\mathbf{f}}$

- **(Errors of Shift Operations)** ($v = v_I \gg r$ with r the number of bits)

$$\varepsilon(v) = \varepsilon(v_I) \cdot 2^{-r} + \varepsilon_{\gg} \quad , \quad \textit{with} \quad \varepsilon_{\gg} \leq 2^{-\mathbf{f}_I} - 2^{-\mathbf{f}} \text{ and } \mathbf{f} = \mathbf{f}_I - r$$

Constraint Generation for DNNs

- DNN with ℓ layers: $\mathscr{L}_0, \ldots, \mathscr{L}_{\ell-1}$
- **Layer sizes:** $\mathscr{I}_k$ inputs, $\mathscr{O}_k$ outputs
- **Variables:** inputs x_j^k, outputs y_i^k, errors $\varepsilon(x_j^k), \varepsilon(y_i^k)$
- **Unknown precisions:** $\mathbf{f}_{x_j}^k, \mathbf{f}_{y_i}^k, \mathbf{f}_i^k \in \mathbb{Z}$
- **Constraints:**

$$\varepsilon(x_j^k) \leq 2^{-\mathbf{f}_{x_j}^k}, \quad \varepsilon(y_i^k) \leq 2^{-\mathbf{f}_{y_i}^k}$$

- **Integer constraints** $\rightarrow$ simple for Z3

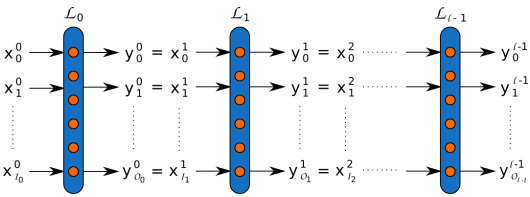
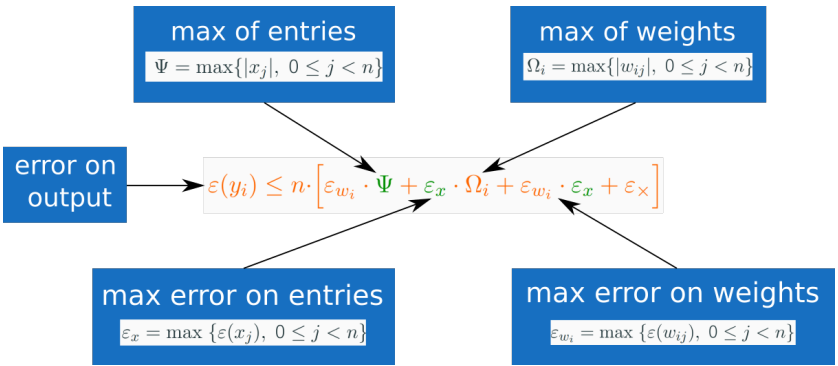


Figure 1: DNN with ℓ layers

Errors in Fully Connected Layers[3]

$$y_i = \sum_{j=0}^{n-1} w_{ij} \cdot x_j, \quad 0 \leq i < L$$



[3] Dorra Ben Khalifa, Matthieu Martel: *Efficient Implementation of Neural Networks Usual Layers on Fixed-Point Architectures.* LCTES 2024: 12-22

Preliminary Lemma

✎ **Lemma** Let $a > 0$ and $b > 0$ be two real numbers, then

$$\log_2(a+b) \leq \max\big(\log_2(a), \log_2(b)\big) + 1$$

✎ **Proof** Since $a > 0$ and $b > 0$, $a + b \leq 2 \cdot \max(a, b)$ and by monotony of the $\log_2$ function,

$$\log_2(a+b) \leq \log_2\big(2 \cdot \max(a,b)\big) \leq \log_2\big(\max(a,b)\big) + 1$$

Using again the monotony of the $\log_2$ function, we know that $\log_2\big(\max(a,b)\big) = \max\big(\log_2(a), \log_2(b)\big)$ Consequently,

$$\log_2(a+b) \leq \max\big(\log_2(a), \log_2(b)\big) + 1$$

■

Preliminary Corollary

✎ **Corollary**

Let $a_1, \ldots, a_n$ be n positive real numbers, then

$$\log_2(a_1 + \ldots + a_n) \leq \max\left(\log_2(a_1), \ldots, \log_2(a_n)\right) + n - 1$$

✎ **Proof** By induction on n, using previous Lemma. ■

Constraints for Fully Connected Layers

- $\mathbf{f}_i^k = \lceil \log_2(\varepsilon_{w_i}) \rceil$: the precision of the i^{th} neuron of $\mathscr{L}_k$ (INTEGER)
- $\mathbf{f}_{x_i}^k = \lceil \log_2(\varepsilon_{x_i}) \rceil$: the precision of the i^{th} input (INTEGER)
- $\mathrm{i}_{\Omega_i} = \lceil \log_2(\Omega_i) \rceil$ and $\mathrm{i}_{\Psi} = \lceil \log_2(\Psi) \rceil$ (INTEGER)

✎ Let $2^{-\mathbf{f}_{y_i}^k}$ be an upper bound on the error introduced by the i^{th} neuron of the layer, since $\varepsilon_{w_i} \leq 2^{-\mathbf{f}_i^k}$ and $\varepsilon_{x_i} \leq 2^{-\mathbf{f}_{x_i}^k}$, we have

$$n \cdot \left[2^{-\mathbf{f}_i^k} \cdot \Psi + 2^{-\mathbf{f}_{x_i}^k} \cdot \Omega_i + 2^{-\mathbf{f}_i^k} \cdot 2^{-\mathbf{f}_{x_i}^k} + \; 2^{-\mathbf{f}_i^k} \cdot 2^{-\mathbf{f}_{x_i}^k} - 2^{-\mathbf{f}_i^k} \; \right] \leq 2^{-\mathbf{f}_{y_i}^k}$$

and

$$-\mathbf{f}_{y_i}^k \leq \log_2\left(n \cdot \left[2^{-\mathbf{f}_i^k} \cdot \Psi + 2^{-\mathbf{f}_{x_i}^k} \cdot \Omega_i + 2 \cdot 2^{-\mathbf{f}_i^k} \cdot 2^{-\mathbf{f}_{x_i}^k} - 2^{-\mathbf{f}_i^k}\right]\right)$$

Fully Connected Layers

We have the following equations:

$$i)\ \log_2(2^{-\mathbf{f}_i^k}\cdot\Psi)=\mathrm{i}_\Psi-\mathbf{f}_i^k, \qquad ii)\ \log_2(2^{-\mathbf{f}_{x_i}^k}\cdot\Omega_i)=\mathrm{i}_{\Omega_i}-\mathbf{f}_{x_i}^k,$$

$$iii)\ \log_2(2\cdot 2^{-\mathbf{f}_i^k}\cdot 2^{-\mathbf{f}_{x_i}^k})=1-\mathbf{f}_i^k-\mathbf{f}_{x_i}^k, \qquad iv)\ \log_2(-2^{-\mathbf{f}_i^k})=\mathbf{f}_i^k.$$

Using the corollary, we have to take the maximum of these four quantities but $iii)$ and (iv) are implied by $i)$ and $ii)$. Then using our Lemma we obtain

$$-\mathbf{f}_{y_i}^k \geq \max\left(\mathrm{i}_\Psi-\mathbf{f}_i^k, \mathrm{i}_{\Omega_i}-\mathbf{f}_{x_i}^k\right)+\lfloor\log_2(n)\rfloor+1\ .$$

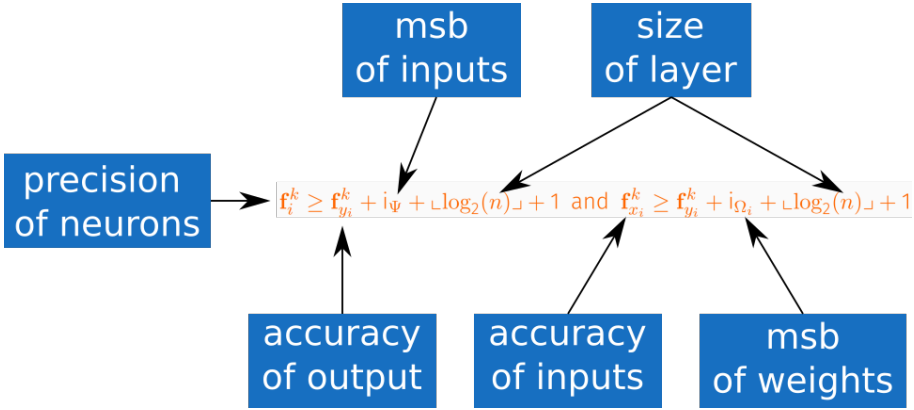
It follows that

$$\mathbf{f}_{y_i}^k \leq \min\left(\mathbf{f}_i^k-\mathrm{i}_\Psi, \mathbf{f}_{x_i}^k-\mathrm{i}_{\Omega_i}\right)-\lfloor\log_2(n)\rfloor-1$$

□

Constraints for Fully Connected Layers

☛ $\mathbf{f}_i^k$, $\mathbf{f}_{x_i}^k$, i_{Ω_i} and i_Ψ INTEGERS



Constraint Generation

$$C_{init} = \left\{ \mathbf{f}^0_{x_j} \leq \Theta_{in} \ : \ 0 \leq j < \mathscr{I}_0 \right\} \cup \left\{ \Theta_{out} \leq \mathbf{f}^{\ell-1}_{y_i} \ : \ 0 \leq i < \mathscr{O}_{\ell-1} \right\} \quad \text{(Init)}$$

$$C_{link} = \left\{ \mathbf{f}^{k+1}_{x_i} \leq \mathbf{f}^k_{y_i} : \ 0 \leq k < \ell - 1, \ 0 \leq i < \mathscr{O}_k \right\} \quad \text{(Link)}$$

$$C^k_{FC} = \left\{ \begin{array}{l} \mathbf{f}^k_i \geq \mathbf{f}^k_{y_i} + \mathsf{i}_\Psi + \llcorner \log_2(n) \lrcorner + 1 \\ \mathbf{f}^k_{x_i} \geq \mathbf{f}^k_{y_i} + \mathsf{i}_{\Omega_i} + \llcorner \log_2(n) \lrcorner + 1 \end{array} \ : \ 0 \leq i < \mathscr{O}_k \right\} \quad \text{(Fully Connected)}$$

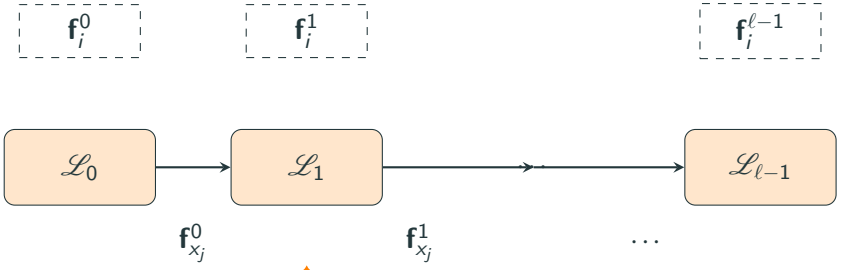
$$C^k_{convo} = \left\{ \begin{array}{l} \mathbf{f}^k \geq \mathbf{f}^k_{y_i} + \mathsf{i}_\Psi + \llcorner \log_2(n) \lrcorner + 2 \\ \mathbf{f}^k_{x_{i+u}} \geq \mathbf{f}^k_{y_i} + \mathsf{i}_\Gamma + \llcorner \log_2(n) \lrcorner + 2 \\ \mathbf{f}^k_{x_{i+u}} \geq \mathbf{f}^k_{y_i} - \mathbf{f}^k + \llcorner \log_2(n) \lrcorner + 3 \end{array} \ : \ \begin{array}{l} 0 \leq u < s_K \\ 0 \leq i < \mathscr{O}_k \end{array} \right\} \quad \text{(Convolution)}$$

$$C^k_{maxpool} = \left\{ \mathbf{f}^k_{y_i} \leq \mathbf{f}^k_{x_{i \times p_y + p}} \ : \ 0 \leq p \leq p_y, \ 0 \leq i < \mathscr{O}_k \right\} \quad \text{(MaxPool)}$$

$$C^k_{upsamp} = \left\{ \mathbf{f}^k_{y_i} \leq \mathbf{f}^k_{x_{\llcorner \frac{i}{p_y} \lrcorner}} \ : \ 0 \leq i < \mathscr{O}_k \right\} \quad \text{(UpSamp)}$$

Cost Function

Goal: Minimize total fractional bits under constraints

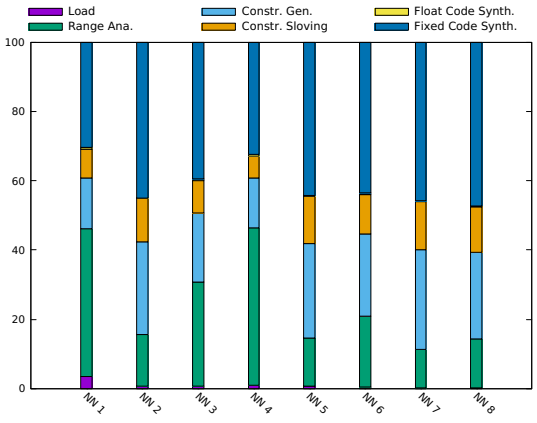


$$\text{cost}(N) = \sum_{0 \le k < \ell} \left(\sum_{0 \le j < \mathscr{I}_k} \mathbf{f}_{x_j}^k + \sum_{0 \le j < \mathscr{O}_k} \mathbf{f}_{x_j}^k + \sum_{0 \le i < \mathscr{O}_k} \mathbf{f}_i^k \right)$$

Description of the NNs

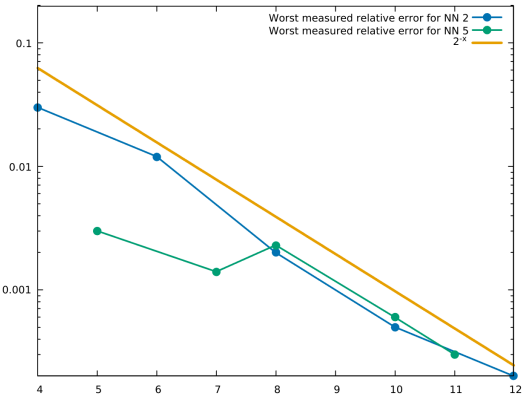
NN	CV	MP	US	FL	FC	ReLU	#Input	#Class	#Param.
1	1	1	-	2	2	2	64	4	126
2	-	-	-	1	1	1	64	12	780
3	1	-	-	1	1	1	144	10	1 020
4	2	1	1	1	1	3	144	10	390
5	-	-	-	2	2	1	64	8	1 176
6	1	-	-	3	3	2	64	6	2 838
7	-	-	-	4	4	3	100	5	10 405
8	1	-	-	3	3	3	144	10	8 120

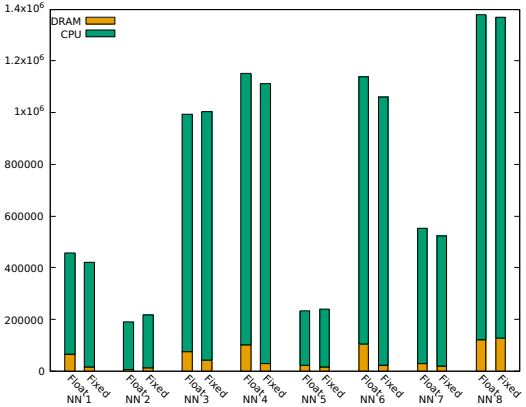
Synthesis Time (%) using POPiNNs




```
[=================================================]
[=     .--'--.                                    =]
[=   /./.|.\.\       Welcome to POPiNNs 1.0       =]
[=        | _Co3_                                 =]
[=        |\((*))    Fixed-Point Code Synthesizer =]
[=        J V =_m_                                =]
[=          /_|___|  For Deep Neural Networks     =]
[=            _||_                                 =]
[=================================================]
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 10, 10, 1)         10

flatten (Flatten)            (None, 100)               0

dense (Dense)                (None, 10)                1010

=================================================================
Total params: 1,020
Trainable params: 1,020
Non-trainable params: 0
_________________________________________________________________
[= Loading TF model in POPiNNs... Elapsed: 0.00365s
[= Range analysis... Elapsed: 0.16217s
[= Constraint generation... Elapsed: 0.12149s
[= Solving constraints with z3... Elapsed: 0.05537s
[= Synthesizing float code... Elapsed: 0.00223s
[= Synthesizing fixed code... Elapsed: 0.21420s
```

Accuracy (left) & Energy of Synthesized Codes (right)

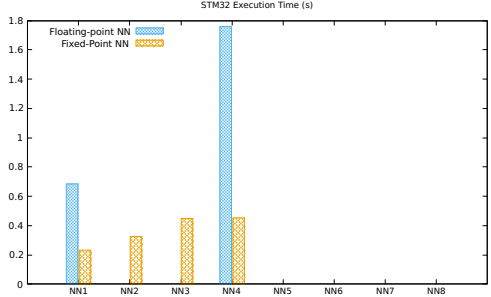


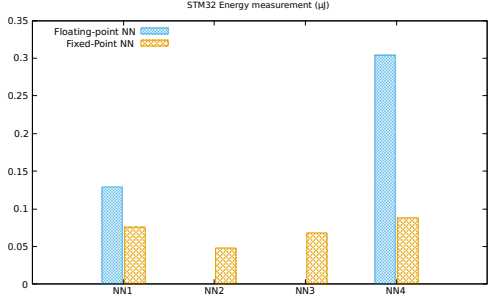


Execution of Synthesized Codes on STM32 Micro-Controllers

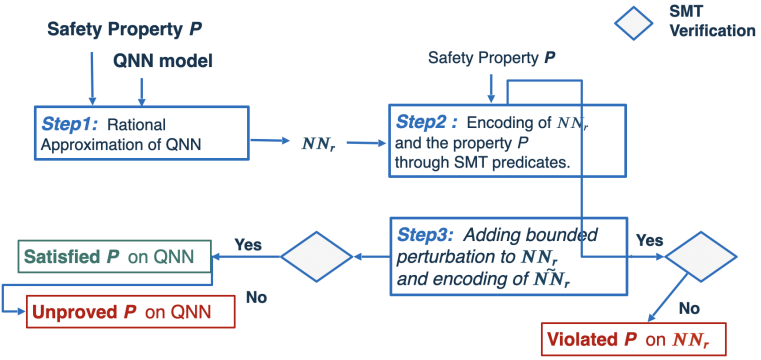
Board: STM32-F756ZG micro-controller

- **32-bit** Arm® Cortex®-M3 CPU (120 MHz max)
- **1 MByte** Flash prg memory, **128Kbyte** RAM
- **Miosix** OS kernel designed to run on 32bit micro-controllers
- STM32 Nucleo Power shield for energy measurement



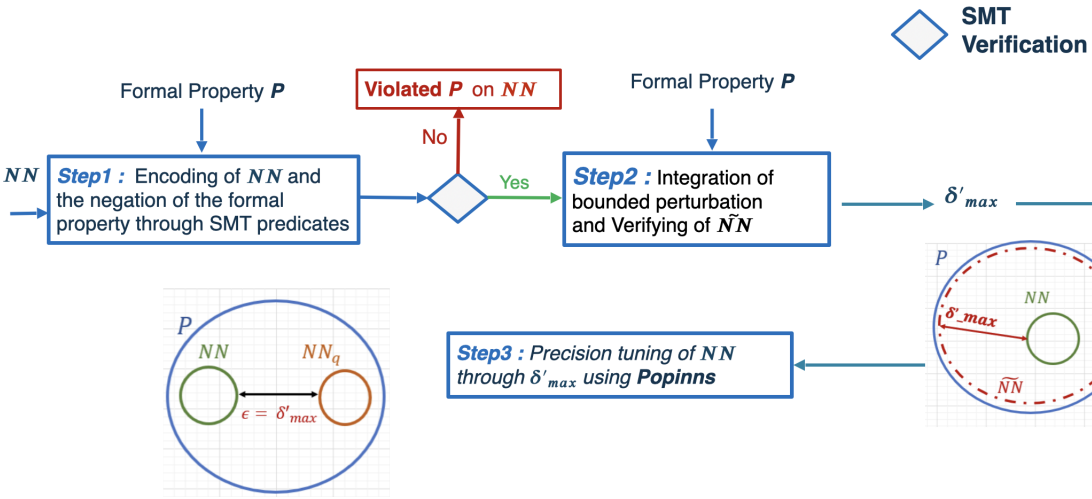


Designing QNN using Formal Methods by W. BACHIRI[4]



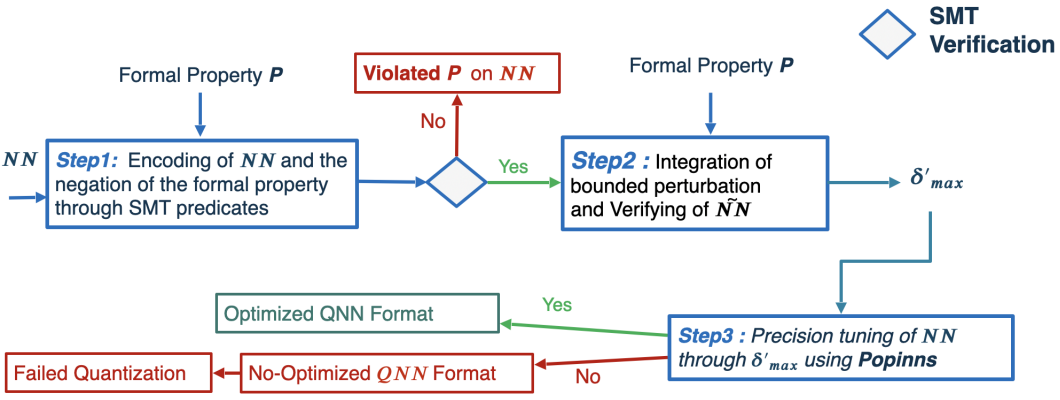
[4]Wahiba Bachiri, "Optimization and Code Verification for Embedded Systems Based on Intelligent Controllers", PhD dissertation 2025

Designing QNN using Formal Methods by W. BACHIRI[4]



[4]Wahiba Bachiri, "Optimization and Code Verification for Embedded Systems Based on Intelligent Controllers", PhD dissertation 2025

Designing QNN using Formal Methods by W. BACHIRI[4]



[4]Wahiba Bachiri, "Optimization and Code Verification for Embedded Systems Based on Intelligent Controllers", PhD dissertation 2025

Experimentations: Apple M1 Ultra, 20 cores, 128 GB RAM

Fractional bits $= \lceil |\log(\delta'_{\max})/\log(2)| \rceil$

ACAS XU NNs	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	bits
ACASXU_1_1	4.0087	SAT	0.0010	0.0012	0.0010	1e-06	–	–	–	–	20
ACASXU_1_2	4.0078	SAT	0.0015	0.0011	–	–	–	–	–	–	10
ACASXU_1_3	4.0069	SAT	0.0012	0.0030	–	–	–	–	–	–	10
ACASXU_1_4	4.0060	SAT	0.0058	0.0058	–	–	–	–	–	–	8
ACASXU_1_5	4.0062	SAT	0.0060	0.0057	–	–	–	–	–	–	8
ACASXU_1_6	4.0074	SAT	0.0028	0.0027	–	–	–	–	–	–	9
ACASXU_1_7	4.0097	5e-06	SAT	SAT	–	–	–	–	–	–	18
ACASXU_1_8	4.0083	1e-05	SAT	SAT	–	–	–	–	–	–	17
ACASXU_1_9	4.0097	1e-05	SAT	SAT	–	–	–	–	–	–	17

SMT Solver: Used **Marabou** for neural network verification (Python 3 API)

Networks: 45 networks $\times$ 6 layers, 50 neurons/layer, ReLU activation

Naming: `ACAS_XU_x_y` $\rightarrow x =$ advisory, $y = \tau \in \{0, 1, 5, 10, 20, 40, 60, 80, 100\}$

Properties: SAT = violated, – = N/A

Execution Results of Popinns

Layer (type)	Output shape	Params
dense	(None, 50)	300
dense_1	(None, 50)	2,550
dense_2	(None, 50)	2,550
dense_3	(None, 50)	2,550
dense_4	(None, 50)	2,550
dense_5	(None, 50)	2,550
dense_6	(None, 5)	255
Total	-	**13,305**

Model Loading: 0.0023 s

Range Analysis: All neuron outputs computed in 0.019 s

Constraint Generation: Time: 0.98 s

Constraint Solving: Z3 solver — Time: 0.866 s

Code Synthesis: FP: 0.007 s FPx: 1.54 s

Take Home Messages

Take Home Messages

What We Can Do Now:

- Generate fixed-point code for DNNs with formal error bounds
- Model error bounds using integer constraints to preserve classifier dominance[5]

What We Aim to Do:

- Extend quantization to more general neural networks
- Target FPGA deployment with generated code
- Combine quantization with other optimization techniques

Thank You for Your Attention!

[5] Ben Khalifa et al., "Towards Bit-Level Dominance Preserving Quantization of Neural Classifiers. SOAP@PLDI 2025