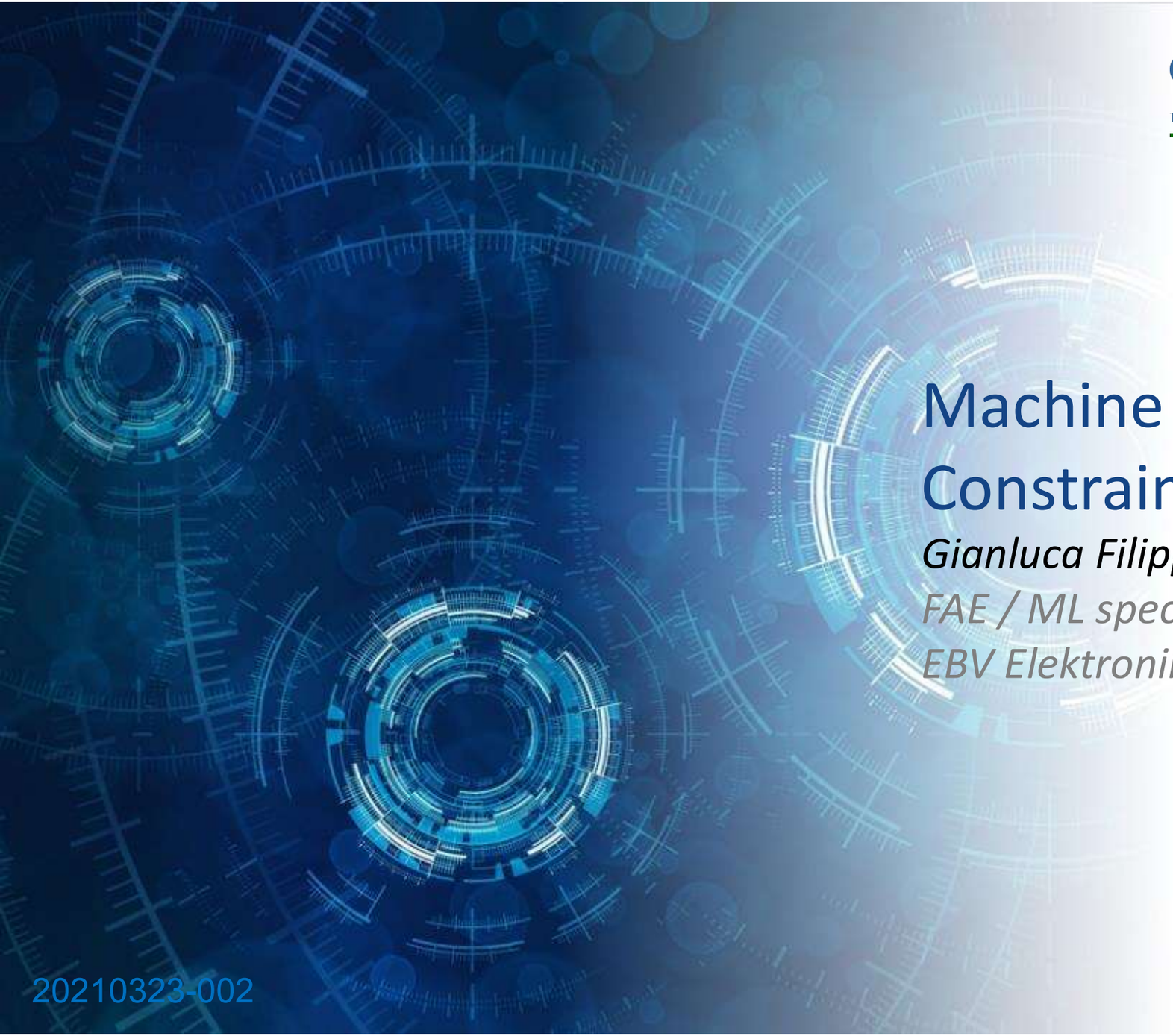


Machine Learning on Resource Constrained Microcontrollers

Gianluca Filippini

FAE / ML specialist

EBV Elektronik

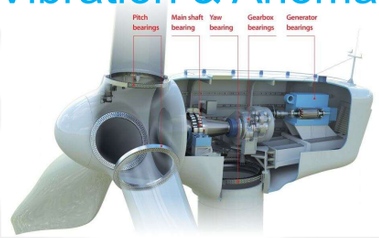
Human “ripples” on MCU+Ai applications

Human-Machine Interaction

Health & Body Tracking

Transportation

Vibration & Anomaly

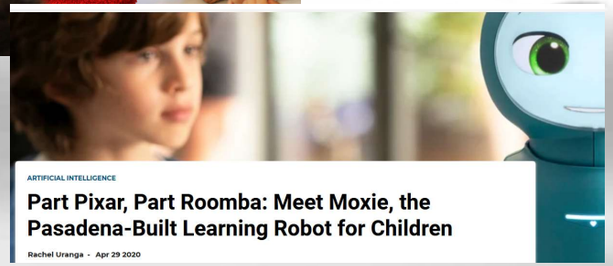


Predictive Maintenance

Medical

https://www.sciencedaily.com/releases/2020/06/200612172204.htm
https://www.ft.com/content/15f0123e-3b7d-11ea-b84f-a62c46f39bc2

Toys & Educational

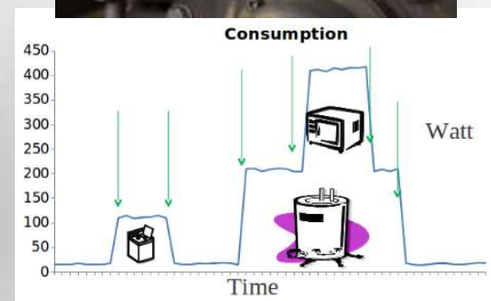


https://dot.la/autism-robot-moxie-ai-2645867544.html

Precision Agriculture

Water Control

Agri-Food

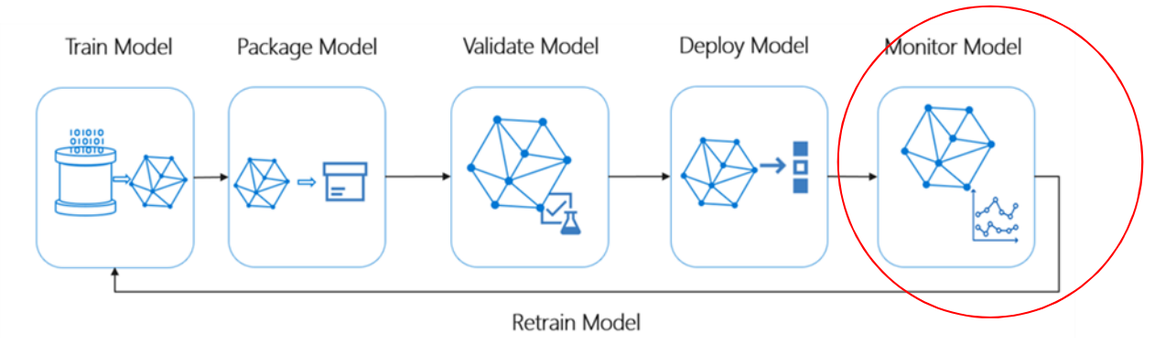


NILM2020

http://nilmworkshop.org/

ML Application: Development & Deployment

https://www.c-sharpcorner.com/blogs/mlops



TrueRNG

Cryptography
Encryption / Decryption

Access control
Authentication

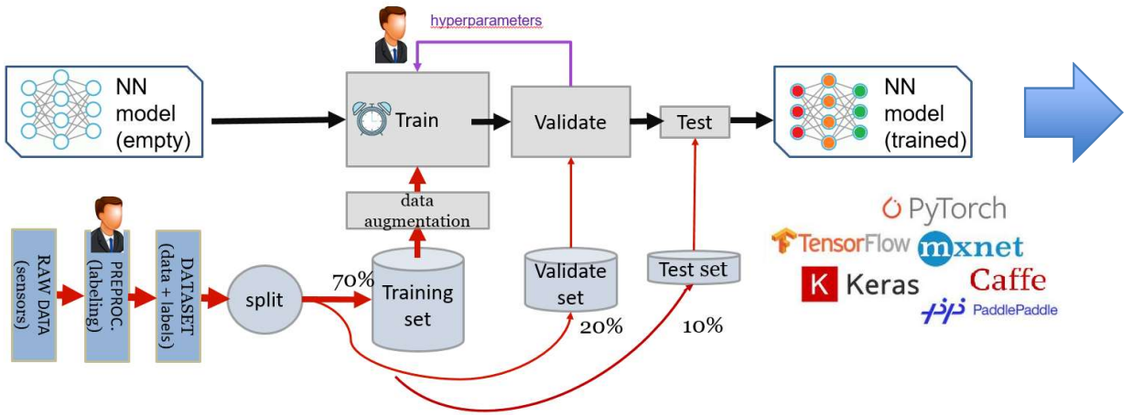
Blockchain

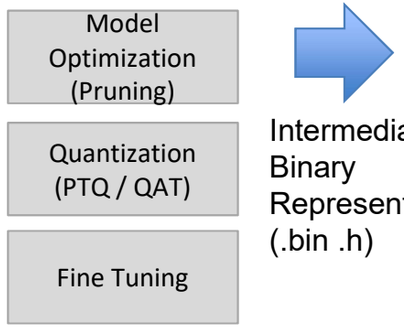
Physical Security
Glitching Hw Attack

Secure Boot / Secure FS

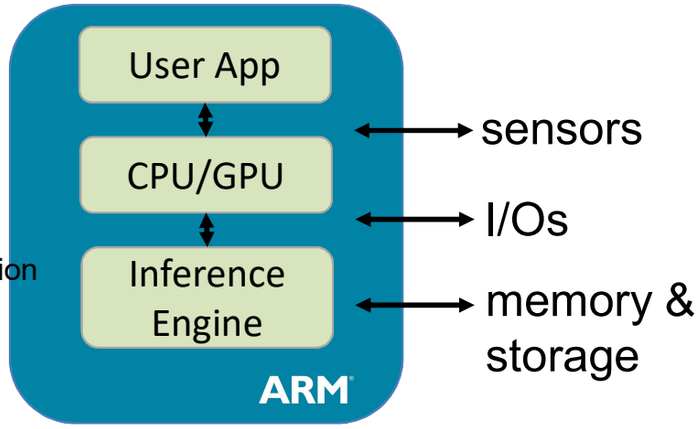
Data ownership / Copyrights

Phase 1 - Training : will generate a FP32 model
Phase 2 - Optimize : model is Quantized/Optimized for a INTx precision
Phase 3 - Inference: On microcontroller after deployment





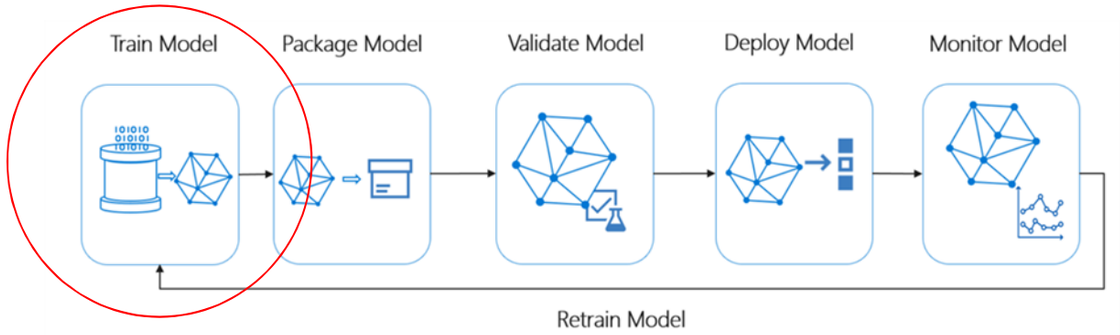
Intermediate Binary Representation (.bin .h)



https://www.eetimes.com/ai-sound-recognition-on-a-cortex-m0-data-is-king/
https://www.darkreading.com/edge/theedge/glitching-the-hardware-attack-that-can-disrupt-secure-software-/b/d-id/1336119

ML Application: Development & Deployment

https://www.c-sharpcorner.com/blogs/mlops



TrueRNG

Cryptography
Encryption / Decryption

Access control
Authentication

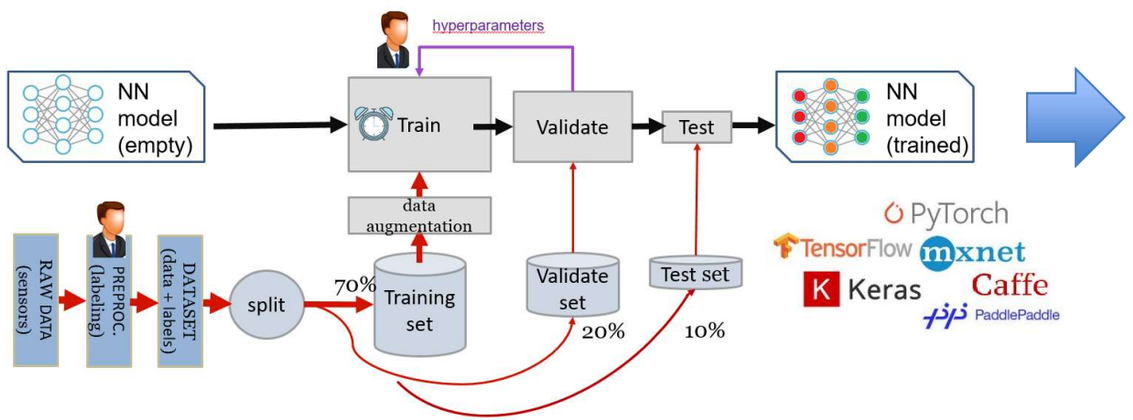
Blockchain

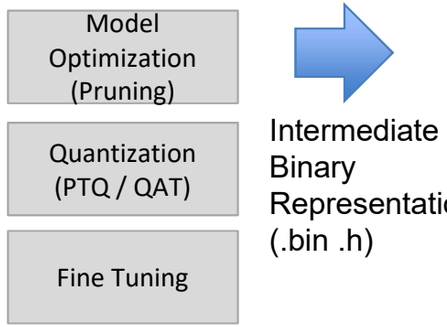
Physical Security
Glitching Hw Attack

Secure Boot / Secure FS

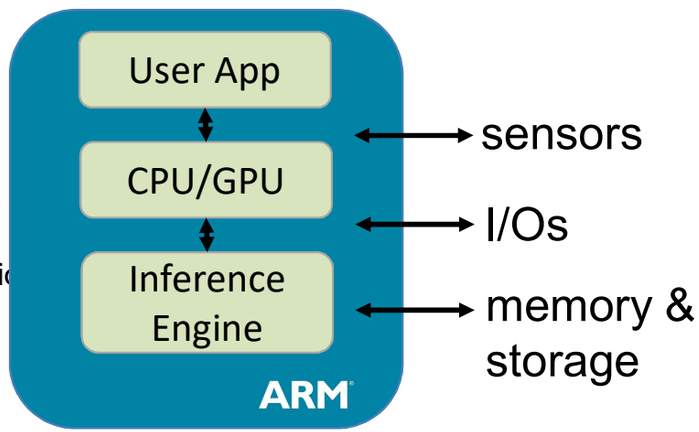
Data ownership / Copyrights

Phase 1 - Training : will generate a FP32 model
Phase 2 - Optimize : model is Quantized/Optimized for a INTx precision
Phase 3 - Inference: On microcontroller after deployment



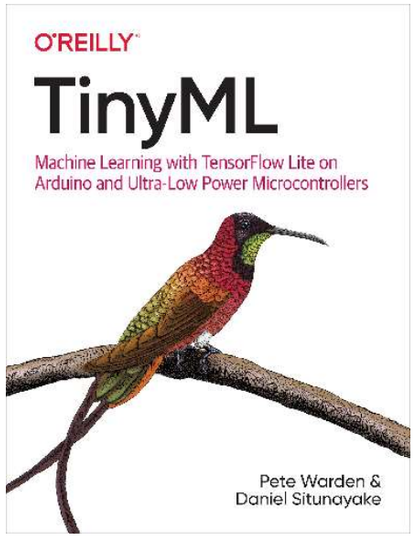


Intermediate Binary Representati (.bin .h)



https://www.eetimes.com/ai-sound-recognition-on-a-cortex-m0-data-is-king/
https://www.darkreading.com/edge/theedge/glitching-the-hardware-attack-that-can-disrupt-secure-software-/b/d-id/1336119

TinyML: Machine Learning for deeply embedded devices



An emerging area in ML that addresses deployment of models that run on small, low-powered microcontrollers. Became a “de-facto reference” also thanks to the wide adoption of the book written by Pete Warden and Daniel Situnayake.

https://www.tinyml.org/

Todays is part of study courses from universities:

Vijay Janapa Reddi
Associate Professor
@ Hardward Univ.

Pete Warden
Engineer
@ Google

Daniel Situnayake
Founding TinyML Engineer
@ Edge Impulse

https://sites.google.com/g.harvard.edu/tinyml/home

Constrained microcontrollers have restricted resources: power, memory, storage and CPU. We use *specific* variants of the main ML tools to address these devices.

TensorFlow → TensorFlow Lite

https://www.tensorflow.org/lite/guide
https://www.tensorflow.org/lite/microcontrollers

Daniel Situnayake:
How do I train my first machine learning model?
https://www.youtube.com/watch?v=DhHw17Z-lvI

•The TensorFlow Lite converter, which converts TensorFlow models into an efficient form for use by the interpreter, and can introduce optimizations to improve binary size and performance.

•The TensorFlow Lite interpreter, which runs specially optimized models on many different hardware types, including mobile phones, embedded Linux devices, and microcontrollers.

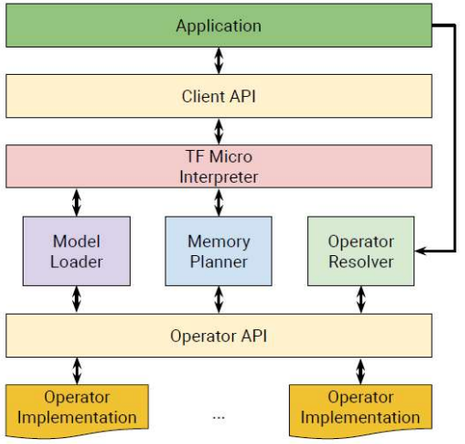
https://petewarden.com/2018/06/11/why-the-future-of-machine-learning-is-tiny/

Tensorflow Lite Micro: an optimized sw architecture

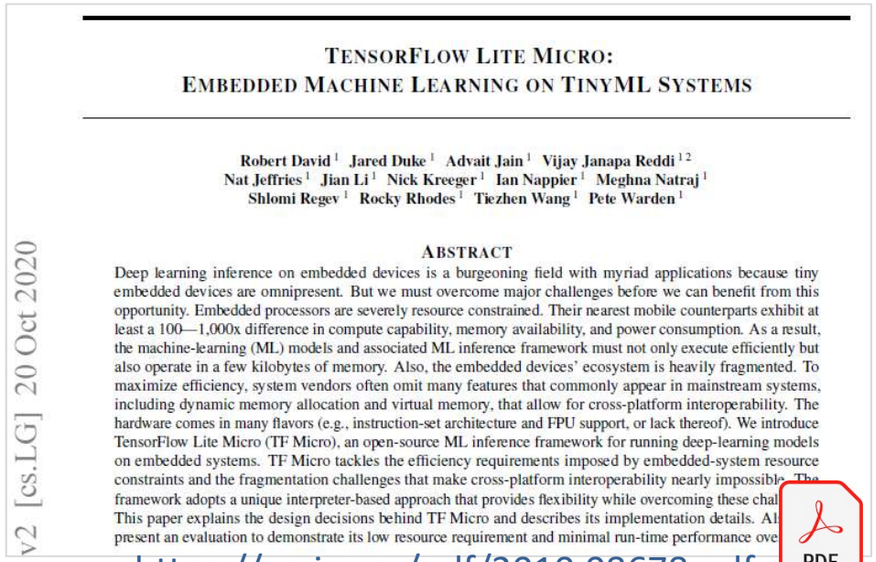
The TFLiteMicro article on arxiv.org presents details on challenges, design principles and implementation of the interpreter running on the MCU (also bare-metal).

https://arxiv.org/abs/2010.08678

The framework does not require operating system support, any standard C or C++ libraries, or dynamic memory allocation – features that are commonly taken for granted in non-embedded system domains. There are restrictions on supported ops:



https://www.tensorflow.org/lite/guide/ops_compatibility



TENSORFLOW LITE MICRO:
EMBEDDED MACHINE LEARNING ON TINYML SYSTEMS

Robert David [1] Jared Duke [1] Advait Jain [1] Vijay Janapa Reddi [1 2]
Nat Jeffries [1] Jian Li [1] Nick Kreeger [1] Ian Nappier [1] Meghna Natraj [1]
Shlomi Regev [1] Rocky Rhodes [1] Tiezhen Wang [1] Pete Warden [1]

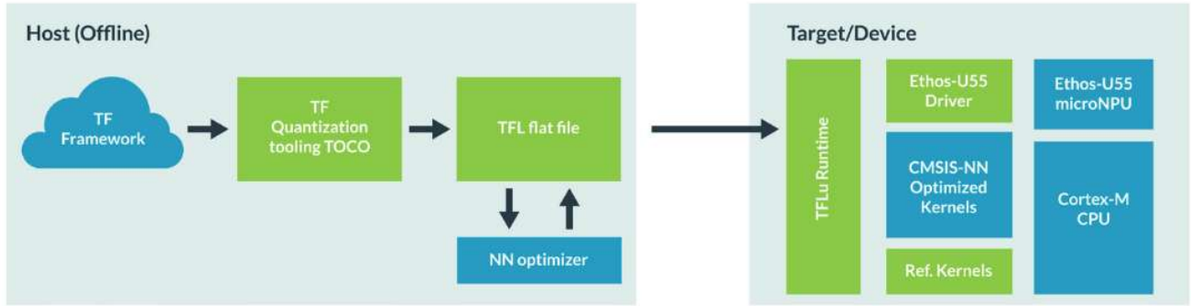
ABSTRACT

Deep learning inference on embedded devices is a burgeoning field with myriad applications because tiny embedded devices are omnipresent. But we must overcome major challenges before we can benefit from this opportunity. Embedded processors are severely resource constrained. Their nearest mobile counterparts exhibit at least a 100—1,000x difference in compute capability, memory availability, and power consumption. As a result, the machine-learning (ML) models and associated ML inference framework must not only execute efficiently but also operate in a few kilobytes of memory. Also, the embedded devices' ecosystem is heavily fragmented. To maximize efficiency, system vendors often omit many features that commonly appear in mainstream systems, including dynamic memory allocation and virtual memory, that allow for cross-platform interoperability. The hardware comes in many flavors (e.g., instruction-set architecture and FPU support, or lack thereof). We introduce TensorFlow Lite Micro (TF Micro), an open-source ML inference framework for running deep-learning models on embedded systems. TF Micro tackles the efficiency requirements imposed by embedded-system resource constraints and the fragmentation challenges that make cross-platform interoperability nearly impossible. The framework adopts a unique interpreter-based approach that provides flexibility while overcoming these chal
This paper explains the design decisions behind TF Micro and describes its implementation details. Al
present an evaluation to demonstrate its low resource requirement and minimal run-time performance ove

v2 [cs.LG] 20 Oct 2020

https://arxiv.org/pdf/2010.08678.pdf

ARM's white paper "Accelerating Machine Learning Compute for the IoT and Embedded Market" introduces results in embedded hw optimization to accelerate execution of TF models. These improvements are both software and hardware (*heterogeneous*)



https://armkeil.blob.core.windows.net/developer/Files/pdf/ethos/Arm_Accelerating_ML_Compute_for_Embedded_Market_white_paper.pdf

Tensorflow Lite Micro: improving execution on MCU

Vendors have produced custom optimizations by improving few key components of the TFLite framework. Optimizing low-level libraries and rewriting specific operators with assembly instructions that are supported on the device we improve the execution of the ML model.

GCC Arm® 8-2018-q4

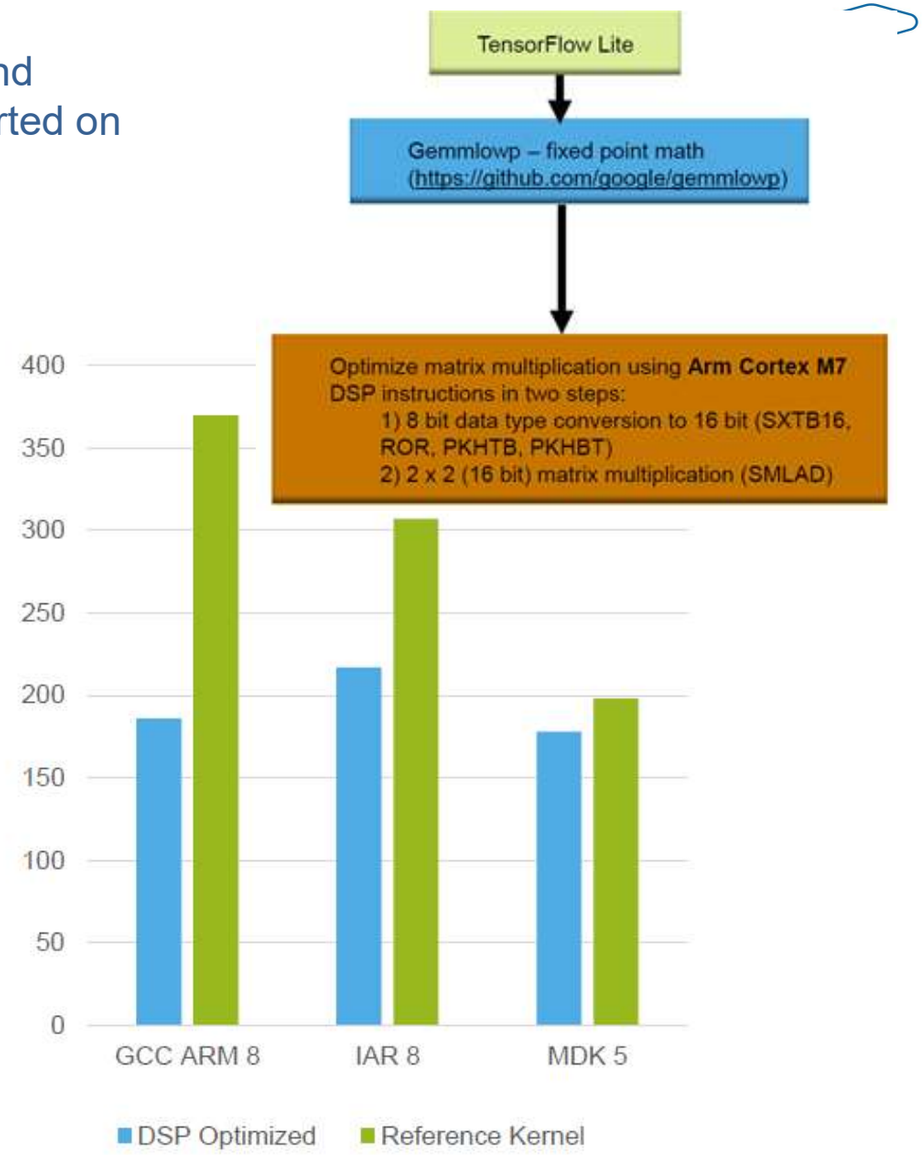
	DSP Optimized (-O2)	Reference Kernel (-O2)
Label Image	186 ms	370 ms
CIFAR-10	61 ms	229 ms

IAR EW 8.32.3

	DSP Optimized	Original
Label Image	217 ms	307 ms
CIFAR-10	67 ms	159 ms

Keil MDK 5.27

	DSP Optimized	Original
Label Image	178 ms	198 ms
CIFAR-10	64 ms	87 ms



GLOW: Graph Lowering Compiler for Hardware Accelerators

A machine learning compiler that accelerates the performance of deep learning frameworks on different hardware platforms.

Started by Facebook (~2018) as a tool to seamlessly deploy PyTorch models on a large variety of hw (from server to embedded devices)

https://ai.facebook.com/tools/glow/

The original presentation provides these motivations:

CPUs and GPUs are inefficient

CPUs and GPUs work hard to extract parallelism.

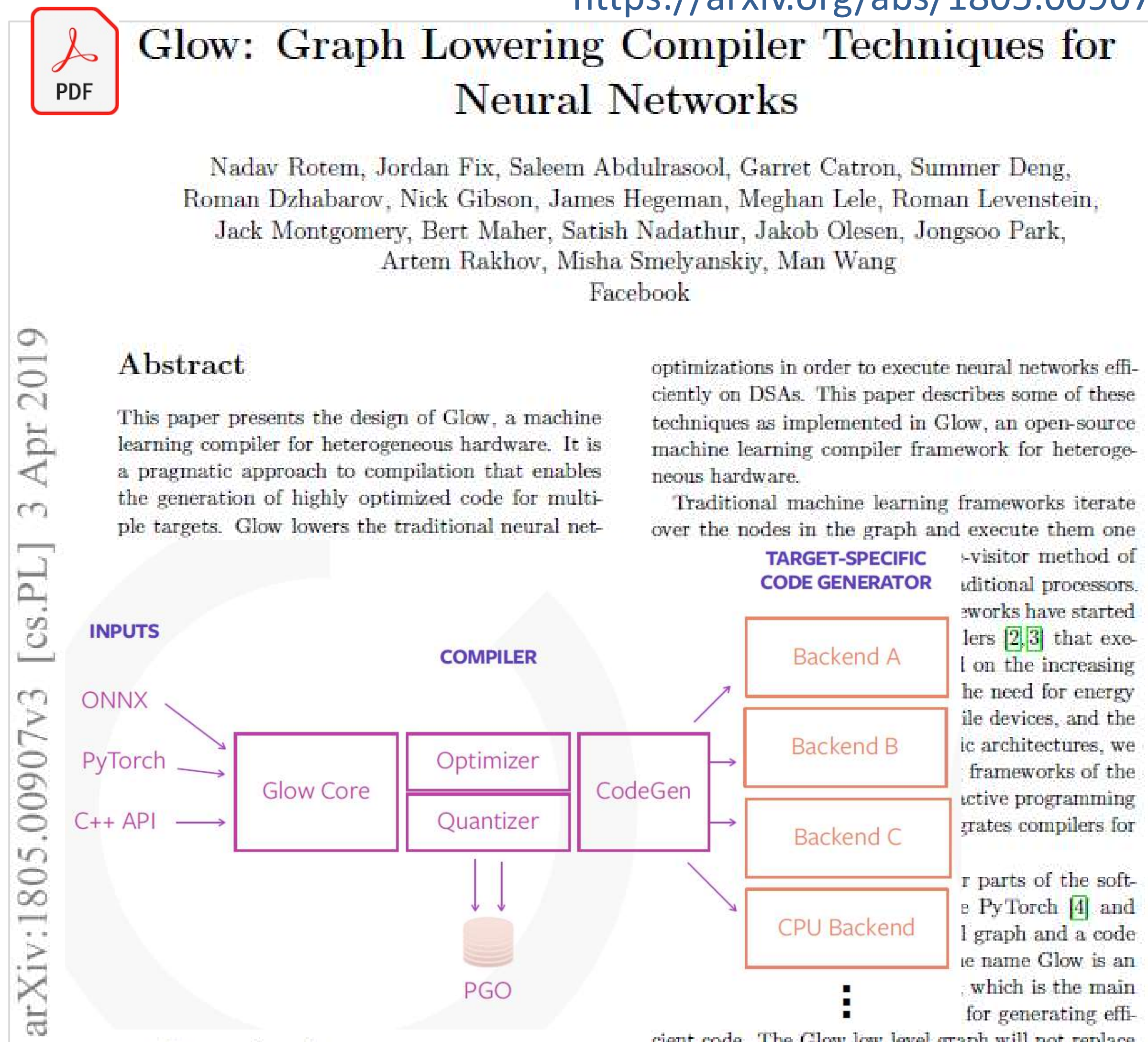
- Matrix operations are very regular and expose lots of parallelism. Easy to accelerate.
- No need to waste power/area on useless features.

Accelerators are efficient because they are specialized

- Have many arithmetic execution units.
- Use dedicated local memories.
- Reduce the arithmetic bit-widths.
- Use a specialized programming model.

Glow | Graph Lowering Compiler for Hardware Accelerators

https://arxiv.org/abs/1805.00907


arXiv:1805.00907v3 [cs.PL] 3 Apr 2019

Glow: Graph Lowering Compiler Techniques for Neural Networks

Nadav Rotem, Jordan Fix, Saleem Abdulrasool, Garret Catron, Summer Deng, Roman Dzhabarov, Nick Gibson, James Hegeman, Meghan Lele, Roman Levenstein, Jack Montgomery, Bert Maher, Satish Nadathur, Jakob Olesen, Jongsoo Park, Artem Rakhov, Misha Smelyanskiy, Man Wang
Facebook

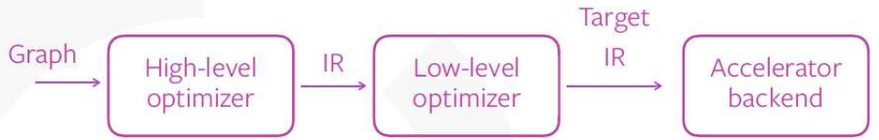
Abstract

This paper presents the design of Glow, a machine learning compiler for heterogeneous hardware. It is a pragmatic approach to compilation that enables the generation of highly optimized code for multiple targets. Glow lowers the traditional neural net- … optimizations in order to execute neural networks efficiently on DSAs. This paper describes some of these techniques as implemented in Glow, an open-source machine learning compiler framework for heterogeneous hardware.

Traditional machine learning frameworks iterate over the nodes in the graph and execute them one …

… cient code. The Glow low-level graph will not replace

GLOW: Model Compilation Pipeline



Performs high-level graph optimizations. Focus on linear-algebra kind of optimizations.

Performs low-level IR optimizations. Focus on buffer and memory reuse optimizations.

Performs target-specific lowering and optimizations for specific accelerator.

Glow | Graph Lowering Compiler for Hardware Accelerators

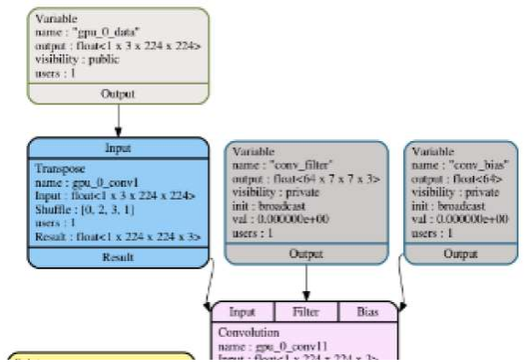
Generates machine code by compiling NN layers optimized for a specific MCU target

Parses graph at compile-time and the compiled library contains only the required optimized code

High-Level Graph

Static-shaped data-flow graph.

- Enables high-level domain-specific optimizations.
- Example: Change the matrix layout, merge batchnorm into conv, eliminate numeric re-scale



Low-Level Instructions

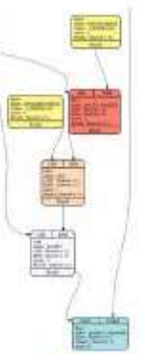
- Linear instruction-based address-only representation.
- Operands are typed pointers to buffers.
- Memory optimizations: Instruction scheduling, Buffer sharing, shortening buffer lifetime.

```
declare {
  %input = WeightVar float<10 x 5000> mutable
  %rnn_initial_state = WeightVar float<10 x 20> muta
  %rnn_Whh = WeightVar float<20 x 20> mutable
  %result = WeightVar float<100 x 500> mutable
}
program {
  %X_0 = allocactivation  { Ty: float<10 x 500>}
  %X_01 = extracttensor @out %X_0, @in %input { Offs
  %mergeLHS11 = allocactivation  { Ty: float<100 x 5
  %mergeLHS111 = splat @out %mergeLHS11 { Value: 0.0
  %mergeLHS112 = inserttensor @inout %mergeLHS11, @i
  %bigMatMul1 res = allocactivation  { Ty: float<100
```

Graph Lowering

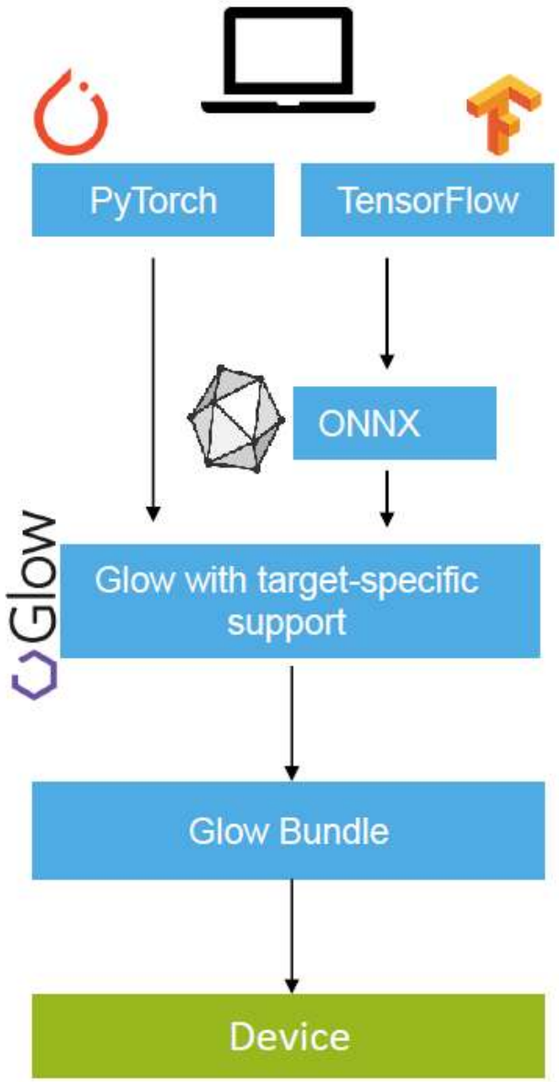
- ML frameworks support hundreds of op kinds, implemented in C and CUDA.
- Writing hundreds of ops for accelerators isn't scalable.
- Glow lowers complex high-level nodes into primitive nodes.

BEFORE



AFTER

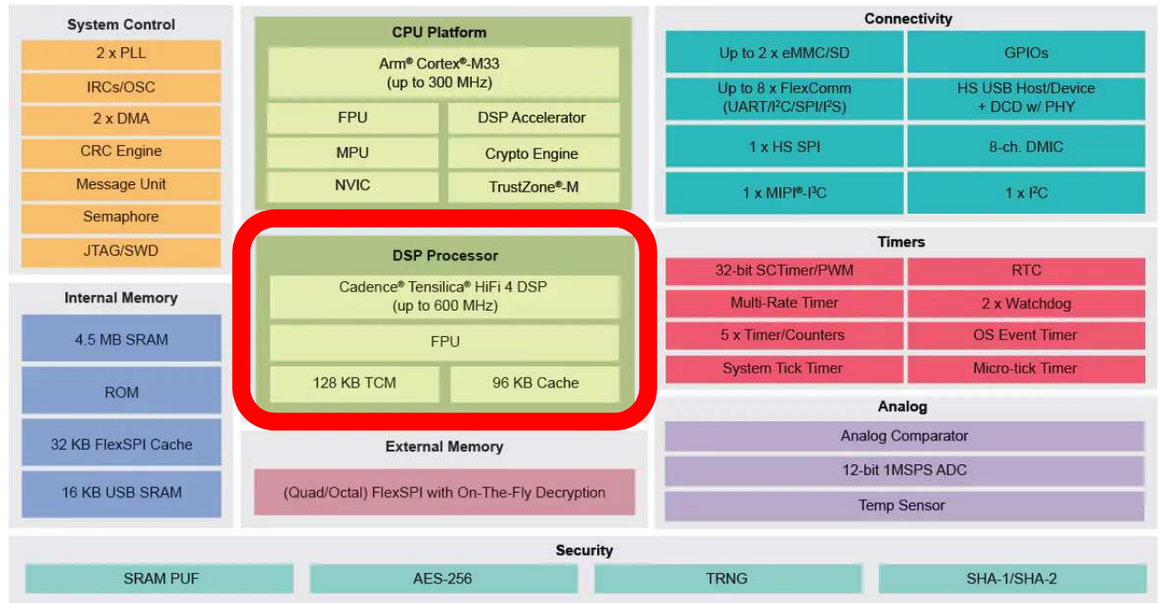
GLOW: Embedded Hardware accelerators



MCU architectures are starting to include specific hw blocks to accelerate execution of NN and more specifically CNN (conv net)

Vendors are extending the support of Glow for Tensorflow models via the intermediate format of ONNX.

- Up to 300 MHz Arm Cortex-M33 core
- **Up to 600 MHz Tensilica HiFi-4 DSP**
- 4.5 MB of SRAM
- Audio and sensor processing



GLOW: Embedded Hardware accelerators

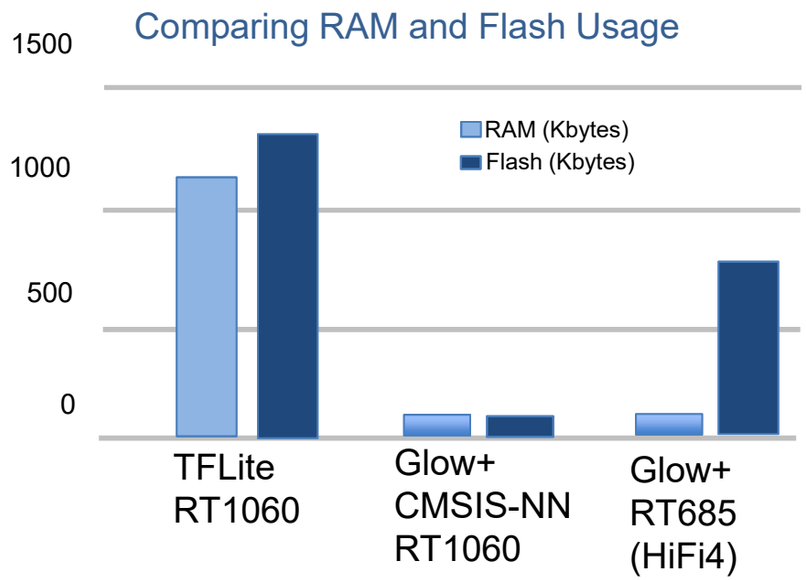
Faster execution of the CNN model is important to improve overall performance on data analysis but also on energy consumption.

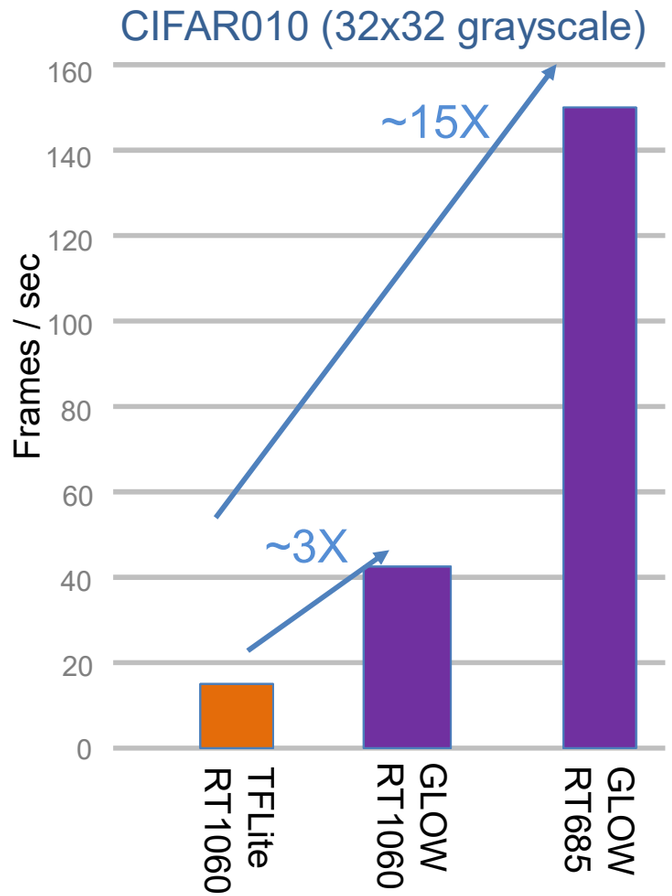
For IoT+Ai projects we can run the inference in a shorter time and return to a power-saving state immediately

Using a dedicated hw accelerator might require custom libraries and custom drivers increasing the binary footprint of the final application.

i.MX RT1060
Arm Cortex-M7@600Mhz, NO DSP

i.MX RT685
Arm Cortex-M33@300Mhz, Tensilica HiFi4





https://media.nxp.com/news-releases/news-release-details/industrys-first-mcu-based-implementation-glow-neural-network/

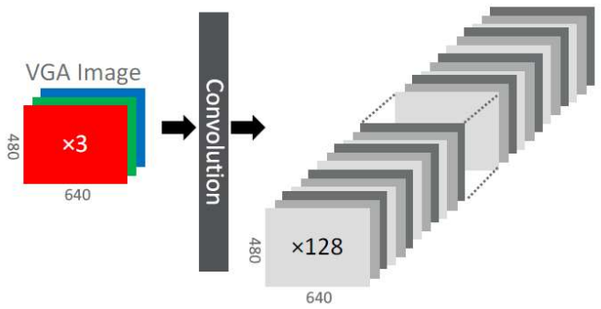
CNN: a quest for energy efficient hardware

The most used & powerful type of NN for deeply embedded devices is CNN. These networks heavily involve matrix multiplication.

Trend for embedded application in vision, audio, medical and signal conditioning is to increase the size of the input layer for common CNN, but input data size is a key factor for MCU with a low amount of memory.

Matrix multiplication on MCU requires frequent access to external memory. On standard hardware this is often a triple-nested multiplication loop.

Resolution	Reference	Memory for 128 ch. (max. intermediate)
32×32	CIFAR-10	128 KB
160×160	Intel Movidius	3,200 KB
224×224	ImageNet	6,272 KB
320×240	QVGA	9,600 KB
640×480	VGA	38,400 KB



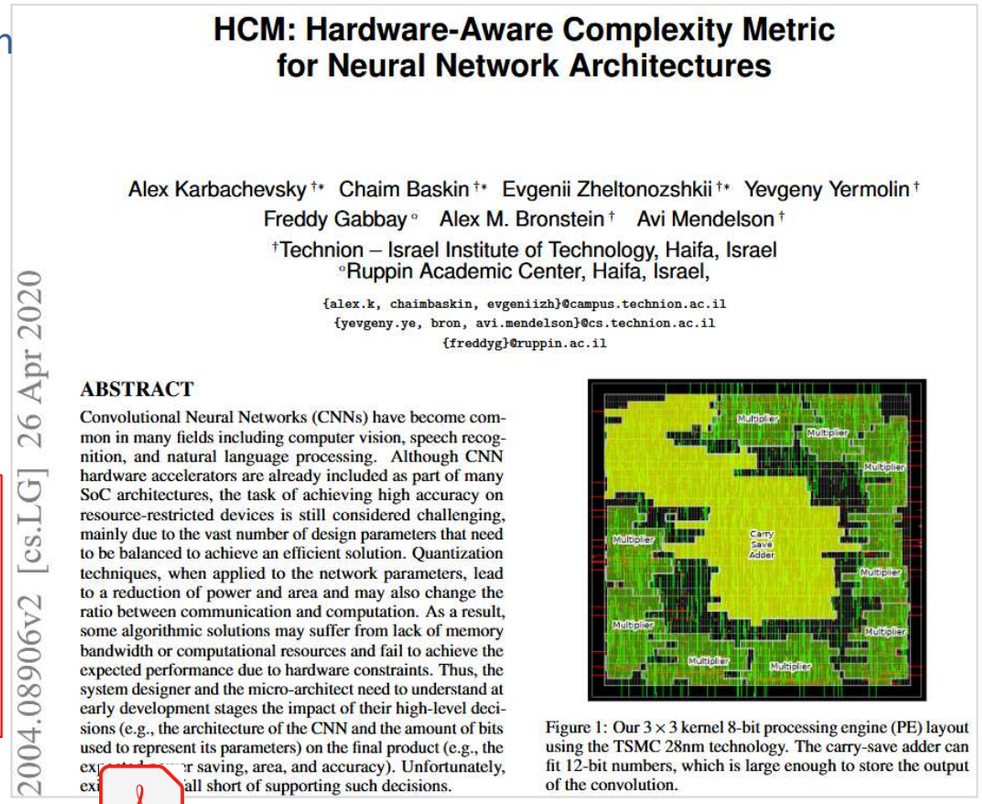

```
matrix_mul(matrix_f32_t *a, matrix_f32_t *b, matrix_f32_t *c)
{
    uint32_t m = a->nrows;
    uint32_t n = a->ncols;
    uint32_t p = b->ncols;

    c->nrows = m;
    c->ncols = p;

    int i, j, k;

    for (i = 0; i < m; i++) {
        for (j = 0; j < p; j++) {
            f_t sum = 0;
            for (k = 0; k < n; k++) {
                sum += a->elements[i * n + k] * b->elements[k * p + j];
            }

            c->elements[i * p + j] = sum;
        }
    }
}
```



HCM: Hardware-Aware Complexity Metric for Neural Network Architectures

Alex Karbachevsky †* Chaim Baskin †* Evgenii Zheltonozshkii †* Yevgeny Yermolin †
Freddy Gabbay ◦ Alex M. Bronstein † Avi Mendelson †
†Technion – Israel Institute of Technology, Haifa, Israel
◦Ruppin Academic Center, Haifa, Israel,
{alex.k, chaimbaskin, evgeniizh}@campus.technion.ac.il
{yevgeny.ye, bron, avi.mendelson}@cs.technion.ac.il
{freddyg}@ruppin.ac.il

ABSTRACT

Convolutional Neural Networks (CNNs) have become common in many fields including computer vision, speech recognition, and natural language processing. Although CNN hardware accelerators are already included as part of many SoC architectures, the task of achieving high accuracy on resource-restricted devices is still considered challenging, mainly due to the vast number of design parameters that need to be balanced to achieve an efficient solution. Quantization techniques, when applied to the network parameters, lead to a reduction of power and area and may also change the ratio between communication and computation. As a result, some algorithmic solutions may suffer from lack of memory bandwidth or computational resources and fail to achieve the expected performance due to hardware constraints. Thus, the system designer and the micro-architect need to understand at early development stages the impact of their high-level decisions (e.g., the architecture of the CNN and the amount of bits used to represent its parameters) on the final product (e.g., the ex… saving, area, and accuracy). Unfortunately, ex… all short of supporting such decisions.

Figure 1: Our 3 × 3 kernel 8-bit processing engine (PE) layout using the TSMC 28nm technology. The carry-save adder can fit 12-bit numbers, which is large enough to store the output of the convolution.

PDF

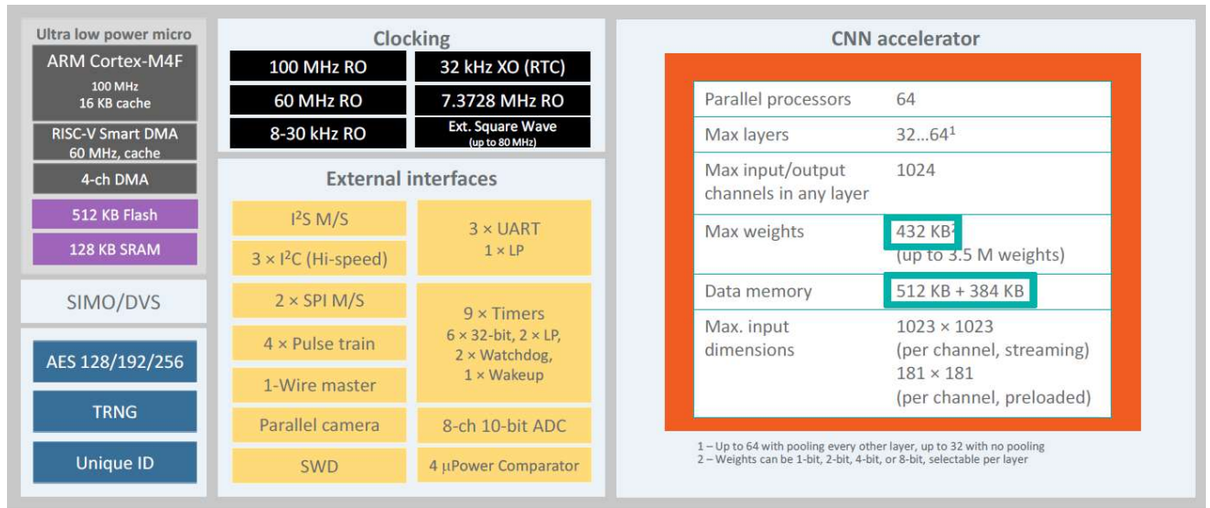
https://arxiv.org/pdf/2004.08906.pdf

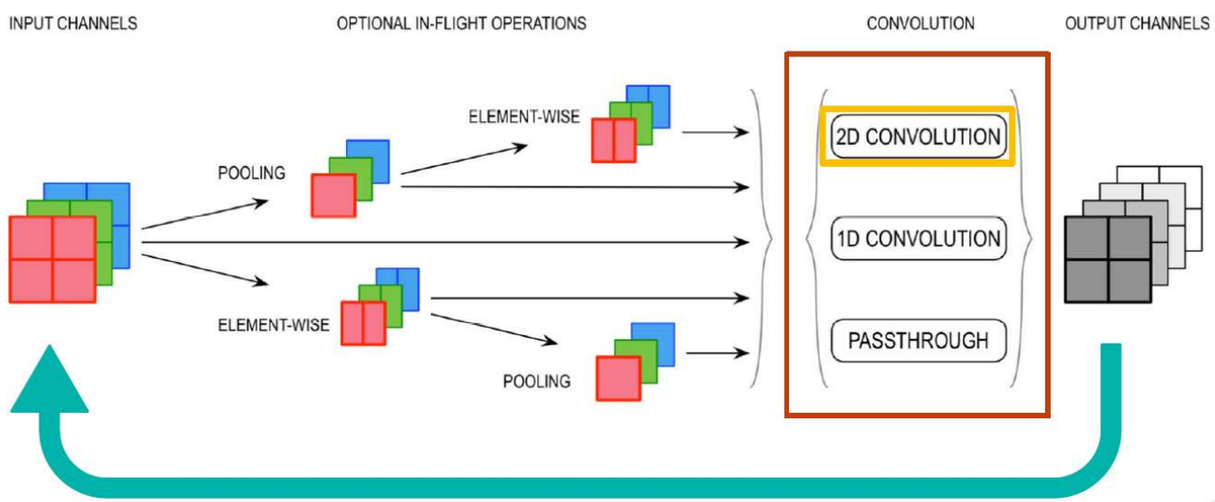
CNN: MAX7800 hybrid architecture

A specific architecture which includes a dedicated accelerator for CNN operators and also includes local memory (on die) for intermediate results and weights.

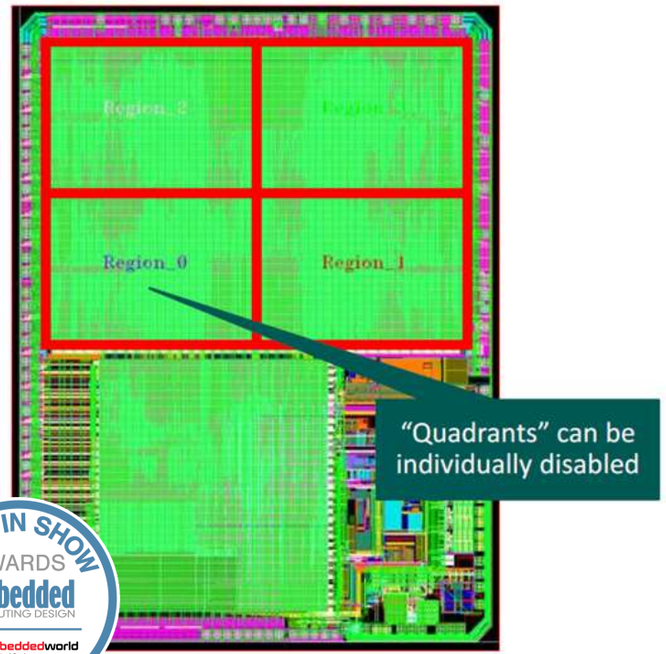
NOTE: only few key operators are accelerated. Flexibility on not supported operators still comes from software extension on the main MCU

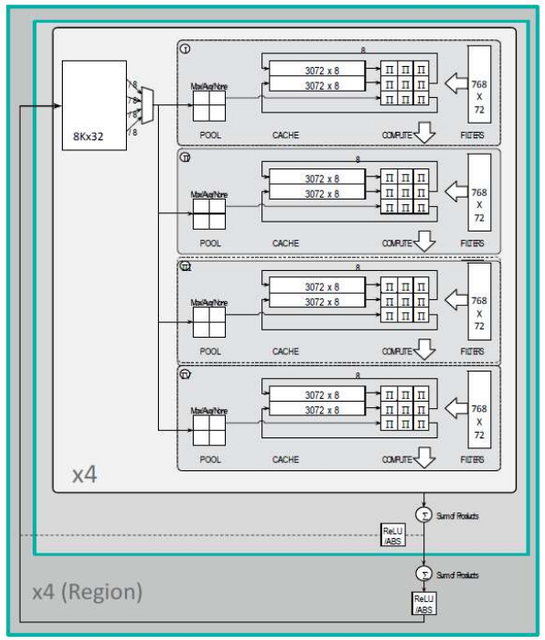
- **Conv2d** (1 × 1, 3 × 3)
- **Conv1d** (1 to 9)
- **ConvTranspose2d** (3 × 3)
- **AvgPool, MaxPool** (up to 16 × 16)
- **Flatten, Linear**
- **Activation: ReLU, Abs, None**
- Elementwise (up to 16) add, sub, binary OR, binary XOR
- Sequence Pool / Eltwise / Conv / Act counts as one layer
- 1, 2, 4, and 8-bit weights selectable per layer, 8-bit bias (optional)
- 8-bit data (clipping at activation stage) with optional 32-bit output for last layer
- Output shift (<<, >>) per layer (before clipping)
- **Padding** 0, 1, or 2
- **Stride** up to [16, 16]
- RISC-V core as “Smart DMA”
- Streaming mode with FIFOs

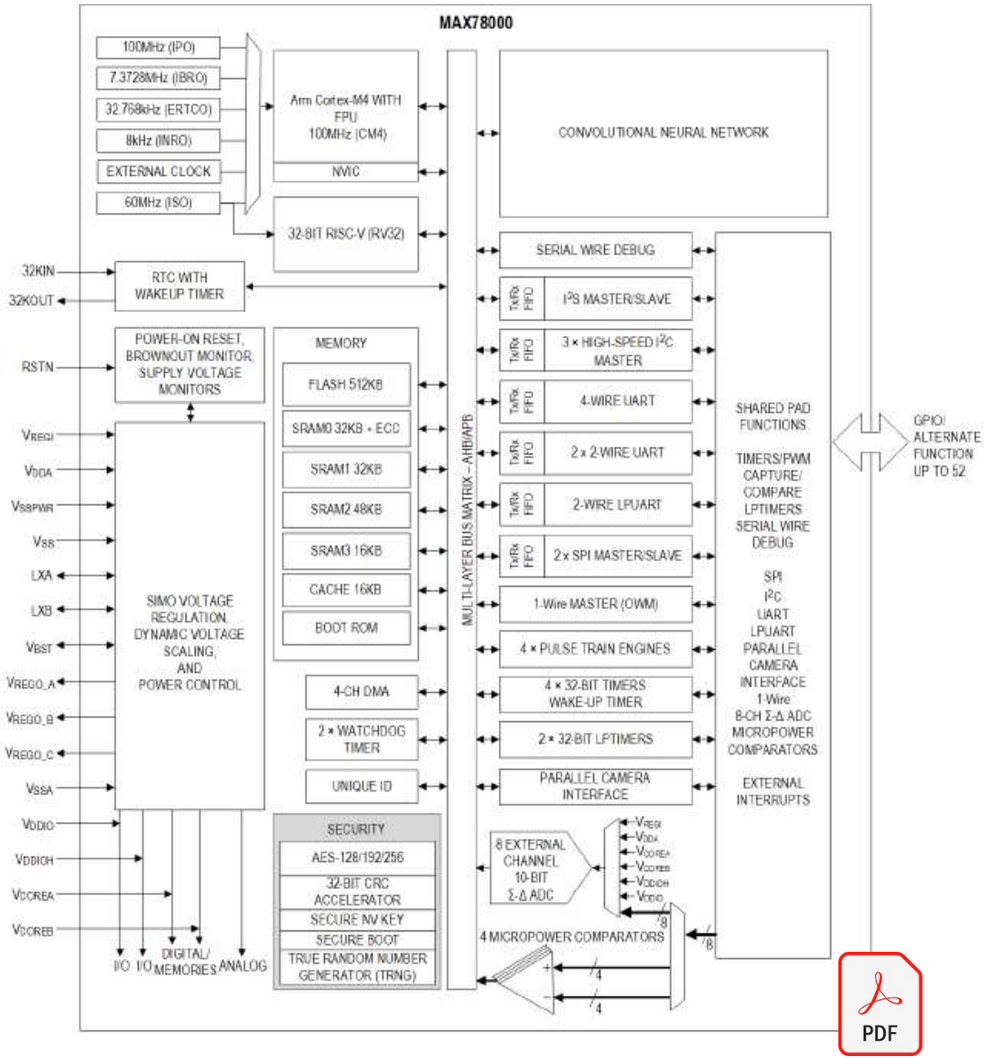


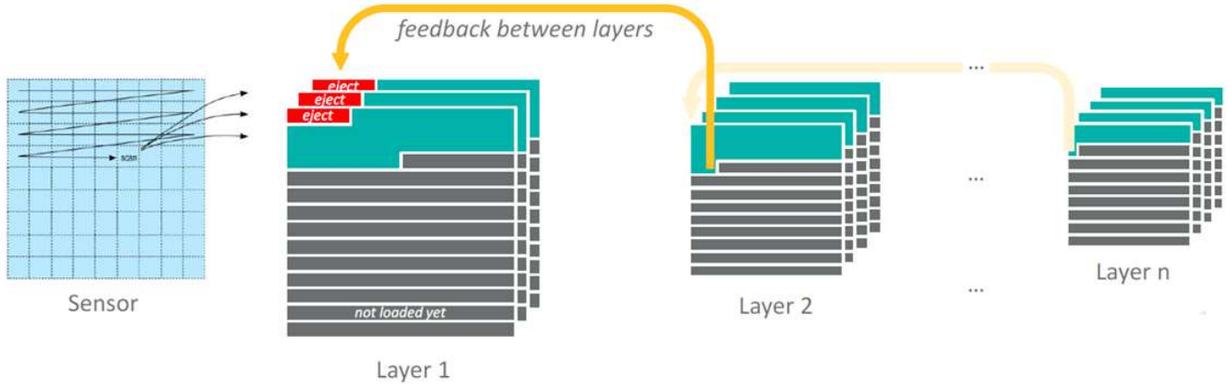


CNN: MAX7800 streaming mode





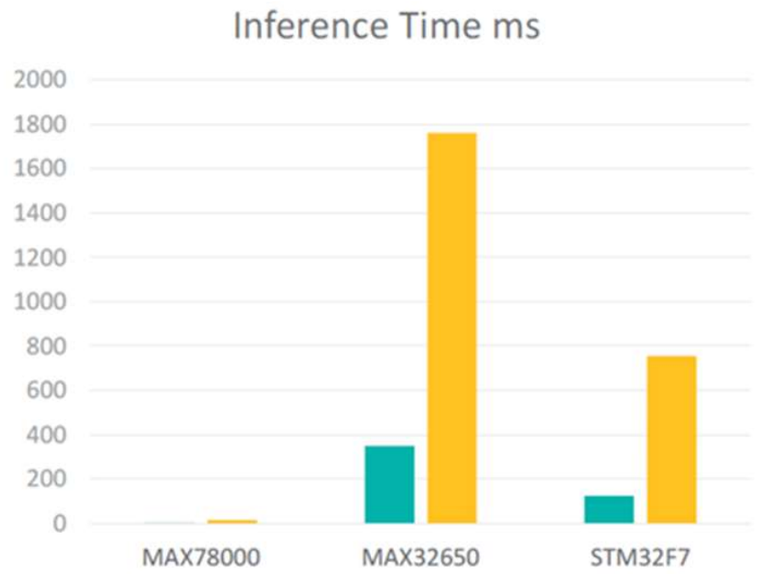


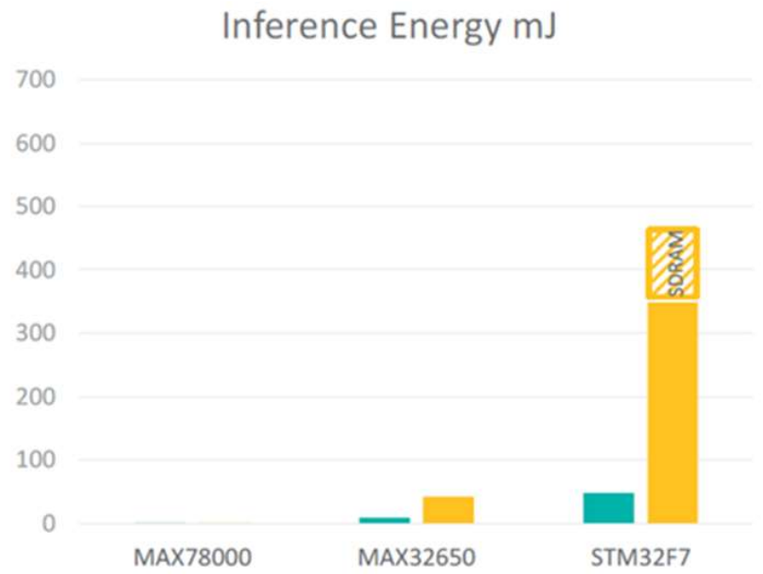


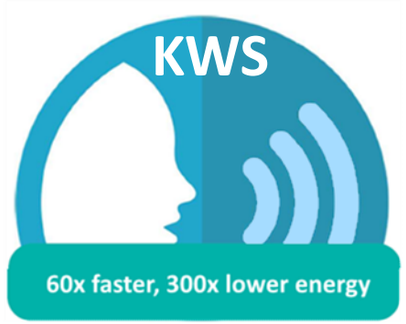
https://datasheets.maximintegrated.com/en/ds/MAX78000.pdf

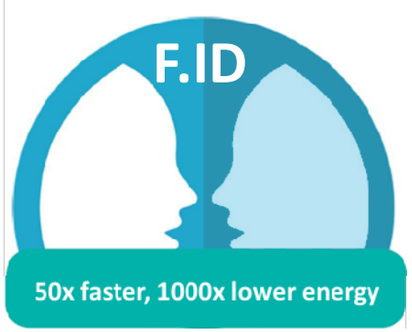
CNN: MAX7800 performance

Keyword spotting (KWS) and Face detection/identification are few of the most common use cases for TinyML. Custom hw architectures for CNN will provide an important gain in speed and energy used to perform the sw task in comparison to pure-software implementation. Speed improvements of >100x are doable with a small tradeoff on accuracy (<1-2%)









Network	MACC	MAX78000 CNN at 50 MHz[1]	MAX32650[2] Cortex-M4 at 120 MHz	STM32F7[2] Cortex-M7 at 216 MHz
KWS20	13,801,088	2.0ms, 0.14mJ	350ms, 8.37mJ	123ms, 47.5mJ[3]
FaceID	55,234,560	13.89ms, 0.40mJ	1760ms, 42.1mJ	754ms, 348 + 116mJ[4]

[1]28 billion operations/second
[2]ARM DSP with CMSIS-NN, running exact same INT8 network as MAX78000
[3]STMF722ZE, internal memories
[4]STMF746NG, external SDRAM at 25% typical active power

https://github.com/MaximIntegratedAI

There is One More Thing …

Rita Cucchiara

(Master Degree in Electronic Engineering '89; PhD in Computer Engineering '93 - University of Bologna).

Since 2005 is Full Professor of Information Processing Systems at the **University of Modena and Reggio Emilia**, where she is supervisor of the **Aimagelab Laboratory** and the future **"AI Academy"**; her main expertise are **Artificial Vision** and **Deep Learning** with more than 350 publications in peer review and international Journals.

She is also member of the **IIT (Italian Institute of Technology)** Board of Directors and from 2018 she is Director of **National Labs AIIS - Artificial Intelligence for Intelligent Systems for CINI** (National Interuniversity Consortium for Information Technology)

La forza dell'AI è nella forza di chi l'ha creata e di chi lo farà in futuro. L'intelligenza Artificiale siamo noi.

AImage Lab https://aimagelab.ing.unimore.it/imagelab/

PDF https://www.mise.gov.it/index.php/it/strategia-intelligenza-artificiale/contesto
https://www.mise.gov.it/images/stories/documenti/Proposte-per-una-strategia-italiana-2019.pdf

Thank You.

ai@ebv.com

Gianluca Filippini

EBV / FAE -ML Specialist