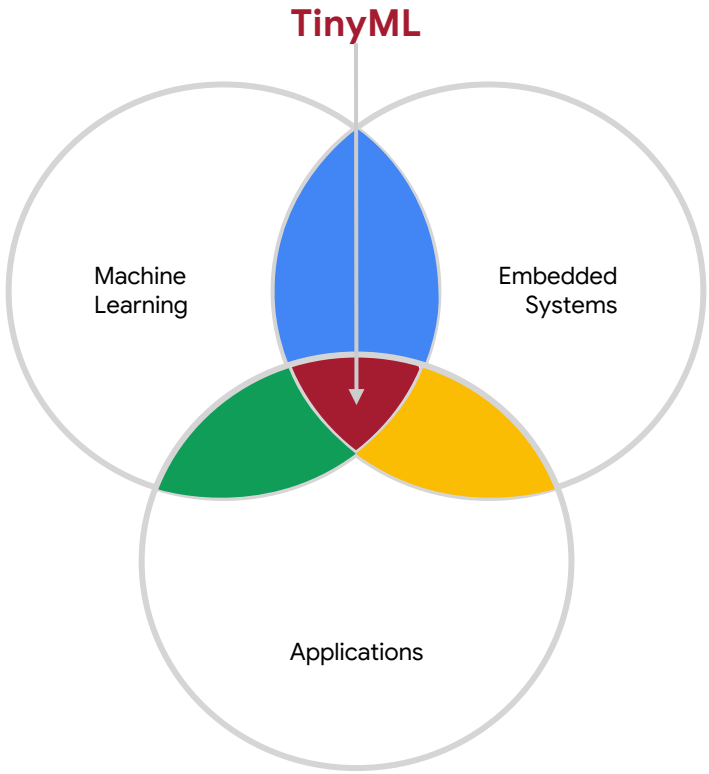



CFU Playground: Full-Stack Open-Source Framework for Tiny Machine Learning (TinyML) Acceleration on FPGAs

---

Shvetank Prakash, Tim Callahan, Joseph Bushagour, Colby Banbury, Alan V. Green, Pete Warden, Tim Ansell, Vijay Janapa Reddi

Build Your Own ML Processor Today:
https://github.com/google/CFU-Playground


TinyML
Machine Learning
Embedded Systems
Applications


amazon

TinyML Application Areas

Home

Office

Industry

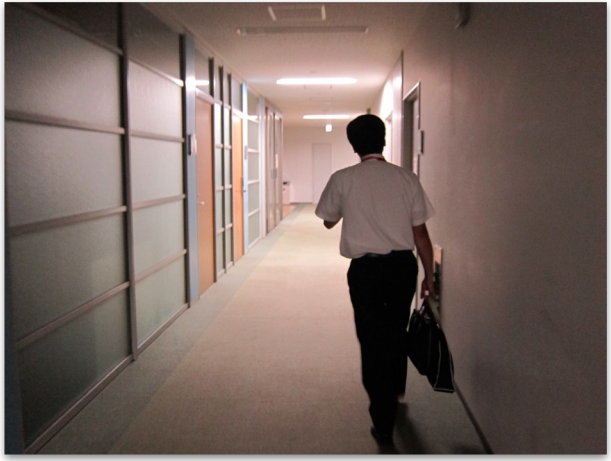
TinyML Application Areas

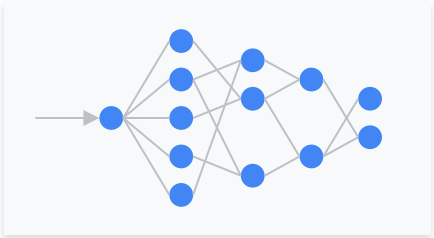
Home

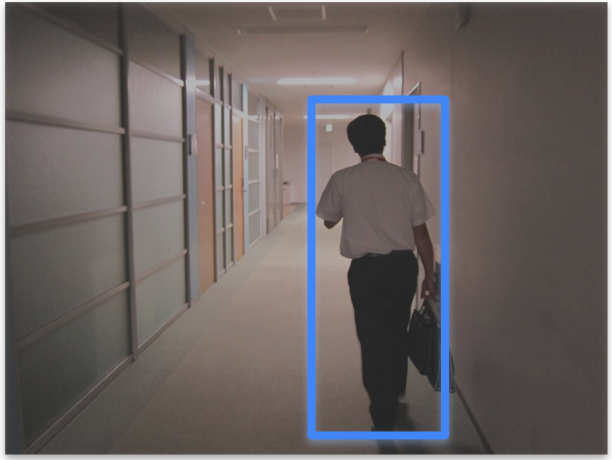
Office

Industry

TinyML Application Areas







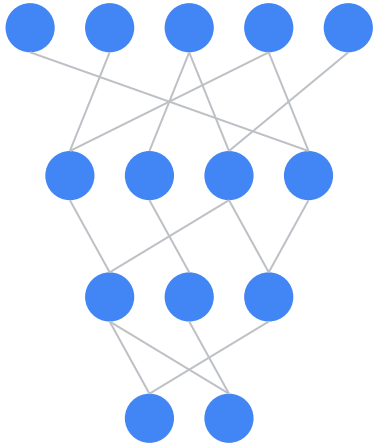
TinyML Application Areas

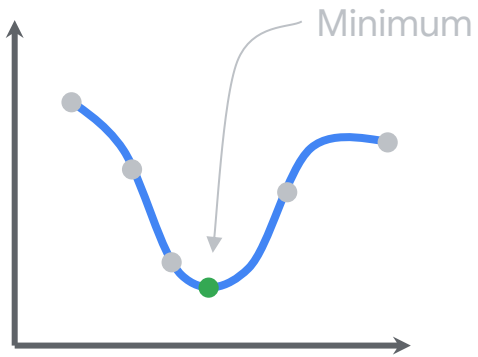
Home

Office

Industry

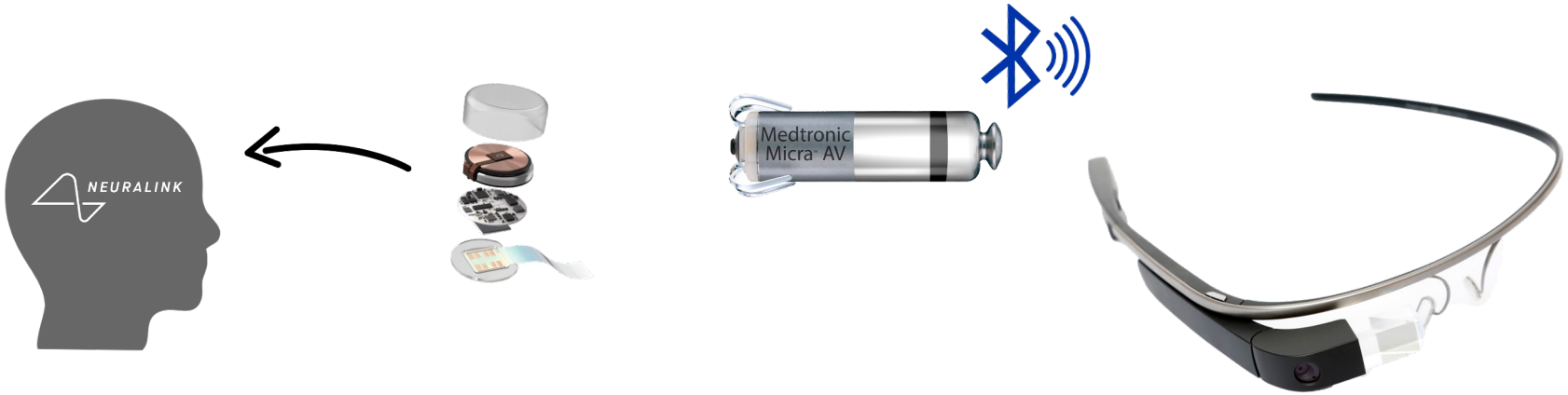
TinyML Application Areas



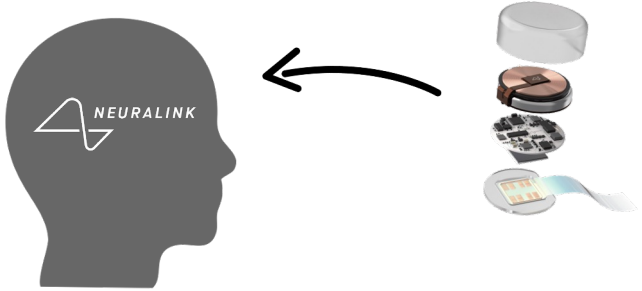


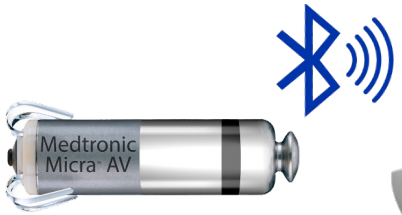
More Forward Looking Applications

More Forward Looking Applications

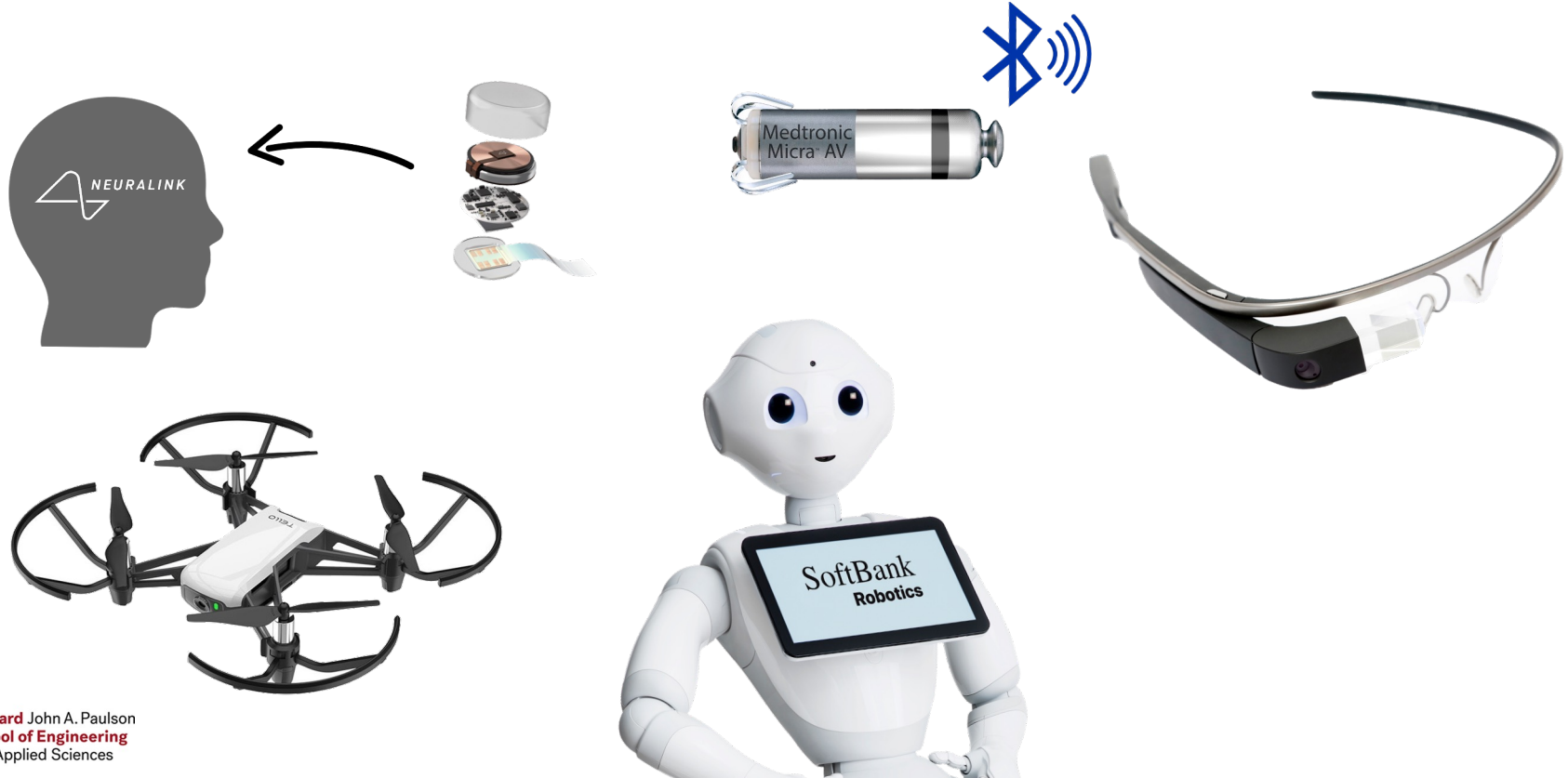


More Forward Looking Applications





More Forward Looking Applications



Challenges of TinyML

	Cloud & Mobile	>	Tiny
Platform			
Compute	1GHz–4GHz	~10X	1MHz–400MHz

Challenges of TinyML

	Cloud & Mobile	>	Tiny
Platform			
Compute	1GHz–4GHz	~10X	1MHz–400MHz
Memory	512MB–64GB	~10000X	2KB–512KB

Challenges of TinyML

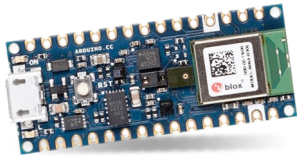
	Cloud & Mobile	>	Tiny
Platform			
Compute	1GHz–4GHz	~10X	1MHz–400MHz
Memory	512MB–64GB	~10000X	2KB–512KB
Storage	64GB–4TB	~100000X	32KB–2MB

Challenges of TinyML

	Cloud & Mobile	>	Tiny
Platform			
Compute	1GHz–4GHz	~10X	1MHz–400MHz
Memory	512MB–64GB	~10000X	2KB–512KB
Storage	64GB–4TB	~100000X	32KB–2MB
Power	30W–100W	~1000X	150µW–23.5mW

Keyword Spotting Model

One Conv2D followed by a **single** Dense Layer!

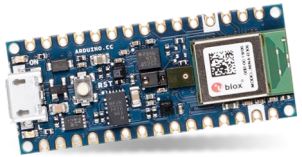


TinyML hardware typically have only **KBs** of RAM (memory)

Keyword Spotting Model

One Conv2D followed by a single Dense Layer!

<17 KBs after quantization for deployment



TinyML hardware typically have only **KBs** of RAM (memory)

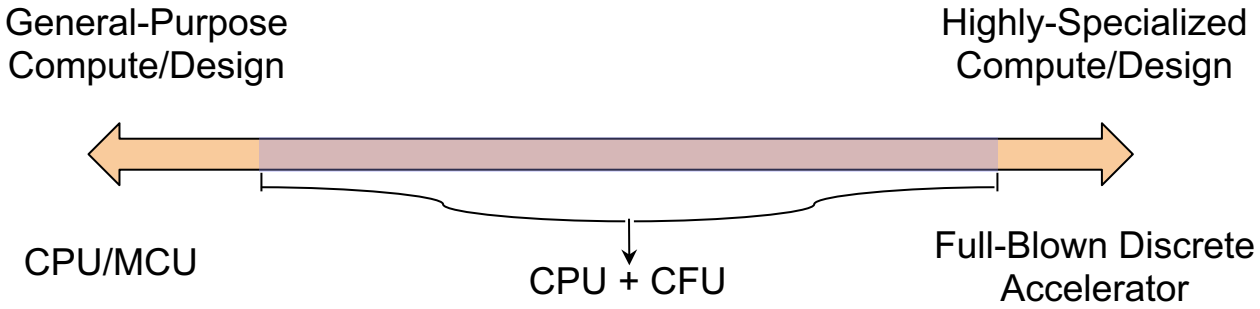
Agenda

1. **Tiny Machine Learning**
2. The Need for Agile and Full-Stack Frameworks
3. Custom Function Units and CFU Playground
4. FPGA Acceleration for Image Classification and Keyword Spotting
5. Design Space Exploration: CFU vs. CPU

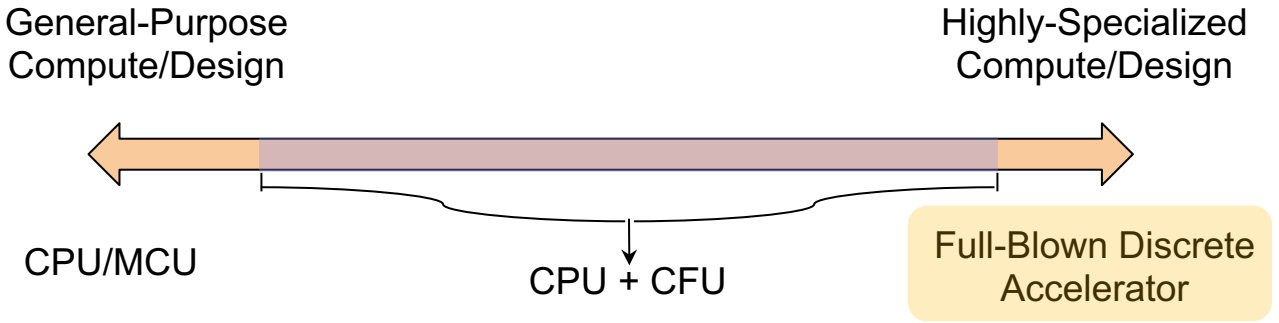
Agenda

1. Tiny Machine Learning
2. **The Need for Agile and Full-Stack Frameworks**
3. Custom Function Units and CFU Playground
4. FPGA Acceleration for Image Classification and Keyword Spotting
5. Design Space Exploration: CFU vs. CPU

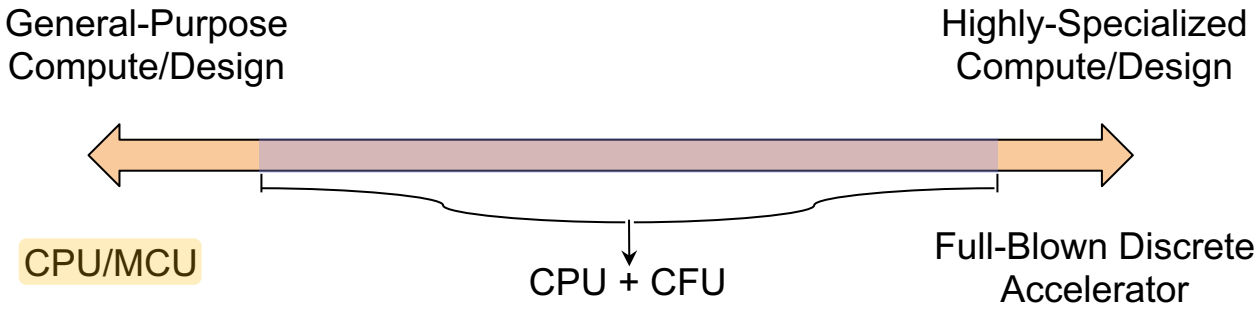
The Need for Agile and Full-Stack Frameworks



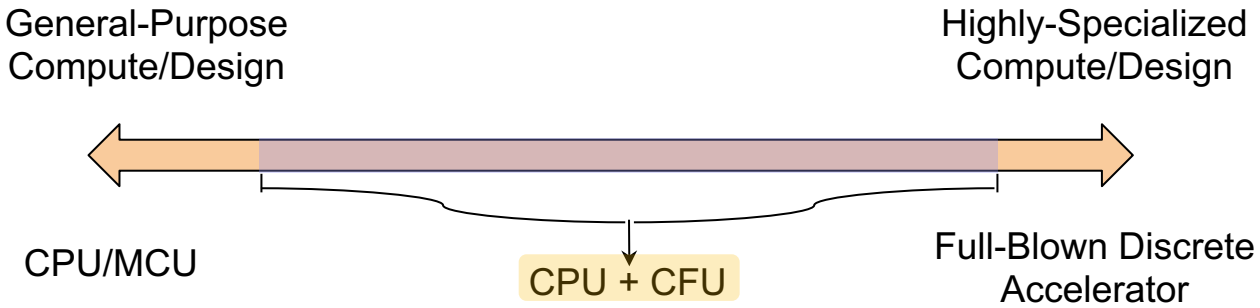
The Need for Agile and Full-Stack Frameworks



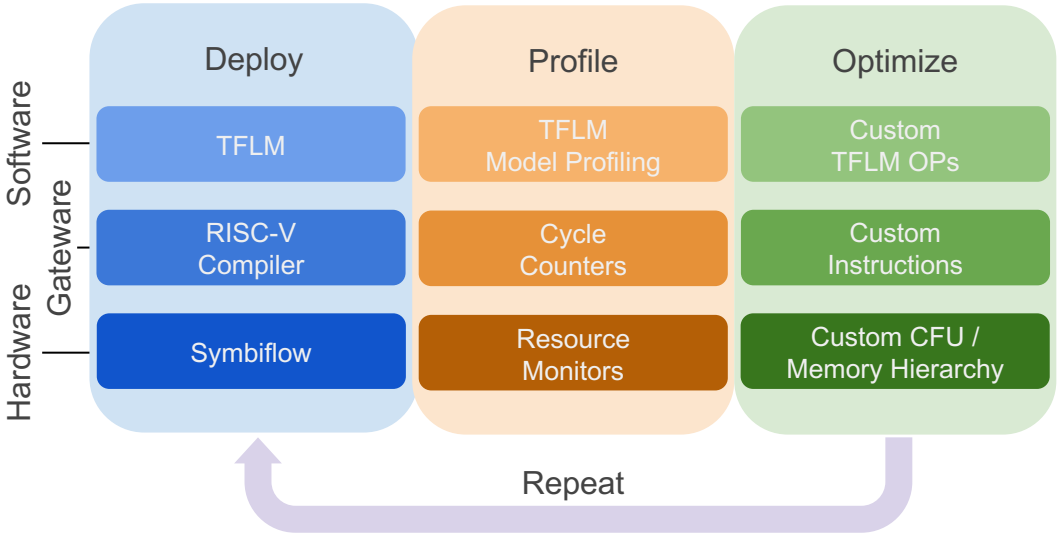
The Need for Agile and Full-Stack Frameworks



The Need for Agile and Full-Stack Frameworks

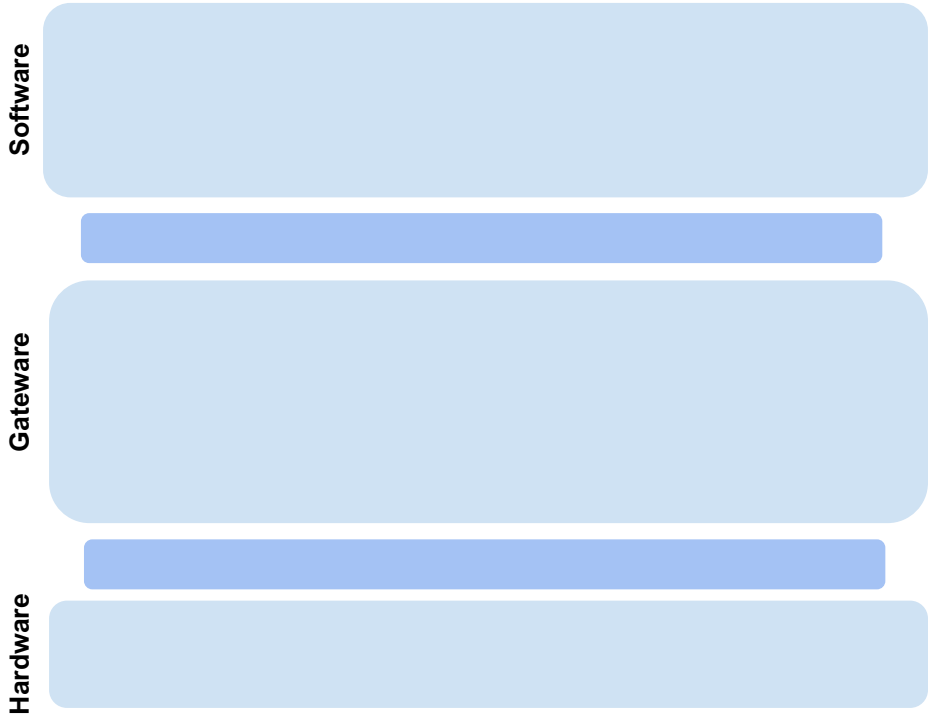


The Need for Agile and Full-Stack Frameworks



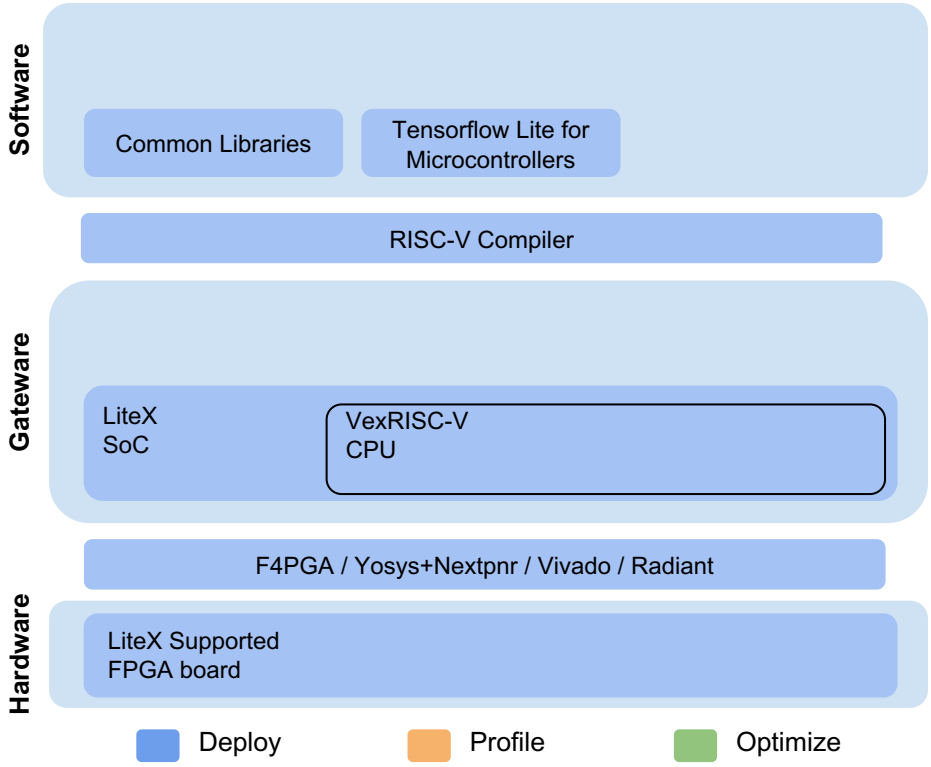
Agile Design Methodology

The Need for Agile and Full-Stack Frameworks



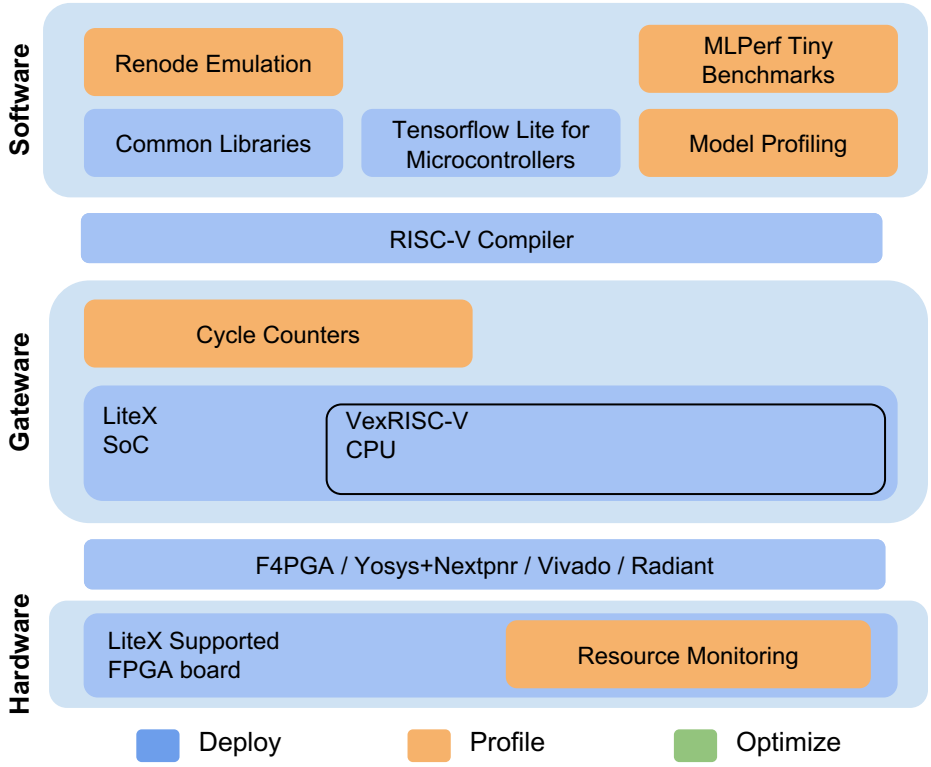
Full-Stack Open-Source Framework

The Need for Agile and Full-Stack Frameworks



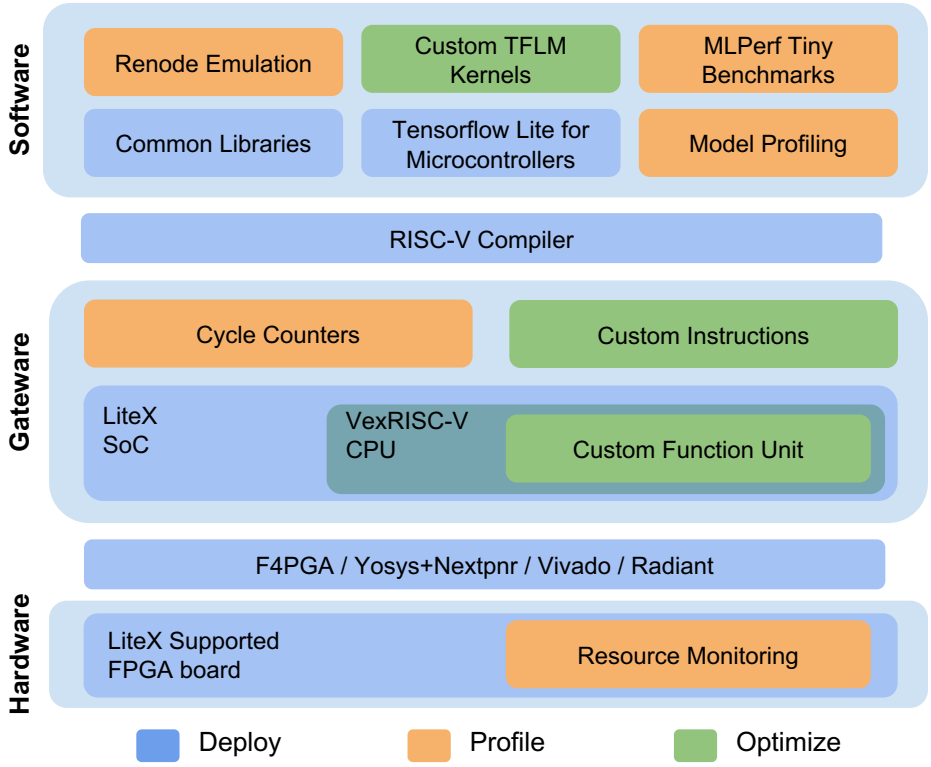
Full-Stack Open-Source Framework

The Need for Agile and Full-Stack Frameworks



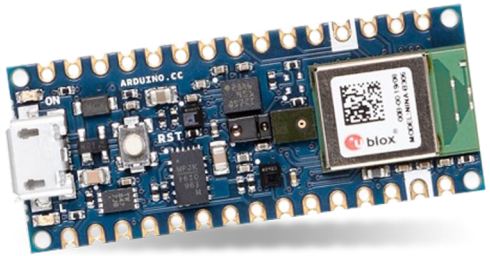
Full-Stack Open-Source Framework

The Need for Agile and Full-Stack Frameworks



Full-Stack Open-Source Framework

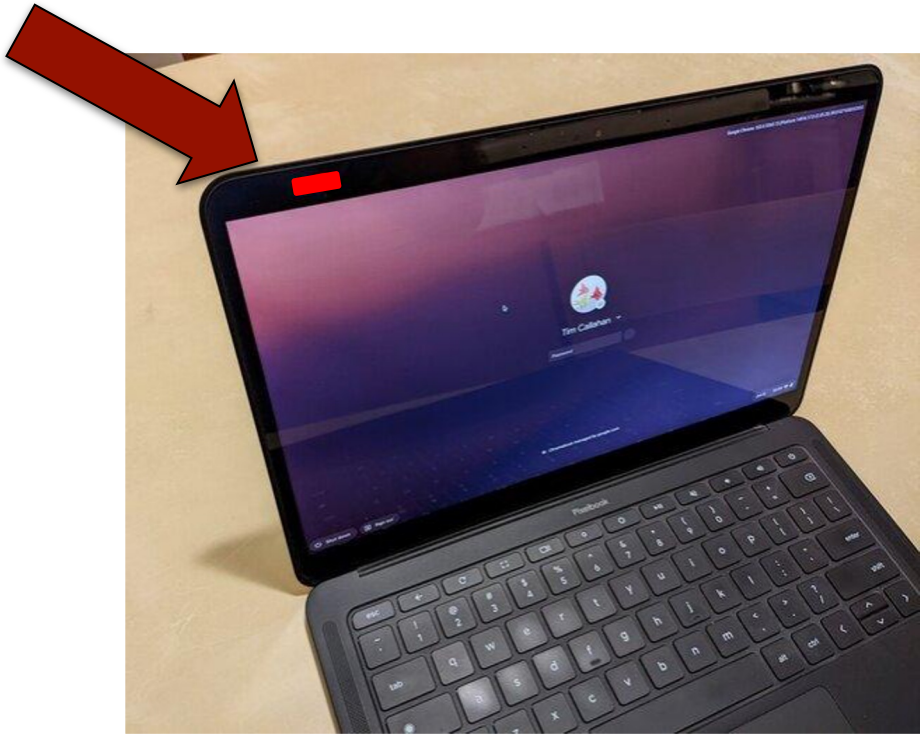
Accelerating TinyML on FPGAs



MCUs: **KBs** of RAM, **Fixed/slow** processor

Specialized Hardware Customization (on FPGAs)

Real World Use Case

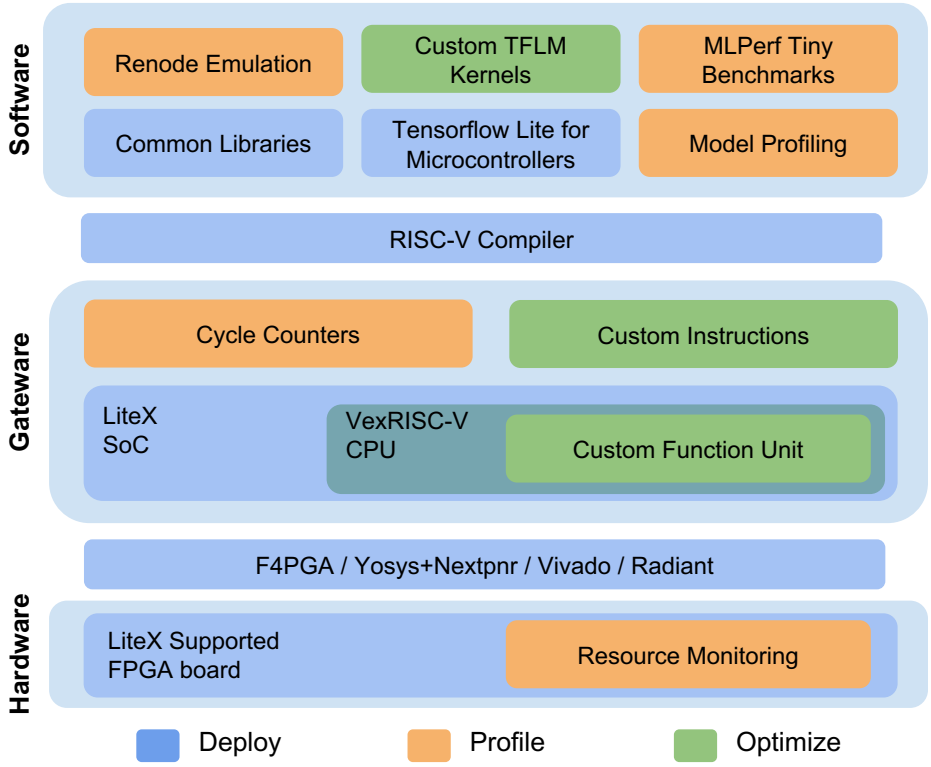


Chromebook Sensor Designed with CFU Playground

Agenda

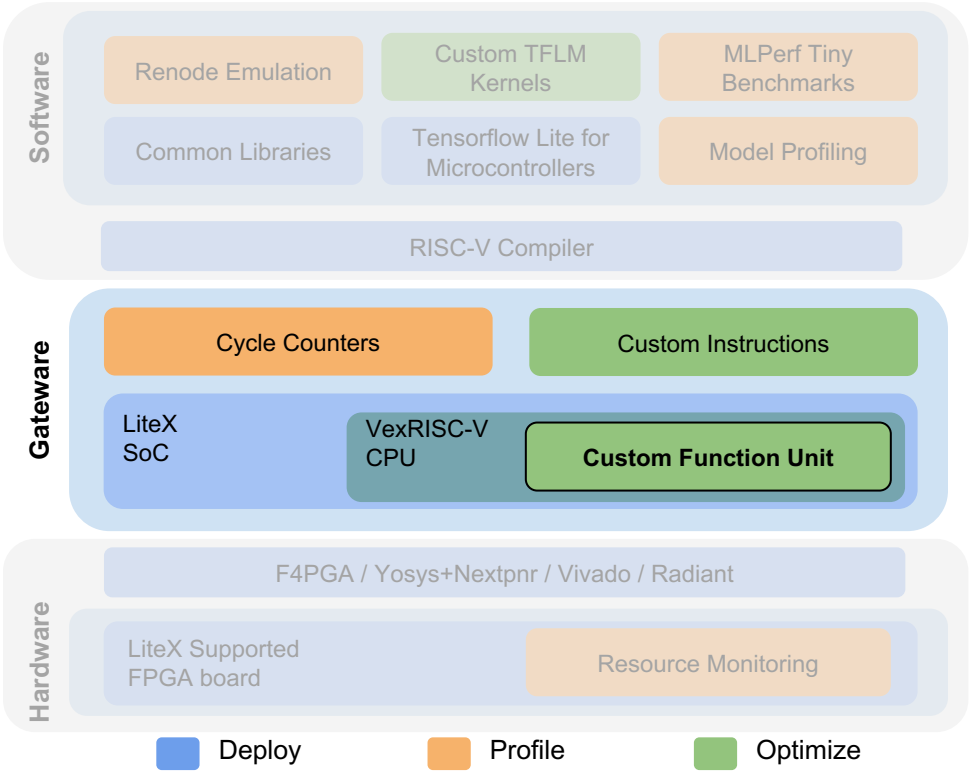
1. Tiny Machine Learning
2. The Need for Agile and Full-Stack Frameworks
3. **Custom Function Units and CFU Playground**
4. FPGA Acceleration for Image Classification and Keyword Spotting
5. Design Space Exploration: CFU vs. CPU

CFU Playground



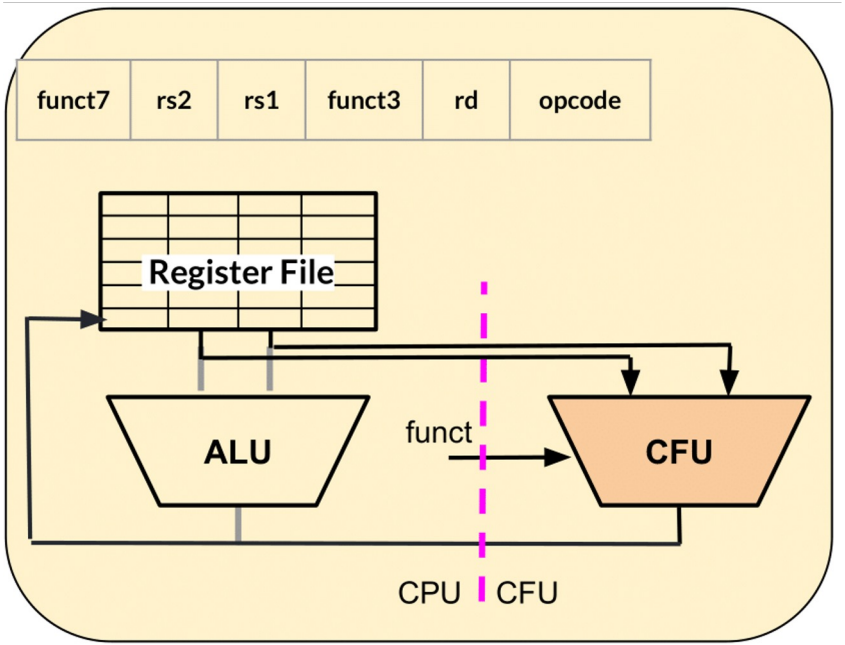
Full-Stack Open-Source Framework

CFU Playground



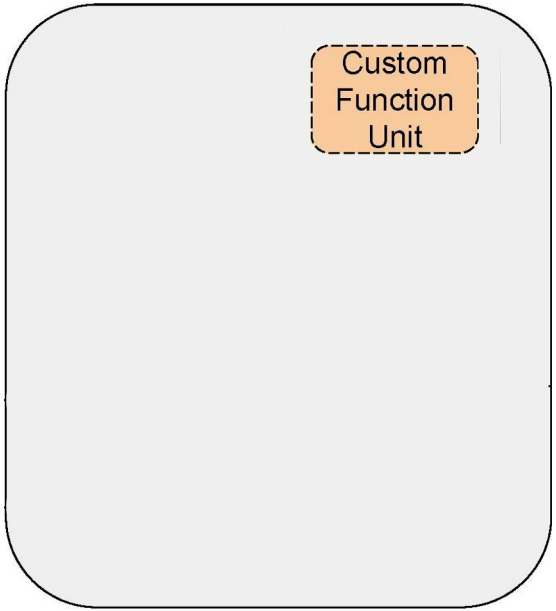
Full-Stack Open-Source Framework

Custom Function Units (CFU)

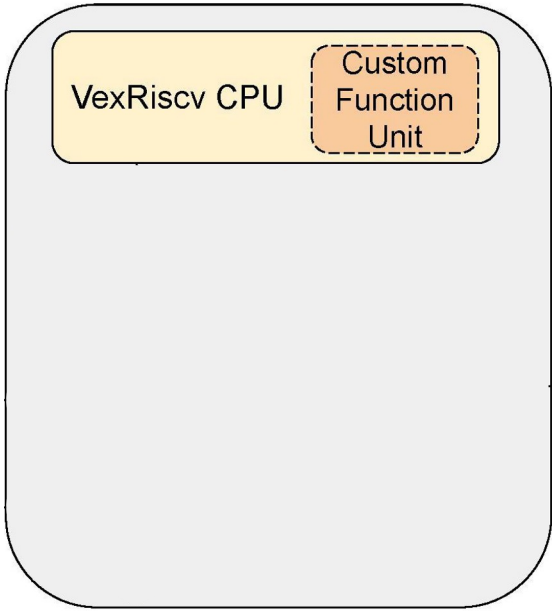


Acceleration via Custom Function Unit (CFU)

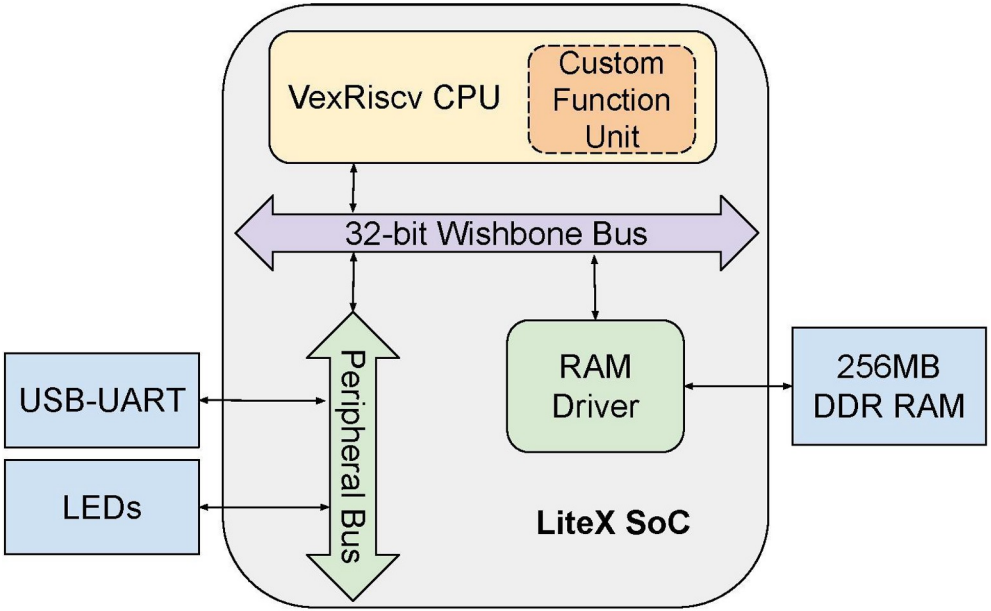
System On-Chip (SoC) Integration



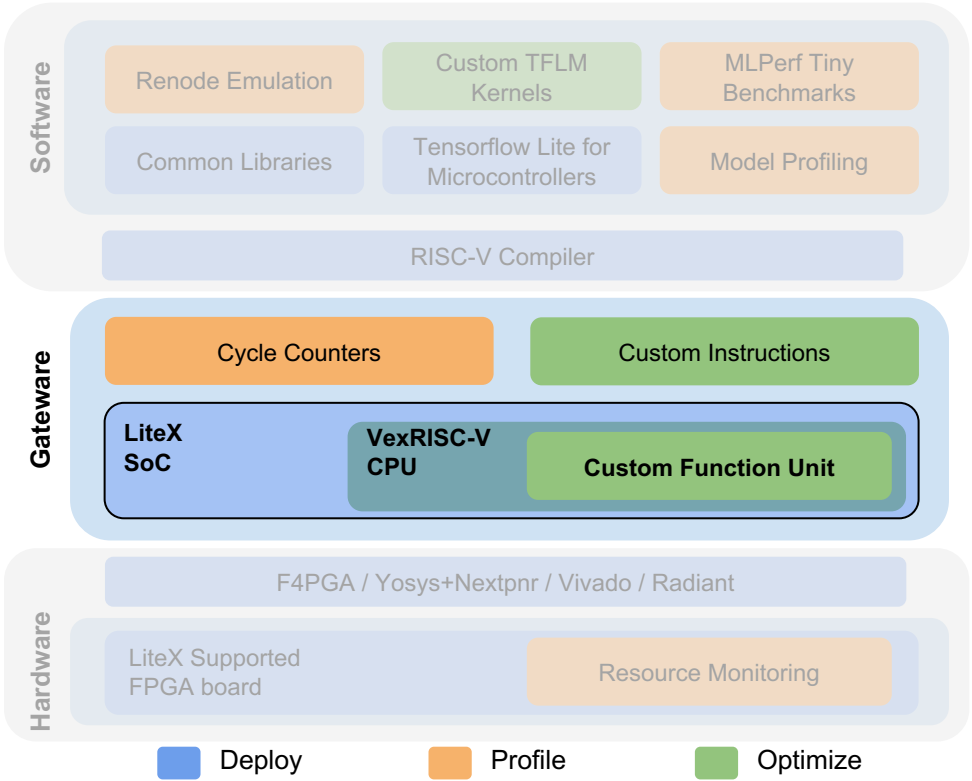
System On-Chip (SoC) Integration



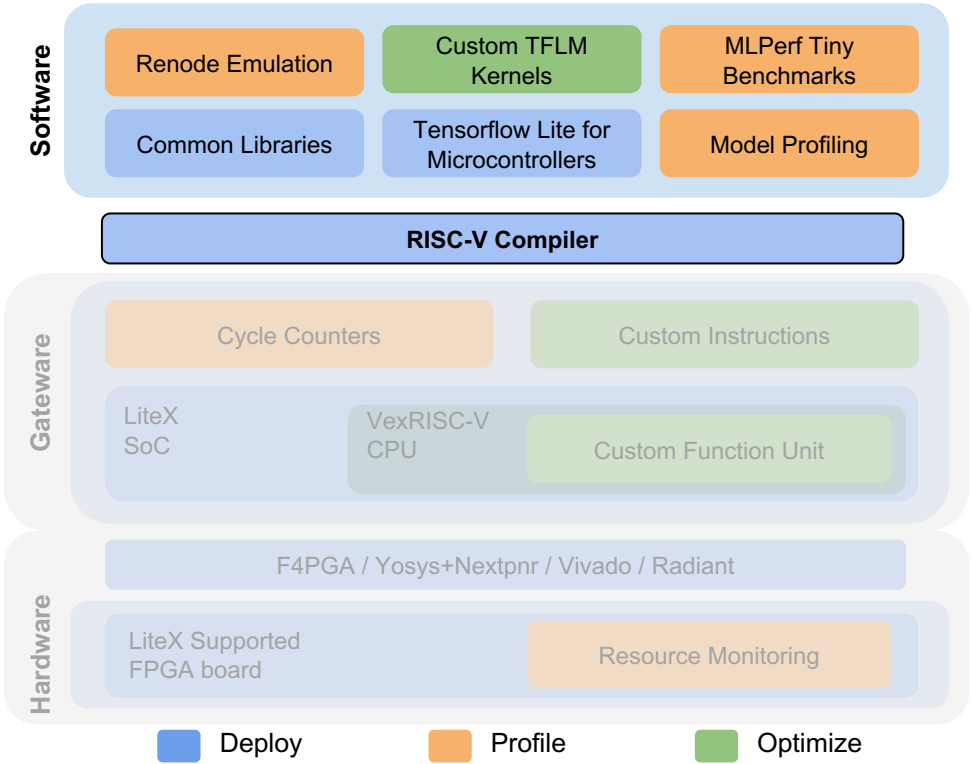
System On-Chip (SoC) Integration



System On-Chip (SoC) Integration



CFU Software Interface



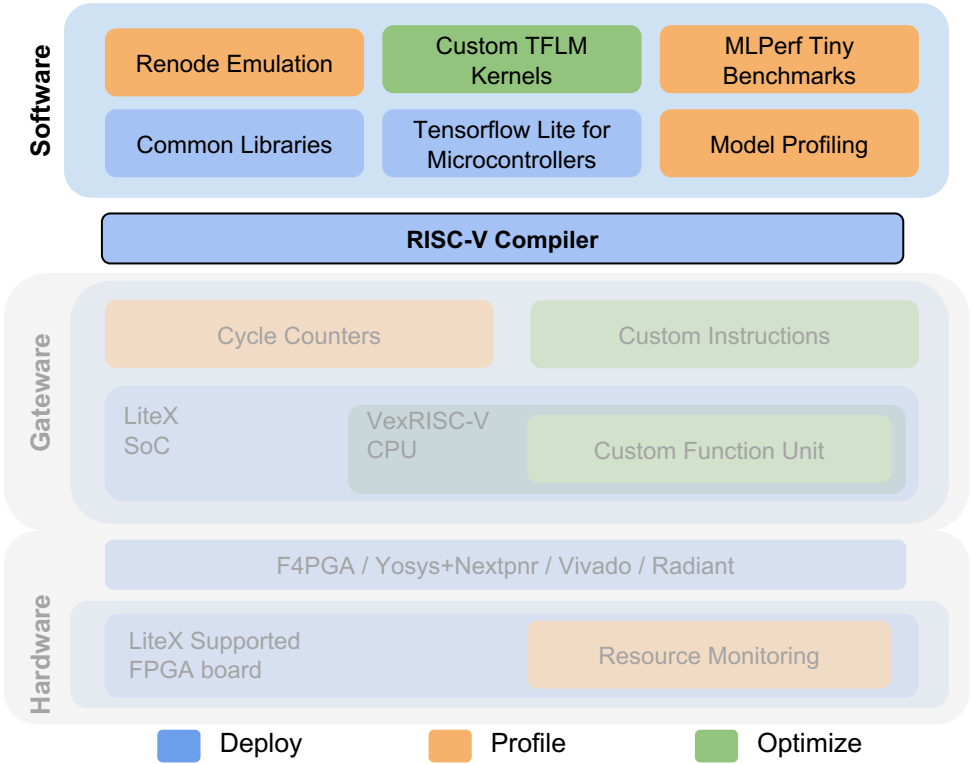
Access new instruction as C function call:

```
rslt = cfu_op(funct3, funct7, op1, op2);
```

Compile-time constants

C / C++ variables / expressions

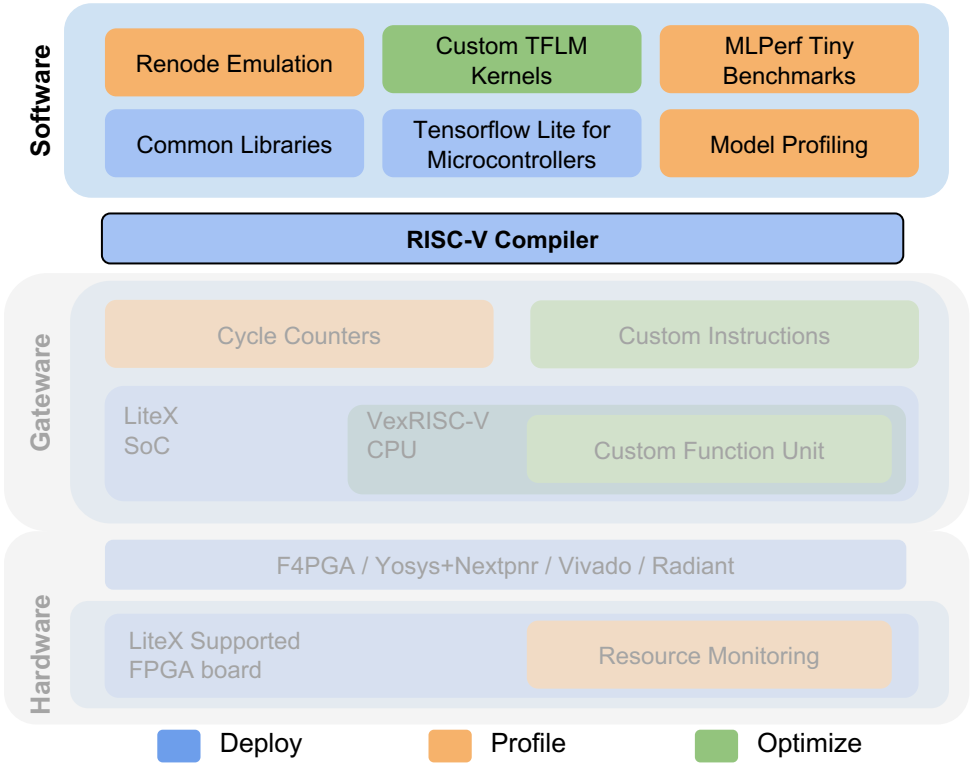
CFU Software Interface



Custom instruction macros intermix with plain C code:

```
t1 = *x;
t2 = cfu_op(0, 0, t1, b);
t3 = cfu_op(1, 0, t2, b);
*x = t3;
```

CFU Software Interface



Custom instruction macros intermix with plain C code:

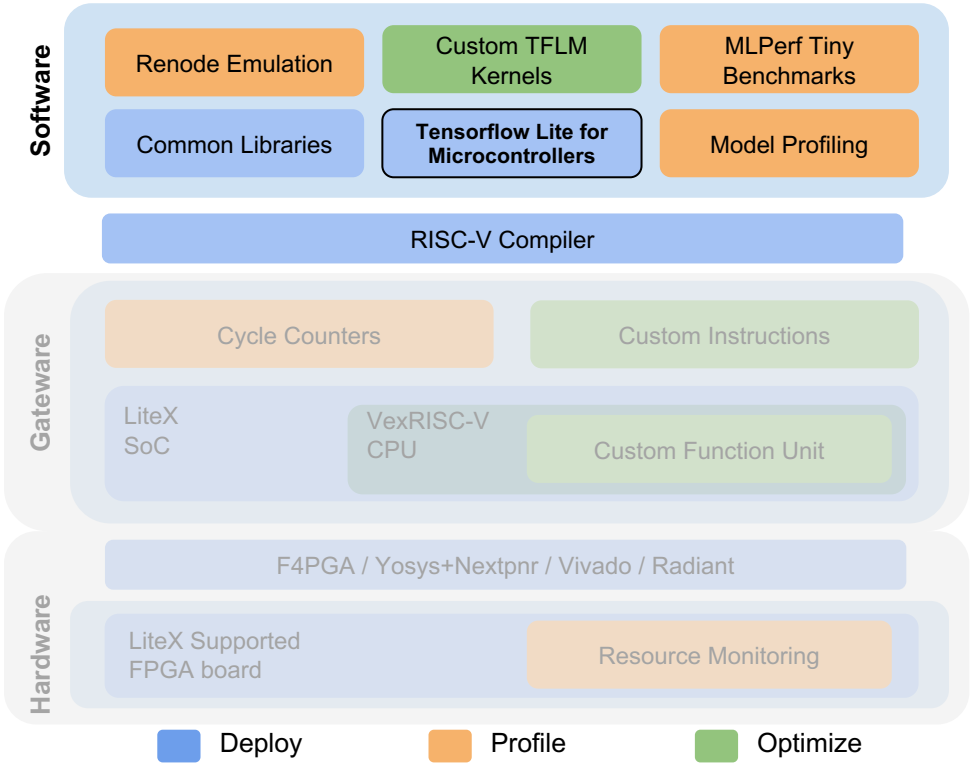
```
t1 = *x;
t2 = cfu_op(0, 0, t1, b);
t3 = cfu_op(1, 0, t2, b);
*x = t3;
```

Compiled and disassembled:

```
400001a0:   00812783    lw          a5,8(sp)
400001a4:   00d7878b    cfu[0,0]    a5, a5, a3
400001a8:   00d7978b    cfu[0,1]    a5, a5, a3
400001ac:   00f12423    sw          a5,8(sp)
```

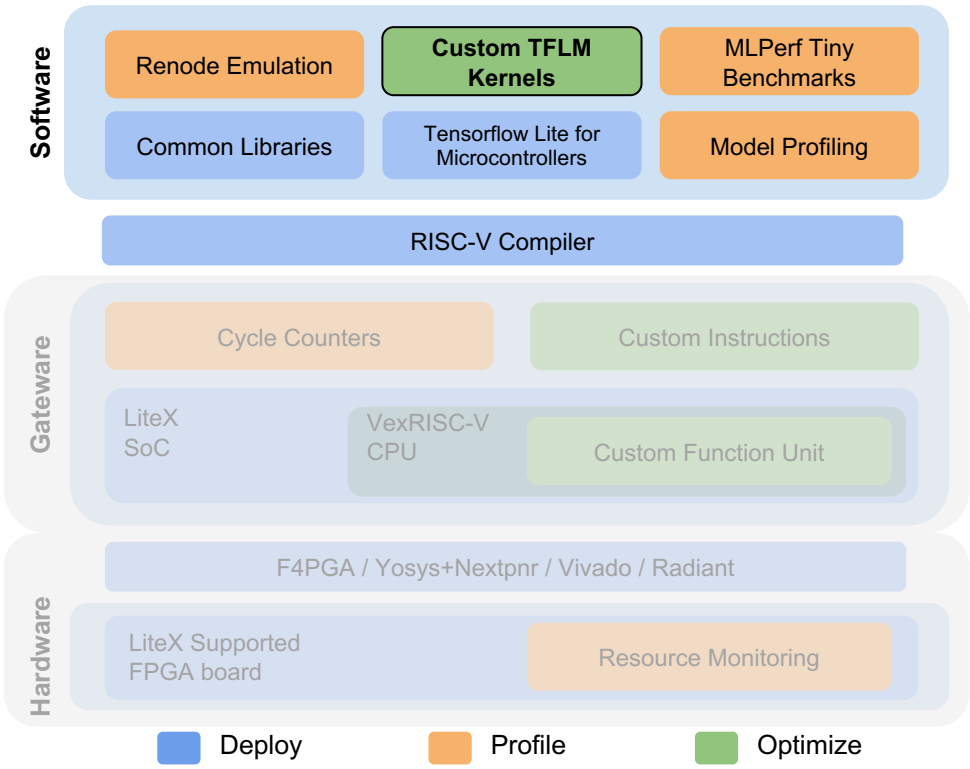
No overhead!

ML Deployment Framework



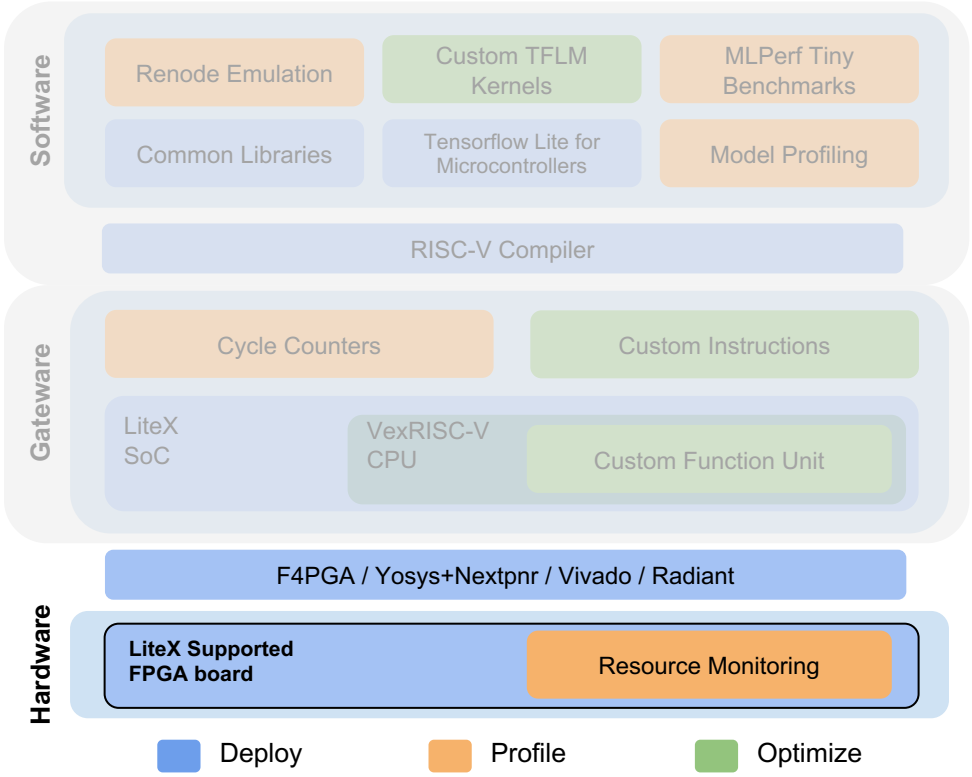

```
const int32_t input_offset = params.input_offset;  // r = s(q - Z)

for (int batch = 0; batch < batches; ++batch) {
  for (int out_y = 0; out_y < output_height; ++out_y) {
    const int in_y_origin = (out_y * stride_height) - pad_height;
    for (int out_x = 0; out_x < output_width; ++out_x) {
      const int in_x_origin = (out_x * stride_width) - pad_width;
      for (int out_channel = 0; out_channel < output_depth; ++out_channel) {
        int32_t acc = 0;
        for (int filter_y = 0; filter_y < filter_height; ++filter_y) {
          const int in_y = in_y_origin + dilation_height_factor * filter_y;
          for (int filter_x = 0; filter_x < filter_width; ++filter_x) {
            const int in_x = in_x_origin + dilation_width_factor * filter_x;

            // Zero padding by omitting the areas outside the image.
            const bool is_point_inside_image =
                (in_x >= 0) && (in_x < input_width) && (in_y >= 0) &&
                (in_y < input_height);

            if (!is_point_inside_image) {
              continue;
            }

            for (int in_channel = 0; in_channel < input_depth; ++in_channel) {
              int32_t input_val = input_data[Offset(input_shape, batch, in_y,
                                                    in_x, in_channel)];
              int32_t filter_val = filter_data[Offset(
                  filter_shape, out_channel, filter_y, filter_x, in_channel)];

              acc += filter_val * (input_val + input_offset);
            }
          }
        }
        (use acc)
```

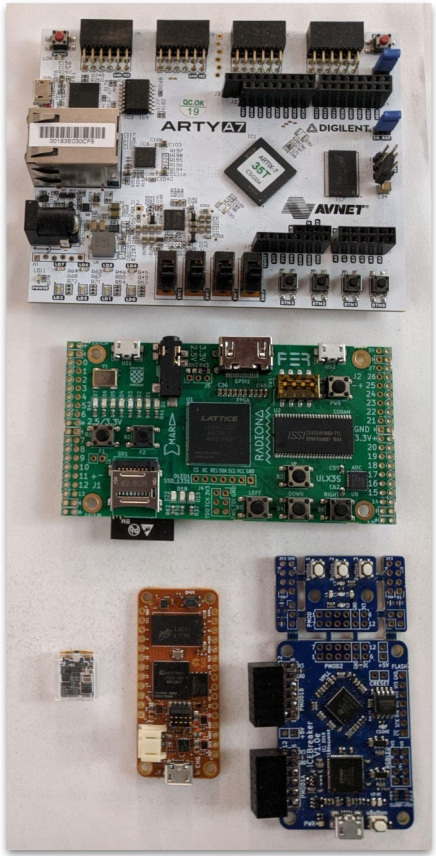
Accelerated Kernels




```
const int32_t input_offset = params.input_offset;  // r = s(q - Z)
// CFU: copy input_offset into the CFU
cfu_init_offset(input_offset);

for (int batch = 0; batch < batches; ++batch) {
  for (int out_y = 0; out_y < output_height; ++out_y) {
    const int in_y_origin = (out_y * stride_height) - pad_height;
    for (int out_x = 0; out_x < output_width; ++out_x) {
      const int in_x_origin = (out_x * stride_width) - pad_width;
      for (int out_channel = 0; out_channel < output_depth; ++out_channel) {

        //int32_t acc = 0;
        // CFU: set the CFU internal acc to ZERO
        cfu_clear_acc();

        for (int filter_y = 0; filter_y < filter_height; ++filter_y) {
          const int in_y = in_y_origin + dilation_height_factor * filter_y;
          for (int filter_x = 0; filter_x < filter_width; ++filter_x) {
            const int in_x = in_x_origin + dilation_width_factor * filter_x;

            ...

            for (int in_channel = 0; in_channel < input_depth; ++in_channel) {
              int32_t input_val = input_data[Offset(input_shape, batch, in_y,
                                                    in_x, in_channel)];
              int32_t filter_val = filter_data[Offset(
                  filter_shape, out_channel, filter_y, filter_x, in_channel)];

              // acc += filter_val * (input_val + input_offset);
              // CFU: add-multiply-accumulate in the CFU
              cfu_macc_with_offset(filter_val, input_val);
            }
          }
        }

        // CFU: retrieve final acc value from the CFU
        int32_t acc = cfu_get_acc();
```

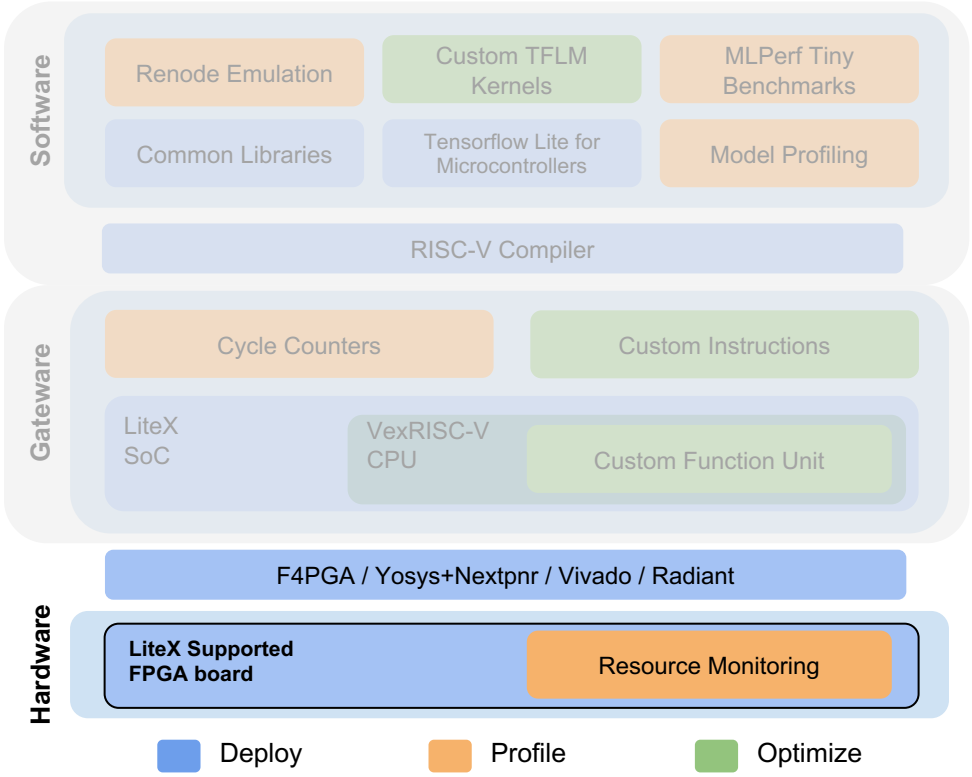
Hardware In-The-Loop

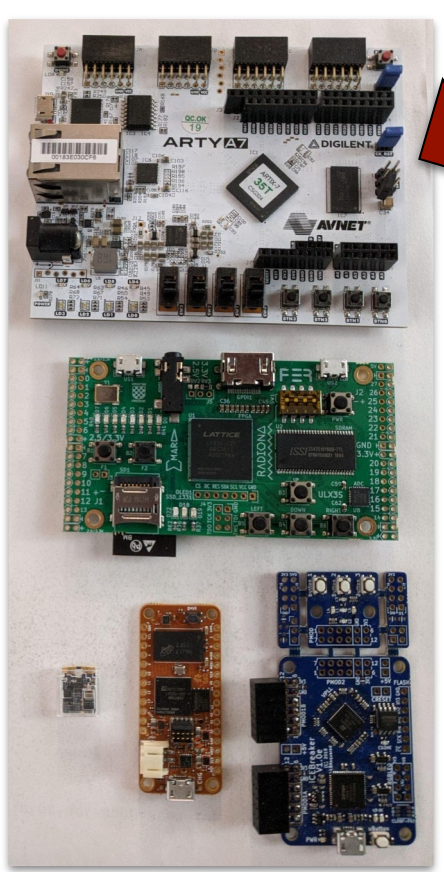




Diverse Family of FPGA Boards

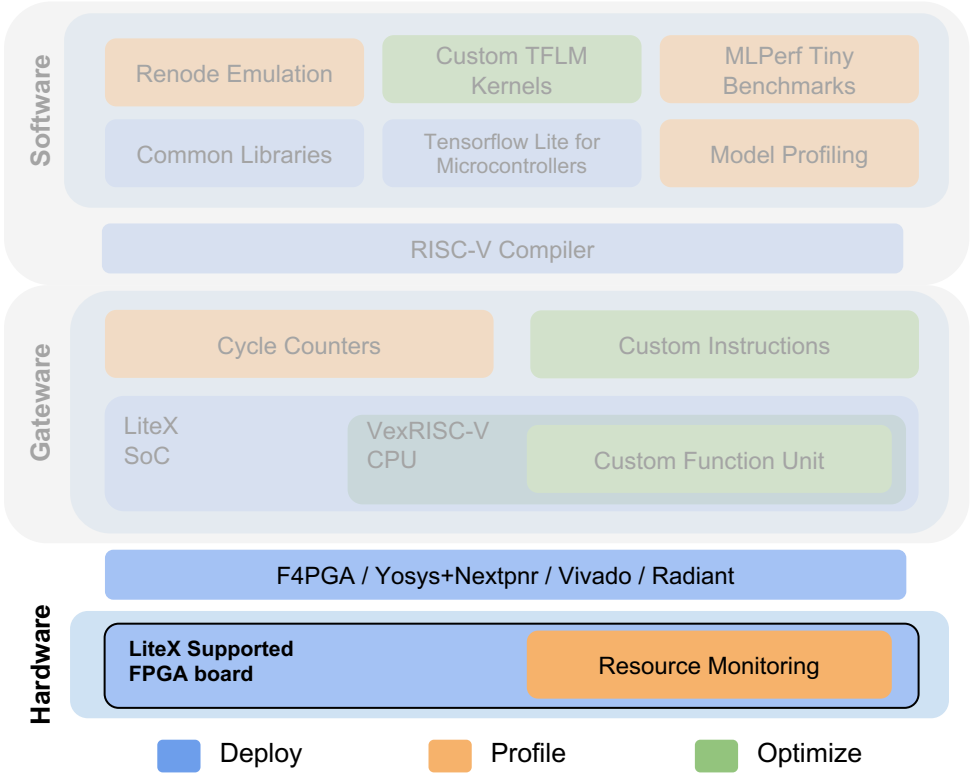
Hardware In-The-Loop

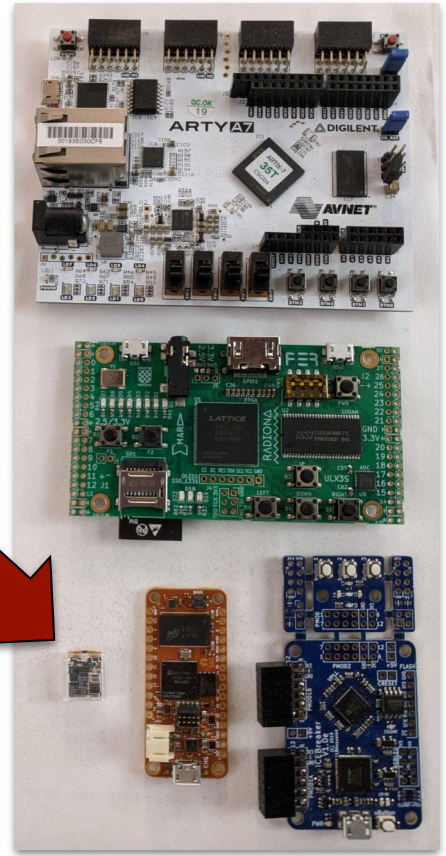




Diverse Family of FPGA Boards

Hardware In-The-Loop



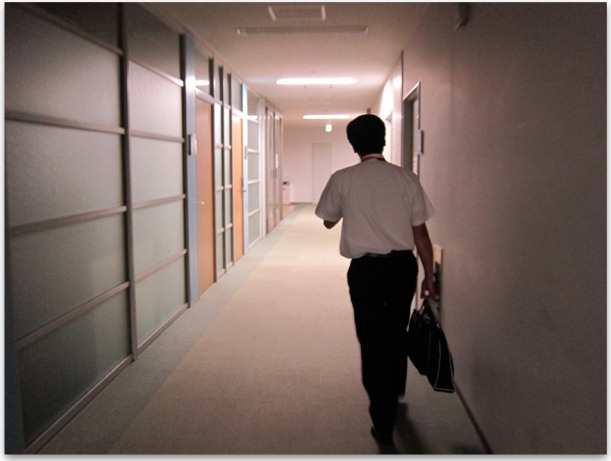


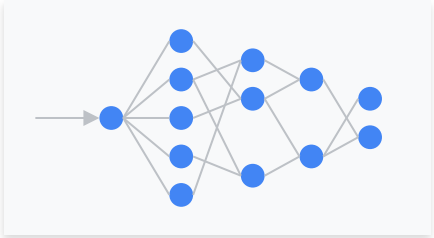
Diverse Family of FPGA Boards

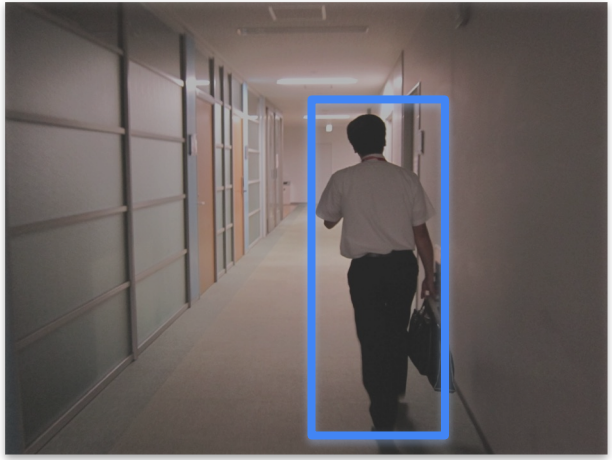
Agenda

1. Tiny Machine Learning
2. The Need for Agile and Full-Stack Frameworks
3. Custom Function Units and CFU Playground
4. **FPGA Acceleration for Image Classification and Keyword Spotting**
5. Design Space Exploration: CFU vs. CPU

FPGA Acceleration for Image Classification

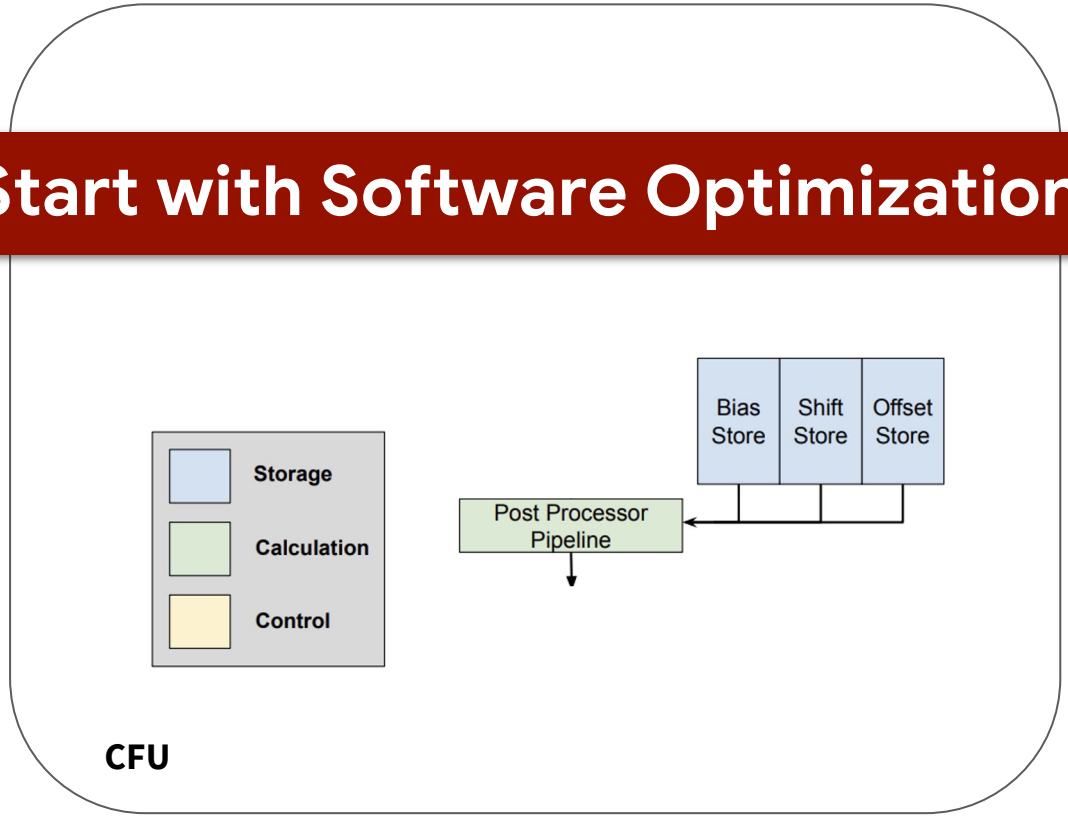




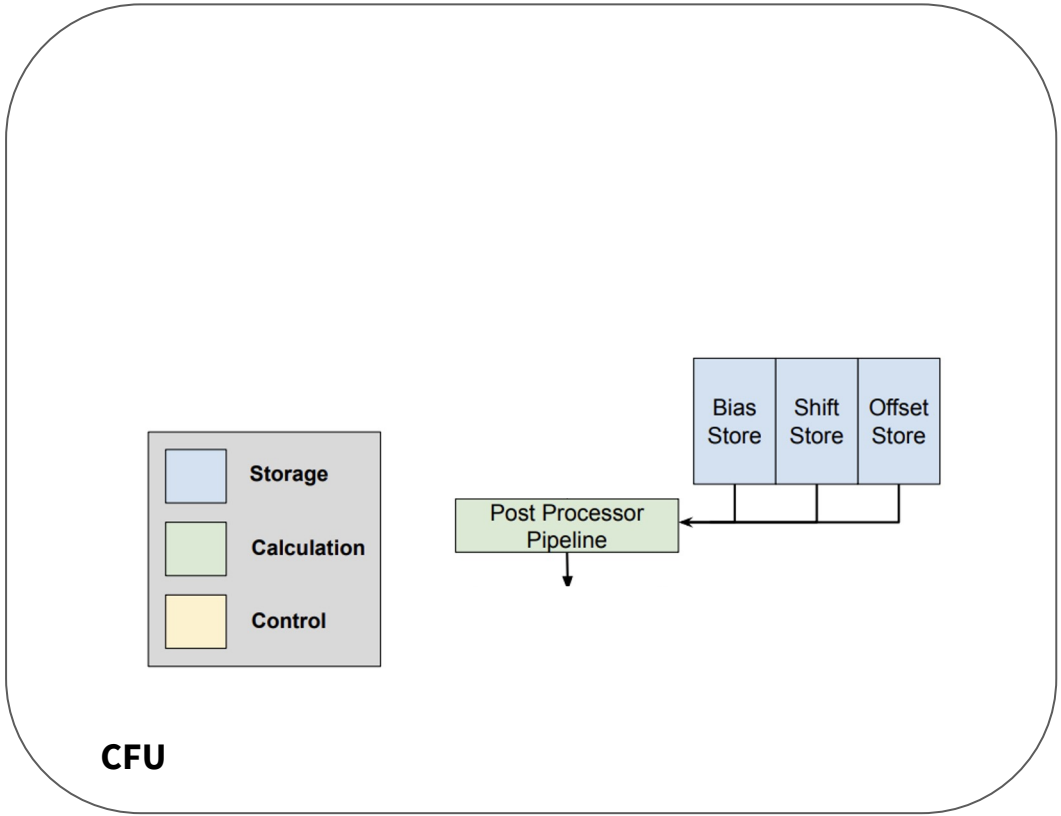


FPGA Acceleration for Image Classification

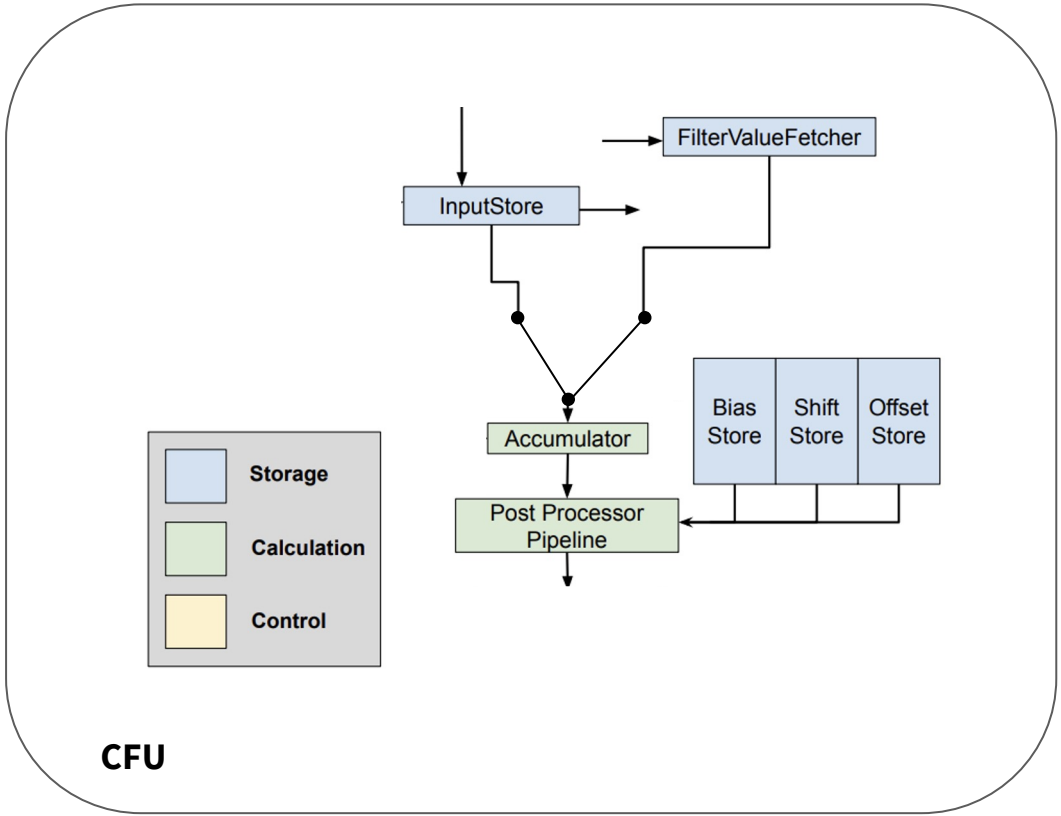
Start with Software Optimizations!



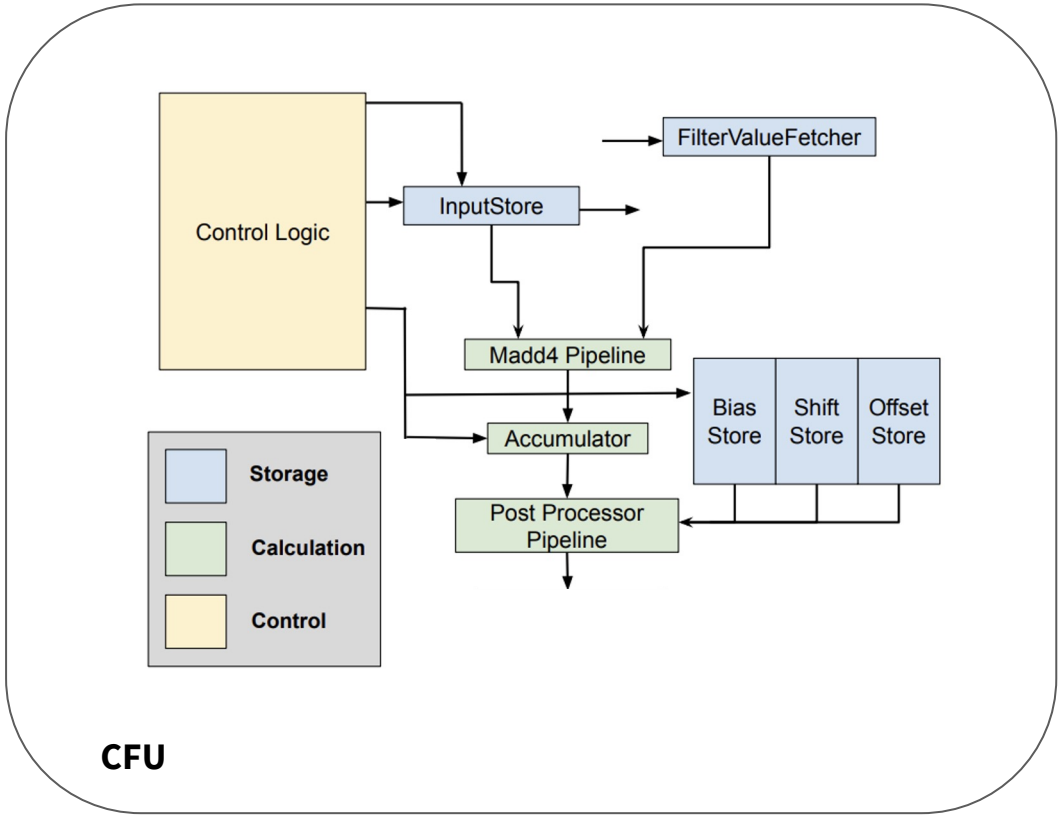
FPGA Acceleration for Image Classification



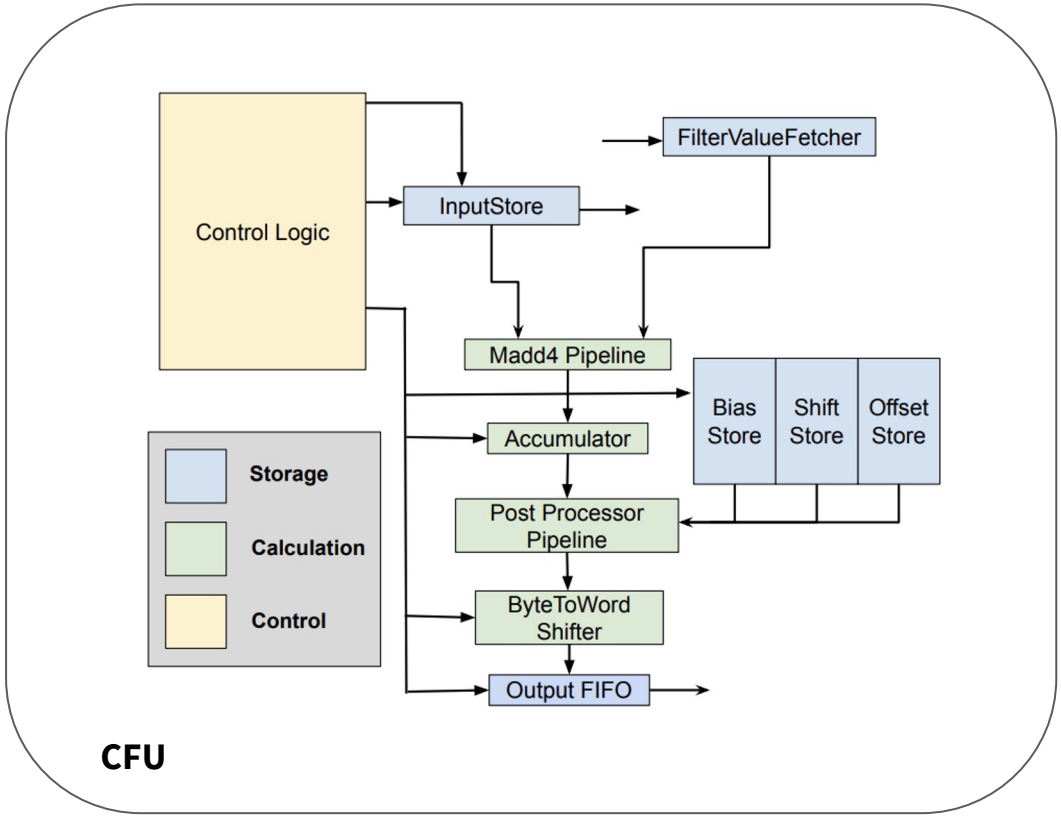
FPGA Acceleration for Image Classification



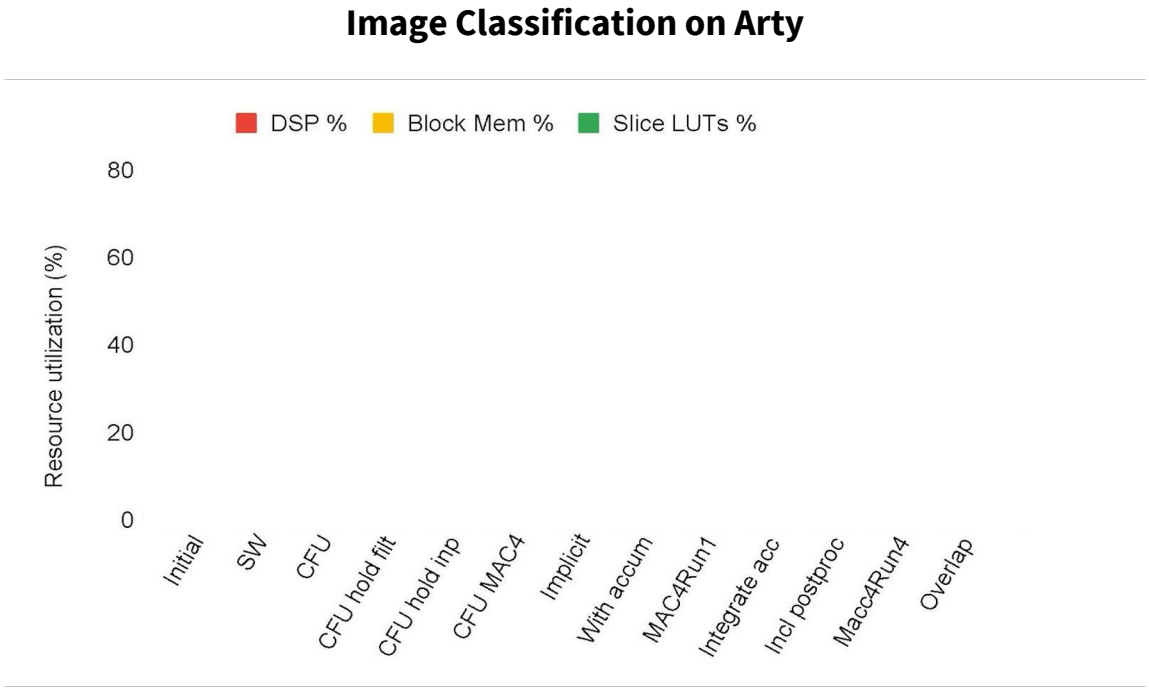
FPGA Acceleration for Image Classification



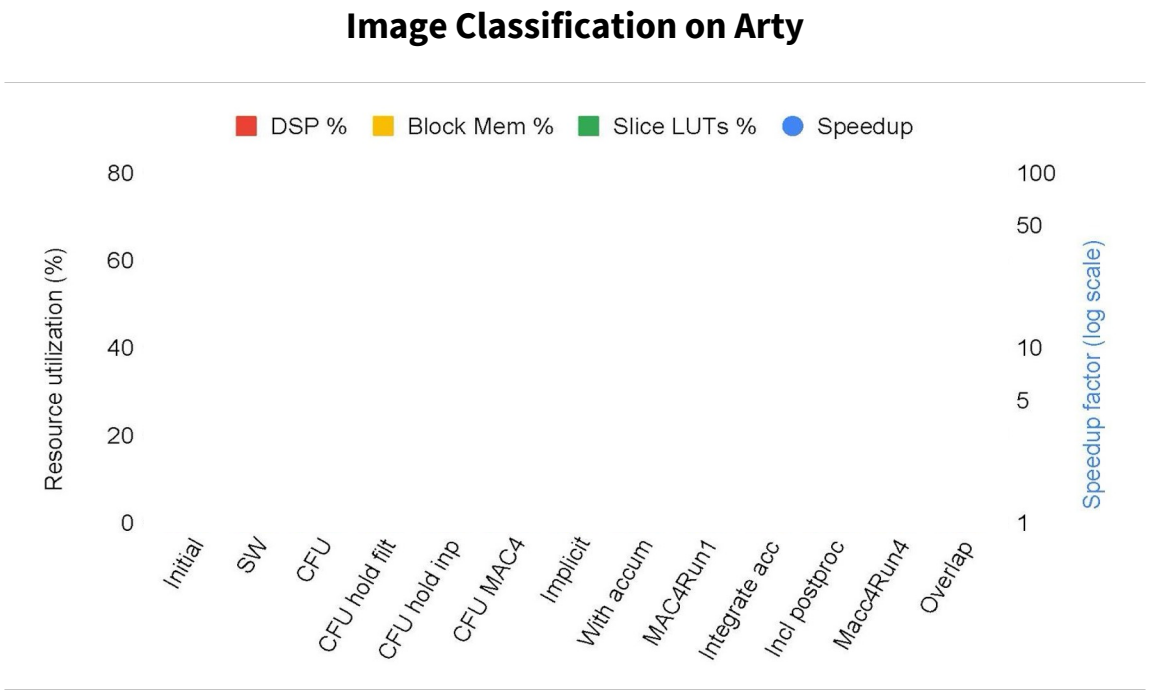
FPGA Acceleration for Image Classification



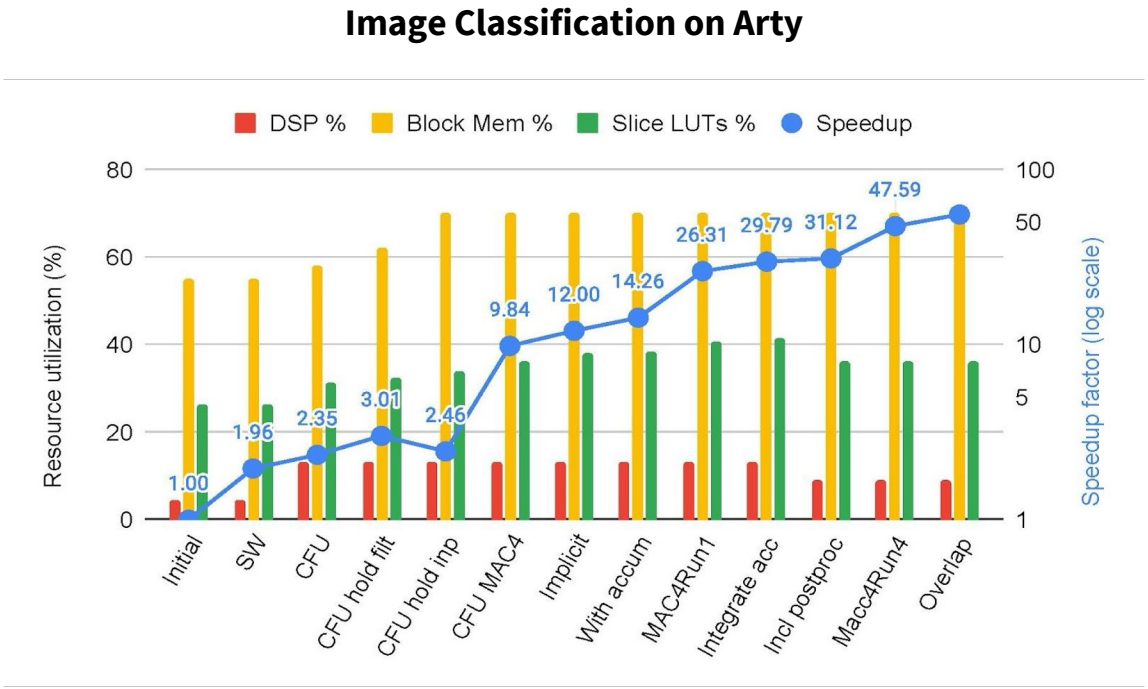
FPGA Acceleration for Image Classification



FPGA Acceleration for Image Classification

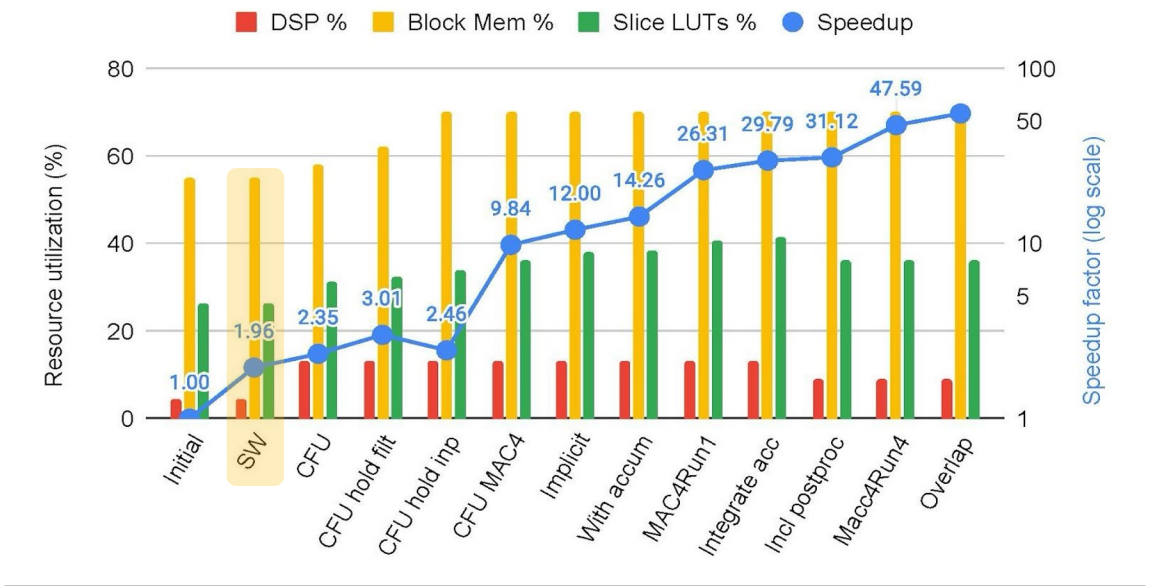


FPGA Acceleration for Image Classification



FPGA Acceleration for Image Classification

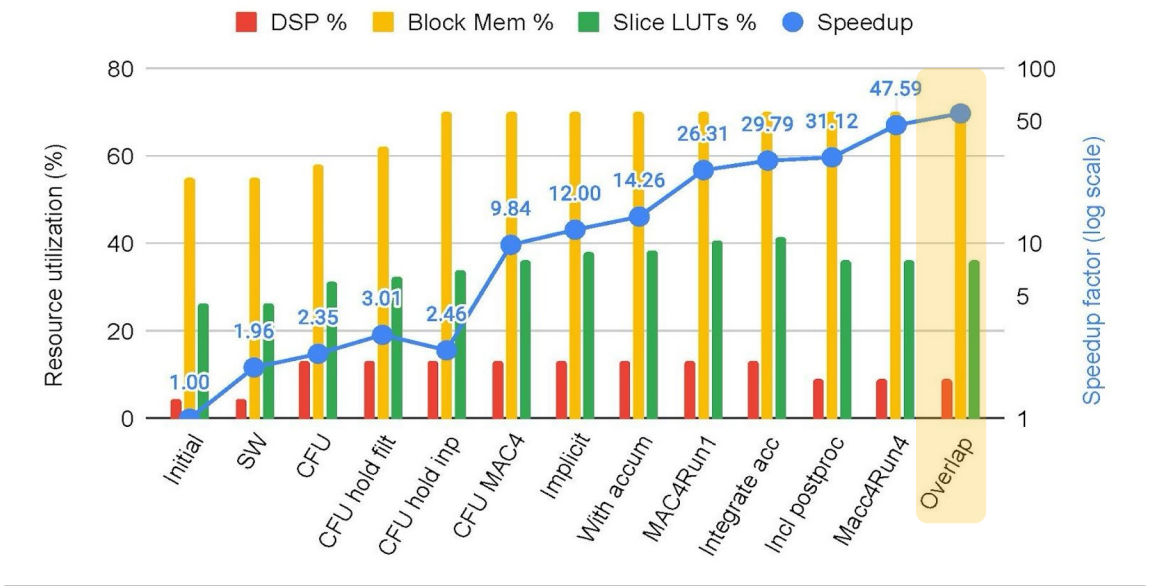
Image Classification on Arty



2x speedup from SW Optimizations

FPGA Acceleration for Image Classification

Image Classification on Arty



Total 55x speedup in 5 weeks

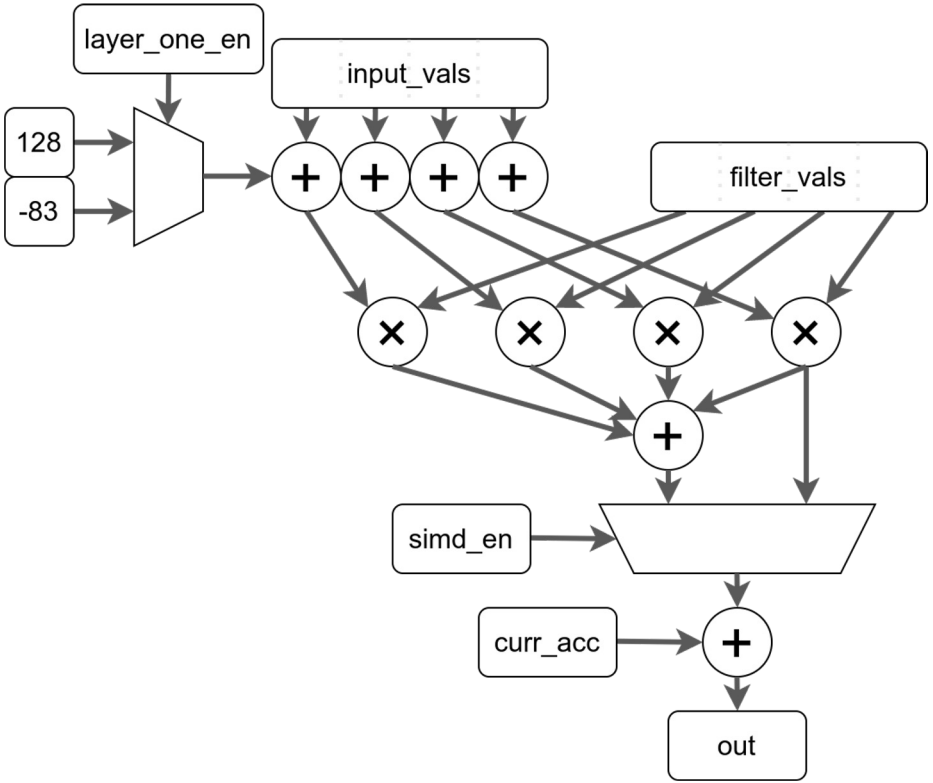
Human Presence Sensor

- In Chromebook:
  - An isolated camera+ML subsystem embedded in the display bezel
- User features:
  - Keep awake while present
  - Dim on leave
  - Wake on approach
  - Eavesdropper warning

FPGA Acceleration for Keyword Spotting

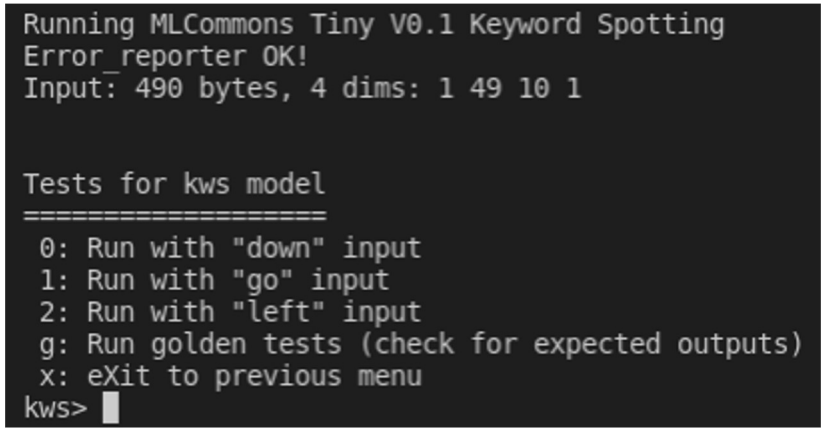
FOMU FPGA

FPGA Acceleration for Keyword Spotting

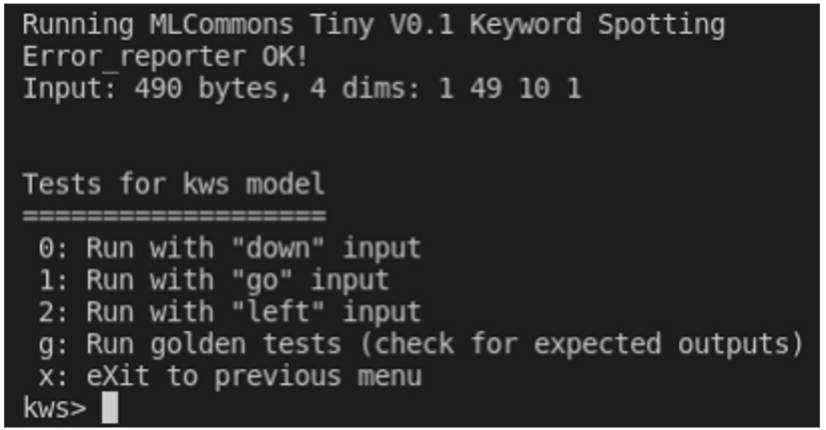


75× speedup on model inference

How it started:

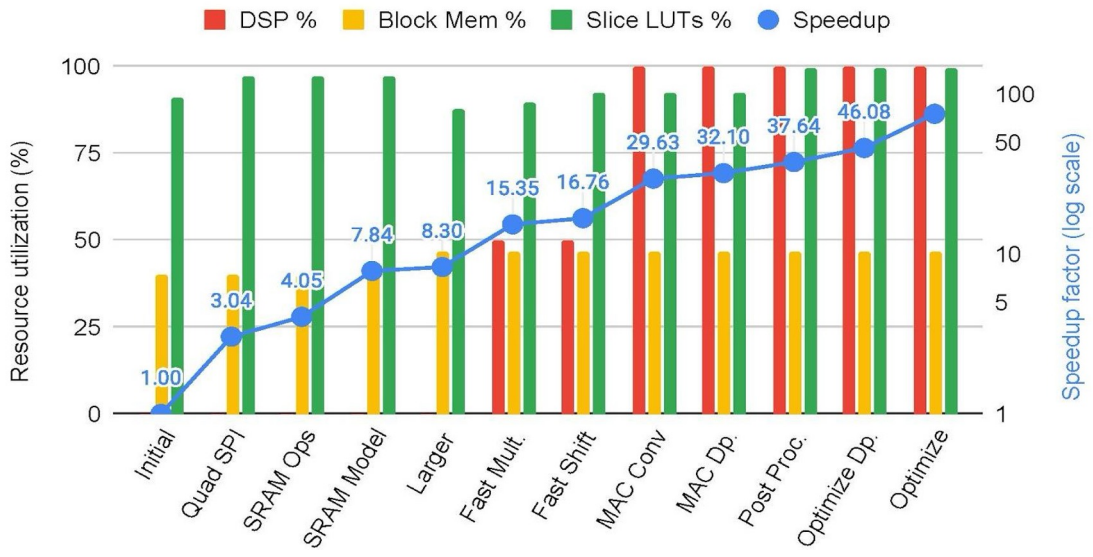


How it's going:



FPGA Acceleration for Keyword Spotting

Keyword Spotting on FOMU



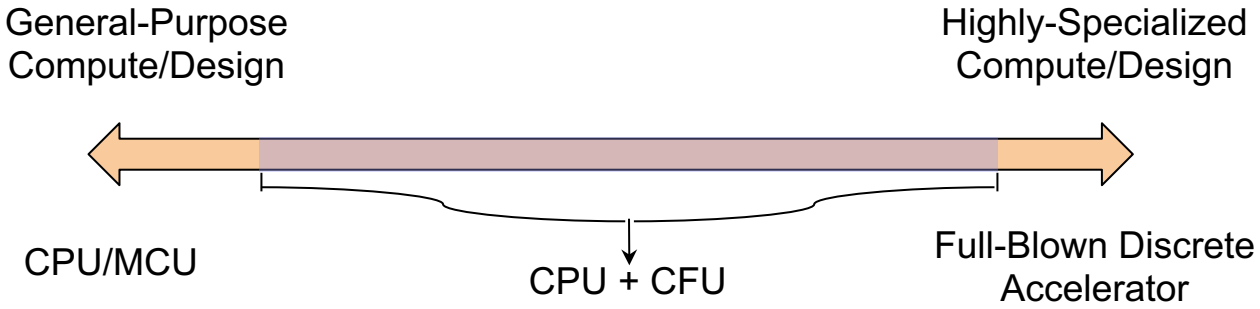
75x speedup in under 4 weeks

Agenda

1. Tiny Machine Learning
2. The Need for Agile and Full-Stack Frameworks
3. Custom Function Units and CFU Playground
4. FPGA Acceleration for Image Classification and Keyword Spotting
5. **Design Space Exploration: CFU vs. CPU**

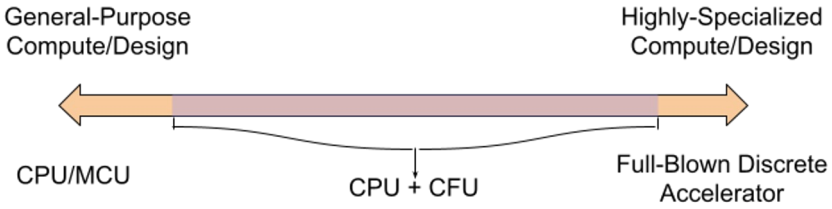
Design Space Exploration

(CFU) Accelerator vs (Soft) CPU

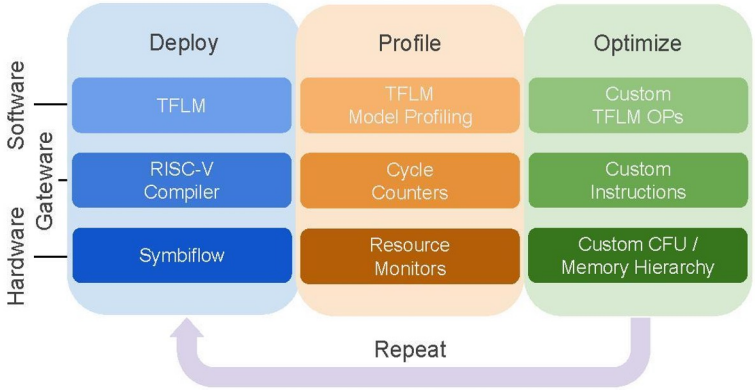


(Manual) Design Space Exploration

(CFU) Accelerator vs (Soft) CPU

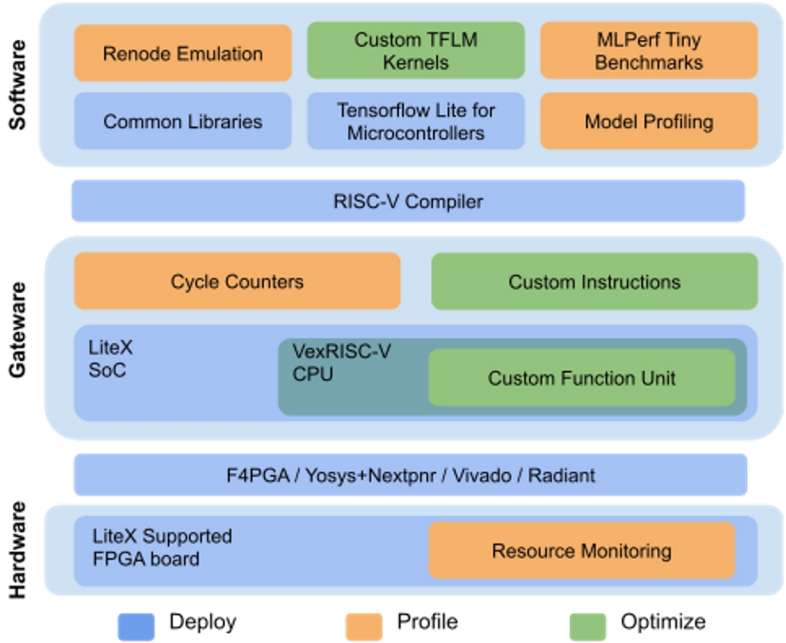


+



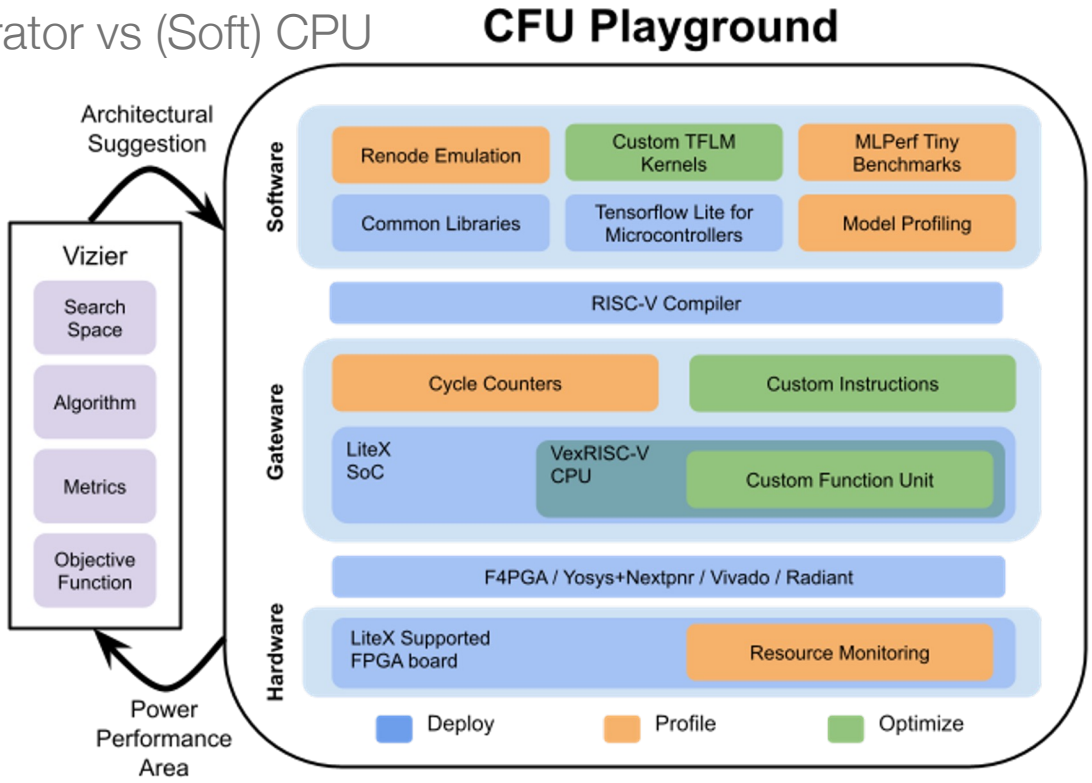
(Automated) Design Space Exploration

(CFU) Accelerator vs (Soft) CPU



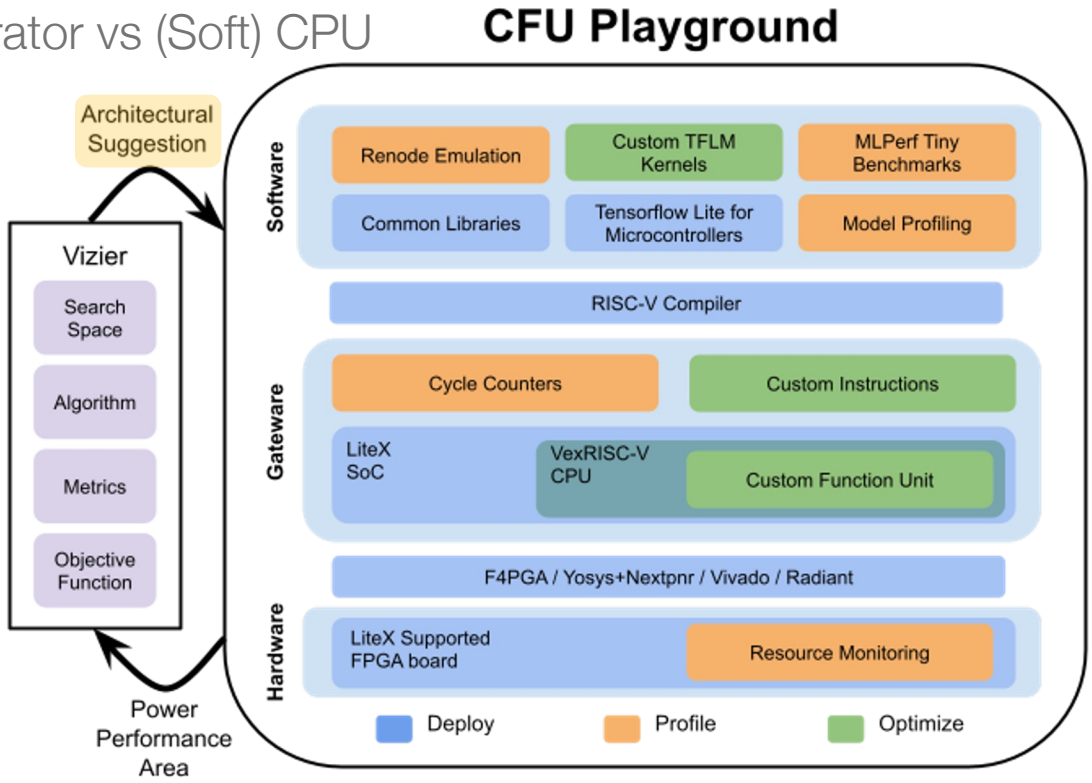
(Automated) Design Space Exploration

(CFU) Accelerator vs (Soft) CPU



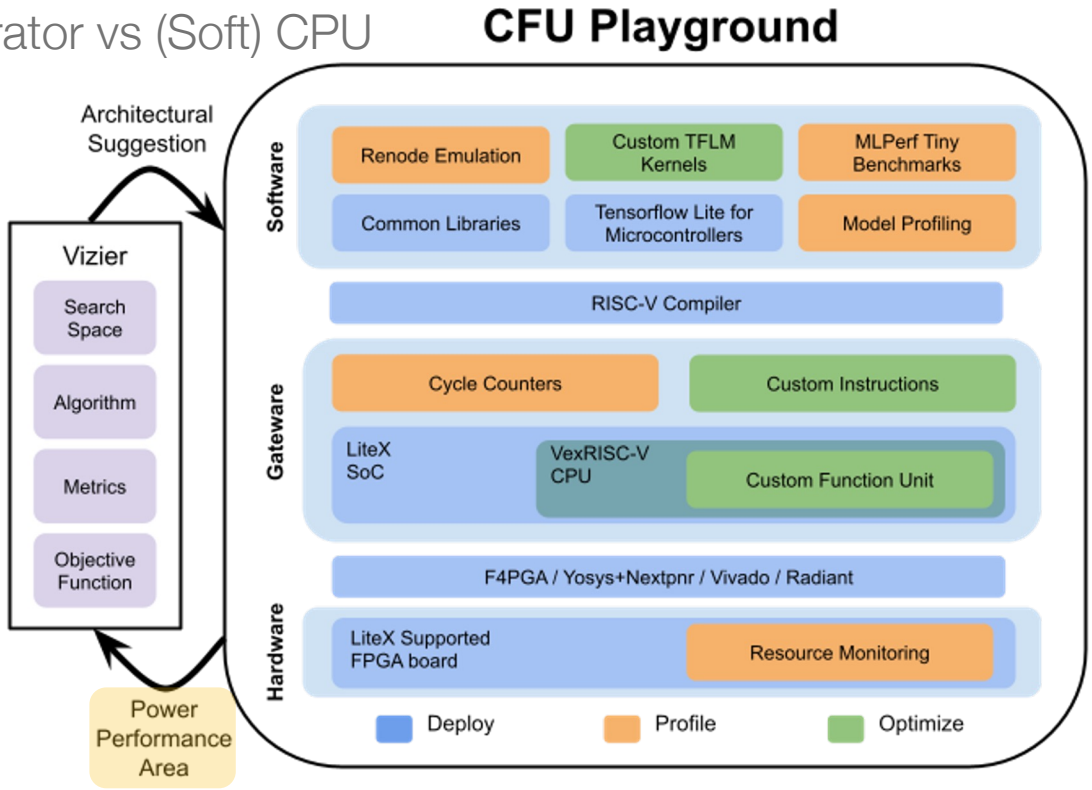
(Automated) Design Space Exploration

(CFU) Accelerator vs (Soft) CPU

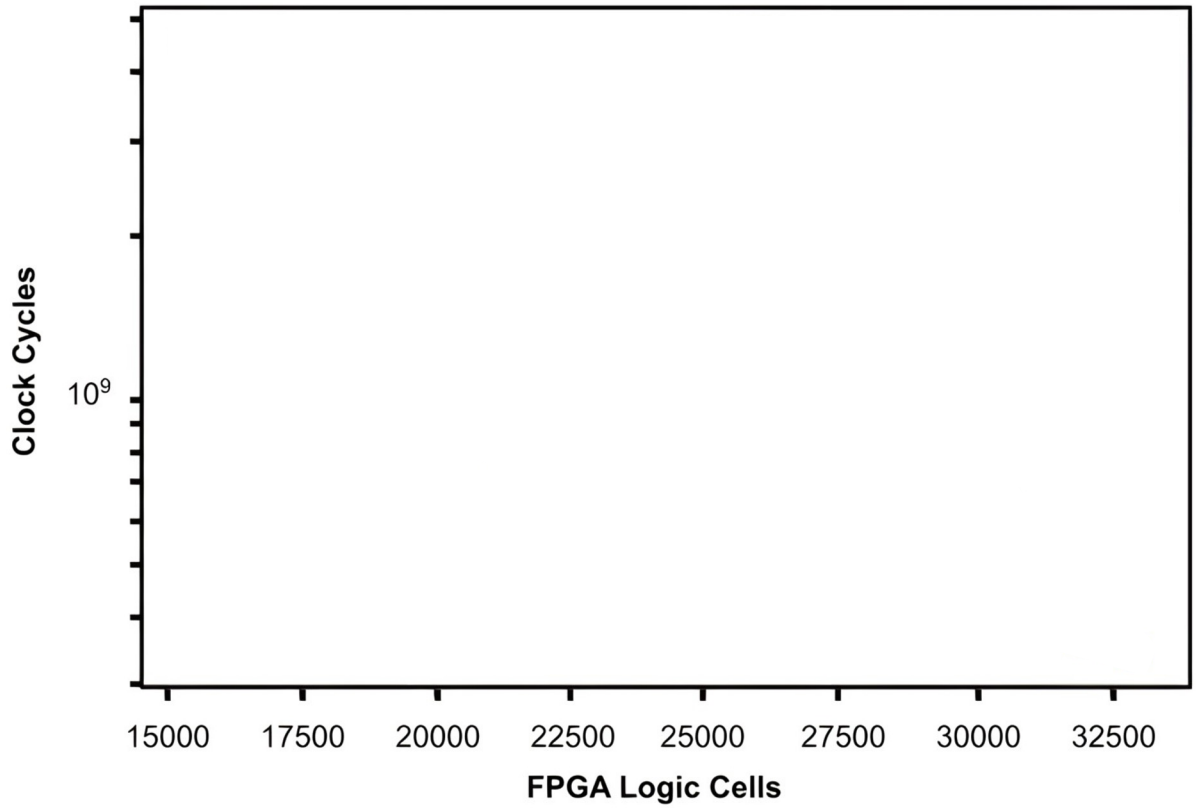


(Automated) Design Space Exploration

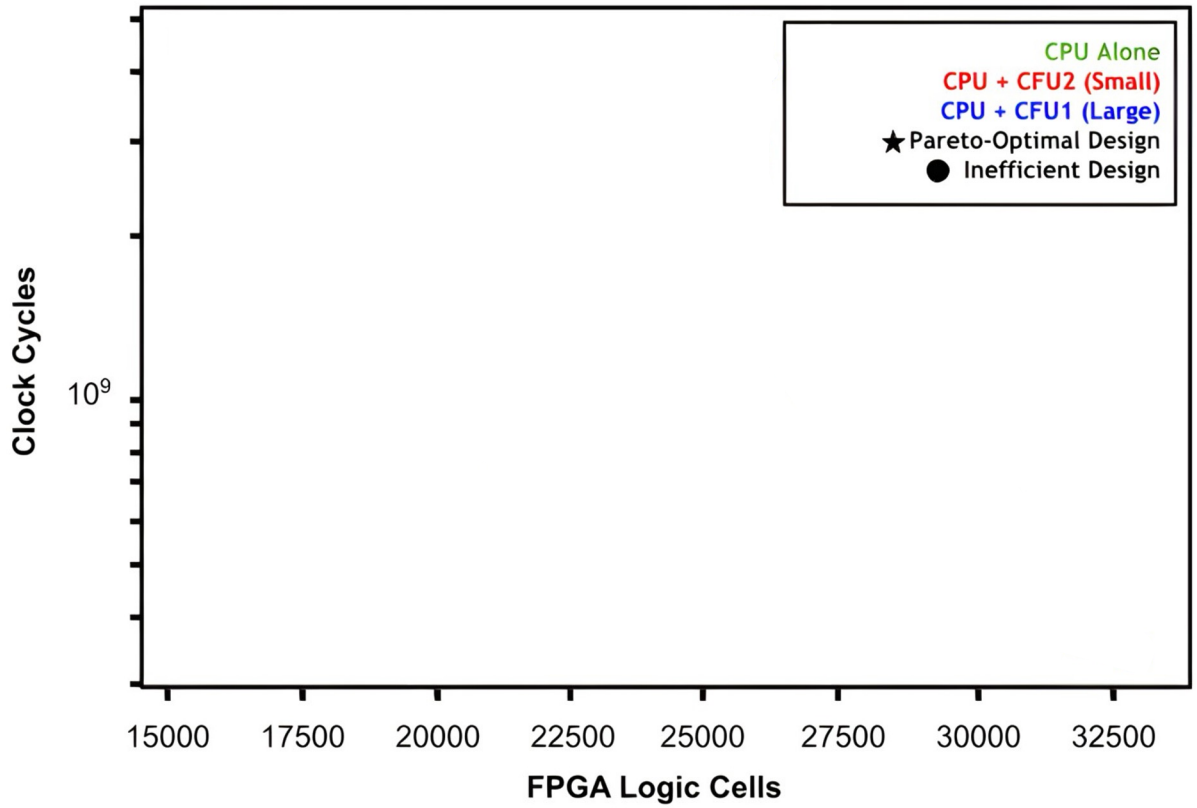
(CFU) Accelerator vs (Soft) CPU



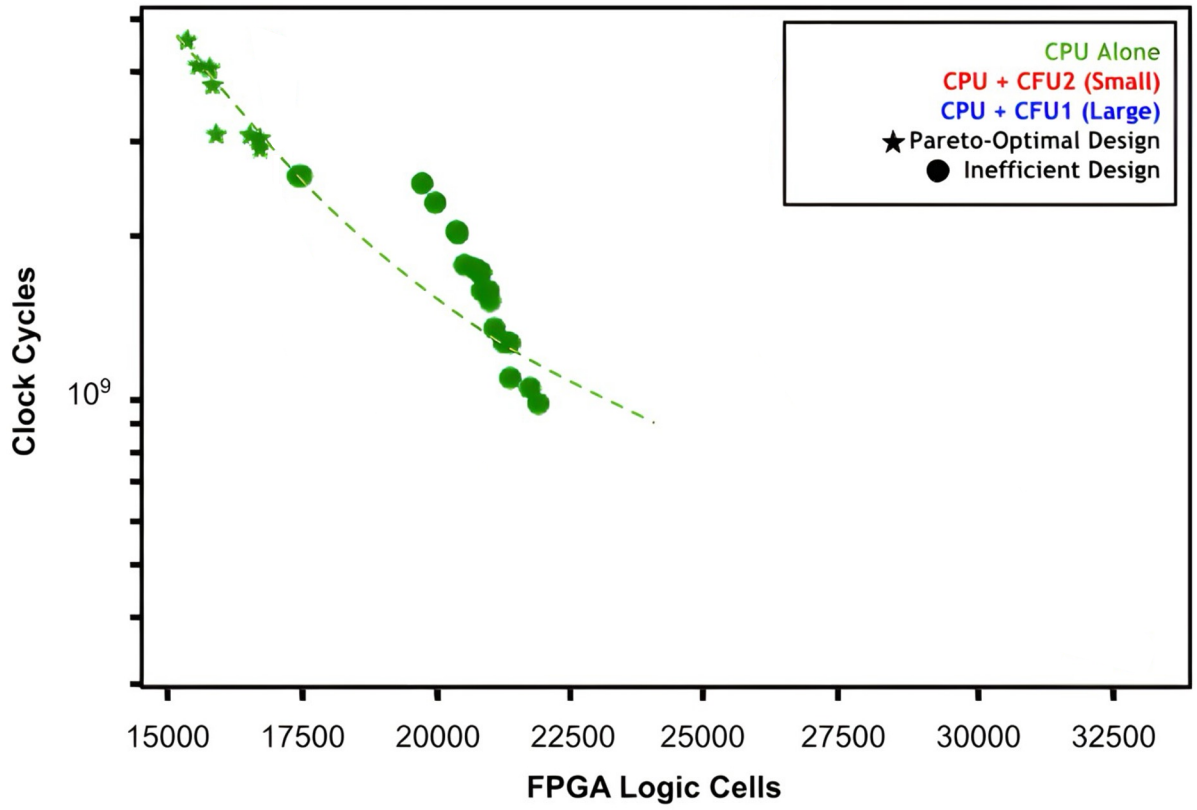
Design Space Exploration: CFU vs CPU



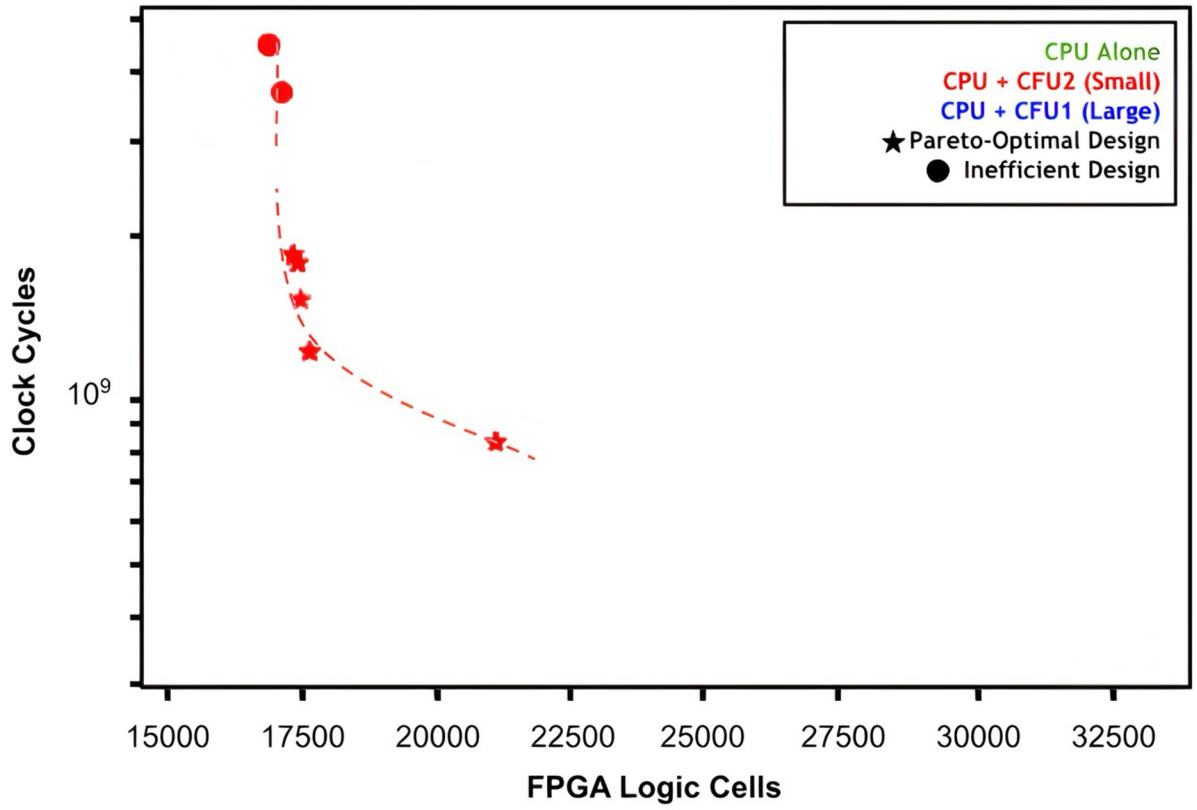
Design Space Exploration: CFU vs CPU



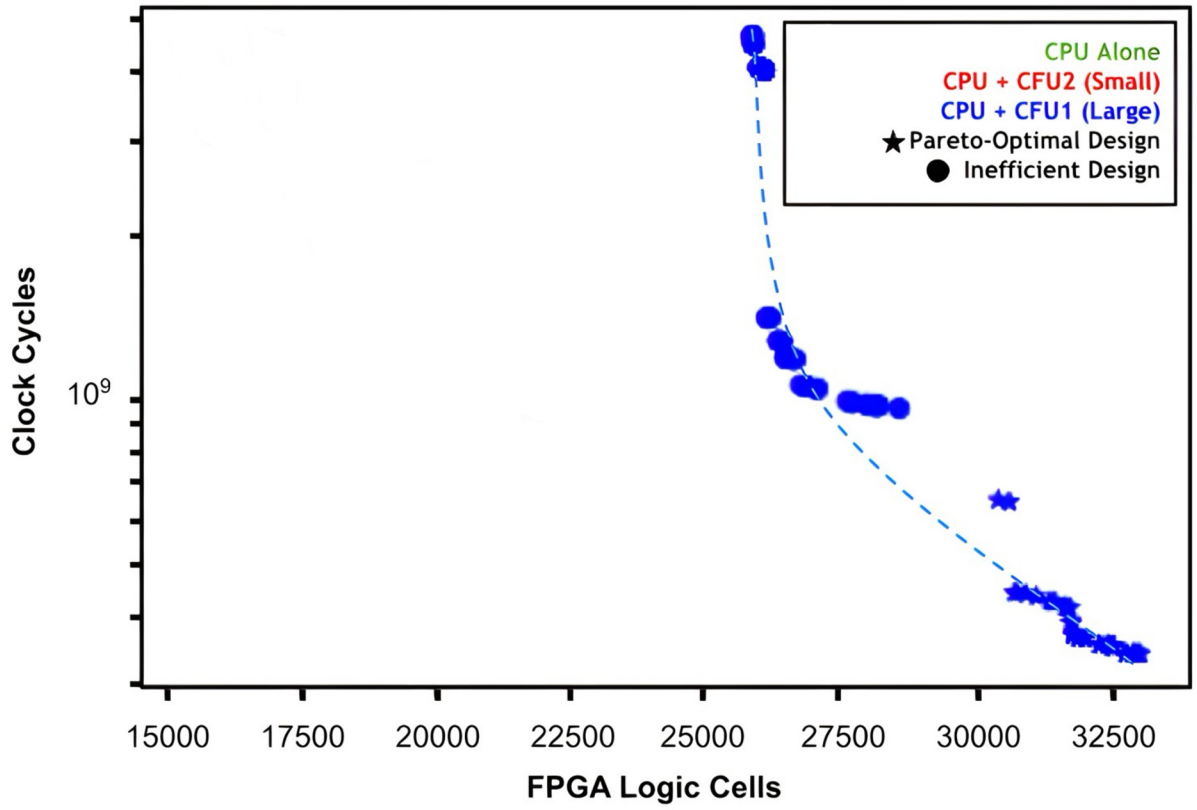
Design Space Exploration: CFU vs CPU



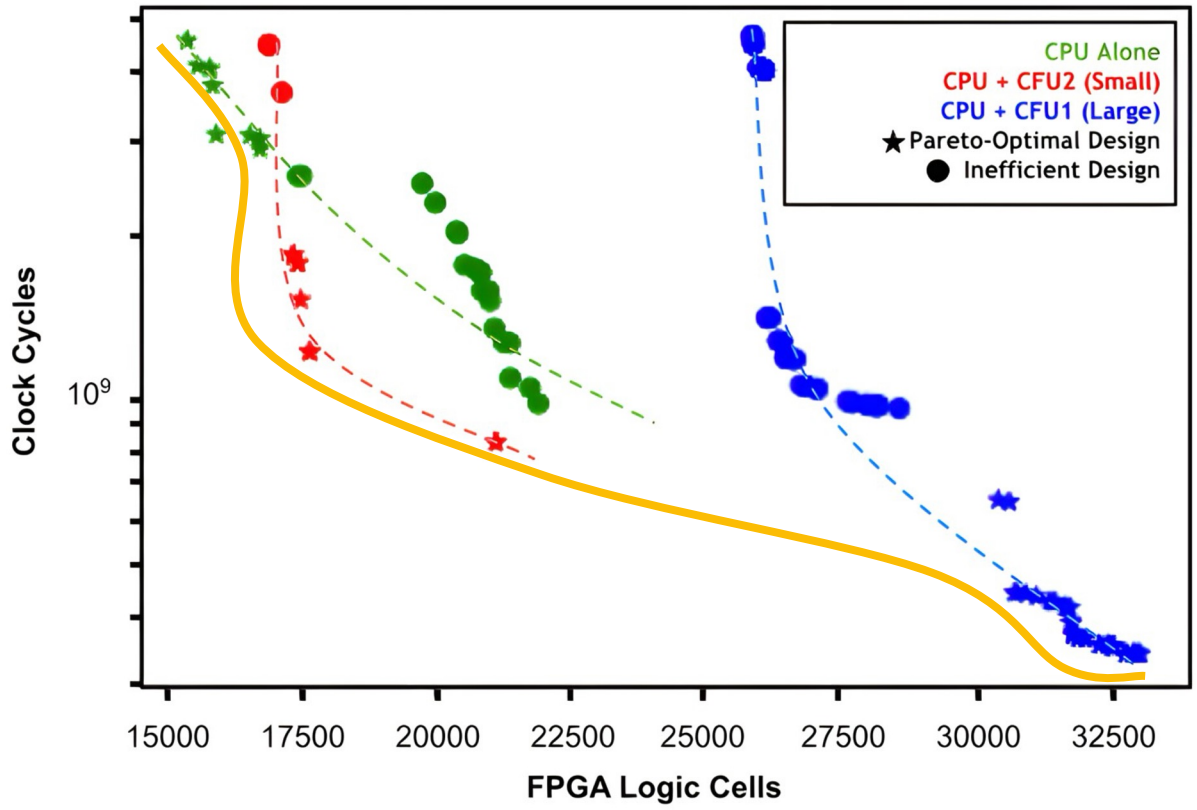
Design Space Exploration: CFU vs CPU



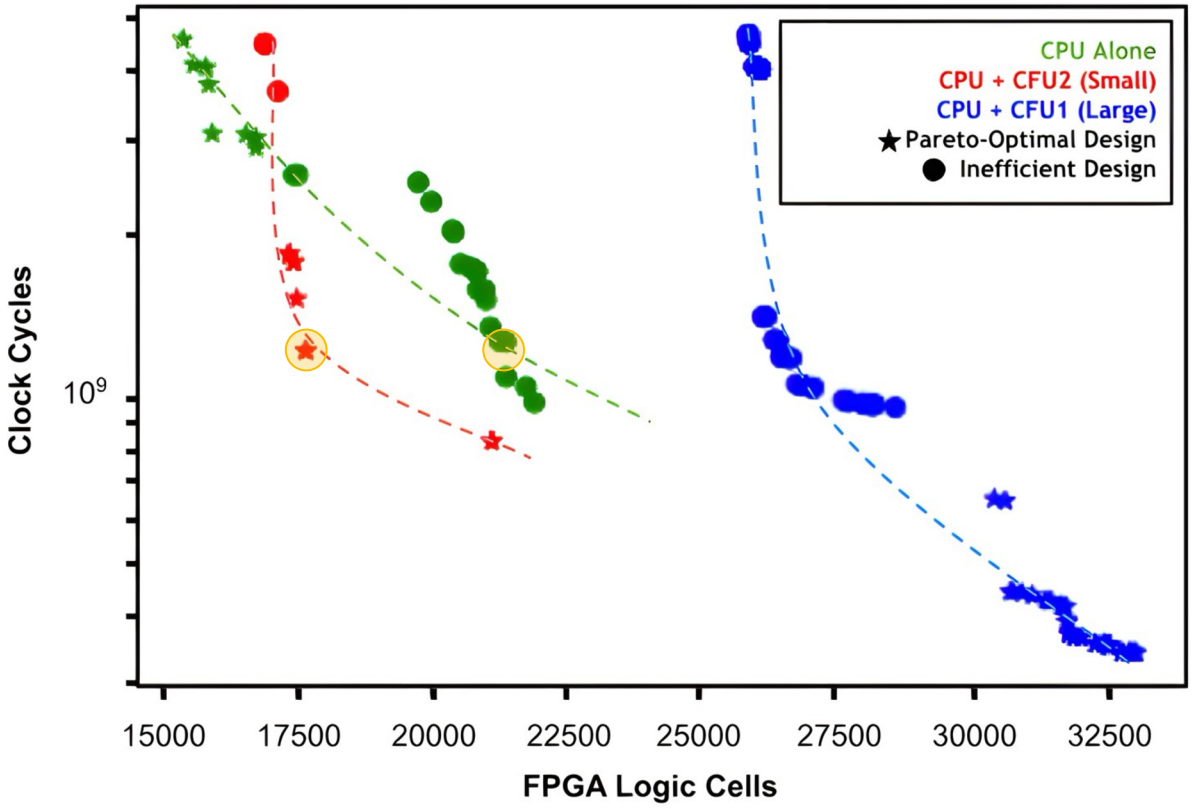
Design Space Exploration: CFU vs CPU



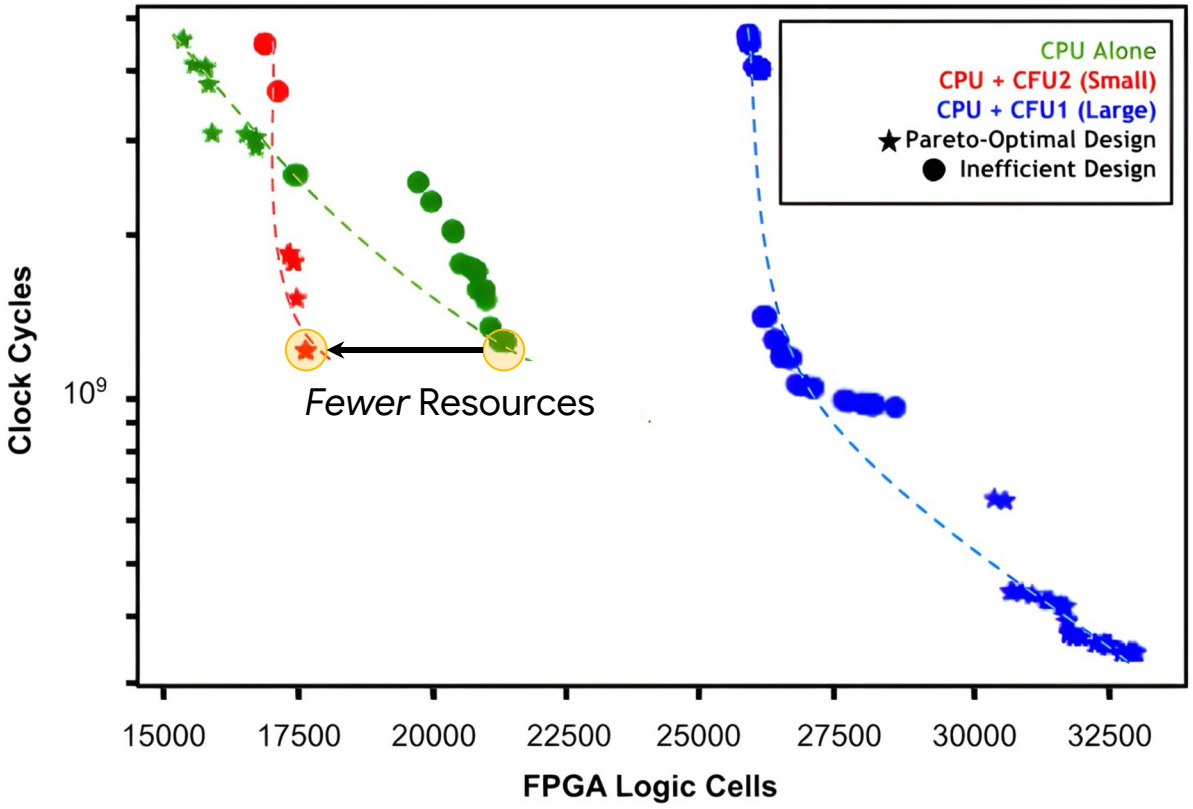
Design Space Exploration: CFU vs CPU



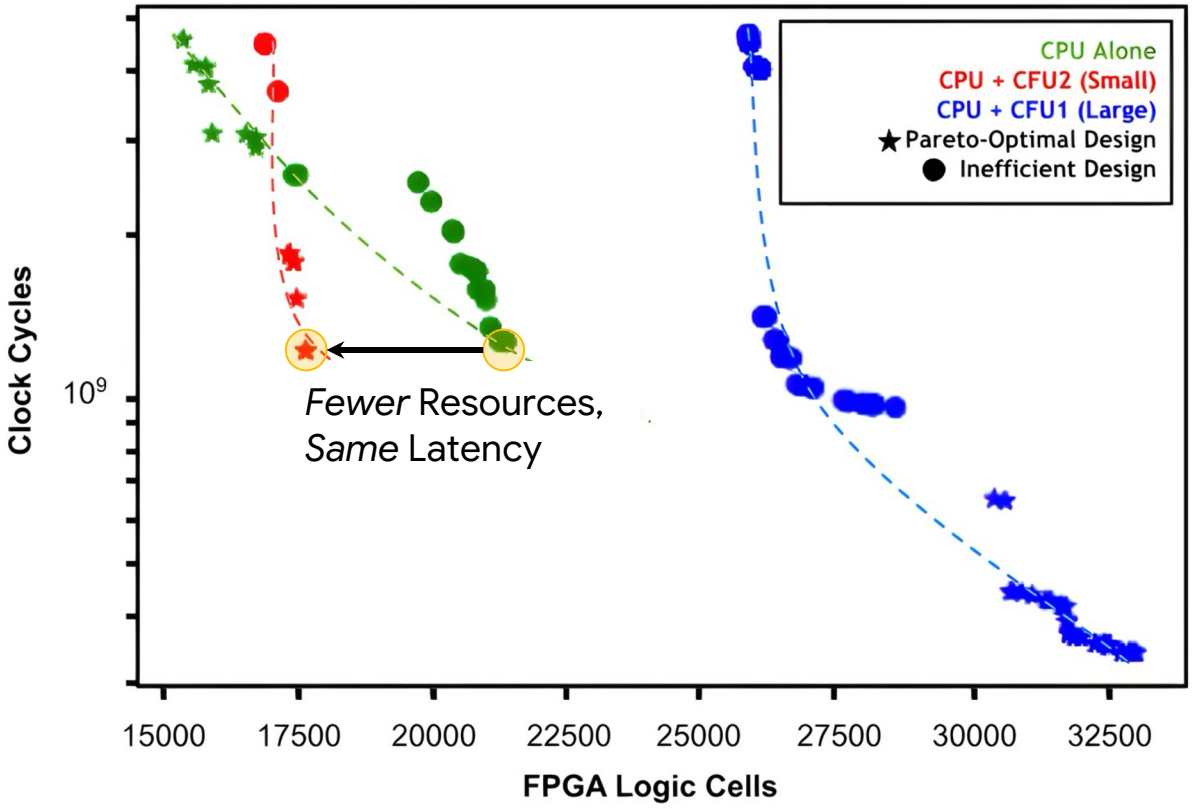
Design Space Exploration: CFU vs CPU



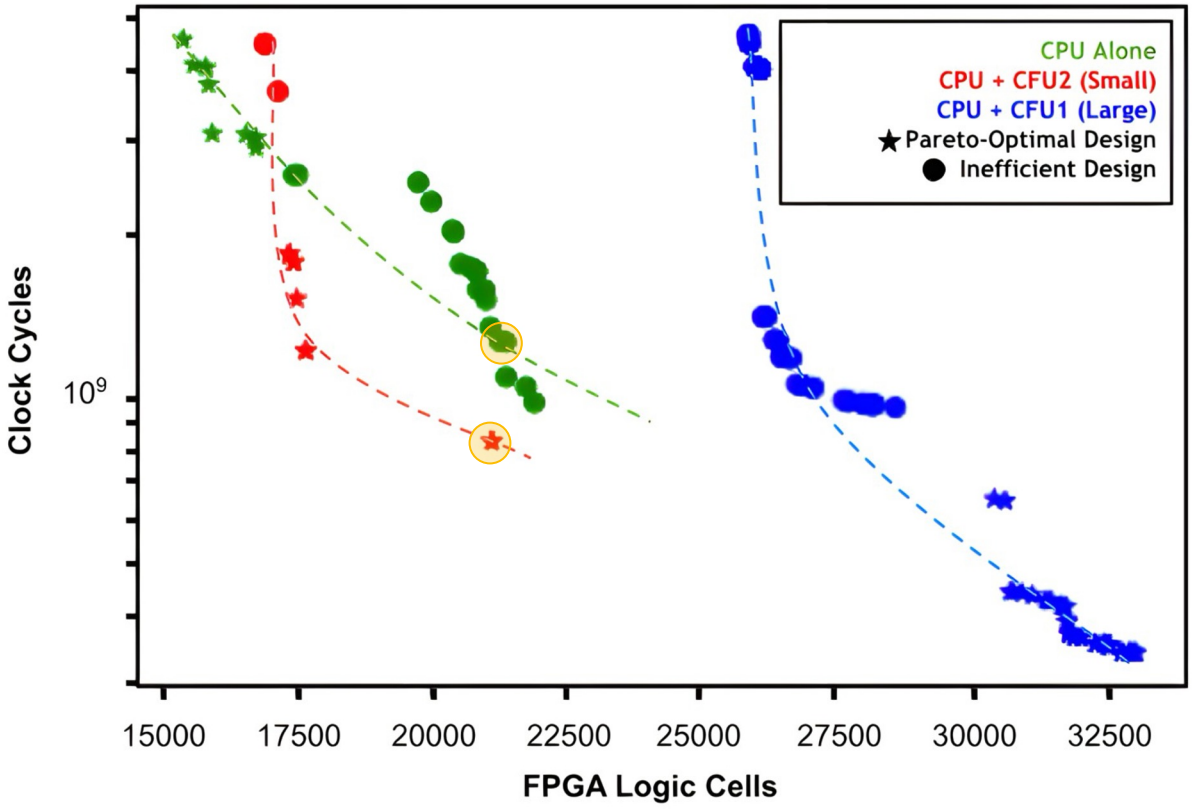
Design Space Exploration: CFU vs CPU



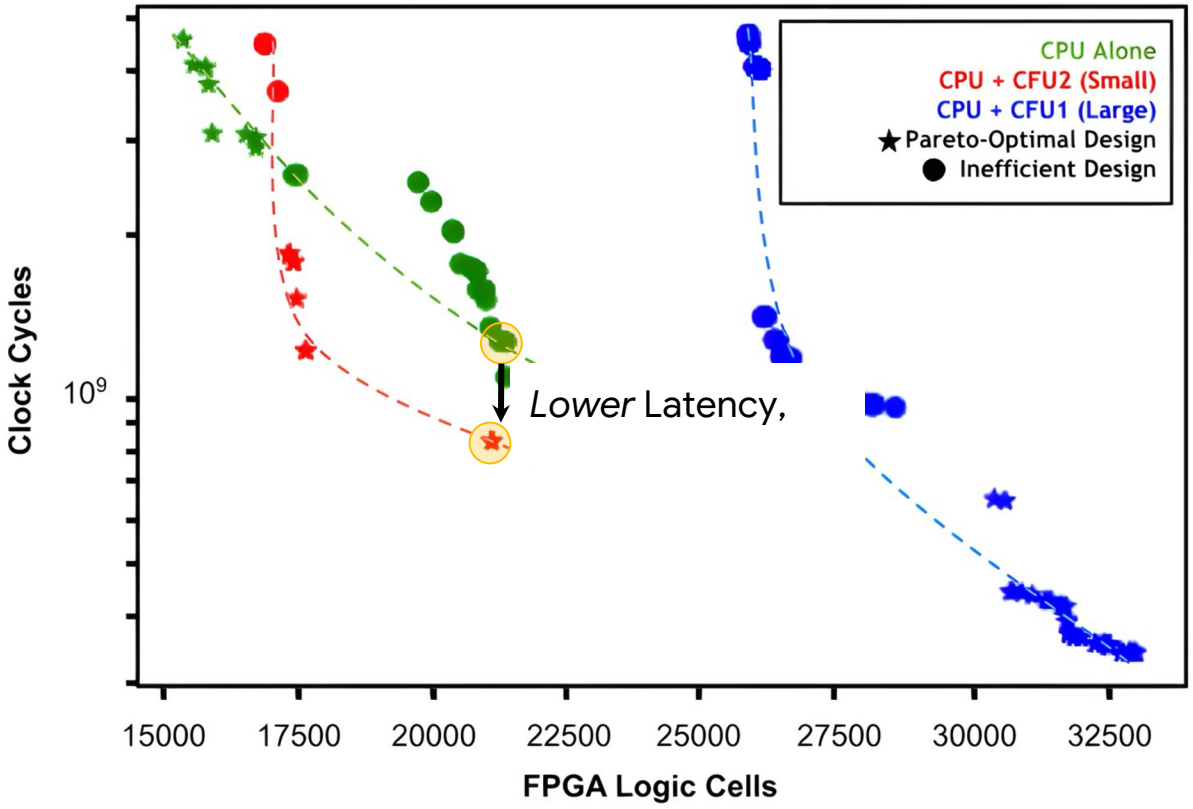
Design Space Exploration: CFU vs CPU



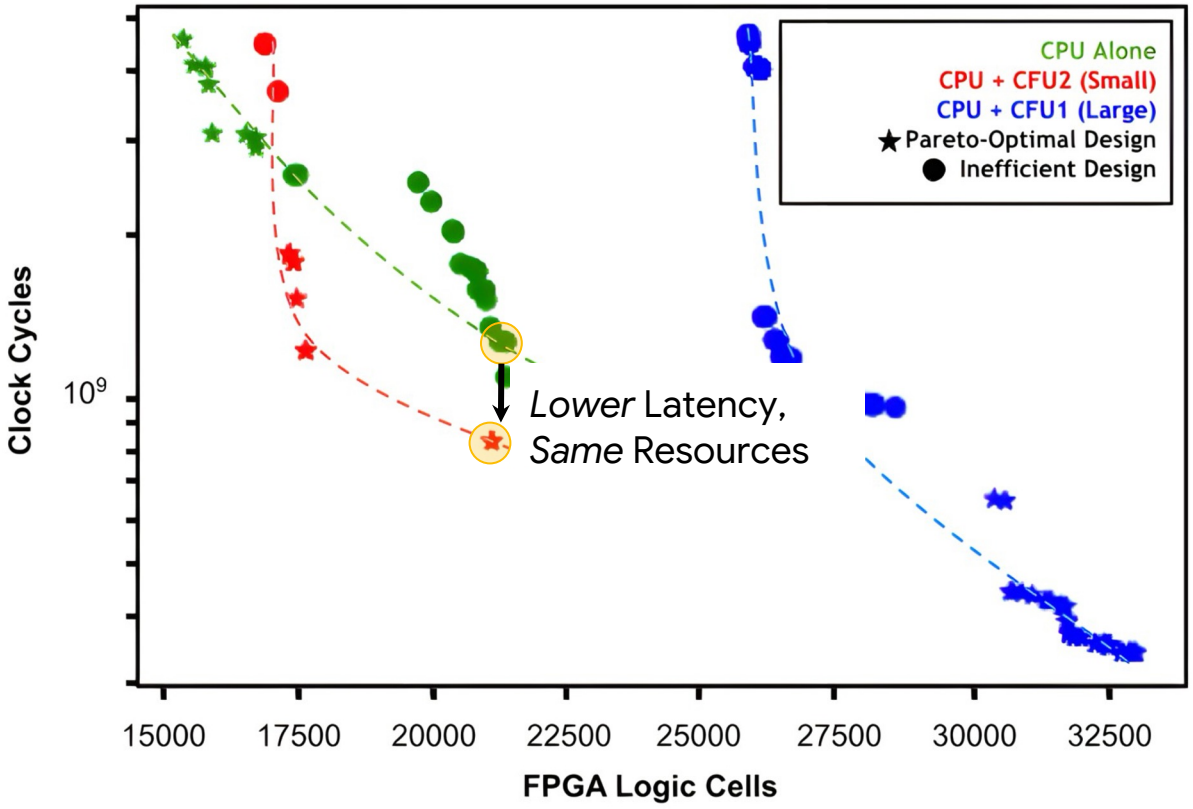
Design Space Exploration: CFU vs CPU



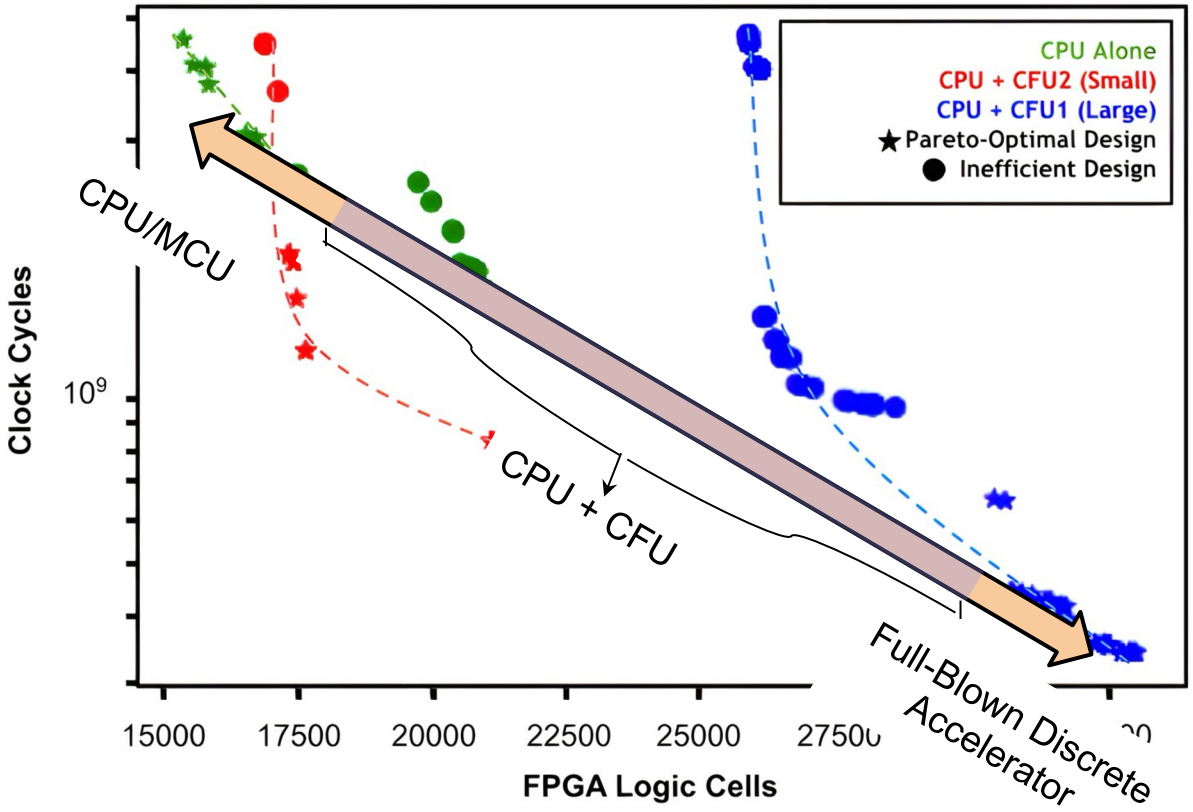
Design Space Exploration: CFU vs CPU



Design Space Exploration: CFU vs CPU



Design Space Exploration: CFU vs CPU



Key Takeaways

1. Full-stack framework that integrates open-source tools to facilitate community-driven research.

1. Agile methodology to iteratively design and evaluate tightly-coupled, bespoke TinyML accelerators.

1. Unique model-specific resource allocation trade-offs between CFU, CPU, and memory.

1. Automated design space exploration of the CPU paired with a CFU using Vizier.

2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)

CFU Playground: Full-Stack Open-Source Framework for Tiny Machine Learning (TinyML) Acceleration on FPGAs

Shvetank Prakash* Tim Callahan† Joseph Bushagour§ Colby Banbury*
Alan V. Green† Pete Warden† Tim Ansell† Vijay Janapa Reddi*
*Harvard University †Google §Purdue University †Stanford University

***Abstract*—Need for the efficient processing of neural networks has given rise to the development of hardware accelerators. The increased adoption of specialized hardware has highlighted the need for more agile design flows for hardware-software co-design and domain-specific optimizations. In this paper, we present CFU Playground— a full-stack open-source framework that enables rapid and iterative design and evaluation of machine learning (ML) accelerators for embedded ML systems. Our tool provides a completely open-source end-to-end flow for hardware-software co-design on FPGAs and future systems research. This full-stack framework gives the users access to explore experimental and bespoke architectures that are customized and co-optimized for embedded ML. Our rapid, deploy-profile-optimization feedback loop lets ML hardware and software developers achieve significant returns out of a relatively small investment in customization. Using CFU Playground's design and evaluation loop, we show substantial speedups between 55× and 75×. The soft CPU coupled with the accelerator opens up a new, rich design space between the two components that we explore in an automated fashion using Vizier, an open-source black-box optimization service.**

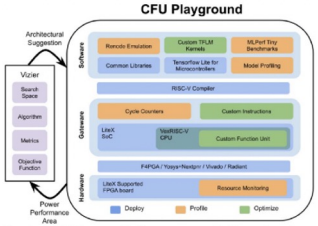


Fig. 1: CFU Playground allows users to design and evaluate model-specific ML enhancements to a "soft" CPU core. The Playground is wrapped around Vizier, an open-source black-box optimization service, to enable ML-driven design space exploration.

I. Introduction

Tiny machine learning (TinyML) is a fast-growing field at the intersection of ML algorithms and low-cost embedded systems. It enables on-device sensor data analytics (vision, audio, IMU, etc.) at ultra-low-power consumption. Processing data close to the sensor allows for an expansive new variety of always-on ML use-cases that preserve bandwidth, latency, and energy while improving responsiveness and maintaining privacy [1]. Given the need for energy efficiency when running ML on these embedded platforms, custom processor support and hardware accelerators for such systems could present the needed solutions. However, the field of ML is still in its infancy and fast-changing. Thus, it is desirable to avoid a massive non-recurring engineering (NRE) cost upfront, especially for low-cost embedded ML systems. Building ASICs is both costly and time-consuming. Moreover, since embedded systems are often task-specific, there is an opportunity to avoid general-purpose ML accelerators and instead explore task and model-specific ML acceleration methods. This setting presents the need for an agile design space exploration tool that allows us to adapt to the changing landscape of ML and hardware.

To enable holistic hardware-software co-design and evaluation of domain-specific performance optimizations easily, we present CFU Playground.[1] It is a full-stack open-source framework for iteratively (deploy→profile→optimize) exploring the design space of lightweight accelerators in an agile manner (Figure 1). The framework is unique in that it couples together various open-source software (TensorFlow Lite Micro/TFLM, GCC), open-source RTL generation IP and toolkits (LiteX, VexRiscv, Migen, Amaranth), and open-source FPGA tools for synthesis, place, and route (yosys, nextpnr, F4PGA/SymbiFlow, etc.). By using open source for the entire stack, we enable the end-user to *customize and co-optimize hardware and software*, resulting in a specialized solution unencumbered by potential licensing restrictions and not tied to a particular FPGA, board, or vendor. CFU Playground yields large returns out of a relatively small investment in customized hardware and is useful for the long tail of low-volume applications.

Yet another novelty of CFU Playground is in its *ability to design custom function units (CFUs)* for distinct ML operations. CFUs represent a novel design space that balances ac-

[1]CFU Playground is available at www.github.com/google/CFU-Playground.

979-8-3503-9739-0/23/$31.00 ©2023 IEEE
DOI 10.1109/ISPASS57527.2023.00024

Thank You!

Build Your Own ML Processor Today: https://github.com/google/CFU-Playground

Contact: Shvetank Prakash
Email: sprakash@g.harvard.edu