Tiny Machine Learning

Vijay Janapa Reddi, Ph. D. | Associate Professor |
John A. Paulson School of Engineering and Applied Sciences | Harvard University |
Web: http://scholar.harvard.edu/vijay-janapa-reddi

Chips & Compilers Symposium at MLSys ‘22, Sep. 1, 2022

TinyML

What is Tiny Machine Learning (**TinyML**)?

TinyML

What is Tiny Machine Learning (**TinyML**)?

TinyML → Fast-growing field of **ML**

What is Tiny Machine Learning (**TinyML**)?

TinyML

Fast-growing field of **ML**

Algorithms, hardware, software

What is Tiny Machine Learning (**TinyML**)?

TinyML

Fast-growing field of **ML**

Algorithms, hardware, software

On-device sensor analytics

What is Tiny Machine Learning (**TinyML**)?

TinyML

Fast-growing field of **ML**

Algorithms, hardware, software

On-device sensor analytics

Low power consumption

What is Tiny Machine Learning (**TinyML**)?

TinyML

Fast-growing field of **ML**

Algorithms, hardware, software

On-device sensor analytics

Low power consumption

Battery-operated

What is Tiny Machine Learning (**TinyML**)?

TinyML

Fast-growing field of **ML**

Algorithms, hardware, software

On-device sensor analytics

Low power consumption

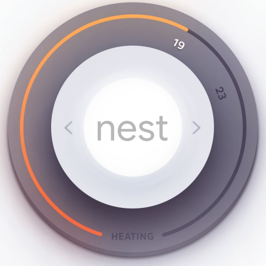
Battery-operated

Always-on ML

Mobile

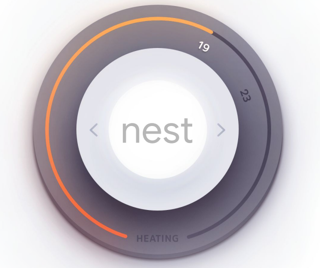
Google Assistant

Google Assistant



IoT 1.0: Internet of Things

Google Assistant



IoT 2.0: Intelligence on Things

Google Assistant

IoT 2.0: Intelligence on Things

Bandwidth
Reliability
Latency
Privacy
Energy

Emerging TinyML Use Cases

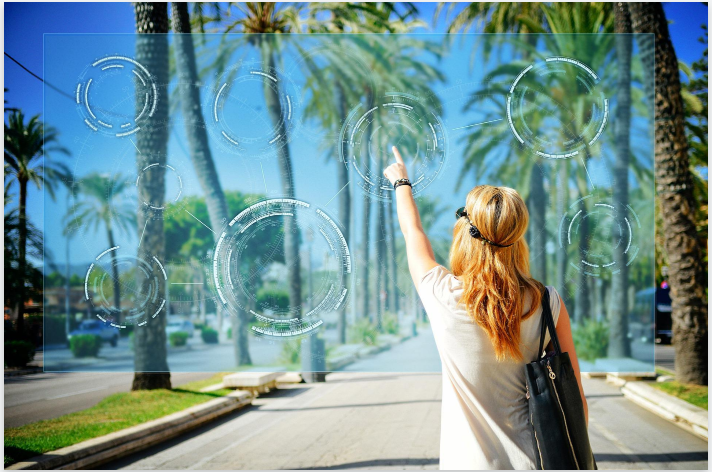
Example: Smart shoes

- Kicking
- Penalty kicking
- Passing
- Dribbling
- ...

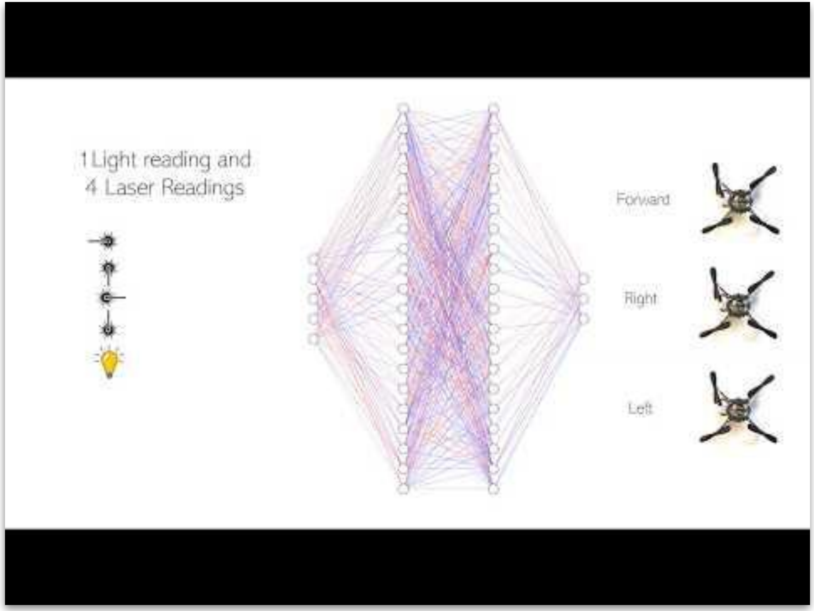
Emerging TinyML Use Cases

Example: Augmented Reality

- Eye tracking
- Hand tracking
- Computer vision
- Superresolution
- ...

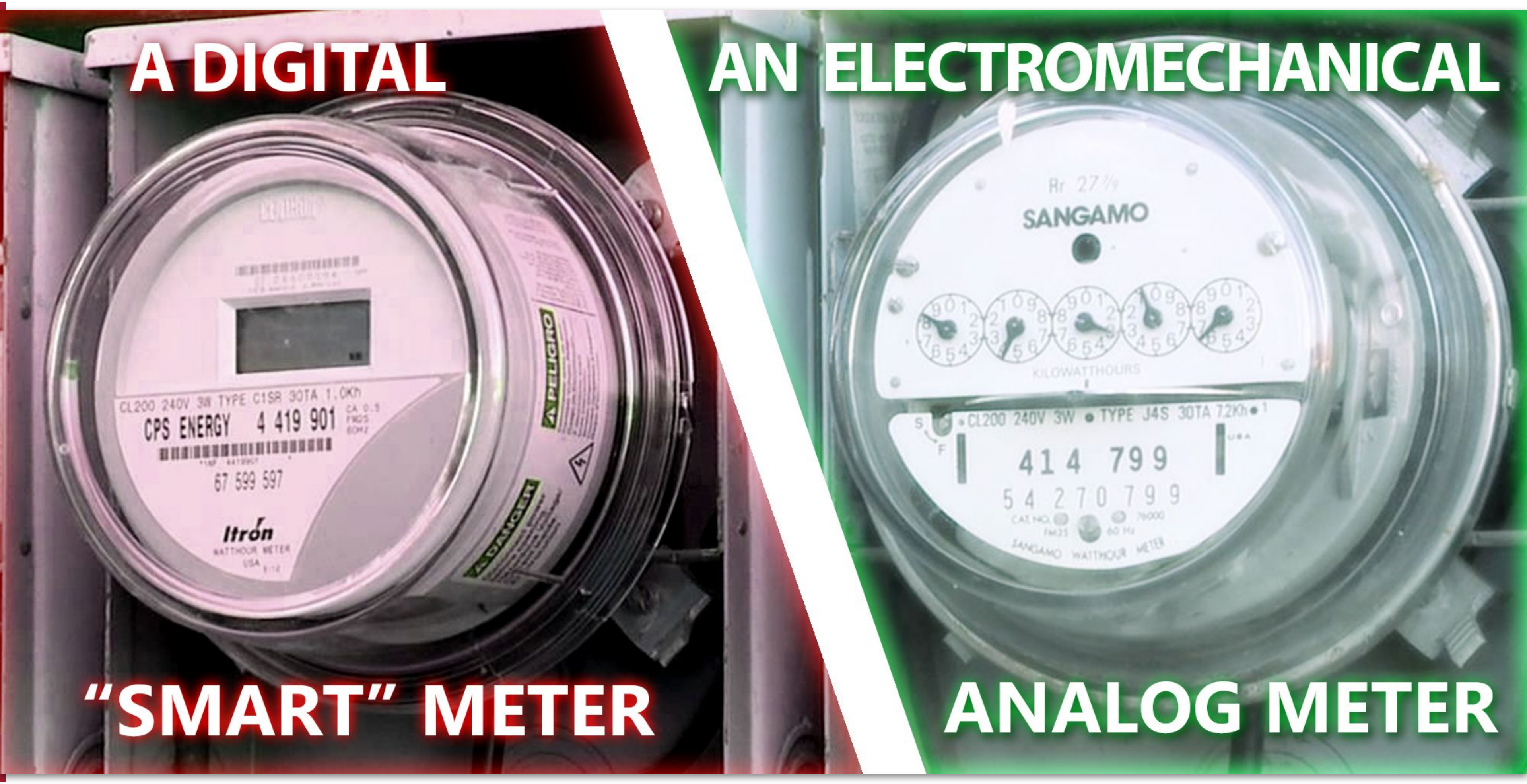


Tiny Robot Learning



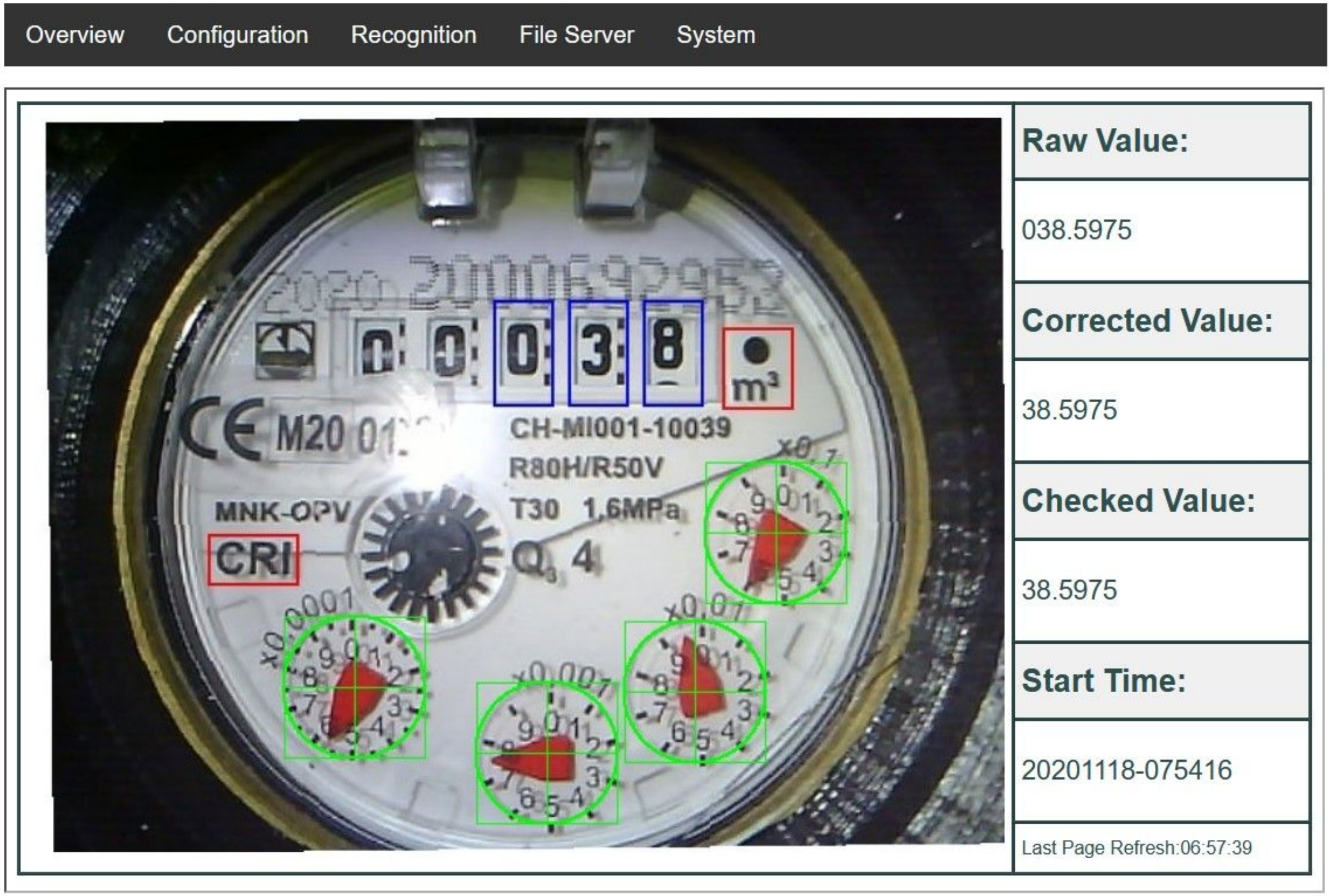
Duisterhof, B.P., Krishnan, S., Cruz, J.J., Banbury, C.R., Fu, W., Faust, A., de Croon, G.C. and Reddi, V.J., 2021, May. Tiny robot learning (tinyrl) for source seeking on a nano quadcopter. In *2021 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 7242-7248). IEEE.

Duisterhof, B.P., Li, S., Burgués, J., Reddi, V.J. and de Croon, G.C., 2021, September. Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 9099-9106). IEEE.


A DIGITAL
"SMART" METER
AN ELECTROMECHANICAL
ANALOG METER
CL200 240V 3W TYPE C1SR 30TA 1.0Kh
CPS ENERGY 4 419 901
67 599 597
Itron
WATTHOUR METER
SANGAMO
KILOWATTHOURS
CL200 240V 3W TYPE J4S 30TA 7.2Kh
414 799
54 270 799
SANGAMO WATTHOUR METER

Digitizer - AI on the edge

An ESP32 all inclusive neural network recognition system for meter digitalization



Rich Array of Sensors

Motion Sensors
Gyroscope, radar, magnetometer, accelerator

Acoustic Sensors
Ultrasonic, Microphones, Geophones, Vibrometers

Environmental Sensors
Temperature, Humidity, Pressure, IR, etc.

Touchscreen Sensors
Capacitive, IR

Image Sensors
Thermal, Image

Biometric Sensors
Fingerprint, Heart rate, etc.

Force Sensors
Pressure, Strain

Rotation Sensors
Encoders

...

No Good Data Left Behind

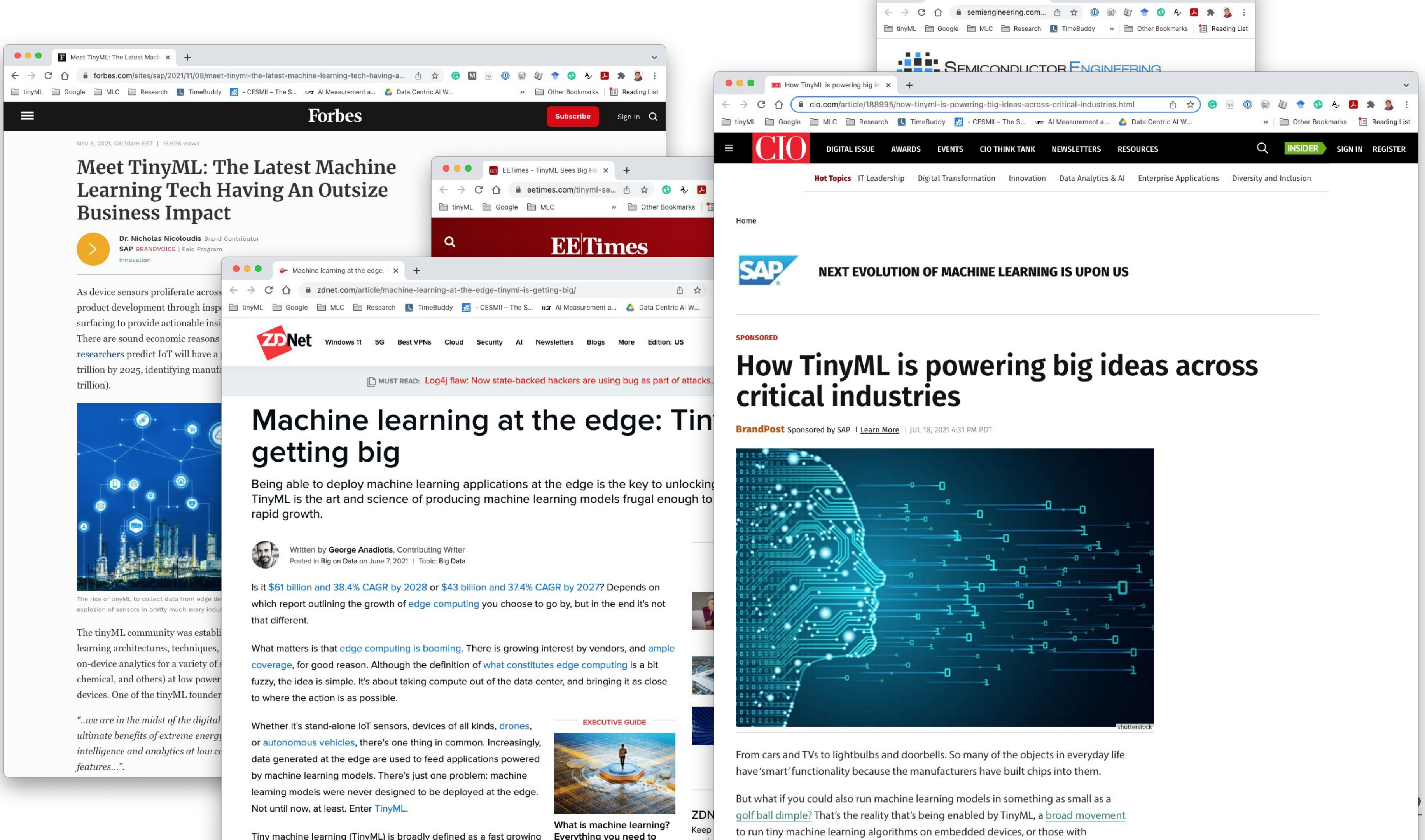
5 Quintillion

bytes of data produced every day by IoT

<1%

of unstructured data is analyzed or used at all

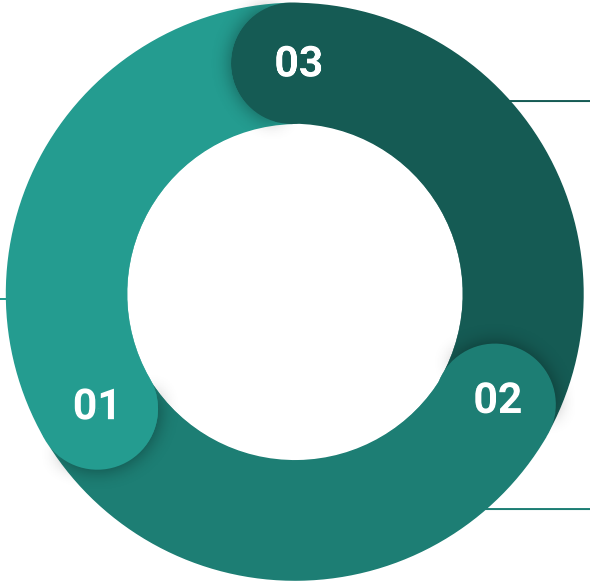
Source: Harvard Business Review, What's Your Data Strategy?, April 18, 2017
Cisco, Internet of Things (IoT) Data Continues to Explode Exponentially. Who Is Using That Data and How?, Feb 5, 2018



Meet TinyML: The Latest Machine Learning Tech Having An Outsize Business Impact
Forbes
Nov 8, 2021, 08:30am EST | 15,696 views
Dr. Nicholas Nicoloudis Brand Contributor
SAP BRANDVOICE | Paid Program
Innovation
EETimes - TinyML Sees Big Ho
EETimes
Semiconductor Engineering
Machine learning at the edge: TinyML is getting big
ZDNet
MUST READ: Log4j flaw: Now state-backed hackers are using bug as part of attacks,
Being able to deploy machine learning applications at the edge is the key to unlocking
TinyML is the art and science of producing machine learning models frugal enough to
rapid growth.
Written by George Anadiotis, Contributing Writer
Posted in Big on Data on June 7, 2021 | Topic: Big Data
Is it $61 billion and 38.4% CAGR by 2028 or $43 billion and 37.4% CAGR by 2027? Depends on which report outlining the growth of edge computing you choose to go by, but in the end it's not that different.
What matters is that edge computing is booming. There is growing interest by vendors, and ample coverage, for good reason. Although the definition of what constitutes edge computing is a bit fuzzy, the idea is simple. It's about taking compute out of the data center, and bringing it as close to where the action is as possible.
Whether it's stand-alone IoT sensors, devices of all kinds, drones, or autonomous vehicles, there's one thing in common. Increasingly, data generated at the edge are used to feed applications powered by machine learning models. There's just one problem: machine learning models were never designed to be deployed at the edge. Not until now, at least. Enter TinyML.
EXECUTIVE GUIDE
What is machine learning? Everything you need to
CIO
How TinyML is powering big ideas across critical industries
SAP
NEXT EVOLUTION OF MACHINE LEARNING IS UPON US
SPONSORED
BrandPost Sponsored by SAP | Learn More | JUL 18, 2021 4:31 PM PDT
From cars and TVs to lightbulbs and doorbells. So many of the objects in everyday life have 'smart' functionality because the manufacturers have built chips into them.
But what if you could also run machine learning models in something as small as a golf ball dimple? That's the reality that's being enabled by TinyML, a broad movement to run tiny machine learning algorithms on embedded devices, or those with

Questions

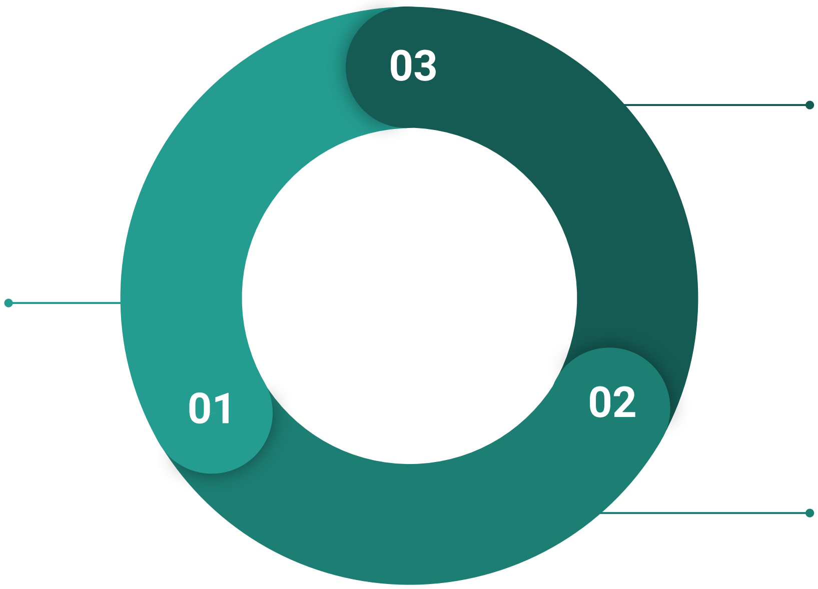
How do we design an open-source ecosystem to enable TinyML to thrive in the face of heterogeneity?



How do we drive hardware and software co-design in a flexible manner across the complete system stack?

How do we benchmark the various TinyML solutions to enable “apples to apples” system comparisons?

Questions



How do we design an open-source ecosystem to enable TinyML to thrive in the face of heterogeneity?

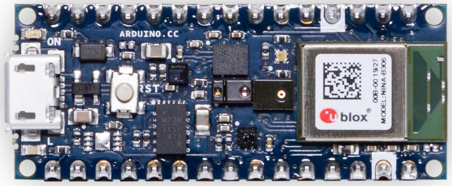
How do we drive hardware and software co-design in a flexible manner across the complete system stack?

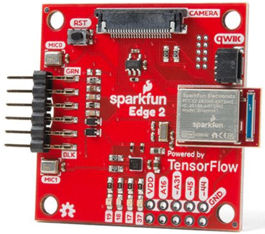
How do we benchmark the various TinyML solutions to enable "apples to apples" system comparisons?

250 Billion

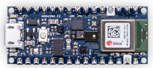
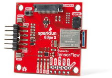
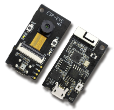
MCUs today


Himax

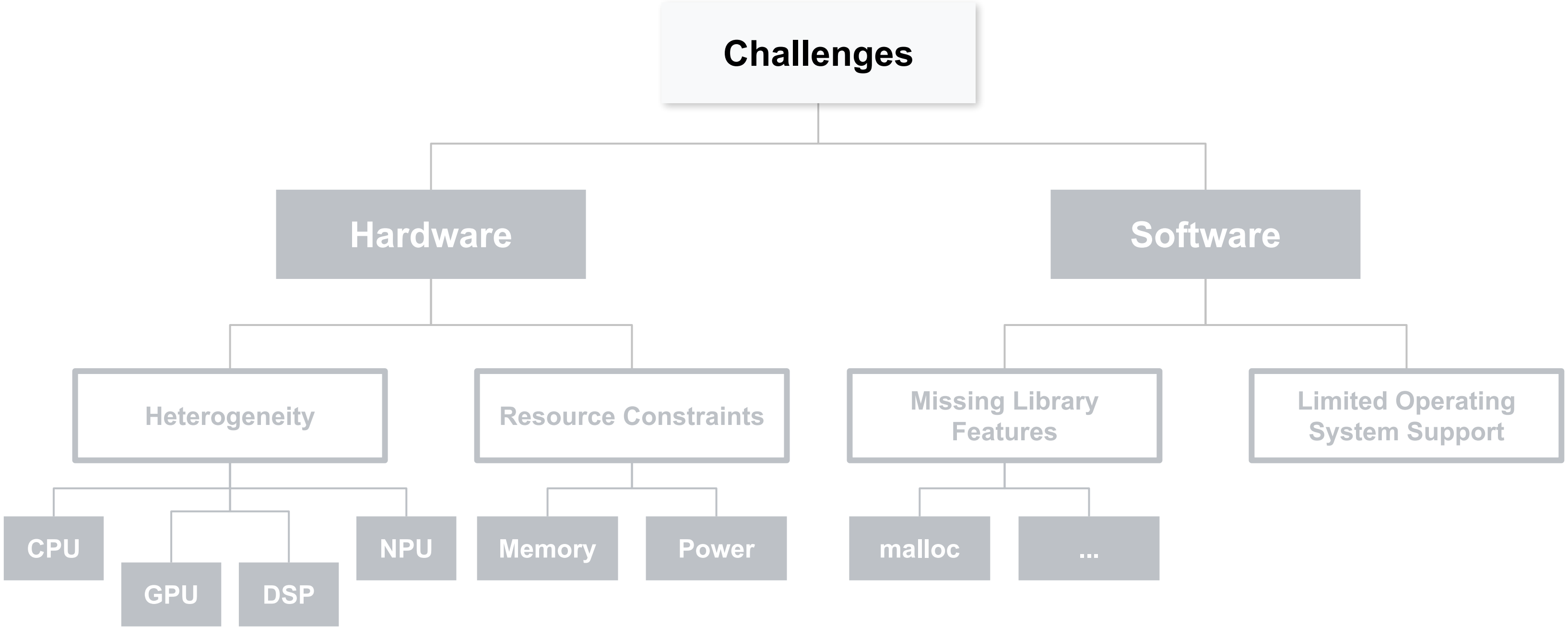

ARDUINO.CC
ON
RST
blox

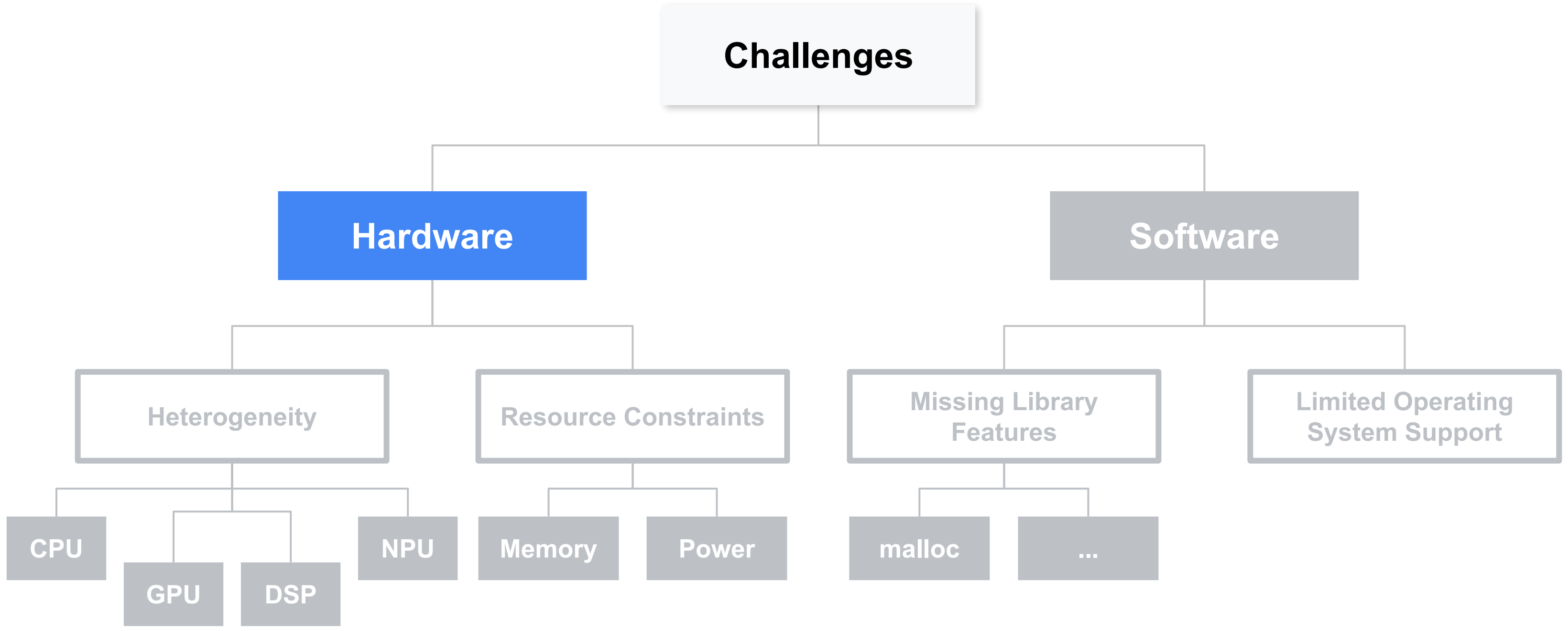

CAMERA
RST
qwiic
sparkfun
Edge 2
Powered by
TensorFlow

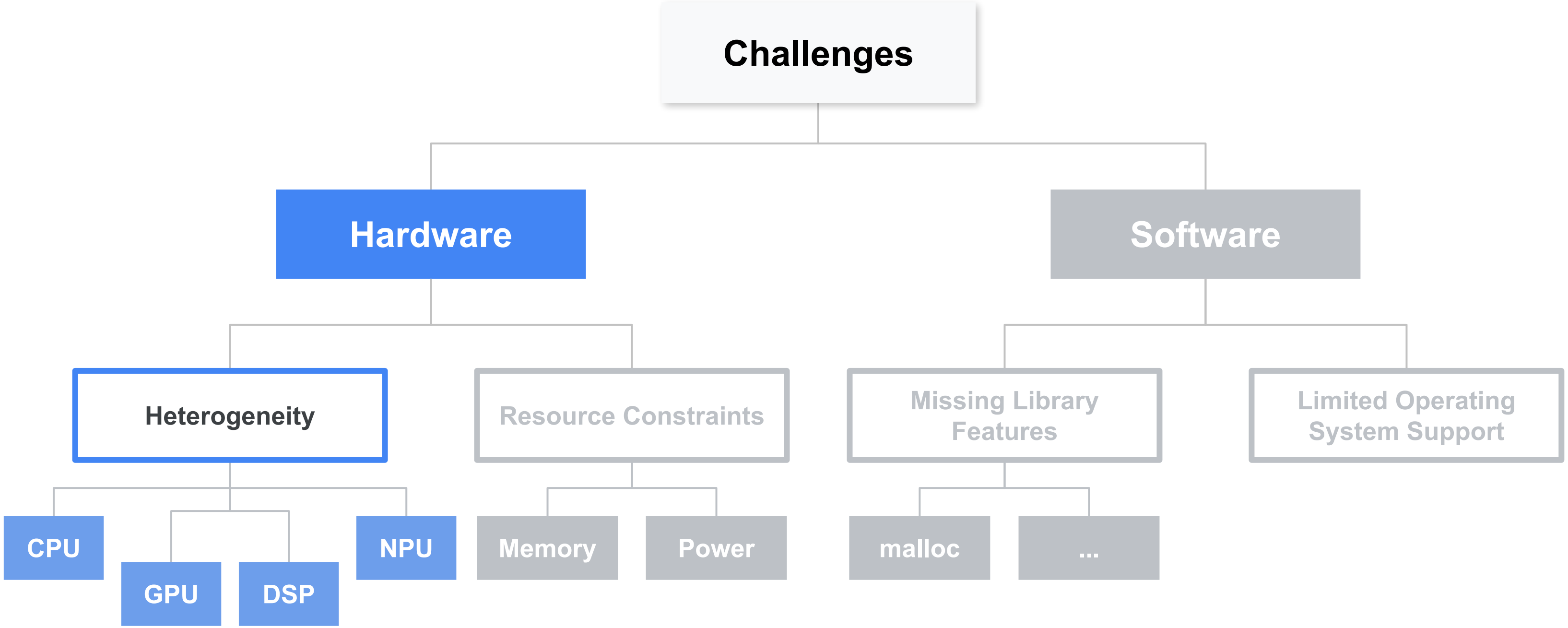

ESP-EYE
V2.1

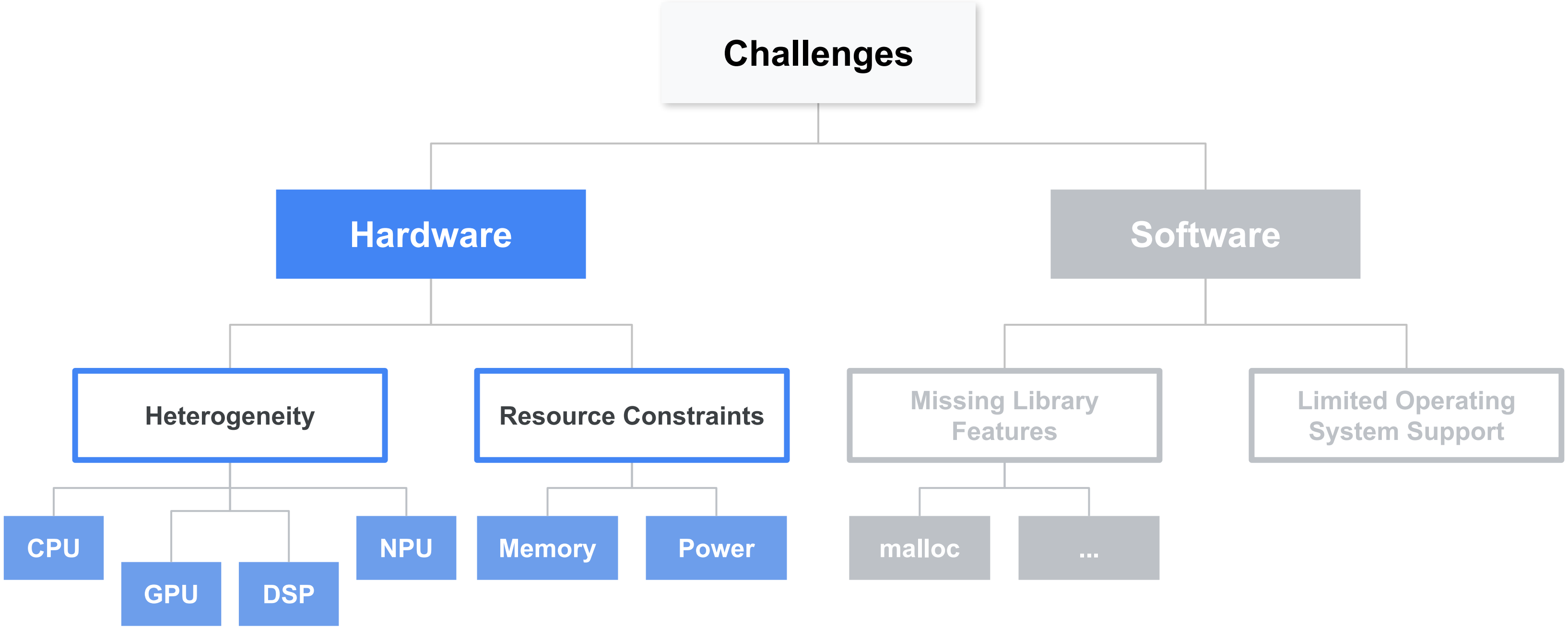
	Board	MCU / ASIC	Clock	Memory	Sensors	Radio
	Himax WE-I Plus EVB	HX6537-A 32-bit EM9D DSP	400 MHz	2MB flash 2MB RAM	Accelerometer, Mic, Camera	None
	Arduino Nano 33 BLE Sense	32-bit nRF52840	64 MHz	1MB flash 256kB RAM	Mic, IMU, Temp, Humidity, Gesture, Pressure, Proximity, Brightness, Color	BLE
	SparkFun Edge 2	32-bit ArtemisV1	48 MHz	1MB flash 384kB RAM	Accelerometer, Mic, Camera	BLE
	Espressif EYE	32-bit ESP32-D0WD	240 MHz	4MB flash 520kB RAM	Mic, Camera	WiFi, BLE

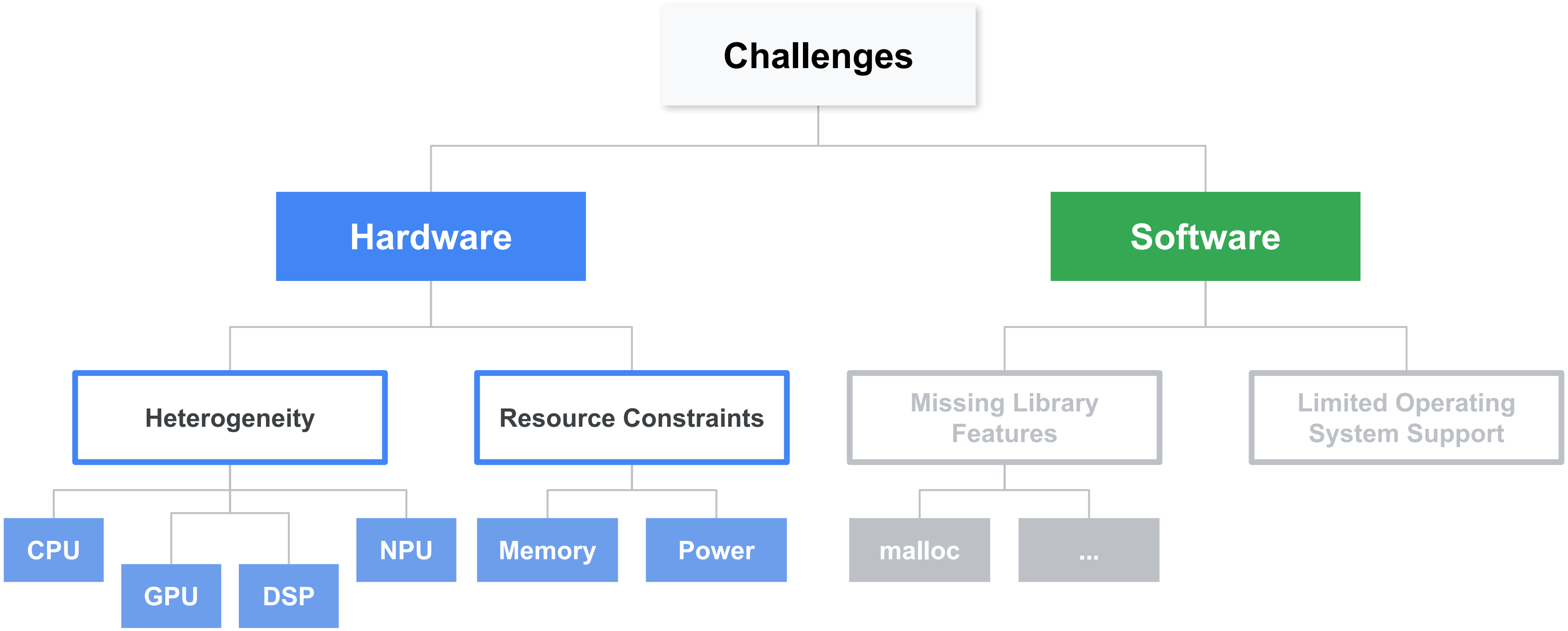
Challenges

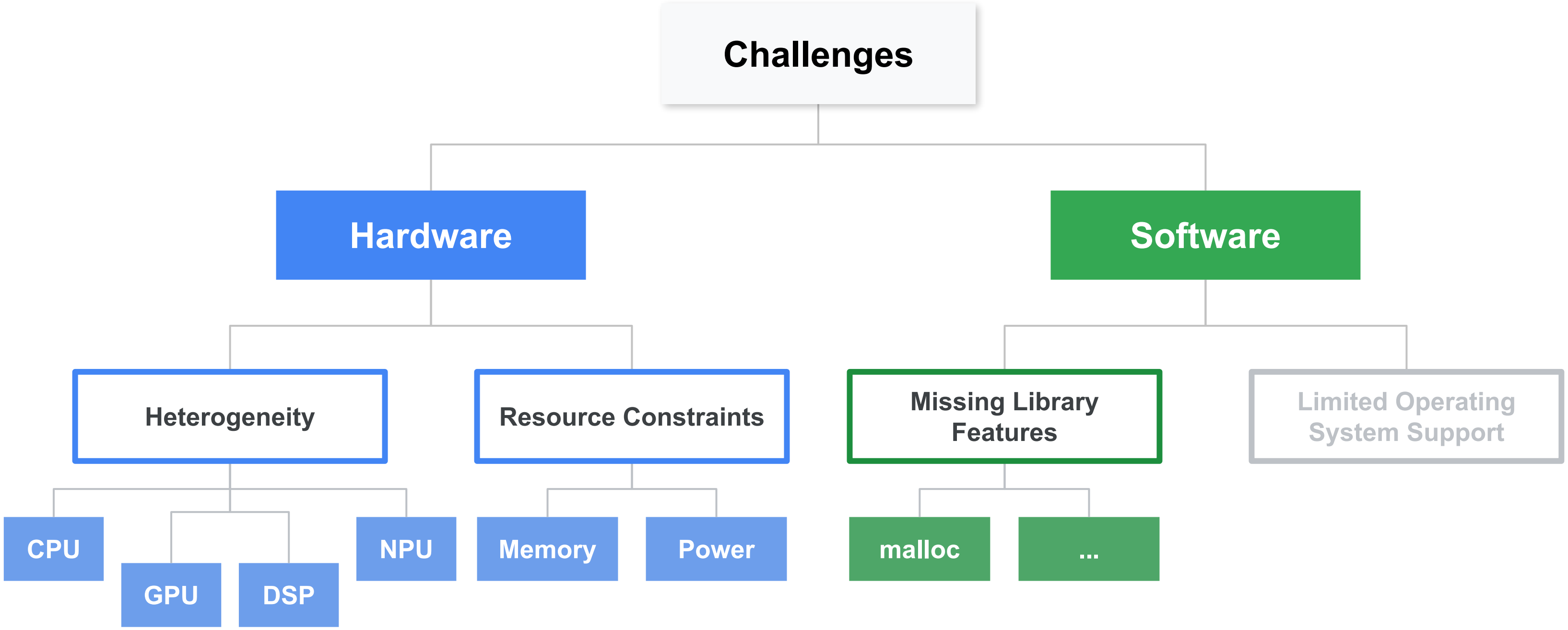

Challenges
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...

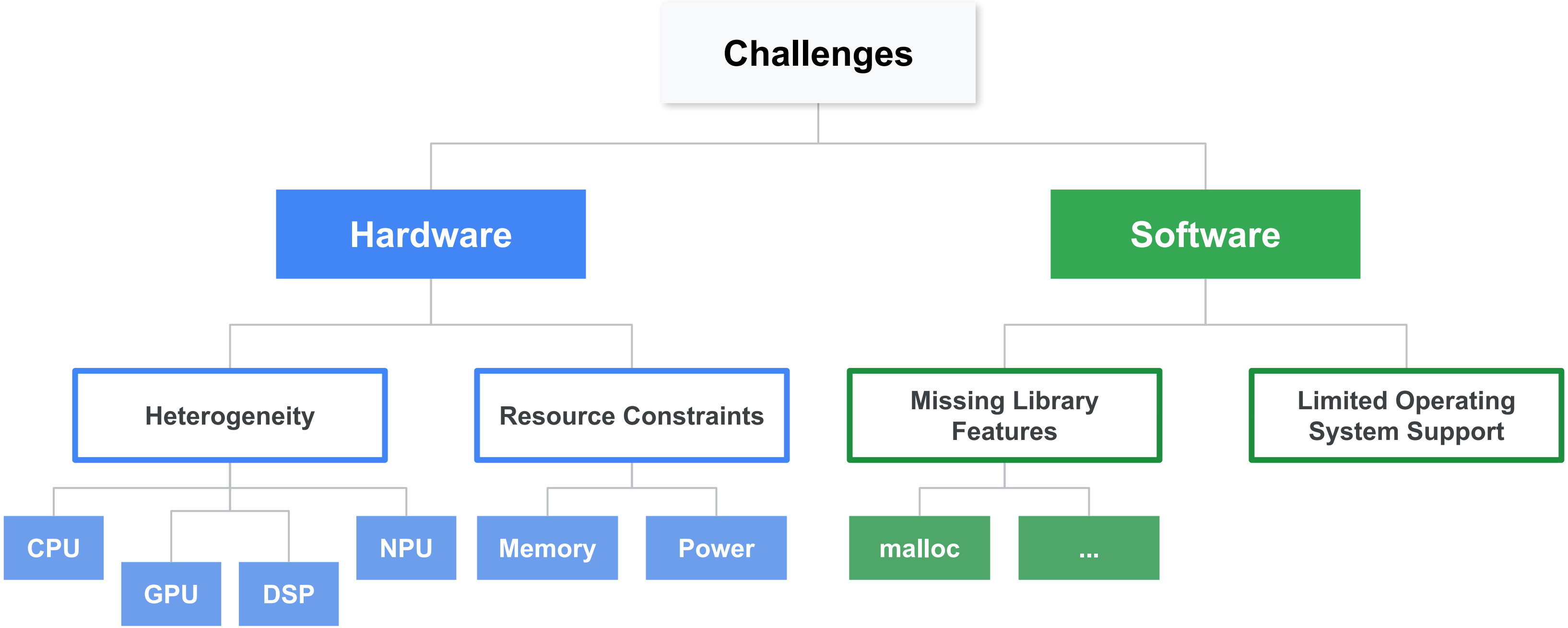

Challenges
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...

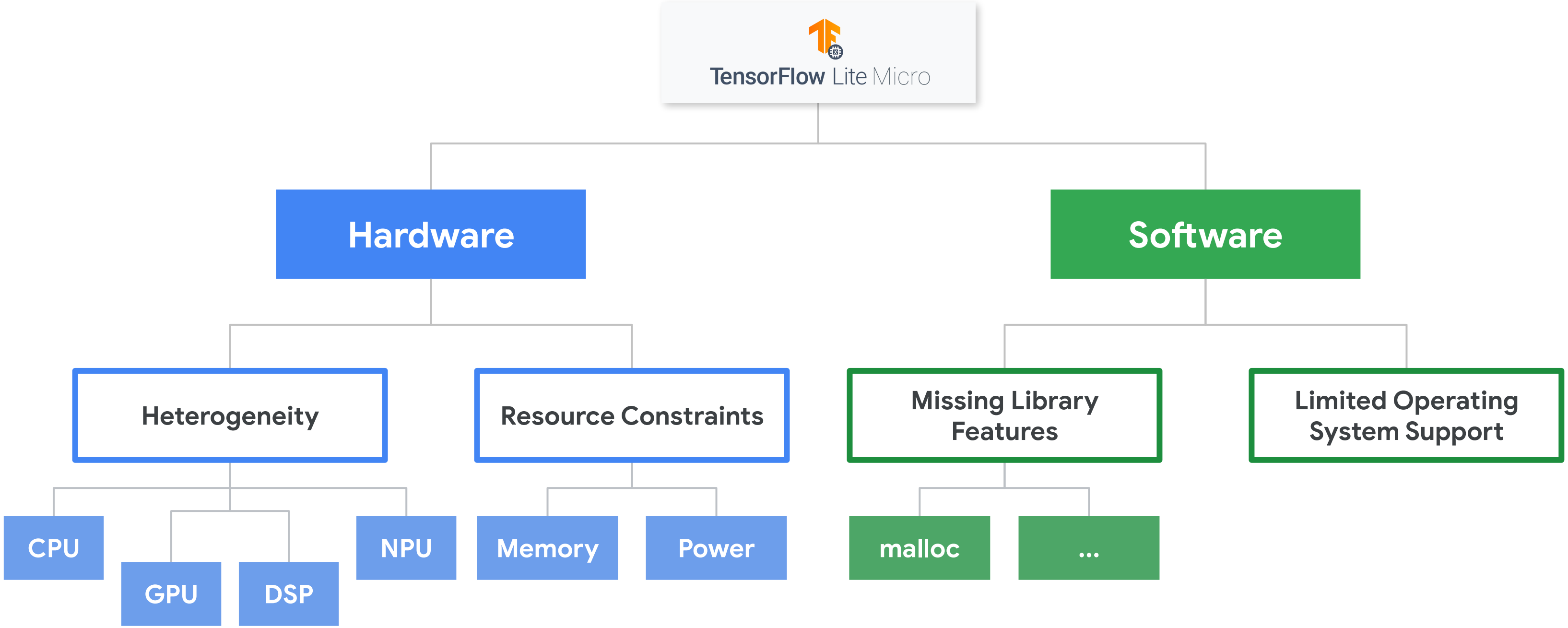

Challenges
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...


Challenges
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...


Challenges
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...


Challenges
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...


Challenges
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...


TensorFlow Lite Micro
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...

TensorFlow Lite Micro

...

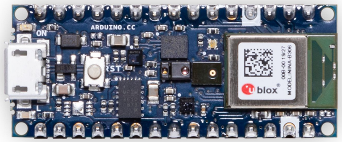
Arduino
BLE Sense 33

Himax
WE-I Plus EVB

SparkFun
Edge 2

Espressif
EYE

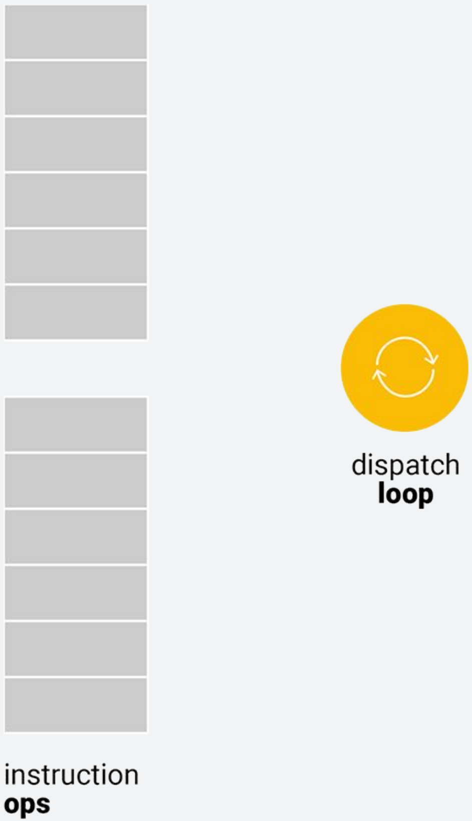
...

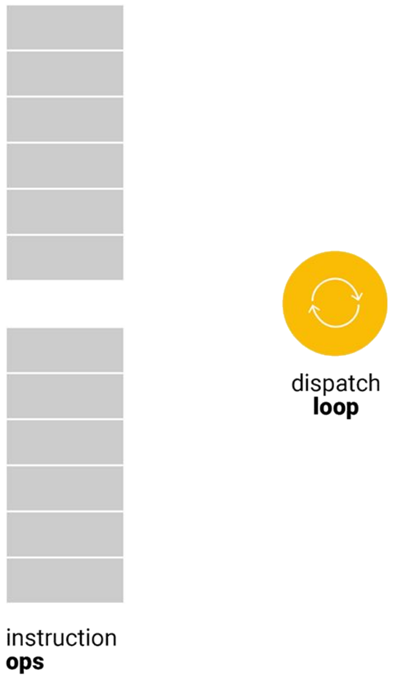


TFLite Micro Design

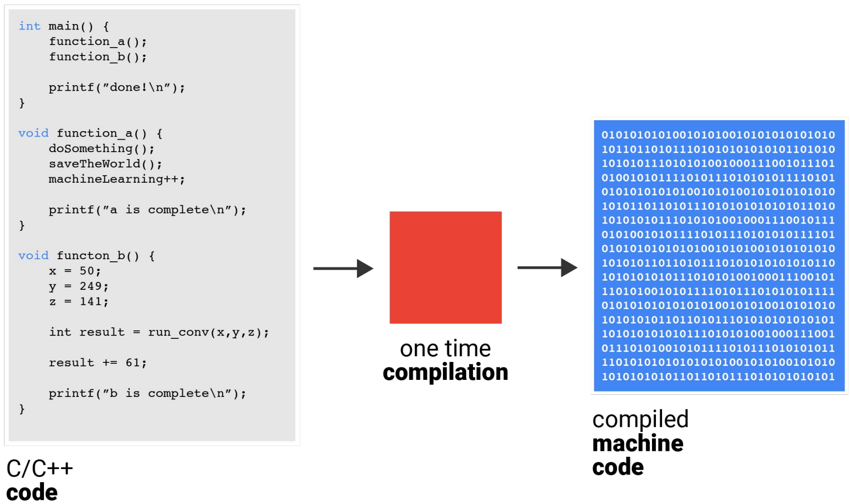
- TFLite Micro uses an **interpreter** design

- Store the model as data and loop through its ops at **runtime**

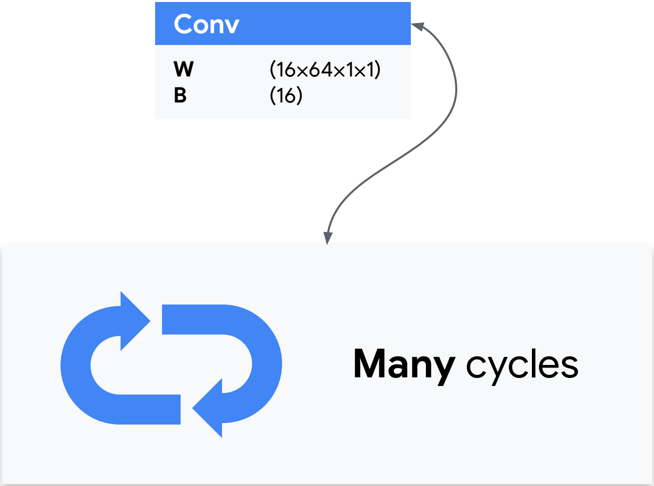




Interpreter
(generally **slower** than compiled code)

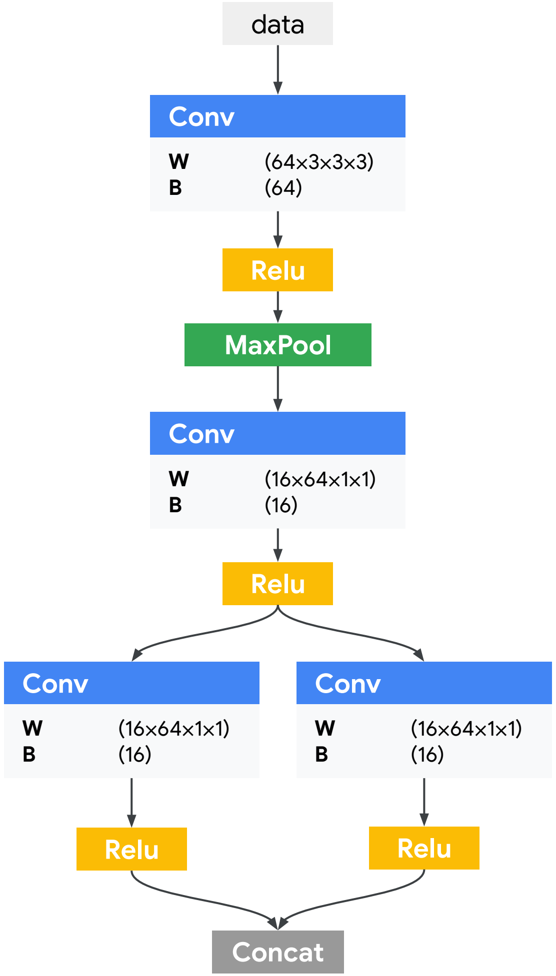


Compiler
(generally **faster** than interpreted code)



ML is **Different**

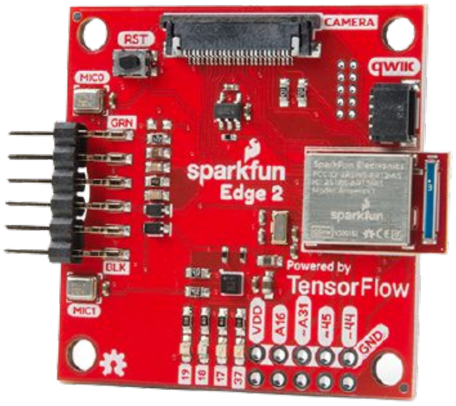
- Each layer like a `Conv` or `softmax` can take tens of thousands or even millions of cycles to complete execution



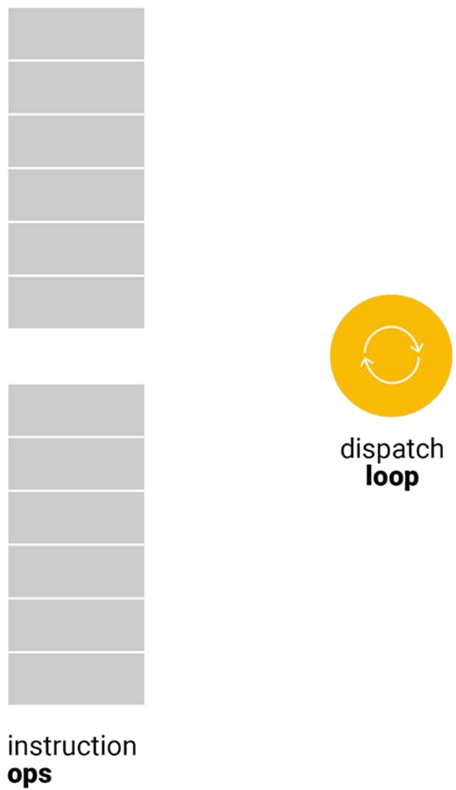
ML is **Different**

- Parsing overhead is **relatively small** for the TFMicro interpreter when we consider the **overall network graph**

Model	Total Cycles	Calculation Cycles	Interpreter Overhead
Visual Wake Words (Ref)	18,990.8K	18,987.1K	**< 0.1%**
Google Hotword (Ref)	36.4K	34.9K	**4.1%**

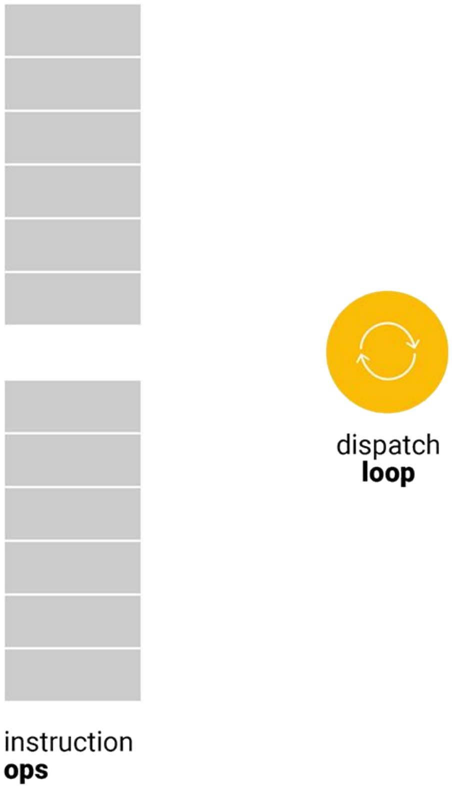


Sparkfun Edge 2
(Apollo 3 **Cortex-M4**)



Interpreter **Advantages**

- Change the model **without recompiling** the code



Interpreter **Advantages**

- Change the model **without recompiling** the code
- **Same operator code** can be used across multiple **different models** in the system

Arduino
BLE Sense 33

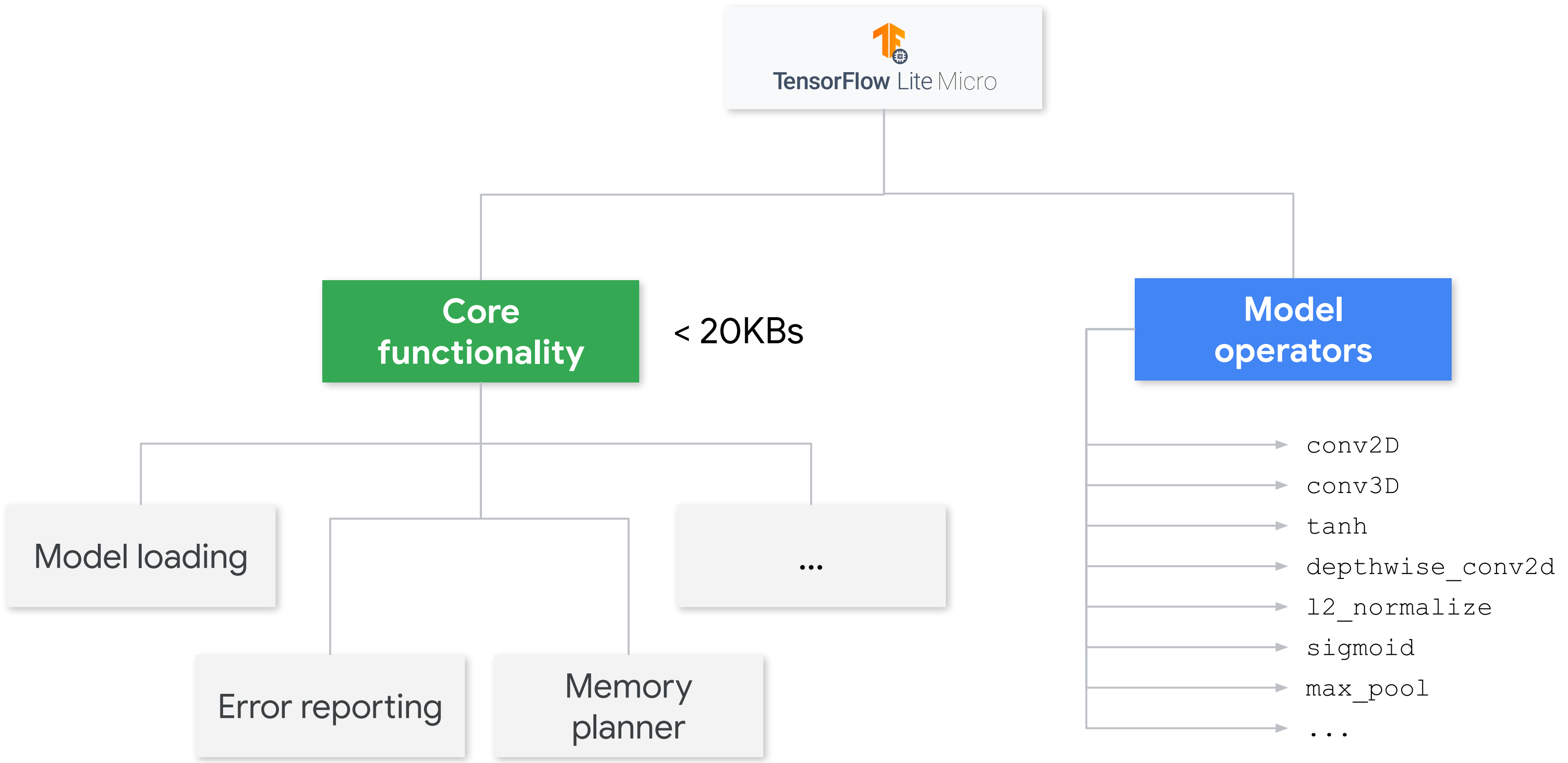
Himax
WE-I Plus EVB

Espressif
EYE

SparkFun
Edge 2

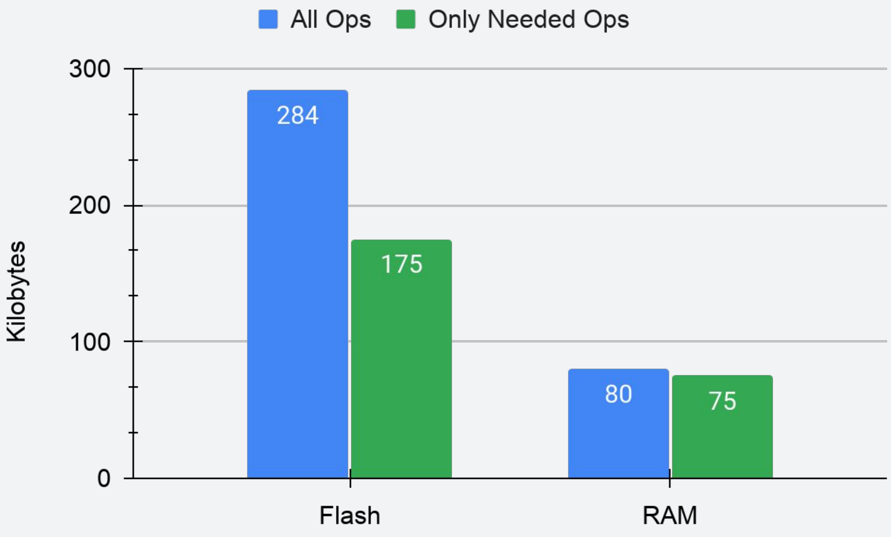
Interpreter **Advantages**

- Same **portable** model serialization format can be used **across a lots of systems**.


TensorFlow Lite Micro
Core functionality
< 20KBs
Model operators
Model loading
...
Error reporting
Memory planner
conv2D
conv3D
tanh
depthwise_conv2d
l2_normalize
sigmoid
max_pool
...

Memory **Improvements**

- Selective op registration **reduces memory consumption by 30%**
- **Memory reduction varies by model**, depending on the operators used by the model



TensorFlow Lite Micro in a Nutshell

Built to fit on **embedded systems**:

- Very **small binary footprint**
- **No** dynamic memory allocation
- **No** dependencies on complex parts of the standard C/C++ libraries
- **No** operating system dependencies, **can run on bare metal**
- Designed to be **portable** across a wide variety of systems

arXiv:2010.08678v3 [cs.LG] 13 Mar 2021

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

TensorFlow Lite Micro (TFLM) is an open-source ML inference framework for running deep-learning models on embedded systems. TFLM 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 unique challenges. In this paper, we explain the design decisions behind TFLM and describe its implementation. We present an evaluation of TFLM to demonstrate its low resource requirements and minimal run-time performance overheads.

1 Introduction

Tiny machine learning (TinyML) is a burgeoning field at the intersection of embedded systems and machine learning. The world has over 250 billion microcontrollers (IC Insights, 2020), with strong growth projected over coming years. As such, a new range of embedded applications are emerging for neural networks. Because these models are extremely small (few hundred KBs), running on microcontrollers or DSP-based embedded subsystems, they can operate continuously with minimal impact on device battery life.

The most well-known and widely deployed example of this new TinyML technology is keyword spotting, also called hotword or wakeword detection (Chen et al., 2014; Gruenstein et al., 2017; Zhang et al., 2017). Amazon, Apple, Google, and others use tiny neural networks on billions of devices to run always-on inferences for keyword detection—and this is far from the only TinyML application. Low-latency analysis and modeling of sensor signals from microphones, low-power image sensors, accelerometers, gyros, PPG optical sensors, and other devices enable consumer and industrial applications, including predictive maintenance (Goebel et al., 2020; Susto et al., 2014), acoustic-anomaly detection (Koizumi et al., 2019), visual object detection (Chowdhery et al., 2019), and human-activity recognition (Chavarriaga et al., 2013; Zhang & Sawchuk, 2012).

Unlocking machine learning's potential in embedded devices requires overcoming two crucial challenges. First and foremost, embedded systems have no unified TinyML framework. When engineers have deployed neural networks to such systems, they have built one-off frameworks that require manual optimization for each hardware platform. Such custom frameworks have tended to be narrowly focused, lacking features to support multiple applications and lacking portability across a wide range of hardware. The developer experience has therefore been painful, requiring hand optimization of models to run on a specific device. And altering these models to run on another device necessitated manual porting and repeated optimization effort. An important second-order effect of this situation is that the slow pace and high cost of training and deploying models to embedded hardware prevents developers from easily justifying the investment required to build new features.

Another challenge limiting TinyML is that hardware vendors have related but separate needs. Without a generic TinyML framework, evaluating hardware performance in a neutral, vendor-agnostic manner has been difficult. Frameworks are tied to specific devices, and it is hard to determine the source of improvements because they can come from hardware, software, or the complete vertically integrated solution.

The lack of a proper framework has been a barrier to accelerating TinyML adoption and application in products. Beyond deploying a model to an embedded target, the framework must also have a means of training a model on a higher-compute platform. TinyML must exploit a broad ecosystem of tools for ML, as well for orchestrating and debugging models, which are beneficial for production devices.

Prior efforts have attempted to bridge this gap. We can distill the major issues facing the frameworks into the following:

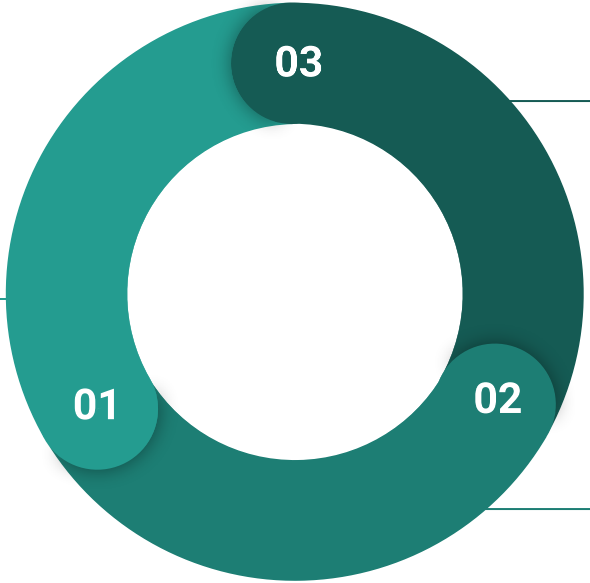
[1]Google [2]Harvard University. Correspondence to: Pete Warden <petewarden@google.com>, Vijay Janapa Reddi <vj@eecs.harvard.edu>.

Proceedings of the 4th MLSys Conference, San Jose, CA, USA, 2021.

David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Wang, T. and Warden, P., 2021. Tensorflow lite micro: Embedded machine learning for tinyml systems. Proceedings of Machine Learning and Systems, 3, pp.800-811.

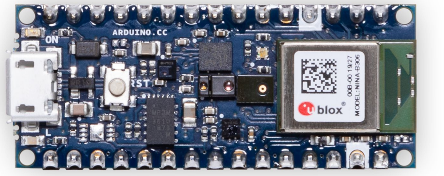
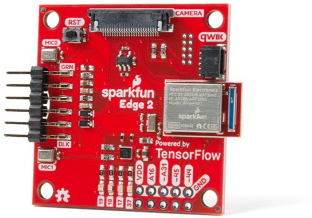
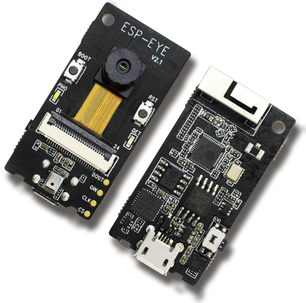
Questions

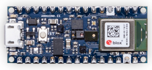
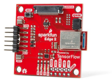
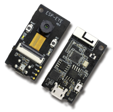
How do we design an open-source ecosystem to enable TinyML to thrive in the face of heterogeneity?

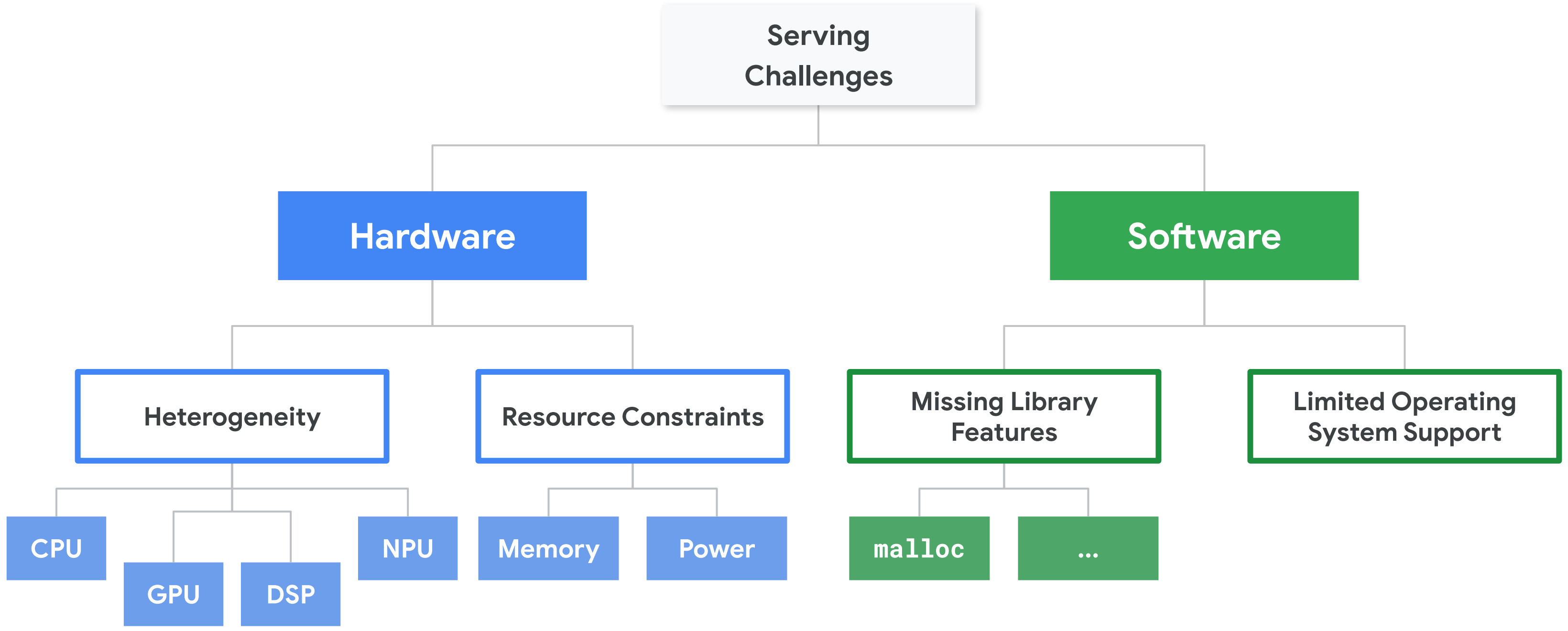


How do we drive hardware and software co-design in a flexible manner across the complete system stack?

How do we benchmark the various TinyML solutions to enable "apples to apples" system comparisons?

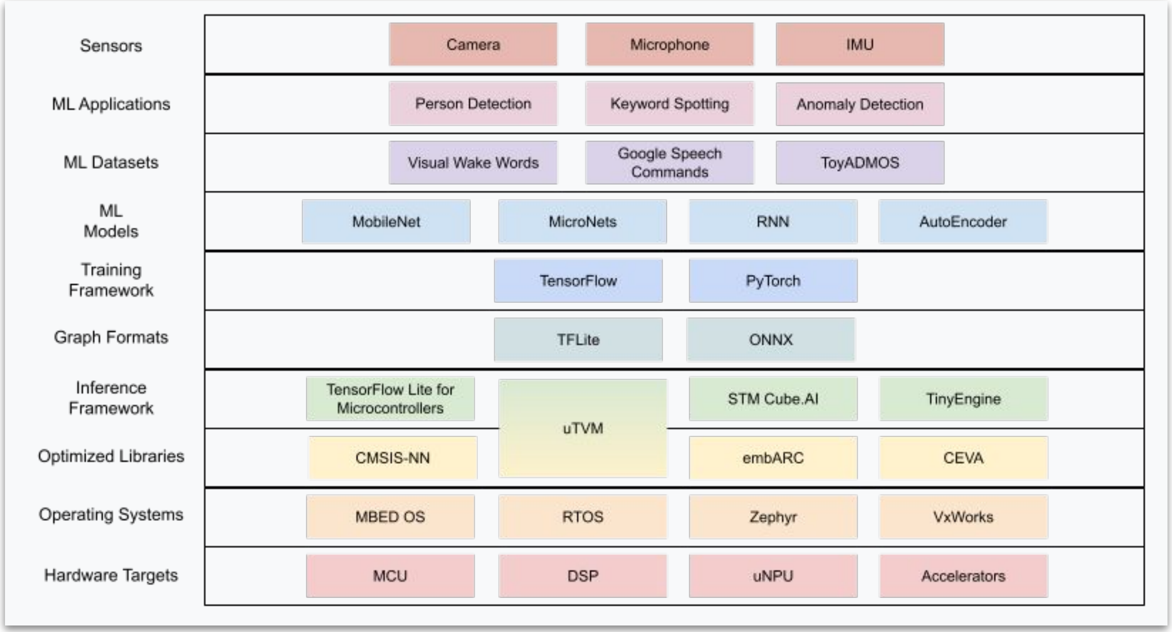

Himax

ARDUINO.CC
blox

sparkfun
Edge 2
Powered by
TensorFlow

ESP-EYE

	Board	MCU / ASIC	Clock	Memory	Sensors	Radio
	Himax WE-I Plus EVB	HX6537-A 32-bit EM9D DSP	400 MHz	2MB flash 2MB RAM	Accelerometer, Mic, Camera	None
	Arduino Nano 33 BLE Sense	32-bit nRF52840	64 MHz	1MB flash 256kB RAM	Mic, IMU, Temp, Humidity, Gesture, Pressure, Proximity, Brightness, Color	BLE
	SparkFun Edge 2	32-bit ArtemisV1	48 MHz	1MB flash 384kB RAM	Accelerometer, Mic, Camera	BLE
	Espressif EYE	32-bit ESP32-D0WD	240 MHz	4MB flash 520kB RAM	Mic, Camera	WiFi, BLE


Serving Challenges
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...

TinyML System Stack is Complicated

- Machine learning system stack is **complicated**
- Many **different** models, datasets, models, frameworks, formats, compilers, libraries, operating systems, targets
- The **cross-product** makes it challenging to decipher system performance



Apples-to-apples comparison

What task?
What model?
What dataset?
What batch size?
What quantization?
What software
libraries?
…

bench·mark

/ˈben(t)SHmärk/

See definitions in:

All | Technology | Surveying

noun

1. a standard or point of reference against which things may be compared or assessed.
   "a benchmark case"
   Similar: standard | point of reference | basis | gauge | criterion | specification
2. a surveyor's mark cut in a wall, pillar, or building and used as a reference point in measuring altitudes.

verb

evaluate or check (something) by comparison with a standard.
"we are **benchmarking** our performance **against** external criteria"

Definitions from Oxford Languages

Feedback

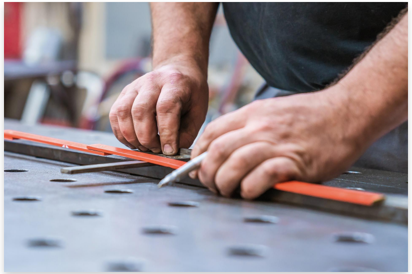
Benchmarking

Use to

- **Compare** solutions
- **Inform** selection
- **Measure** and track progress
- **Raise** the bar, **advance** the field

Requires

- **Methodology** that is both fair and rigorous
- **Community** support and consensus



Provides

- **Standardization** of use cases and workloads
- **Comparability** across heterogeneous HW/SW systems
- **Complex characterization** of system compromises
- **Verifiable and Reproducible** results

Wide Array of ML Tasks

Task Category	Use Case	Model Type	Datasets
Audio	Audio Wake Words Context Recognition Control Words Keyword Detection	DNN CNN RNN LSTM	Speech Commands Audioset ExtraSensory Freesound DCASE
Image	Visual Wake Words Object Detection Gesture Recognition Object Counting Text Recognition	DNN CNN SVM Decision Tree KNN Linear	Visual Wake Words CIFAR10 MNIST ImageNet DVS128 Gesture
Physiological / Behavioral Metrics	Segmentation Anomaly Detection Forecasting Activity Detection	DNN Decision Tree SVM Linear	Physionet HAR DSA Opportunity
Industry Telemetry	Sensing Predictive Maintenance Motor Control	DNN Decision Tree SVM Linear Naive Bayes	UCI Air Quality UCI Gas UCI EMG NASA's PCoE

MLPerf

Goals

MLPerf

Enforce performance result replicability to ensure reliable results

Use **representative workloads**, reflecting production use-cases

Encourage innovation to improve the state-of-the-art of ML

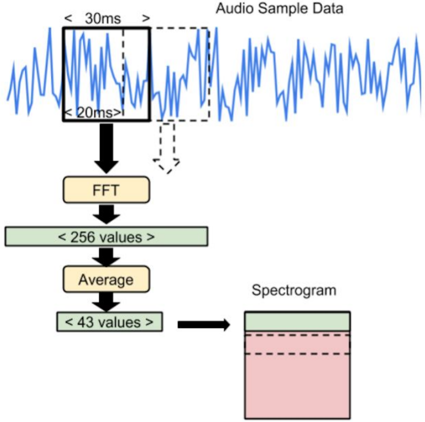
Accelerate progress in ML via **fair and useful measurement**

Serve both the **commercial and research communities**

Keep **benchmarking affordable** so that all can participate

MLPerf “Tiny” Tasks

Keyword Spotting



Warden, Pete. "Speech commands: A dataset for limited-vocabulary speech recognition." *arXiv preprint arXiv:1804.03209* (2018).

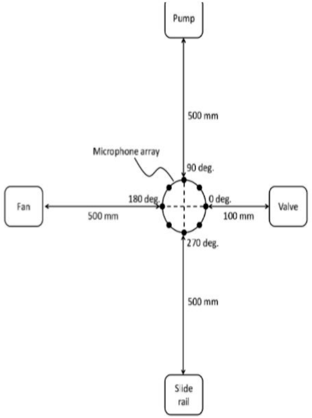
Visual Wake Words

(a) ‘Person’

(b) ‘Not-person’

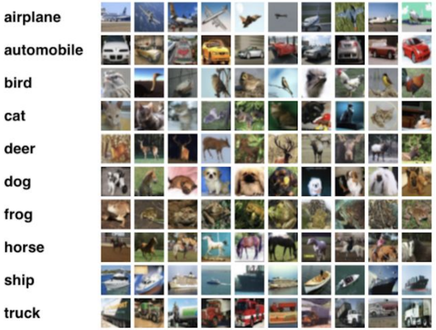
Chowdhery, Aakanksha, et al. "Visual wake words dataset." *arXiv preprint arXiv:1906.05721* (2019).

Anomaly Detection



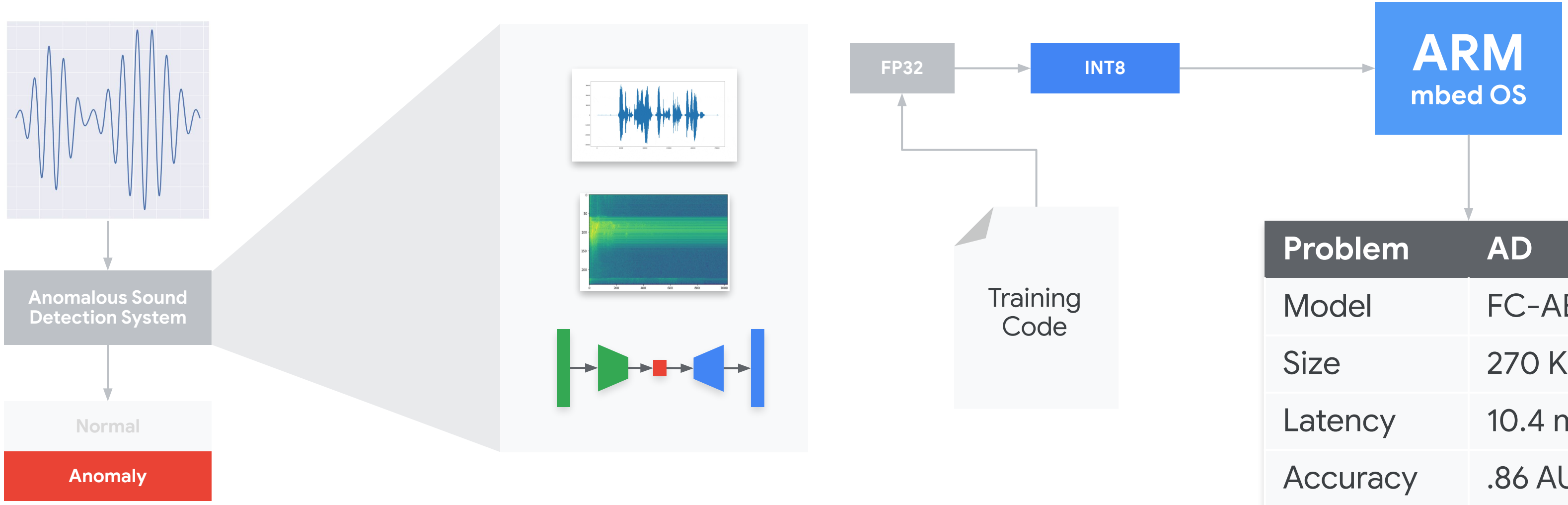
Purohit, Harsh, et al. "MIMII dataset: Sound dataset for malfunctioning industrial machine investigation and inspection." *arXiv preprint arXiv:1909.09347* (2019).

Tiny Image Classification



Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009): 7.

Problem definition → Dataset selection (public domain) → Model selection → Model training code → Derive “Tiny” version: Quantization → Embedded implementation → Benchmarking harness integration → Deploy on device → **Example benchmark run**



Problem	AD
Model	FC-AE
Size	270 Kpar
Latency	10.4 ms/inf.
Accuracy	.86 AUC
Energy	516 µJ/inf.

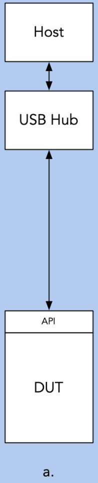
Metrics

Latency

Small fast dataset

Loop of inferences

No data-dependent execution




```
Runtime requirements have been met.
Performance results for window 10:
  # Inferences :          1000
  Runtime      :        10.524 sec.
  Throughput   :        95.020 inf./sec.
Runtime requirements have been met.
----------------------------------------
Median throughput is 95.019 inf./sec.
----------------------------------------
```

Accuracy

Evaluate on larger dataset

Top-1 accuracy & AUC

CLOSED: meet threshold

v.

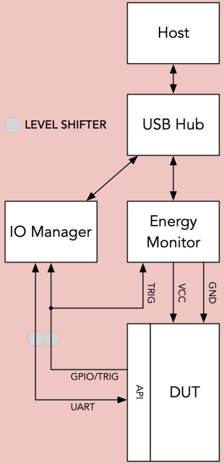
OPEN: part of the metrics

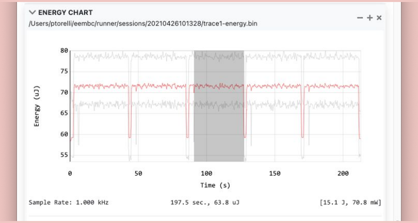
Energy

No "cherry-picking"

Power Monitor setup

Median result





MLPerf Tiny in a Nutshell

Built to benchmark **embedded ML systems**:

- **Standardize best practices** in TinyML benchmarking
- **Measure both ML performance and power** consumption
- Designed to be **portable across a wide variety of systems**

Division	Dataset	Training	Model	Numerics	Framework	Hardware	Demonstrates
Closed	X	X	X	INT-8 PTQ	TensorFlow Lite Micro	ARM MCU	Baseline performance results on the reference platform.
Closed	X	X	X	INT-8 PTQ	TensorFlow Lite Micro	RISC-V MCU	Performance of a RISC-V microcontroller customized for neural network inference.
Closed	X	X	X	FP-32 & INT-8 PTQ	LEIP Framework	RasPi 4	Capabilities of a software-only optimization toolchain that is agnostic of the hardware.
Closed	X	X	X	INT-8 PTQ	Syntiant TDK	Neural Network Accelerator	Ultra-low power hardware efficiency for running deep neural networks.
Open	X	QKeras	✓	Int-6/8 QAT	HLS4ML	FPGA	Rapid end-to-end development of machine learning accelerators on reconfigurable fabrics.

arXiv:2106.07597v4 [cs.LG] 24 Aug 2021

MLPerf Tiny Benchmark

Colby Banbury* Vijay Janapa Reddi* Peter Torelli† Jeremy Holleman‡‖ Nat Jeffries§

Csaba Kiraly¶ Pietro Montino* David Kanter** Sebastian Ahmed†† Danilo Pau‡‡

Urmish Thakker[I] Antonio Torrini[II] Peter Warden§ Jay Cordaro ‡Giuseppe Di Guglielmo[III]

Javier Duarte[IV] Stephen Gibellini‡ Videet Parekh[V] Honson Tran[V] Nhan Tran[VI]

Niu Wenxu[VII] Xu Xuesong[VII]

Abstract

Advancements in ultra-low-power *tiny* machine learning (TinyML) systems promise to unlock an entirely new class of smart applications. However, continued progress is limited by the lack of a widely accepted and easily reproducible benchmark for these systems. To meet this need, we present MLPerf Tiny, the first industry-standard benchmark suite for ultra-low-power tiny machine learning systems. The benchmark suite is the collaborative effort of more than 50 organizations from industry and academia and reflects the needs of the community. MLPerf Tiny measures the accuracy, latency, and energy of machine learning inference to properly evaluate the tradeoffs between systems. Additionally, MLPerf Tiny implements a modular design that enables benchmark submitters to show the benefits of their product, regardless of where it falls on the ML deployment stack, in a fair and reproducible manner. The suite features four benchmarks: keyword spotting, visual wake words, image classification, and anomaly detection.

1 Introduction

Machine learning (ML) inference on the edge is an increasingly attractive prospect due to its potential for increasing energy efficiency [4], privacy, responsiveness, and autonomy of edge devices. Thus far, the field edge ML has predominantly focused on mobile inference, but in recent years, there have been major strides towards expanding the scope of edge ML to ultra-low-power devices. The field, known as "TinyML" [1], achieves ML inference under a milliWatt, and thereby breaks the traditional power barrier preventing widely distributed machine intelligence. By performing inference on-device, and near-sensor, TinyML enables greater responsiveness and privacy while avoiding the energy cost associated with wireless communication, which at this scale is far higher than that of compute [5]. Furthermore, the efficiency of TinyML enables a class of smart, battery-powered, always-on applications that can revolutionize the real-time collection and processing of data. Deploying advanced ML applications at this scale requires the co-optimization of each layer of the ML deployment stack to achieve the maximum efficiency. Due to this complex optimization, the

*Harvard University, †EEMBC, ‡Syntiant ‖UNC Charlotte §Google ¶Digital Catapult *VoiceMed **MLCommons ††Qualcomm ‡‡STMicroelectronics [I]SambaNova Systems [II]Silicon Labs [III]Columbia [IV]UCSD [V]Latent AI [VI]Fermilab [VII]Peng Cheng Labs

Preprint. Under review.

Banbury, C., Reddi, V.J., Torelli, P., Holleman, J., Jeffries, N., Kiraly, C., Montino, P., Kanter, D., Ahmed, S., Pau, D. and Thakker, U., 2021. Mlperf tiny benchmark. NeurIPS'21

Toward Emerging Multi-DNN Models

Pipelined DNNs

Keyword Spotting | Speech Processing

- Back-to-back execution
- Execution dependency

Concurrent DNNs

Eye Tracking | Obstacle Detection | Video Processing

- Concurrent execution
- Execution deadline

Concurrent & Pipelined DNNs

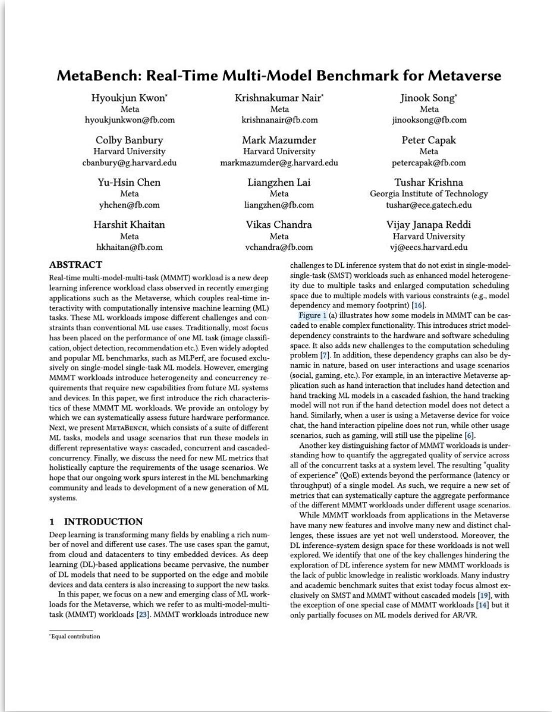
Obstacle Detection

Eye Tracking → Foveated Rendering

- Challenges from both pipelined and concurrent

MetaBench in a Nutshell (Stay Tuned!)

- We **demystify the unique features and challenges of MMMT workloads** for Metaverse applications
- We **provide a taxonomy of MMMT workloads** to understand new classes of deep learning inference workloads and discuss their feature and challenges
- Based on realistic applications, we propose a **real-time MMMT benchmark suite** that models the different Metaverse end-user usage scenarios.
- We also **discuss the need for new scoring metrics** that reflect ML system performance in a useful manner.



MetaBench: Real-Time Multi-Model Benchmark for Metaverse

Hyoukjun Kwon* Meta hyoukjunkwon@fb.com
Krishnakumar Nair* Meta krishnanair@fb.com
Jinook Song* Meta jinooksong@fb.com
Colby Banbury Harvard University cbanbury@g.harvard.edu
Mark Mazumder Harvard University markmazumder@g.harvard.edu
Peter Capak Meta petercapak@fb.com
Yu-Hsin Chen Meta yhchen@fb.com
Liangzhen Lai Meta liangzhen@fb.com
Tushar Krishna Georgia Institute of Technology tushar@ece.gatech.edu
Harshit Khaitan Meta hkhaitan@fb.com
Vikas Chandra Meta vchandra@fb.com
Vijay Janapa Reddi Harvard University vj@eecs.harvard.edu

ABSTRACT

Real-time multi-model-multi-task (MMMT) workload is a new deep learning inference workload class observed in recently emerging applications such as the Metaverse, which couples real-time interactivity with computationally intensive machine learning (ML) tasks. These ML workloads impose different challenges and constraints than conventional ML use cases. Traditionally, most focus has been placed on the performance of one ML task (image classification, object detection, recommendation etc.). Even widely adopted and popular ML benchmarks, such as MLPerf, are focused exclusively on single-model single-task ML models. However, emerging MMMT workloads introduce heterogeneity and concurrency requirements that require new capabilities from future ML systems and devices. In this paper, we first introduce the rich characteristics of these MMMT ML workloads. We provide an ontology by which we can systematically assess future hardware performance. Next, we present MetaBench, which consists of a suite of different ML tasks, models and usage scenarios that run these models in different representative ways: cascaded, concurrent and cascaded-concurrency. Finally, we discuss the need for new ML metrics that holistically capture the requirements of the usage scenarios. We hope that our ongoing work spurs interest in the ML benchmarking community and leads to development of a new generation of ML systems.

1 INTRODUCTION

Deep learning is transforming many fields by enabling a rich number of novel and different use cases. The use cases span the gamut, from cloud and datacenters to tiny embedded devices. As deep learning (DL)-based applications became pervasive, the number of DL models that need to be supported on the edge and mobile devices and data centers is also increasing to support the new tasks.

In this paper, we focus on a new and emerging class of ML workloads for the Metaverse, which we refer to as multi-model-multi-task (MMMT) workloads [23]. MMMT workloads introduce new challenges to DL inference system that do not exist in single-model-single-task (SMST) workloads such as enhanced model heterogeneity due to multiple tasks and enlarged computation scheduling space due to multiple models with various constraints (e.g., model dependency and memory footprint) [16].

Figure 1 (a) illustrates how some models in MMMT can be cascaded to enable complex functionality. This introduces strict model-dependency constraints to the hardware and software scheduling space. It also adds new challenges to the computation scheduling problem [7]. In addition, these dependency graphs can also be dynamic in nature, based on user interactions and usage scenarios (social, gaming, etc.). For example, in an interactive Metaverse application such as hand interaction that includes hand detection and hand tracking ML models in a cascaded fashion, the hand tracking model will not run if the hand detection model does not detect a hand. Similarly, when a user is using a Metaverse device for voice chat, the hand interaction pipeline does not run, while other usage scenarios, such as gaming, will still use the pipeline [6].

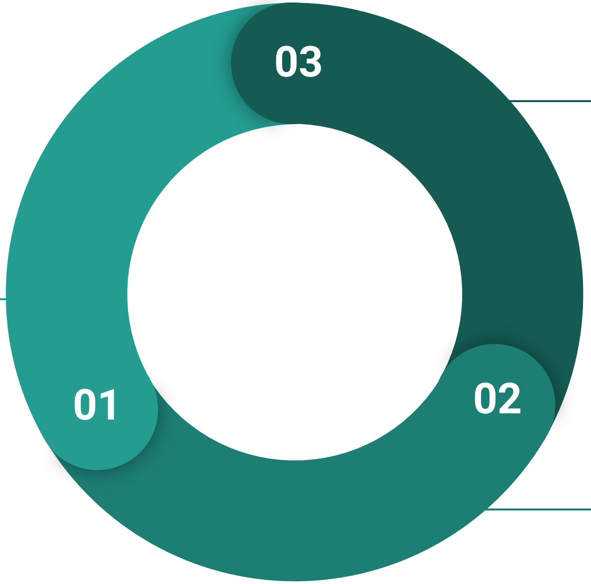
Another key distinguishing factor of MMMT workloads is understanding how to quantify the aggregated quality of service across all of the concurrent tasks at a system level. The resulting "quality of experience" (QoE) extends beyond the performance (latency or throughput) of a single model. As such, we require a new set of metrics that can systematically capture the aggregate performance of the different MMMT workloads under different usage scenarios.

While MMMT workloads from applications in the Metaverse have many new features and involve many new and distinct challenges, these issues are yet not well understood. Moreover, the DL inference-system design space for these workloads is not well explored. We identify that one of the key challenges hindering the exploration of DL inference system for new MMMT workloads is the lack of public knowledge in realistic workloads. Many industry and academic benchmark suites that exist today focus almost exclusively on SMST and MMMT without cascaded models [19], with the exception of one special case of MMMT workloads [14] but it only partially focuses on ML models derived for AR/VR.

*Equal contribution

Questions

How do we design an open-source ecosystem to enable TinyML to thrive in the face of heterogeneity?

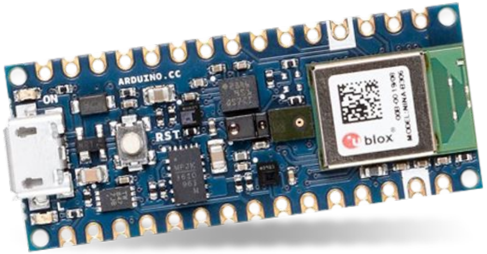


How do we drive hardware and software co-design in a flexible manner across the complete system stack?

How do we benchmark the various TinyML solutions to enable “apples to apples” system comparisons?

The Hardware Lottery

- Sara Hooker's observation that the success of new ML approaches depends on their compatibility with downstream software and hardware. Here you can "make your own luck"!



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

Specialized Hardware Customization (on FPGAs)

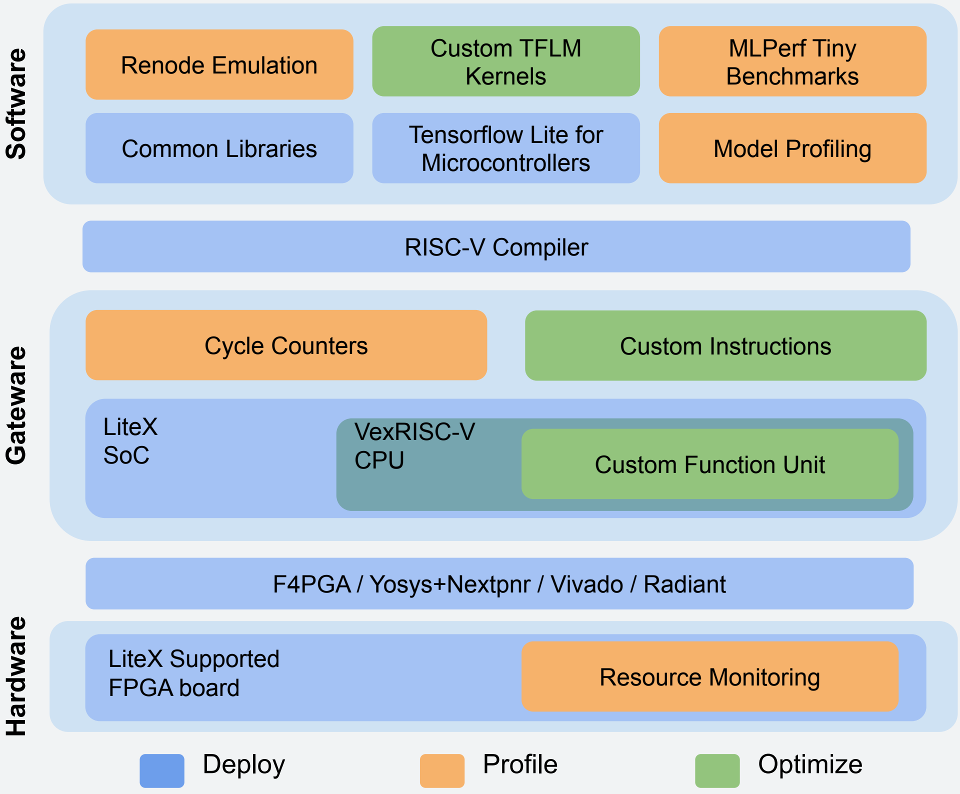
CFU Playground

- **ML library**
    - TensorFlow Lite -- *open source*
- **CPU ISA**
    - RISC-V -- *open source*
- **CPU design**
    - VexRiscv -- *open source*
- **FPGA SoC/IP**
    - LiteX -- *open source*
- **FPGA synth/PnR**
    - SymbiFlow,Yosys, -- *open source*
      Nextpnr, VPR -- *open source*

FPGA vendor tools can be used if you wish

- **Python HW gen**
    - Migen, nMigen -- *open source*
- **Simulation**
    - Renode, Verilator -- *open source*

The only proprietary component is the FPGA itself



Full-Stack Open-Source Framework

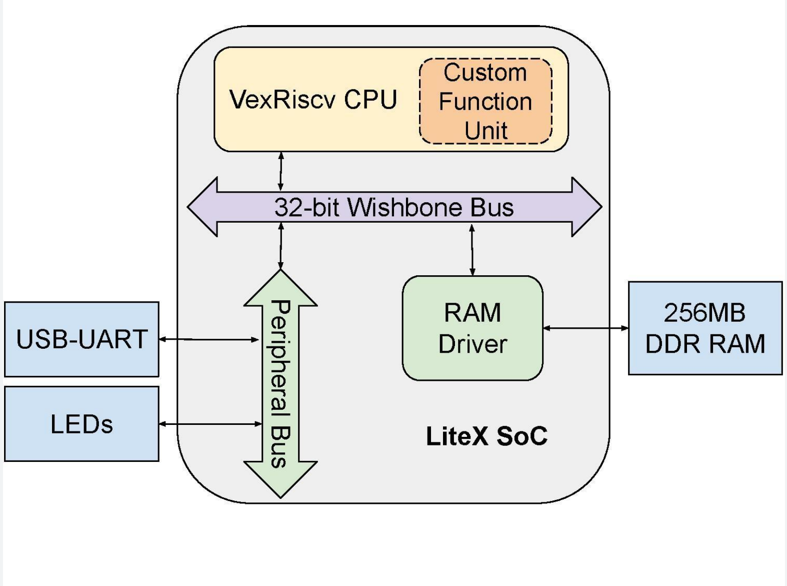
CFU Playground

- **ML library**
  - TensorFlow Lite -- *open source*
- **CPU ISA**
  - RISC-V -- *open source*
- **CPU design**
  - VexRiscv -- *open source*
- **FPGA SoC/IP**
  - LiteX -- *open source*
- **FPGA synth/PnR**
  - SymbiFlow,Yosys, -- *open source*
    Nextpnr, VPR -- *open source*

FPGA vendor tools can be used if you wish

- **Python HW gen**
  - Migen, nMigen -- *open source*
- **Simulation**
  - Renode, Verilator -- *open source*

The only proprietary component is the FPGA itself



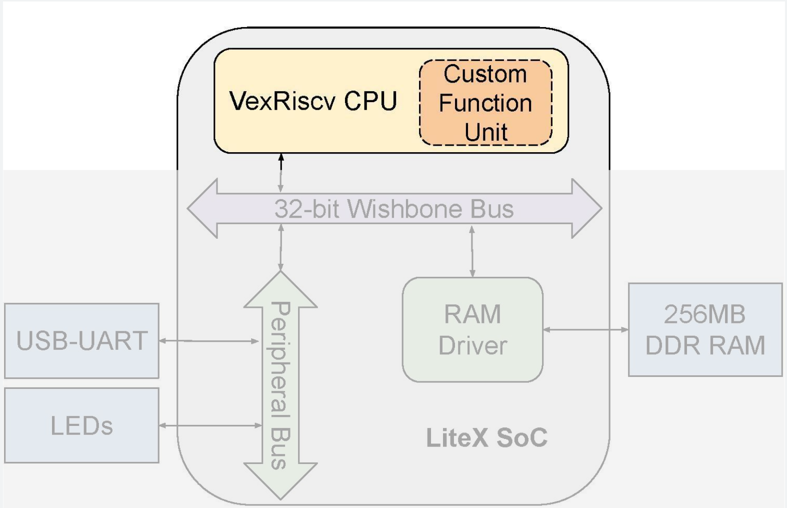
CFU Playground

- **ML library**
  - TensorFlow Lite -- *open source*
- **CPU ISA**
  - RISC-V -- *open source*
- **CPU design**
  - VexRiscv -- *open source*
- **FPGA SoC/IP**
  - LiteX -- *open source*
- **FPGA synth/PnR**
  - SymbiFlow,Yosys, -- *open source*
    Nextpnr, VPR -- *open source*

FPGA vendor tools can be used if you wish

- **Python HW gen**
  - Migen, nMigen -- *open source*
- **Simulation**
  - Renode, Verilator -- *open source*

The only proprietary component is the FPGA itself



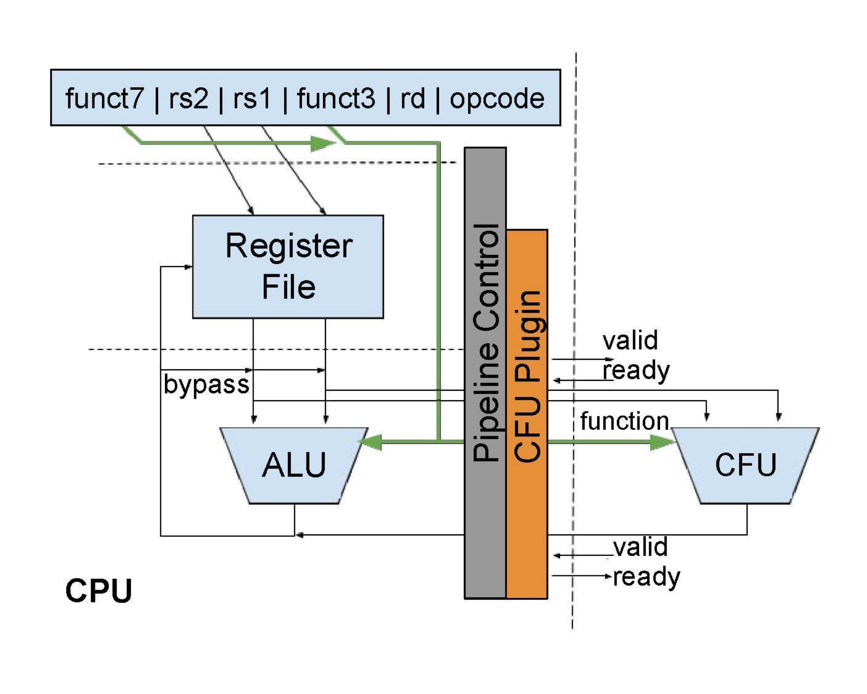
CFU Playground

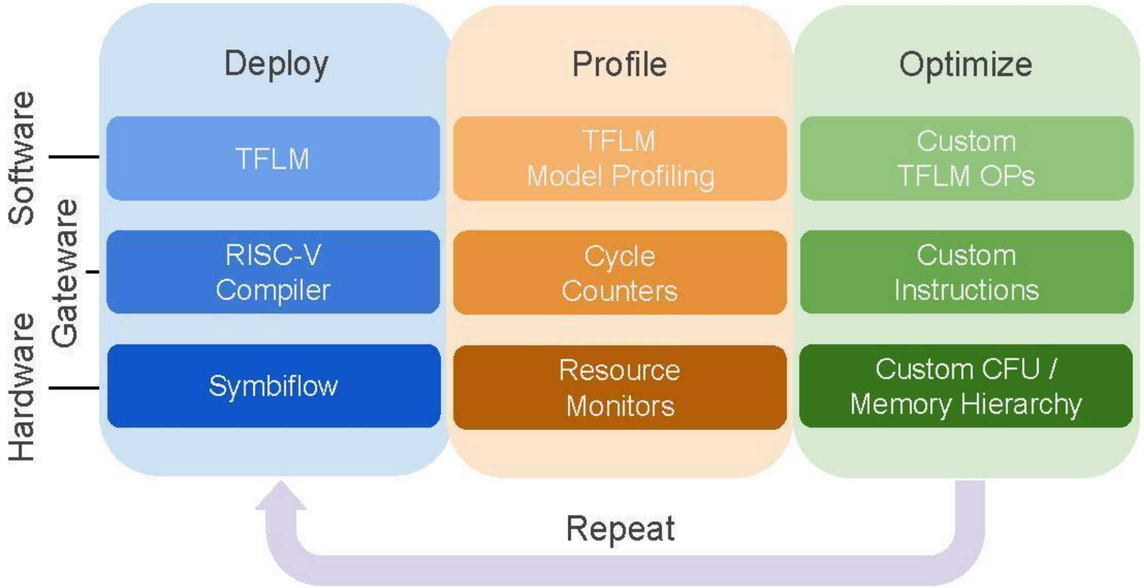
- **ML library**
  - TensorFlow Lite -- *open source*
- **CPU ISA**
  - RISC-V -- *open source*
- **CPU design**
  - VexRiscv -- *open source*
- **FPGA SoC/IP**
  - LiteX -- *open source*
- **FPGA synth/PnR**
  - SymbiFlow,Yosys, -- *open source*
    Nextpnr, VPR -- *open source*

FPGA vendor tools can be used if you wish

- **Python HW gen**
  - Migen, nMigen -- *open source*
- **Simulation**
  - Renode, Verilator -- *open source*

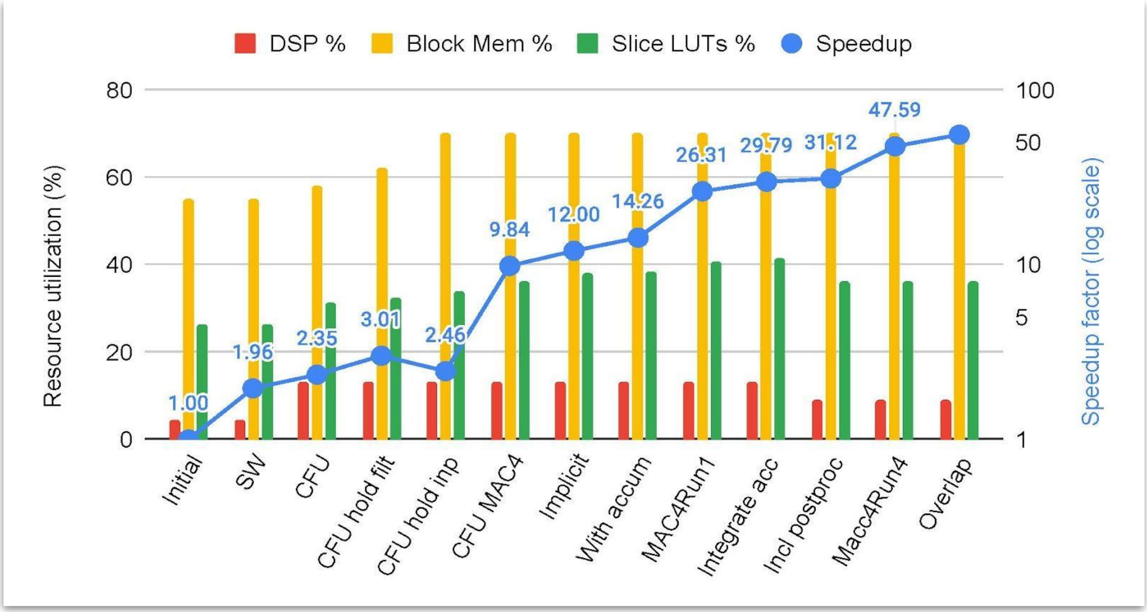
The only proprietary component is the FPGA itself





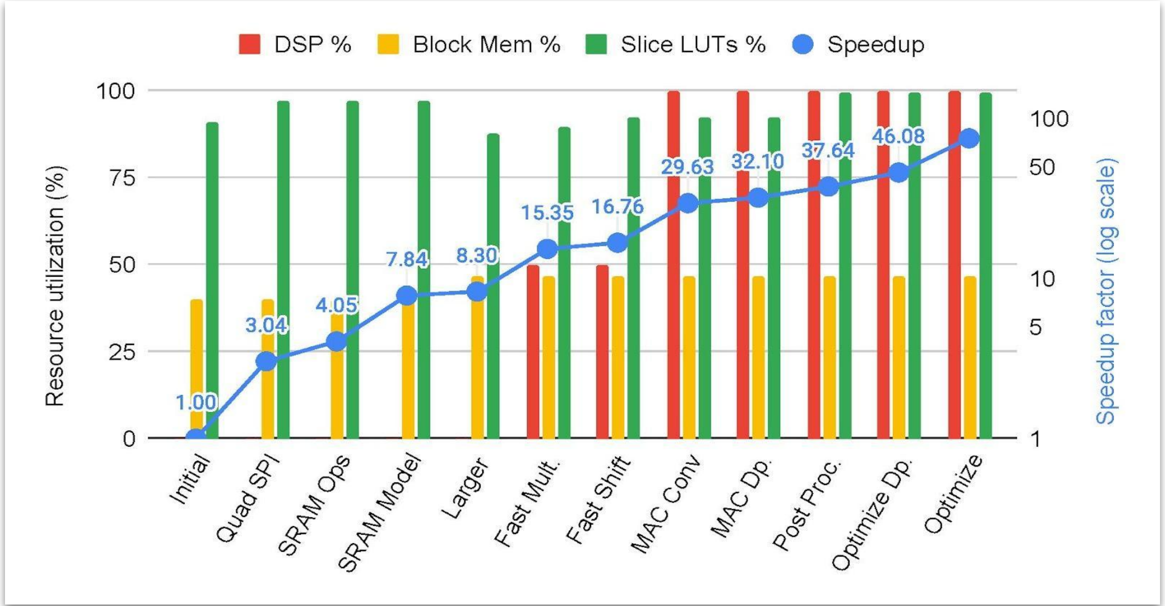
Agile Design Methodology

Image Classification on Arty

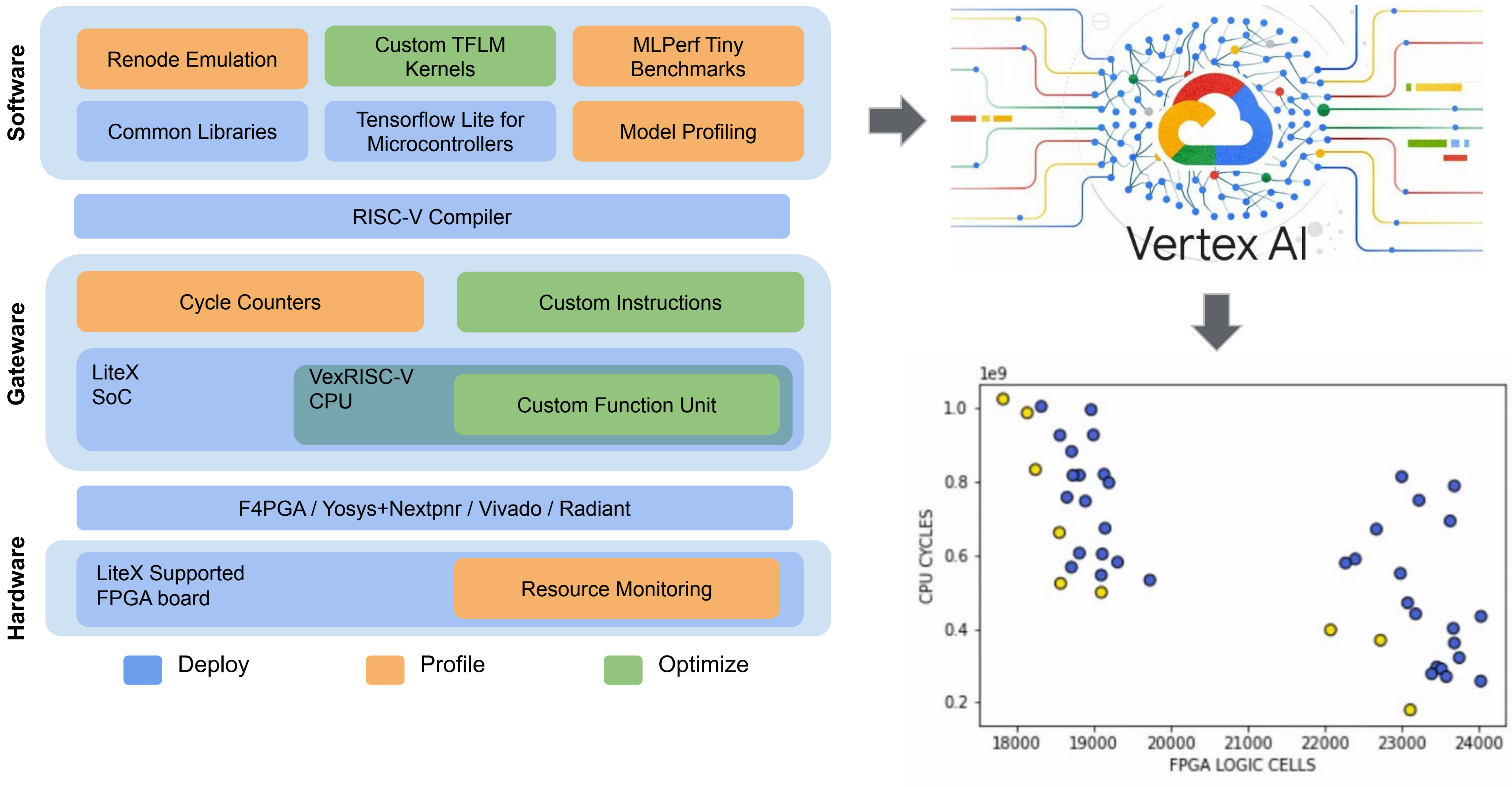


55x speedup in 5 weeks (part-time)

Keyword Spotting on FOMU

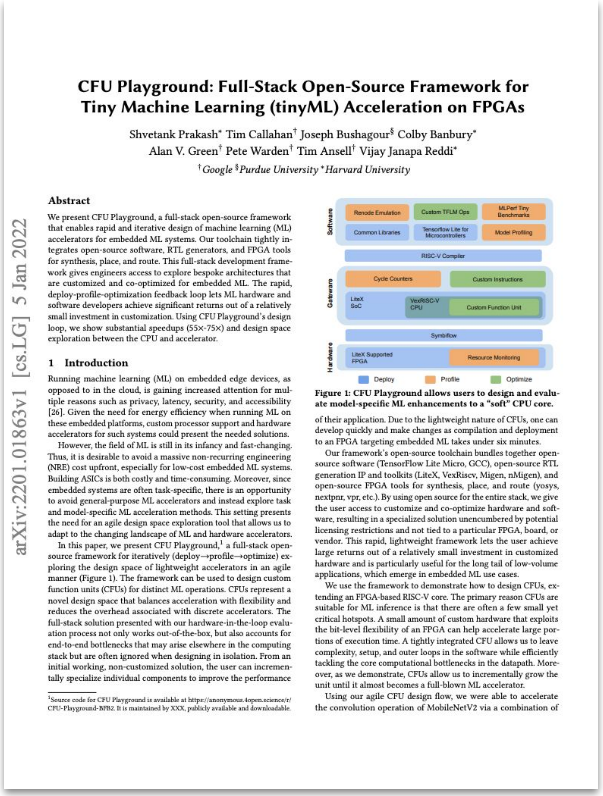


75x speedup in under 4 weeks (intern)


Software
Renode Emulation
Custom TFLM Kernels
MLPerf Tiny Benchmarks
Common Libraries
Tensorflow Lite for Microcontrollers
Model Profiling
RISC-V Compiler
Gateware
Cycle Counters
Custom Instructions
LiteX SoC
VexRISC-V CPU
Custom Function Unit
F4PGA / Yosys+Nextpnr / Vivado / Radiant
Hardware
LiteX Supported FPGA board
Resource Monitoring
Deploy
Profile
Optimize
Vertex AI
1e9
1.0
0.8
0.6
0.4
0.2
CPU CYCLES
18000
19000
20000
21000
22000
23000
24000
FPGA LOGIC CELLS

CFU Playground in a Nutshell

- An **out-of-the-box, full-stack framework that fully integrates open-source tools** across the entire stack to facilitate rich community-driven ecosystem development
- An **agile methodology for developers to progressively and iteratively design bespoke accelerators** for resource-constrained, latency-bound tinyML applications
- Through cross-stack insights, we **demonstrate novel model-specific resource allocation trade-offs between the CFU, CPU, and memory system** that enable optimal ML performance on resource-constrained FPGA platforms



arXiv:2201.01863v1 [cs.LG] 5 Jan 2022

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*
†Google §Purdue University *Harvard University

Abstract

We present CFU Playground, a full-stack open-source framework that enables rapid and iterative design of machine learning (ML) accelerators for embedded ML systems. Our toolchain tightly integrates open-source software, RTL generators, and FPGA tools for synthesis, place, and route. This full-stack development framework gives engineers access to explore bespoke architectures that are customized and co-optimized for embedded ML. The 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 loop, we show substantial speedups (55×-75×) and design space exploration between the CPU and accelerator.

1 Introduction

Running machine learning (ML) on embedded edge devices, as opposed to in the cloud, is gaining increased attention for multiple reasons such as privacy, latency, security, and accessibility [26]. 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 accelerators.

In this paper, we present CFU Playground,[1] 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 can be used to design custom function units (CFUs) for distinct ML operations. CFUs represent a novel design space that balances acceleration with flexibility and reduces the overhead associated with discrete accelerators. The full-stack solution presented with our hardware-in-the-loop evaluation process not only works out-of-the-box, but also accounts for end-to-end bottlenecks that may arise elsewhere in the computing stack but are often ignored when designing in isolation. From an initial working, non-customized solution, the user can incrementally specialize individual components to improve the performance

[1] Source code for CFU Playground is available at https://anonymous.4open.science/r/CFU-Playground-BFB2. It is maintained by XXX, publicly available and downloadable.

Figure 1: CFU Playground allows users to design and evaluate model-specific ML enhancements to a "soft" CPU core.

of their application. Due to the lightweight nature of CFUs, one can develop quickly and make changes as compilation and deployment to an FPGA targeting embedded ML takes under six minutes.

Our framework's open-source toolchain bundles together open-source software (TensorFlow Lite Micro, GCC), open-source RTL generation IP and toolkits (LiteX, VexRiscv, Migen, nMigen), and open-source FPGA tools for synthesis, place, and route (yosys, nextpnr, vpr, etc.). By using open source for the entire stack, we give the user access 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. This rapid, lightweight framework lets the user achieve large returns out of a relatively small investment in customized hardware and is particularly useful for the long tail of low-volume applications, which emerge in embedded ML use cases.

We use the framework to demonstrate how to design CFUs, extending an FPGA-based RISC-V core. The primary reason CFUs are suitable for ML inference is that there are often a few small yet critical hotspots. A small amount of custom hardware that exploits the bit-level flexibility of an FPGA can help accelerate large portions of execution time. A tightly integrated CFU allows us to leave complexity, setup, and outer loops in the software while efficiently tackling the core computational bottlenecks in the datapath. Moreover, as we demonstrate, CFUs allow us to incrementally grow the unit until it almost becomes a full-blown ML accelerator.

Using our agile CFU design flow, we were able to accelerate the convolution operation of MobileNetV2 via a combination of

A Greener Tomorrow with TinyML


1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

1 NO POVERTY

2 ZERO HUNGER

3 GOOD HEALTH AND WELL-BEING

4 QUALITY EDUCATION

5 GENDER EQUALITY

6 CLEAN WATER AND SANITATION

7 AFFORDABLE AND CLEAN ENERGY

8 DECENT WORK AND ECONOMIC GROWTH

9 INDUSTRY, INNOVATION AND INFRASTRUCTURE

10 REDUCED INEQUALITIES

11 SUSTAINABLE CITIES AND COMMUNITIES

12 RESPONSIBLE CONSUMPTION AND PRODUCTION

13 CLIMATE ACTION

14 LIFE BELOW WATER

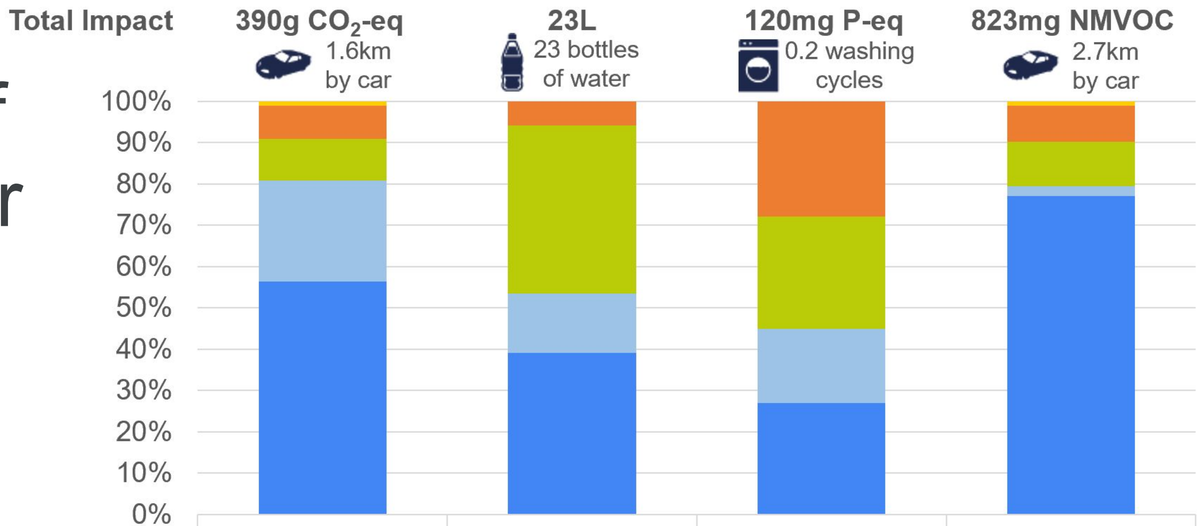
15 LIFE ON LAND

16 PEACE, JUSTICE AND STRONG INSTITUTIONS

17 PARTNERSHIPS FOR THE GOALS

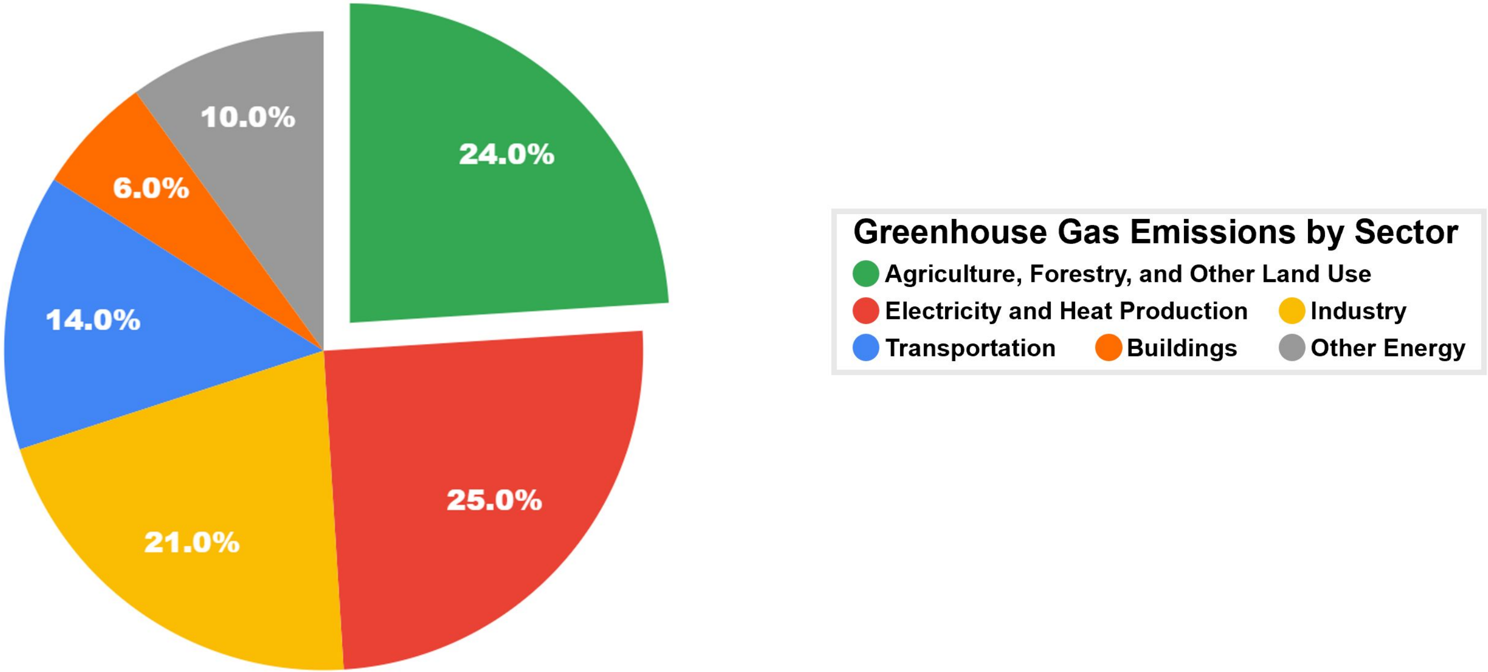
SUSTAINABLE DEVELOPMENT GOALS

Tiny Footprint of a Microcontroller

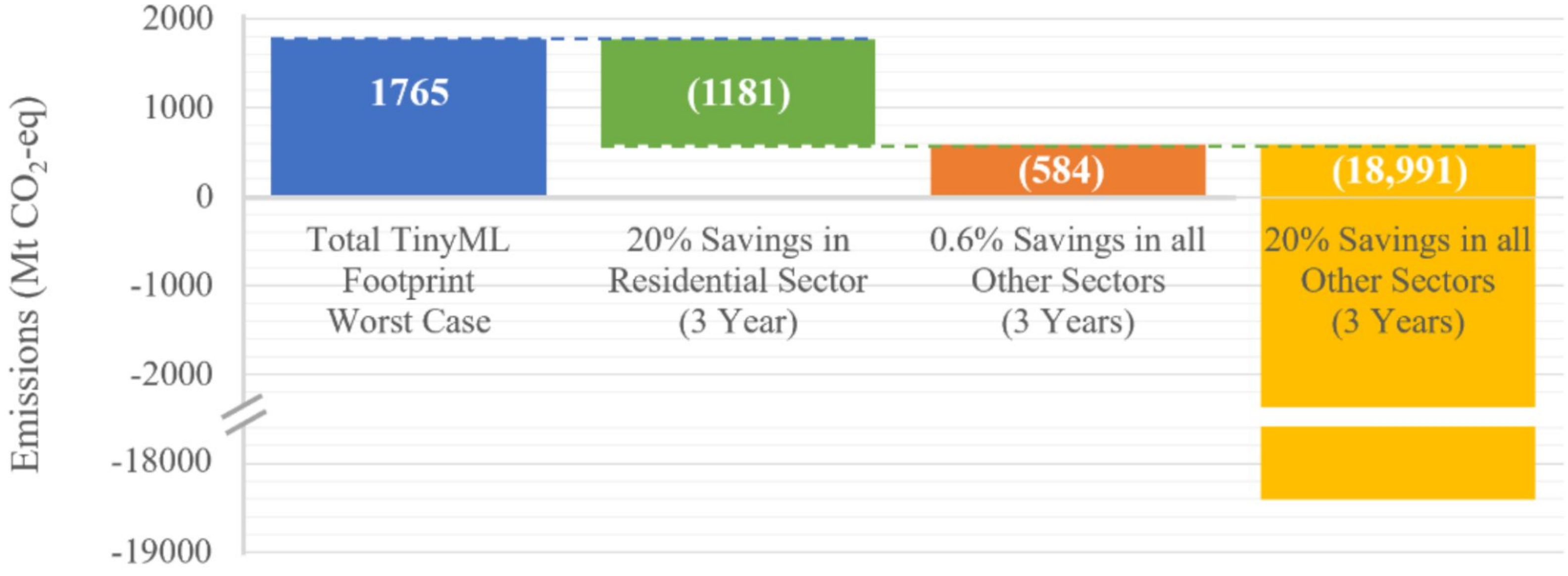


	Climate Change	Water Demand	Freshwater Eutrophication	Protochemical Oxidant Formation
End of Life	<1%	<1%	<1%	<1%
Logistics	1%	<1%	<1%	1%
Use	8%	6%	28%	8%
Raw Materials	10%	41%	27%	10%
Production: Other	24%	15%	18%	2%
Production: Energy Consumption	56%	39%	27%	71%

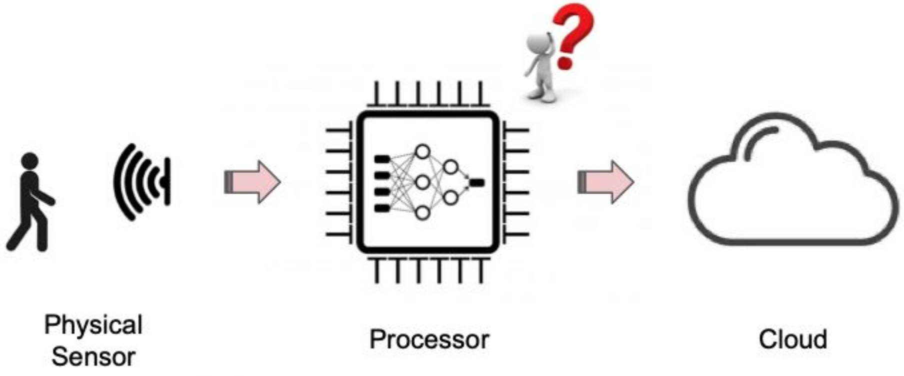
Global CO_2 Emissions by Sectors



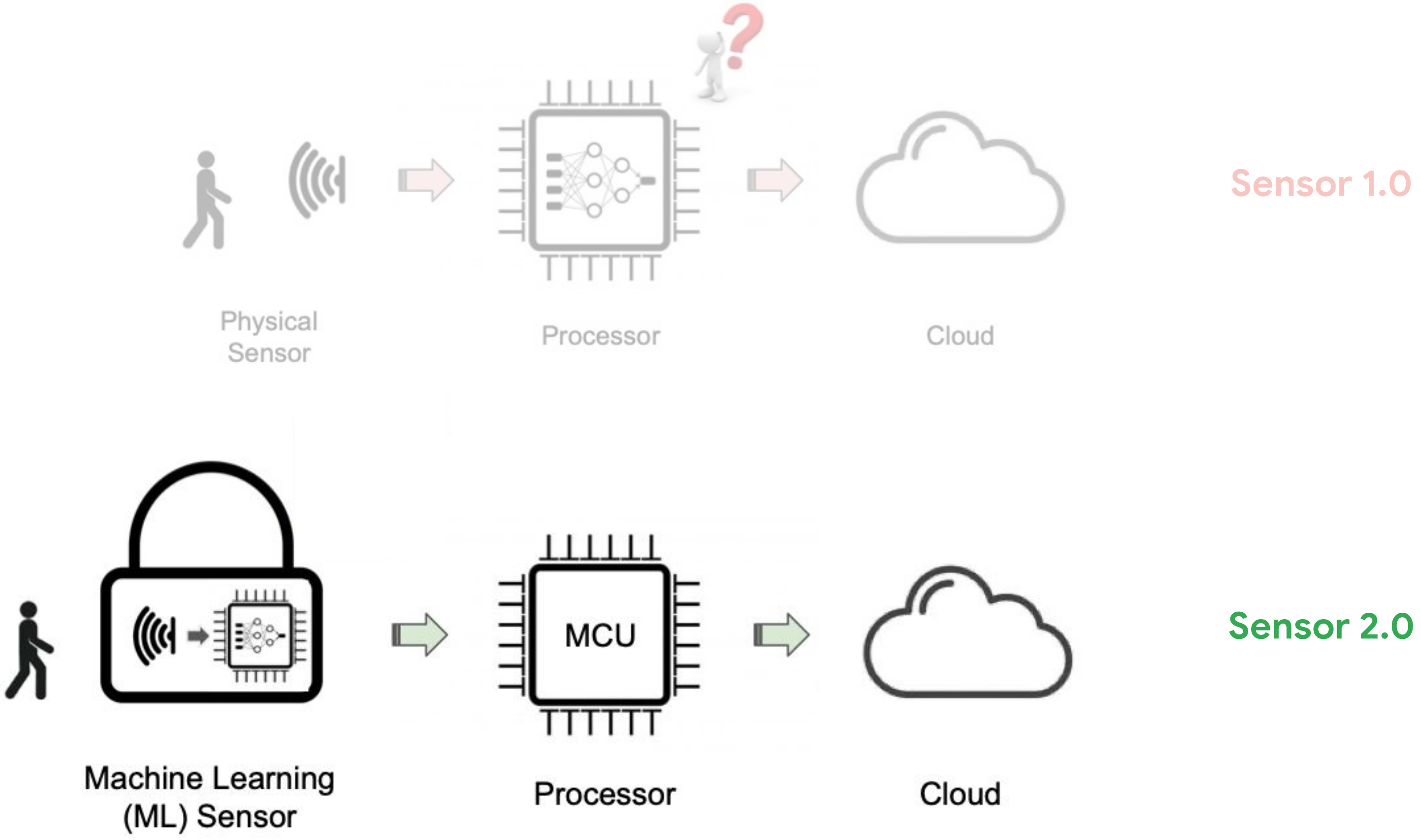
TinyML System - Net Environmental Impact

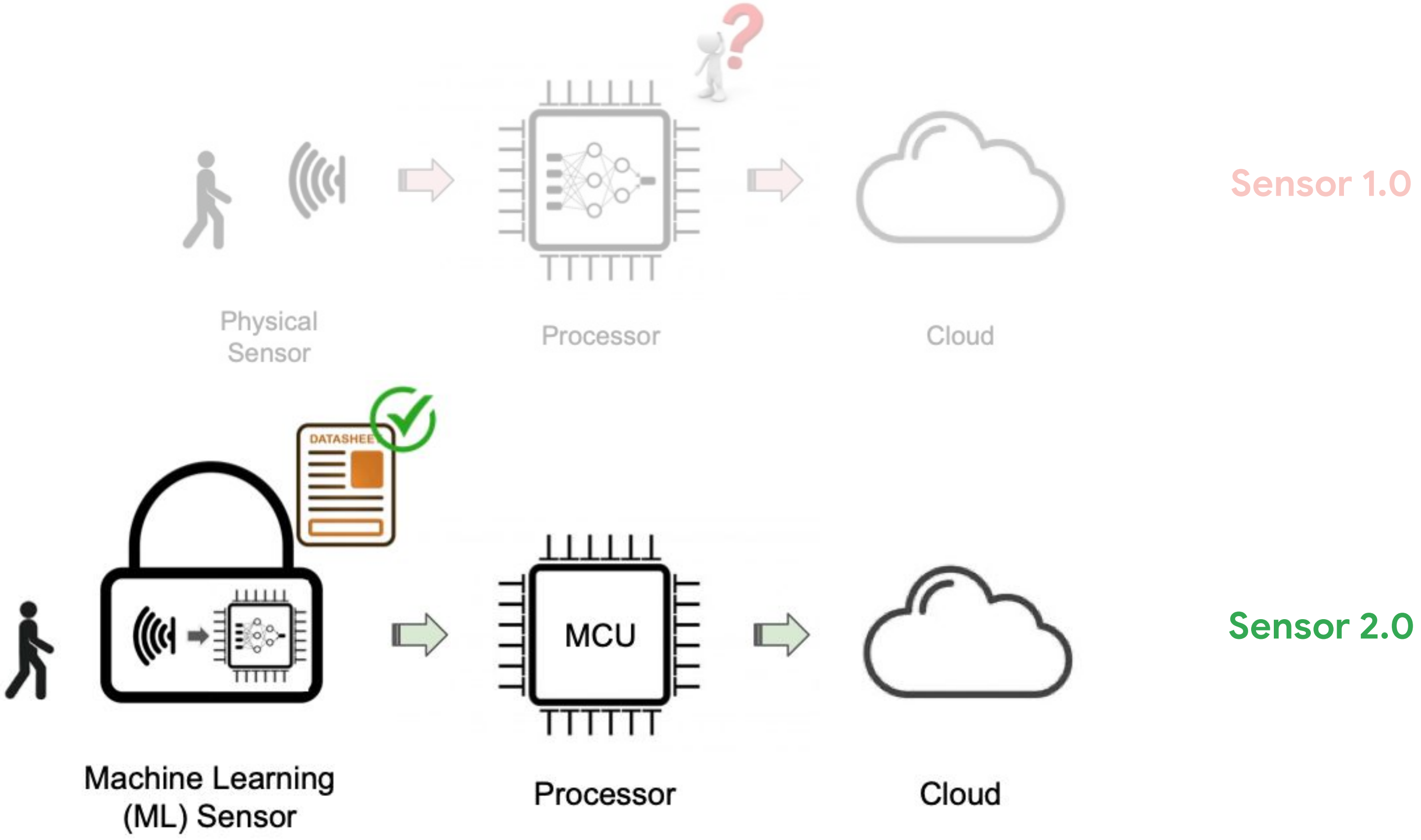


ML Sensors



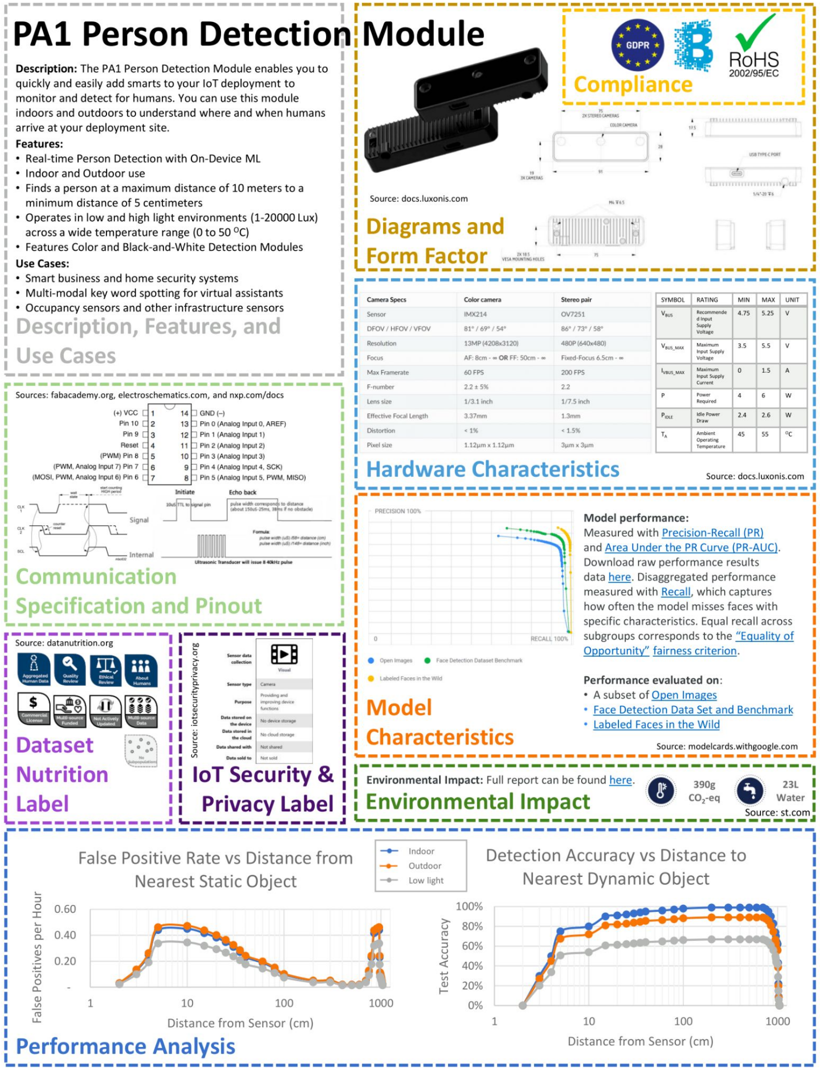
Sensor 1.0


Physical Sensor
Processor
Cloud
Sensor 1.0
MCU
Machine Learning (ML) Sensor
Processor
Cloud
Sensor 2.0


Physical Sensor
Processor
Cloud
Sensor 1.0
DATASHEET
Machine Learning (ML) Sensor
MCU
Processor
Cloud
Sensor 2.0

Datasheets for ML Sensors

ML sensors must be transparent, indicating in a publicly and freely accessible ML sensor datasheet all the relevant information such as fact sheets, model cards, and dataset nutrition labels to supplement the traditional EE hardware information typically available for sensors.



Machine Learning Sensors - ML

mlsensors.org

TinyML | Harvard | MLC | Research | Seeed | CS141 | TimeBuddy | VJs Funding | Enterprise - Suppl... | Geo Chart Exampl... | Other Bookmarks

MLSensors

Machine Learning Sensors

Home | Whitepaper | GitHub | Email | Team

Machine Learning Sensors

An ML sensor is a self-contained system that utilizes on-device machine learning to extract useful information by observing some complex set of phenomena in the physical world and reports it through a simple interface to a wider system.

Machine learning sensors represent a paradigm shift for the future of embedded machine learning applications. Current instantiations of embedded ML suffer from complex integration, lack of modularity, and privacy and security concerns from data

Challenges

Interface

What universal interface is needed for ML Sensors?

Standards

What standards need to be in place for ML Sensors?

Ethics

What ethical considerations are needed for ML Sensors?

Call for Working Group Members

We are actively growing our working group. If you would like to be a part of it please email us at: ml-sensors@googlegroups.com!

Example ML Sensor Datasheet

This illustrative example datasheet highlighting the various sections of an ML Sensor datasheet. On the top, we have the items currently found in standard datasheets: the description, features, use cases, diagrams and form factor, hardware characteristics, and communication specification and pinout. On the bottom, we have the new items that need to be included in an ML sensor datasheet: the ML model characteristics, dataset nutrition label, environmental impact analysis, and end-to-end performance analysis. While we compressed this datasheet into a one-page illustrative example by combining features and data from a mixture of sources, on a real datasheet, we assume each of these sections would be longer and include additional explanatory text to increase the transparency of the device to end-users. Interested users can find the most up-to-date version of the datasheet online at https://github.com/harvard-edge/ML-Sensors.

Machine Learning Sensors

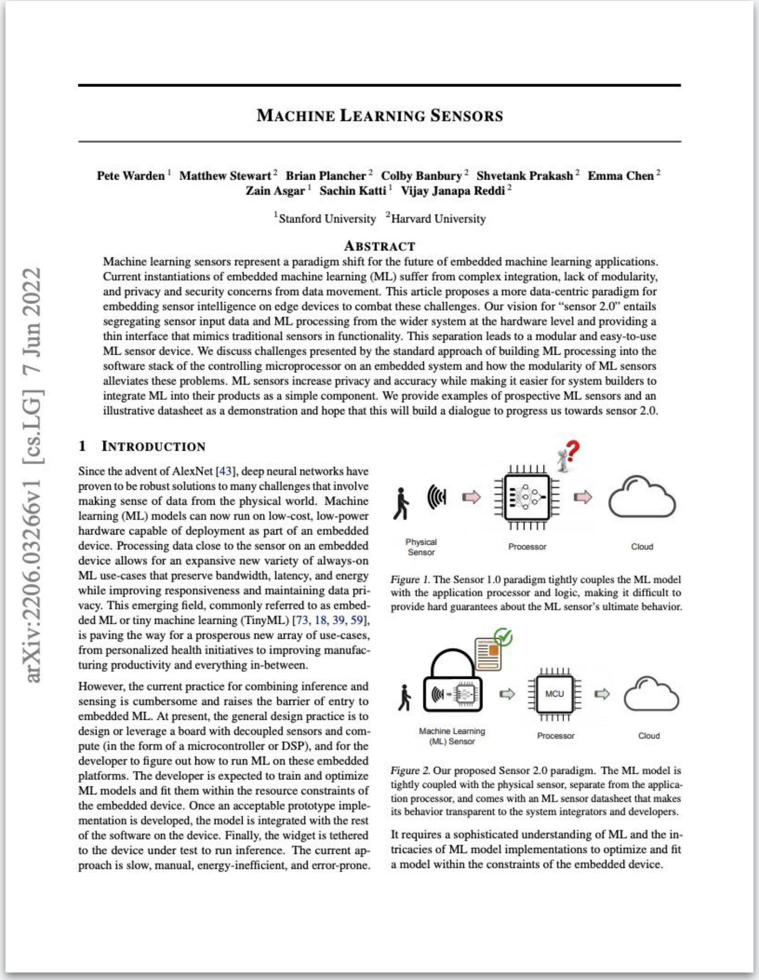
1. We need to **raise the level of abstraction** to enable ease of use for scalable deployment of ML sensors; not everyone should be required to be an systems developer or an engineer to use or leverage ML sensors into their ecosystem.

2. The ML sensor's **design should be inherently data-centric** and defined by its input-output behavior instead of exposing the underlying hardware and software mechanisms that support ML model execution.

3. An ML sensor's **implementation must be clean and complexity-free**. Features such as reusability, software updates, and networking must be thought through to ensure data privacy and secure execution.

4. ML sensors **must be transparent, indicating in a publicly and freely accessible ML sensor datasheet** all the relevant information such as fact sheets, model cards, and dataset nutrition labels to supplement the traditional information available for hardware sensors.

5. We as a community should aim to **foster an open ML sensors ecosystem by maximizing data, model, and hardware transparency** where possible, without necessarily relinquishing any claim to intellectual property.



arXiv:2206.03266v1 [cs.LG] 7 Jun 2022

MACHINE LEARNING SENSORS

Pete Warden [1] Matthew Stewart [2] Brian Plancher [2] Colby Banbury [2] Shvetank Prakash [2] Emma Chen [2] Zain Asgar [1] Sachin Katti [1] Vijay Janapa Reddi [2]

[1]Stanford University [2]Harvard University

ABSTRACT

Machine learning sensors represent a paradigm shift for the future of embedded machine learning applications. Current instantiations of embedded machine learning (ML) suffer from complex integration, lack of modularity, and privacy and security concerns from data movement. This article proposes a more data-centric paradigm for embedding sensor intelligence on edge devices to combat these challenges. Our vision for "sensor 2.0" entails segregating sensor input data and ML processing from the wider system at the hardware level and providing a thin interface that mimics traditional sensors in functionality. This separation leads to a modular and easy-to-use ML sensor device. We discuss challenges presented by the standard approach of building ML processing into the software stack of the controlling microprocessor on an embedded system and how the modularity of ML sensors alleviates these problems. ML sensors increase privacy and accuracy while making it easier for system builders to integrate ML into their products as a simple component. We provide examples of prospective ML sensors and an illustrative datasheet as a demonstration and hope that this will build a dialogue to progress us towards sensor 2.0.

1 INTRODUCTION

Since the advent of AlexNet [43], deep neural networks have proven to be robust solutions to many challenges that involve making sense of data from the physical world. Machine learning (ML) models can now run on low-cost, low-power hardware capable of deployment as part of an embedded device. Processing data close to the sensor on an embedded device allows for an expansive new variety of always-on ML use-cases that preserve bandwidth, latency, and energy while improving responsiveness and maintaining data privacy. This emerging field, commonly referred to as embedded ML or tiny machine learning (TinyML) [73, 18, 39, 59], is paving the way for a prosperous new array of use-cases, from personalized health initiatives to improving manufacturing productivity and everything in-between.

However, the current practice for combining inference and sensing is cumbersome and raises the barrier of entry to embedded ML. At present, the general design practice is to design or leverage a board with decoupled sensors and compute (in the form of a microcontroller or DSP), and for the developer to figure out how to run ML on these embedded platforms. The developer is expected to train and optimize ML models and fit them within the resource constraints of the embedded device. Once an acceptable prototype implementation is developed, the model is integrated with the rest of the software on the device. Finally, the widget is tethered to the device under test to run inference. The current approach is slow, manual, energy-inefficient, and error-prone.

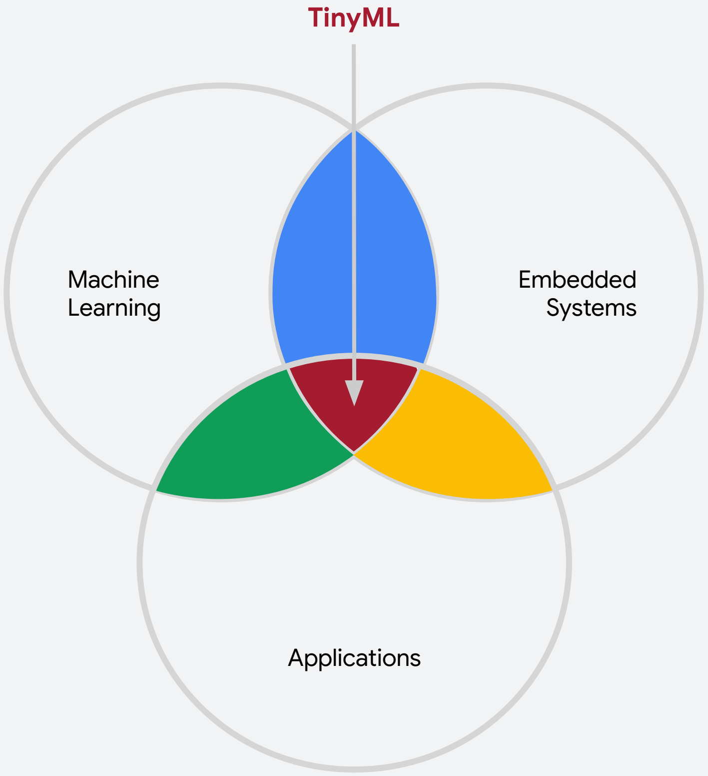
Figure 1. The Sensor 1.0 paradigm tightly couples the ML model with the application processor and logic, making it difficult to provide hard guarantees about the ML sensor's ultimate behavior.

Figure 2. Our proposed Sensor 2.0 paradigm. The ML model is tightly coupled with the physical sensor, separate from the application processor, and comes with an ML sensor datasheet that makes its behavior transparent to the system integrators and developers.

It requires a sophisticated understanding of ML and the intricacies of ML model implementations to optimize and fit a model within the constraints of the embedded device.

Conclusion

1. TinyML has the **potential to dramatically change our future**
2. No free lunch – hardware and software **fragmentation is a serious challenge** to address
3. TinyML **sustainability is crucial** to ensure its broad applicability
4. ML sensors based on TinyML technology must be **transparent**
5. Widening access to applied ML is a must to ensure **equitable access**



The future of ML is tiny and bright,
and its benefits can translate to societal impact.

Conclusion

The Future of ML is Tiny and Bright