

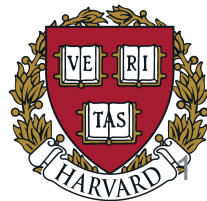
Tiny Machine Learning (tinyML) for Robotics



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>



Conference on Robot Learning, 2021



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

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

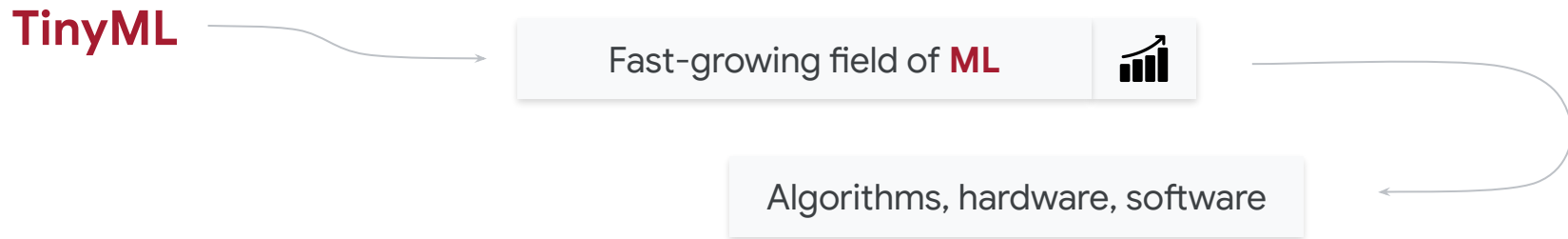
TinyML



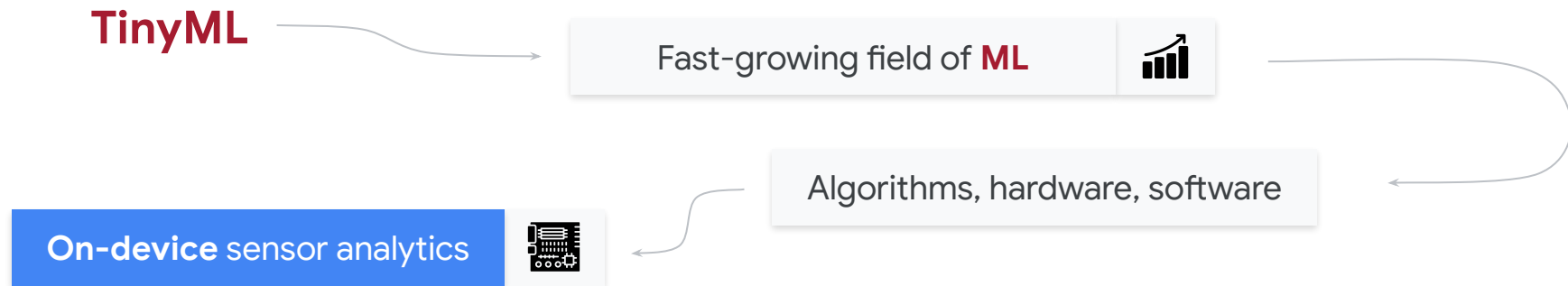
Fast-growing field of **ML**



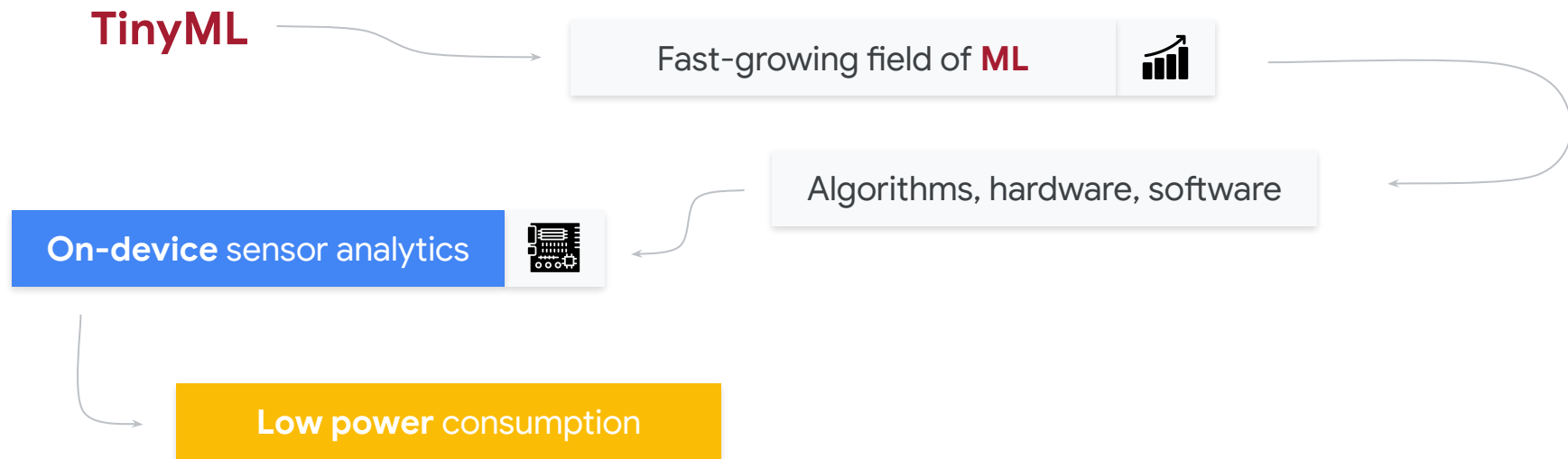
What is Tiny Machine Learning (**TinyML**)?



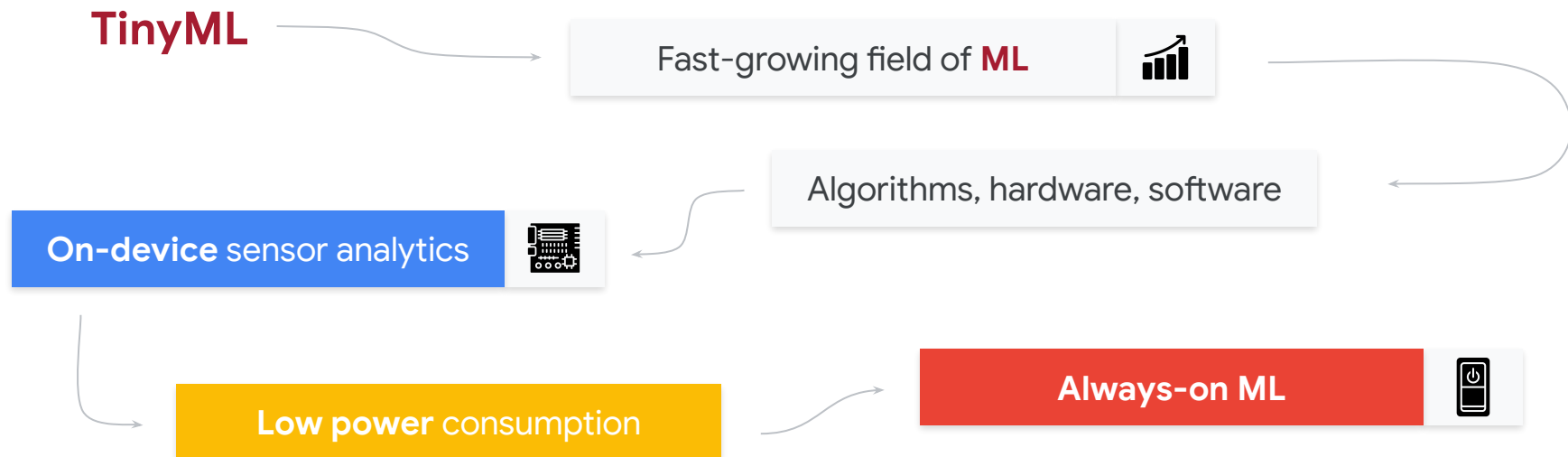
What is Tiny Machine Learning (**TinyML**)?



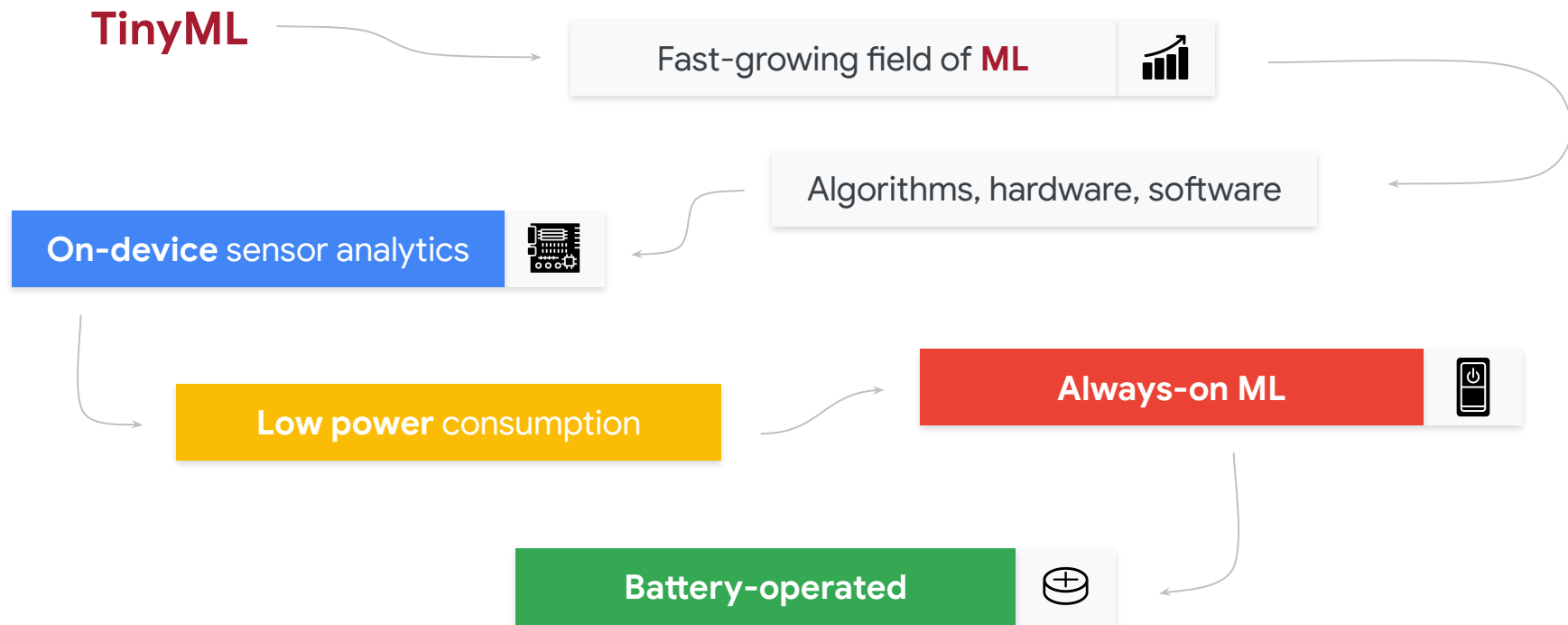
What is Tiny Machine Learning (**TinyML**)?



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



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

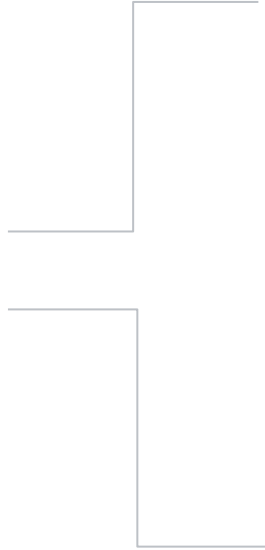




Endpoint Devices



Google Assistant





Robots Have **Sensors**, Tons 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.

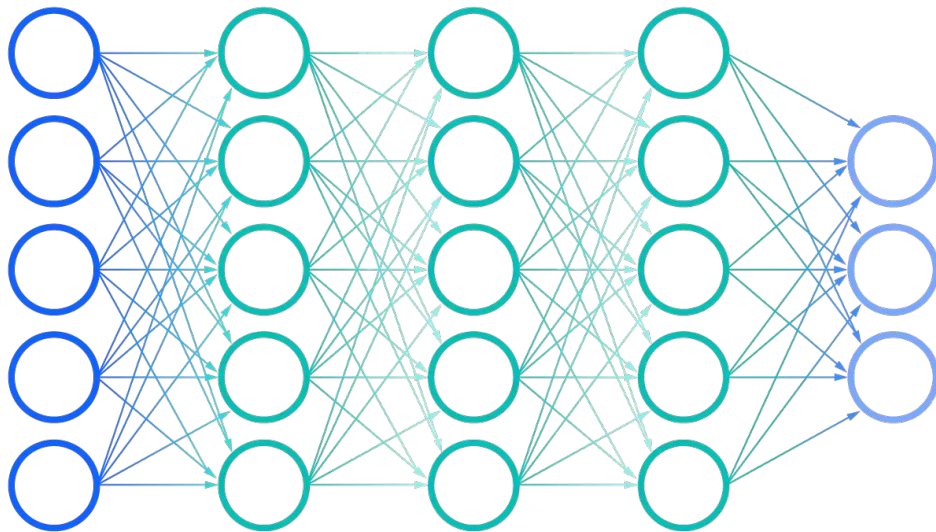
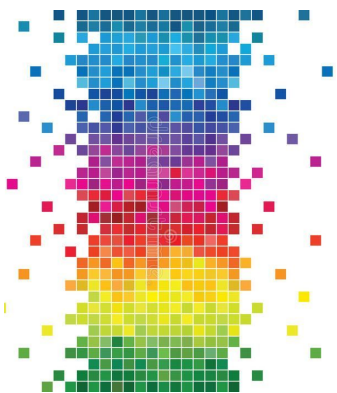
Force Sensors

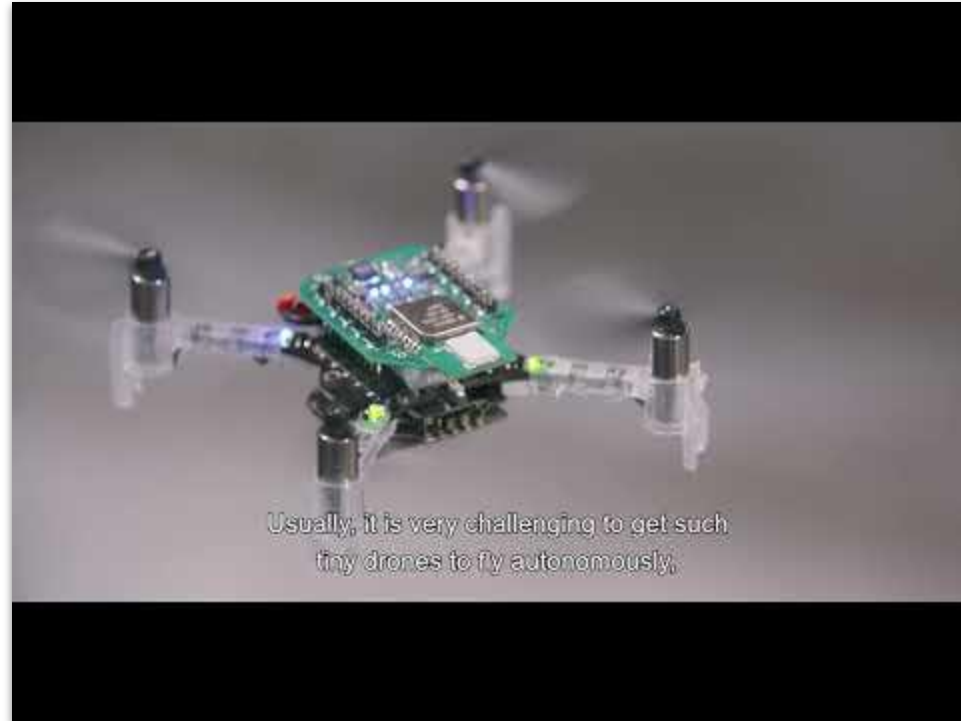
Pressure, Strain

Rotation Sensors

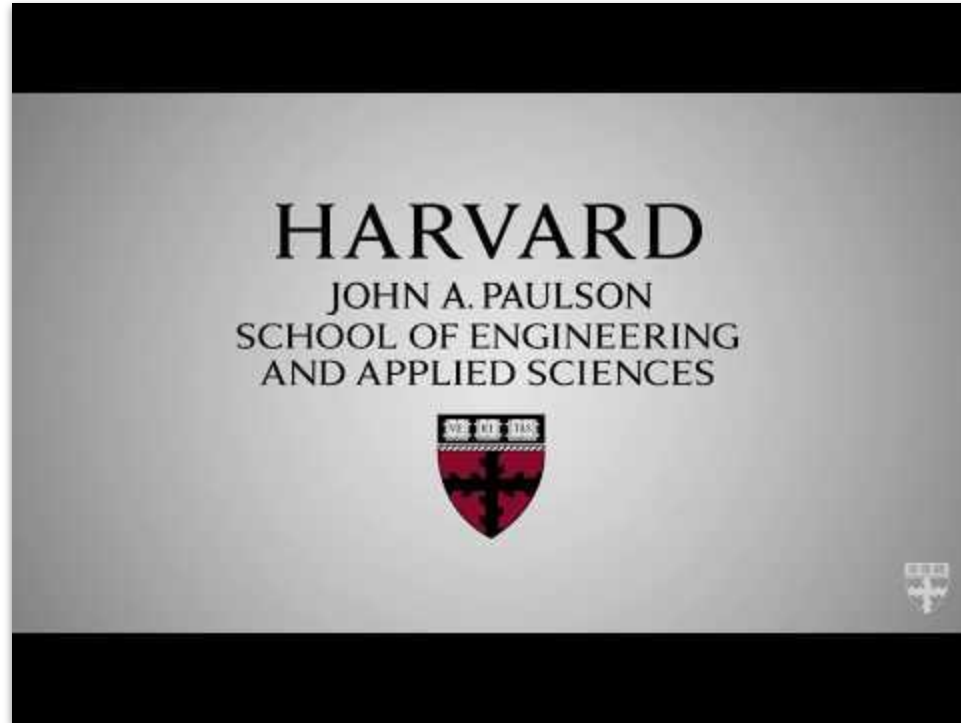
Encoders

...





Duisterhof, B.P., Li, S., Burgués, J., Reddi, V.J. and de Croon, G.C., 2021. Sniffy Bug: A Fully Autonomous Swarm of Gas-Seeking Nano Quadcopters in Cluttered Environments. arXiv preprint arXiv:2107.05490.



Goldberg, B., Zufferey, R., Doshi, N., Helbling, E.F., Whittredge, G., Kovac, M. and Wood, R.J., 2018. Power and control autonomy for high-speed locomotion with an insect-scale legged robot. IEEE Robotics and Automation Letters, 3(2), pp.987-993.

No Good Data Left Behind

5 Quintillion

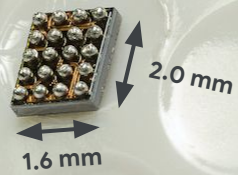
bytes of data produced
every day by IoT

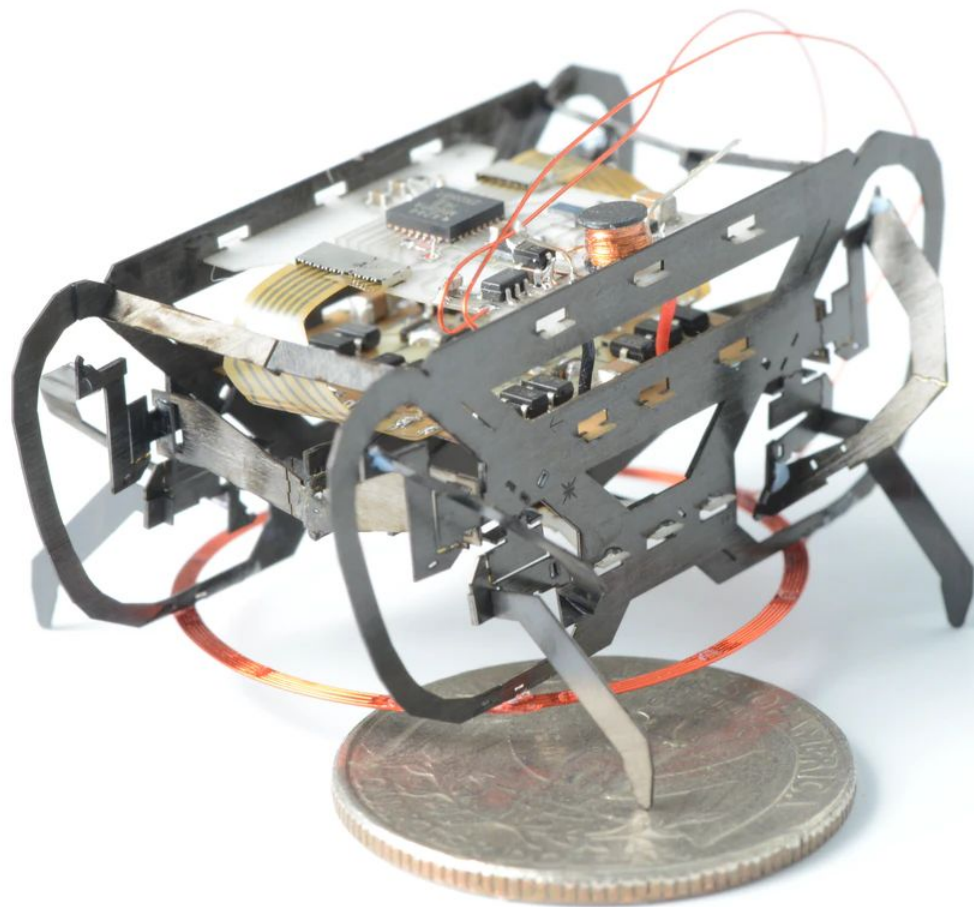
<1%

of unstructured data is
analyzed or used at all



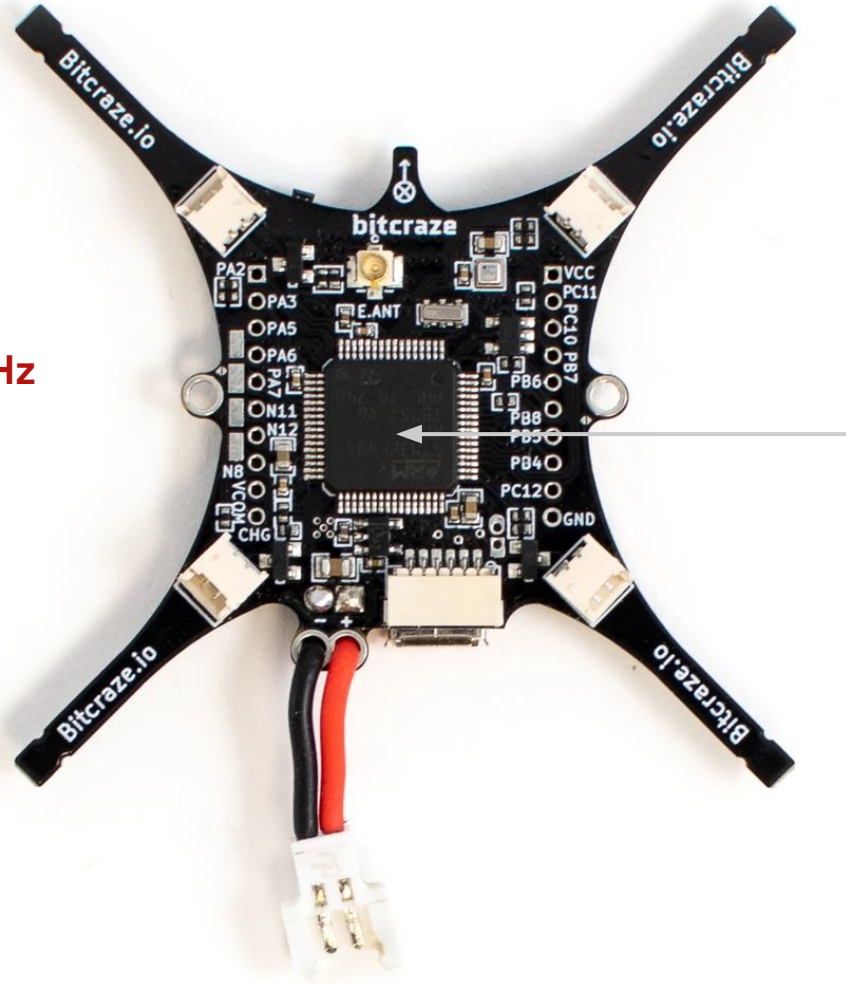
The Future of Machine Learning is Tiny... and Bright





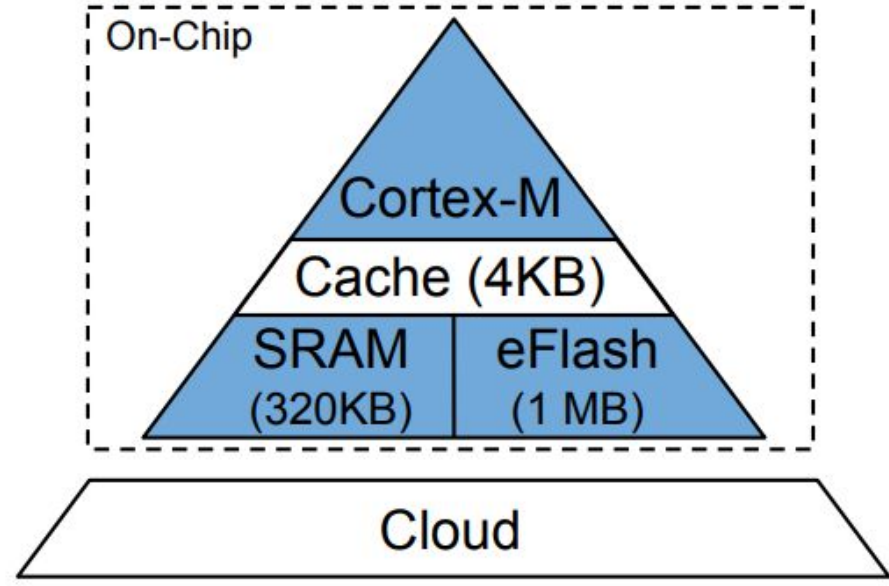
BitCraze CrazyFlie 2.1

- ARM Cortex-M4
- CPU: 1-core & **168 MHz**
- RAM: **196 kB**
- Storage: **1 MB**
- Available RAM: 33 kB
- Weight: 33 grams



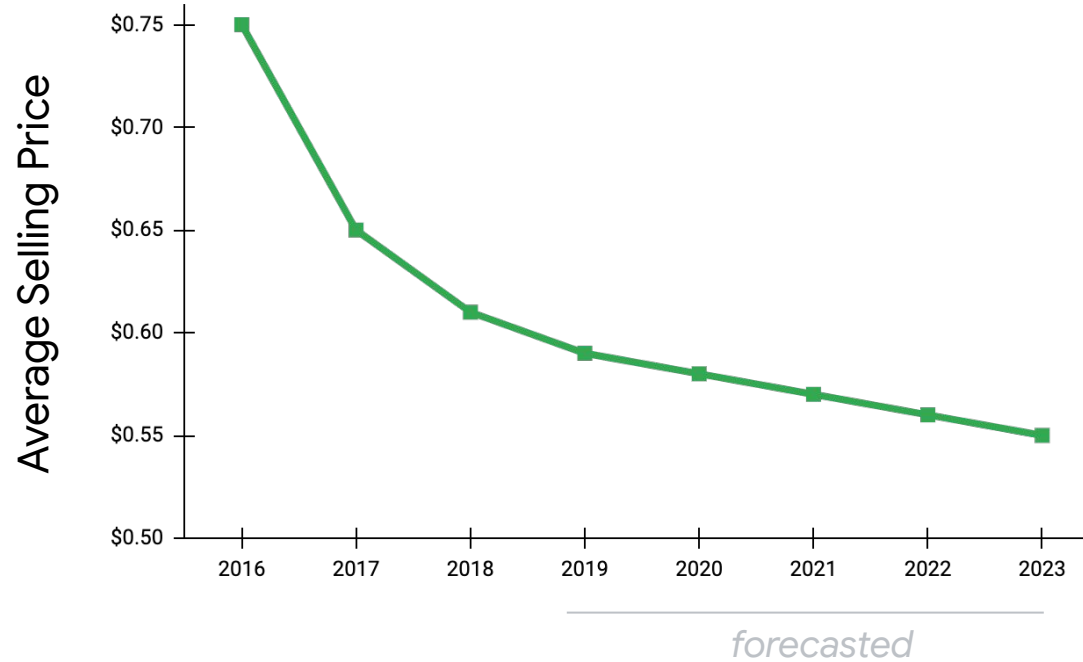
**Tiny
SoC**

250 Billion
MCUs today



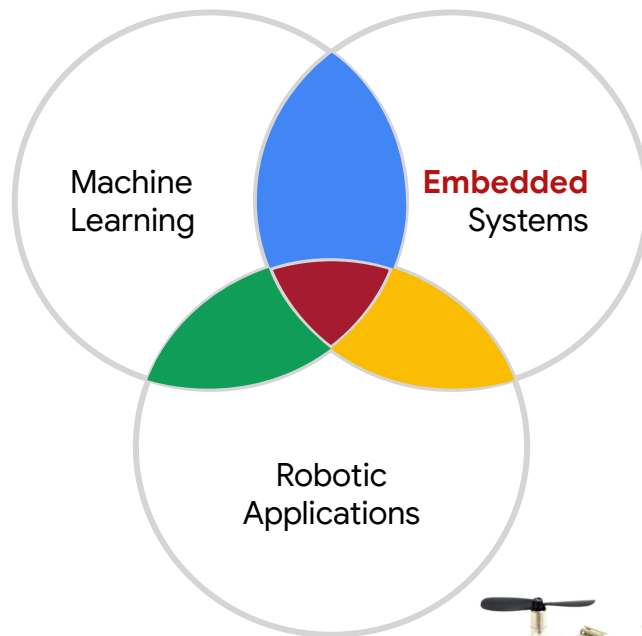
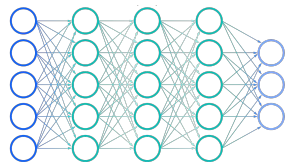
**Tiny
SoC**

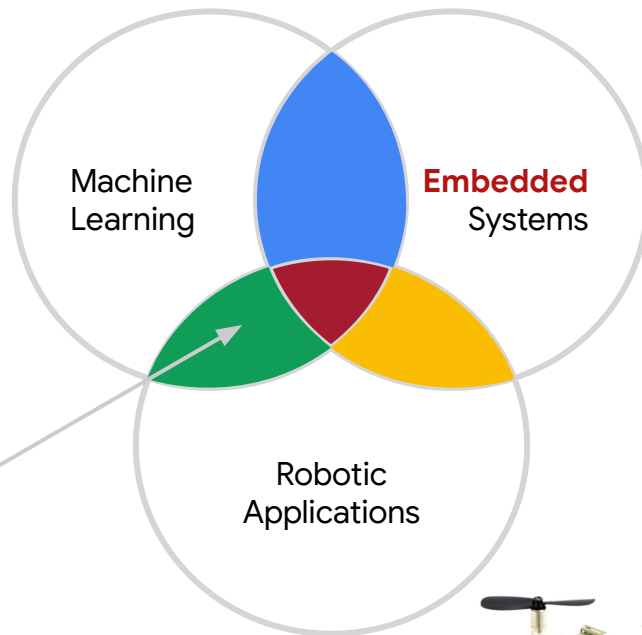
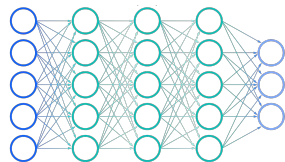
MCU Pricing Forecast



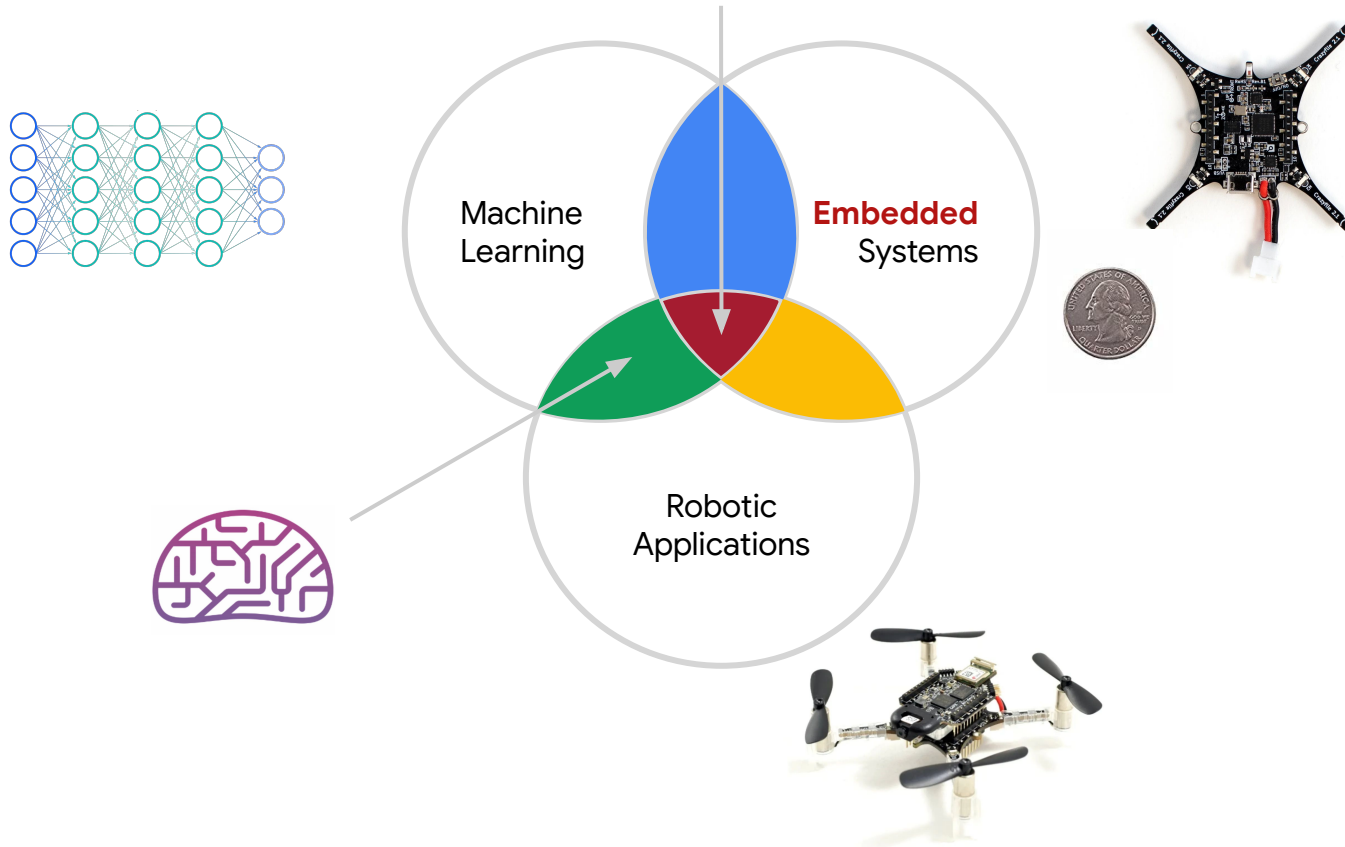


- Tiny machine learning (**tinyML**): ML applications on **low-power, cheap, commodity hardware**.
- Focus on **always-on machine learning use cases for robotics** with rich sensory input.





TinyML for Robotics





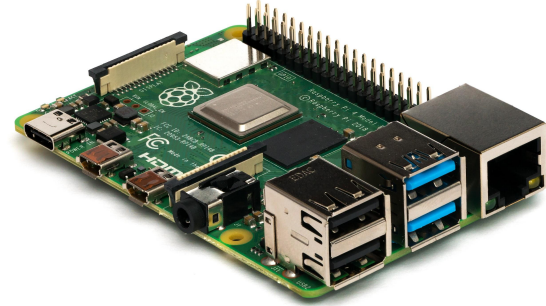
ML Training & Evaluation



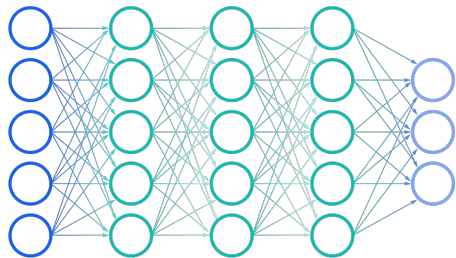
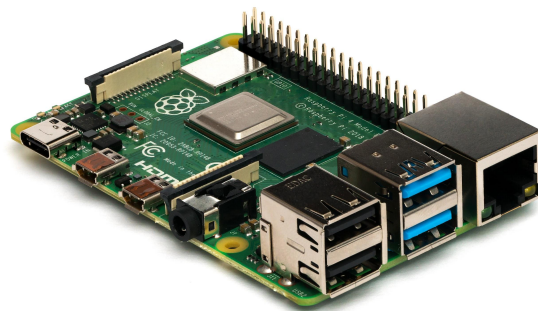
ML Deployment



ML Deployment

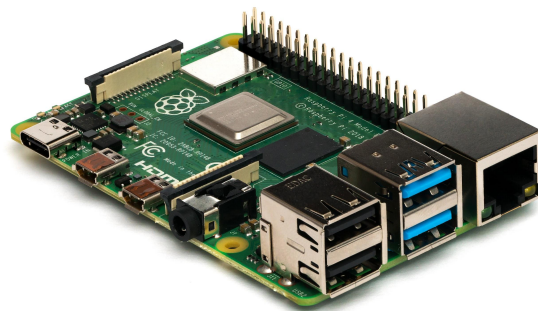


ML Deployment

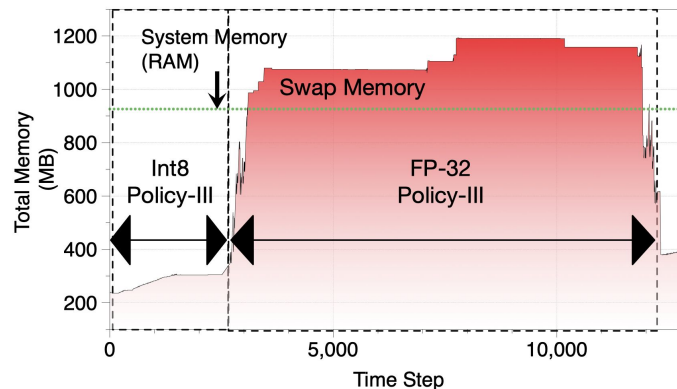


Name	Parameters	fp32 (ms)	fp32 (success)	int8 (ms)	int8 (success)	Delta
Policy III	3L, MLP (4096, 512, 1024)	208 ms	86%	11 ms	75%	19x

ML Deployment

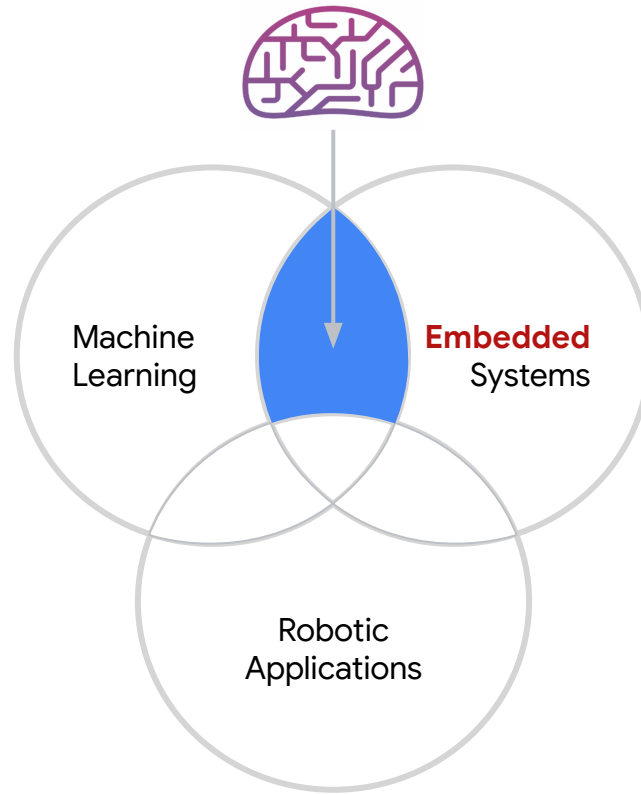


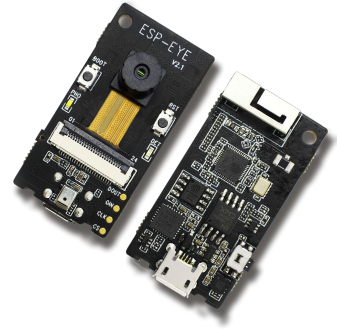
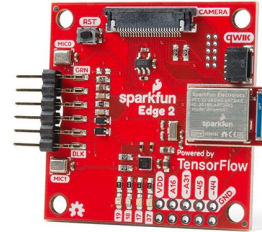
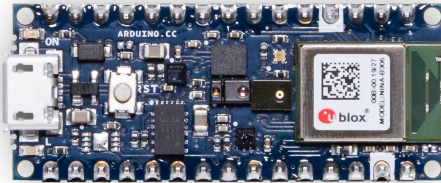
Name	Parameters	fp32 (ms)	fp32 (success)	int8 (ms)	int8 (success)	Delta
Policy III	3L, MLP (4096, 512, 1024)	208 ms	86%	11 ms	75%	19x



TinyML for Robotics \Rightarrow End-to-end ML Workflow

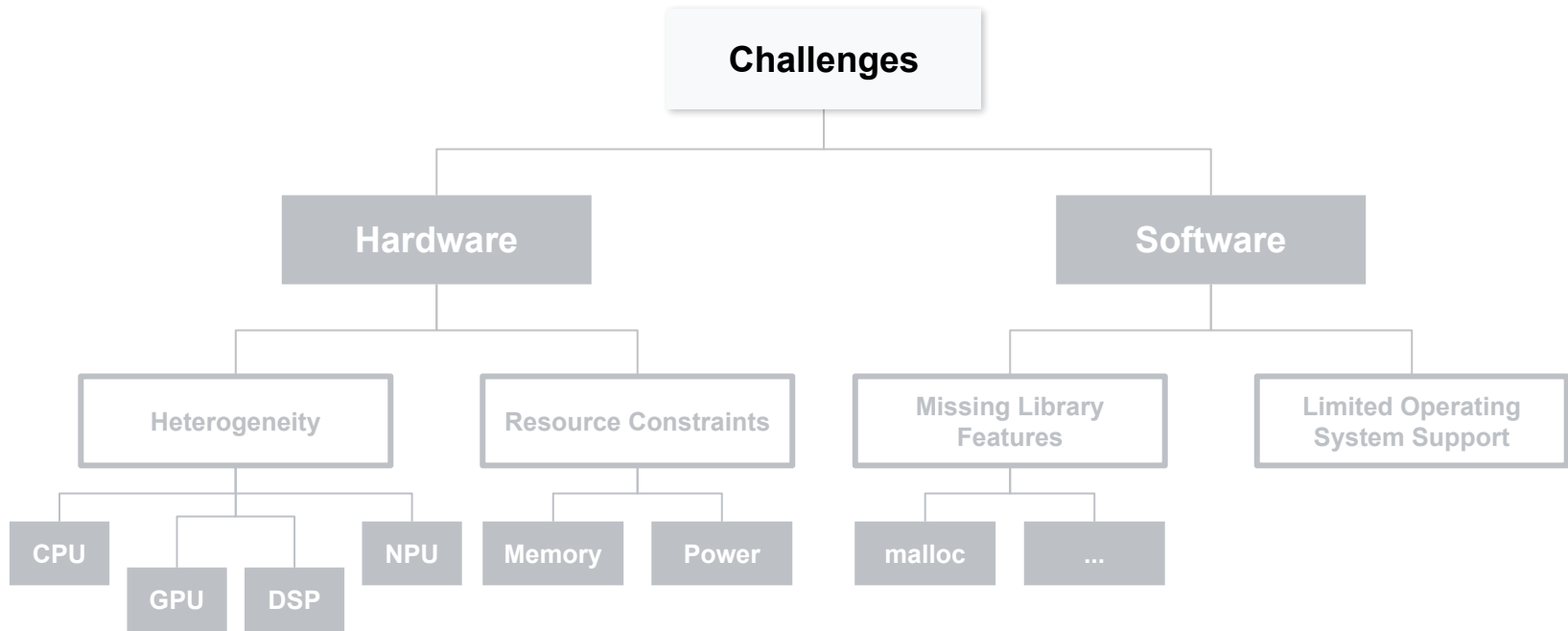


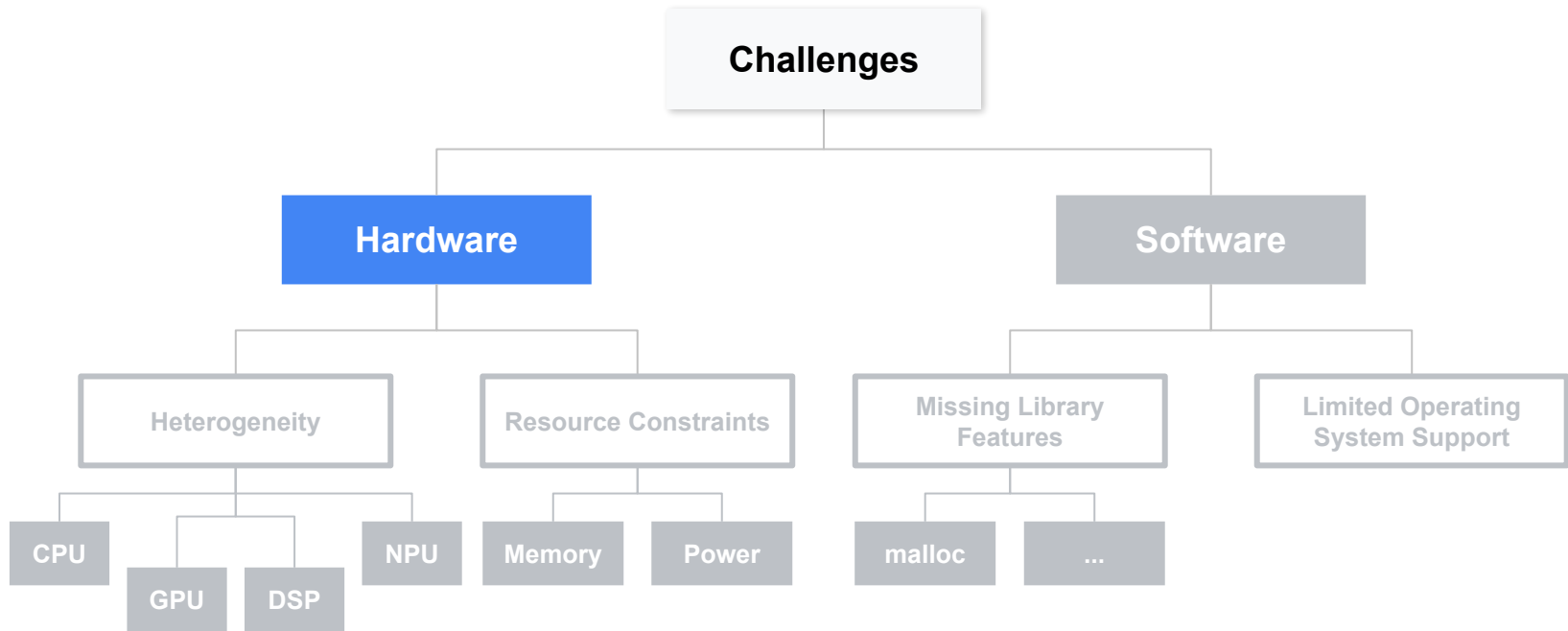


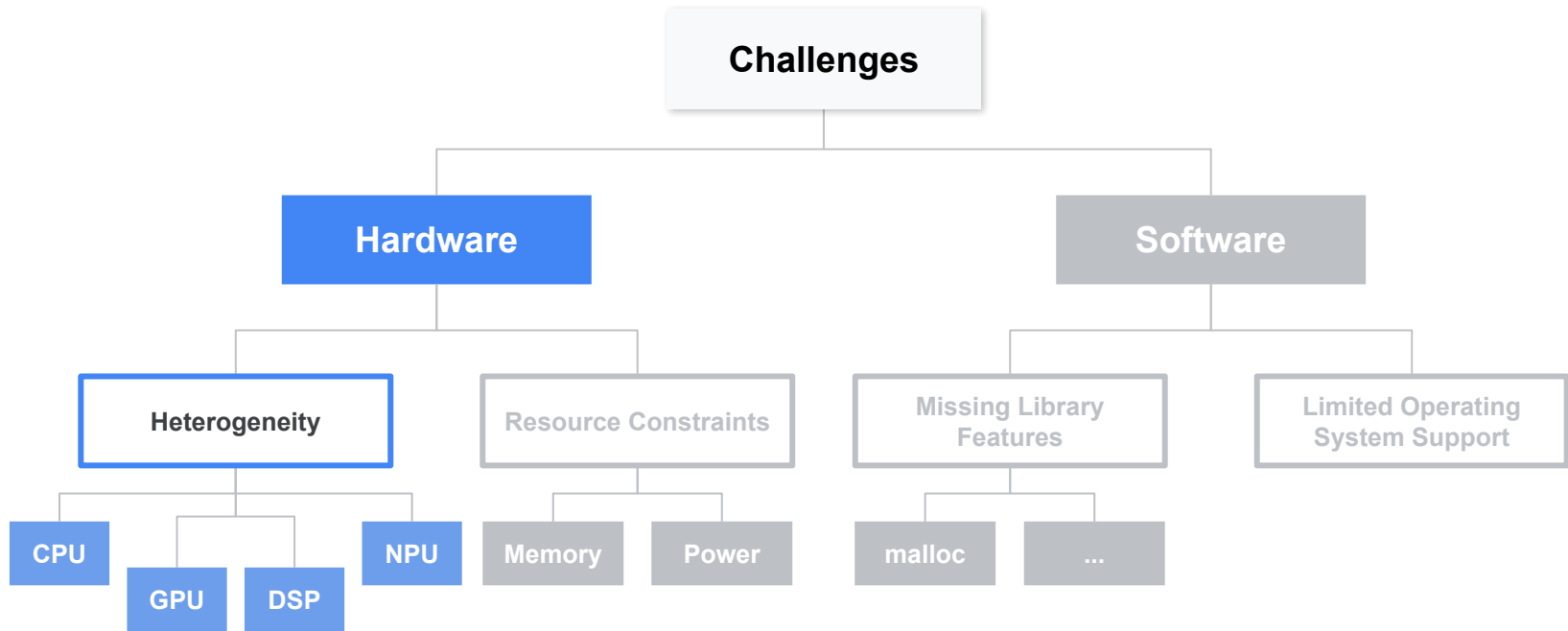


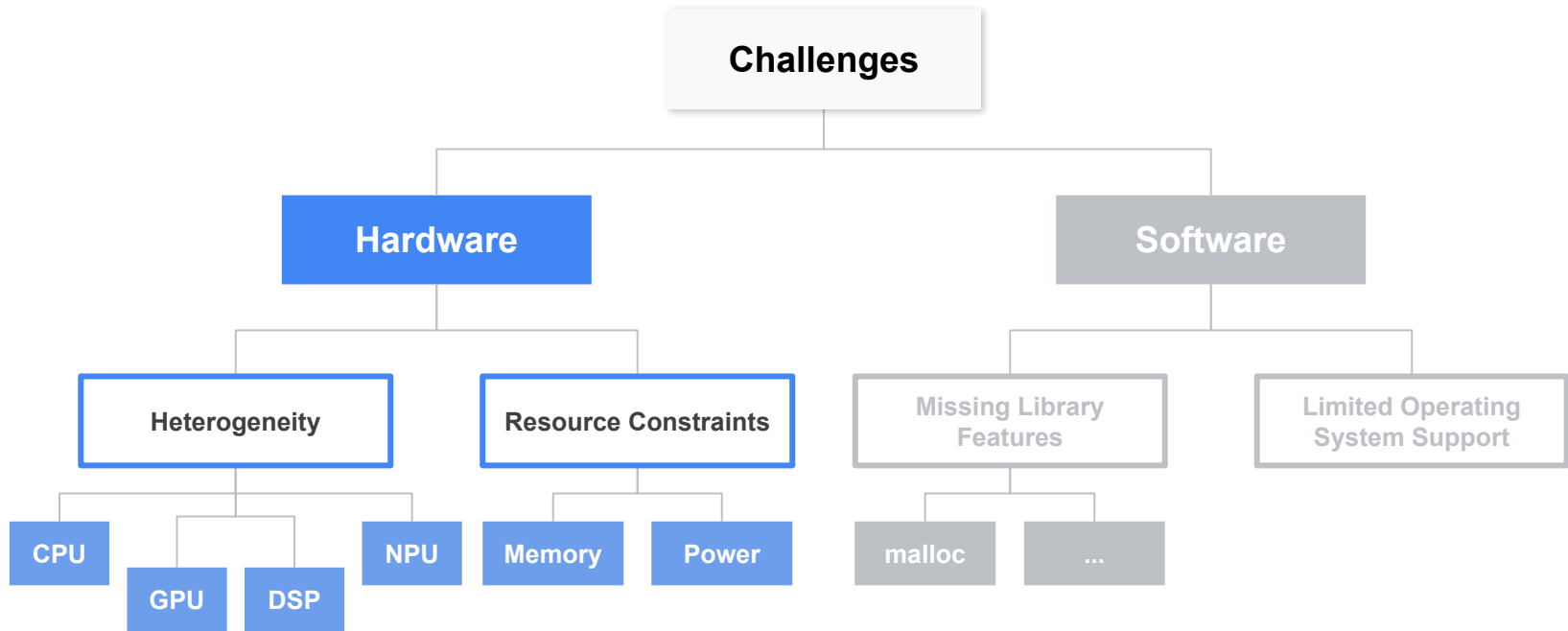


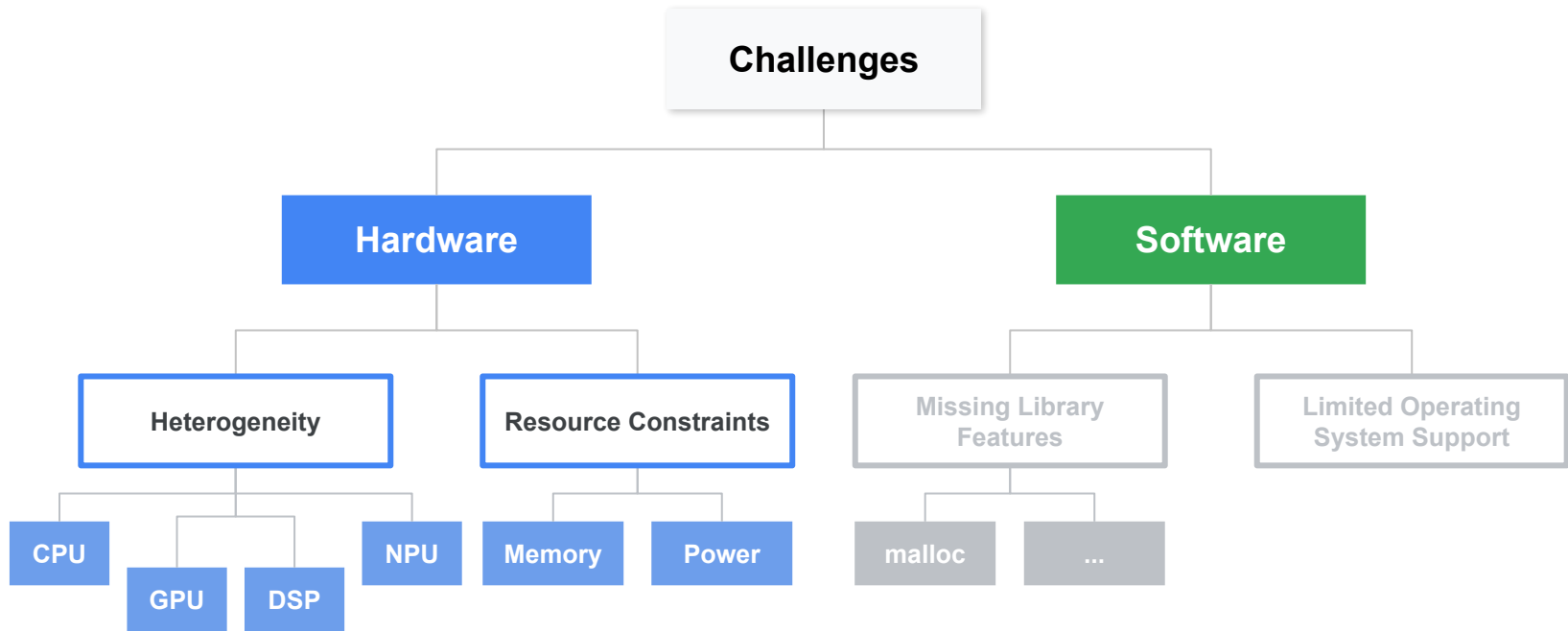
Board	MCU / ASIC	Clock	Memory	Sensors	Radio
Himax WE-I Plus EVB	HX6537-A 32-bit EM9D DSP	400 MHz	2MB flash 2MB RAM	Accelerometer, Mic, Camera	None
Arduino Nano 33 BLE Sense	32-bit nRF52840	64 MHz	1MB flash 256kB RAM	Mic, IMU, Temp, Humidity, Gesture, Pressure, Proximity, Brightness, Color	BLE
SparkFun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

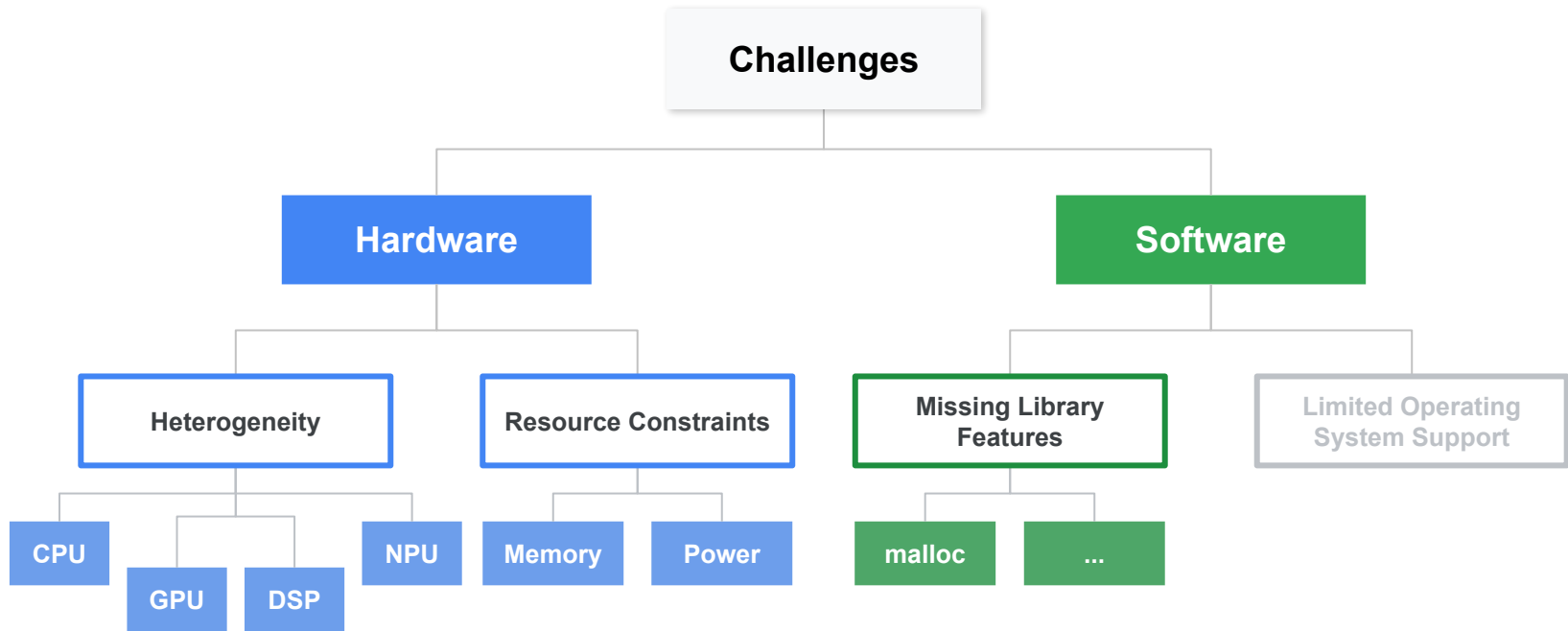


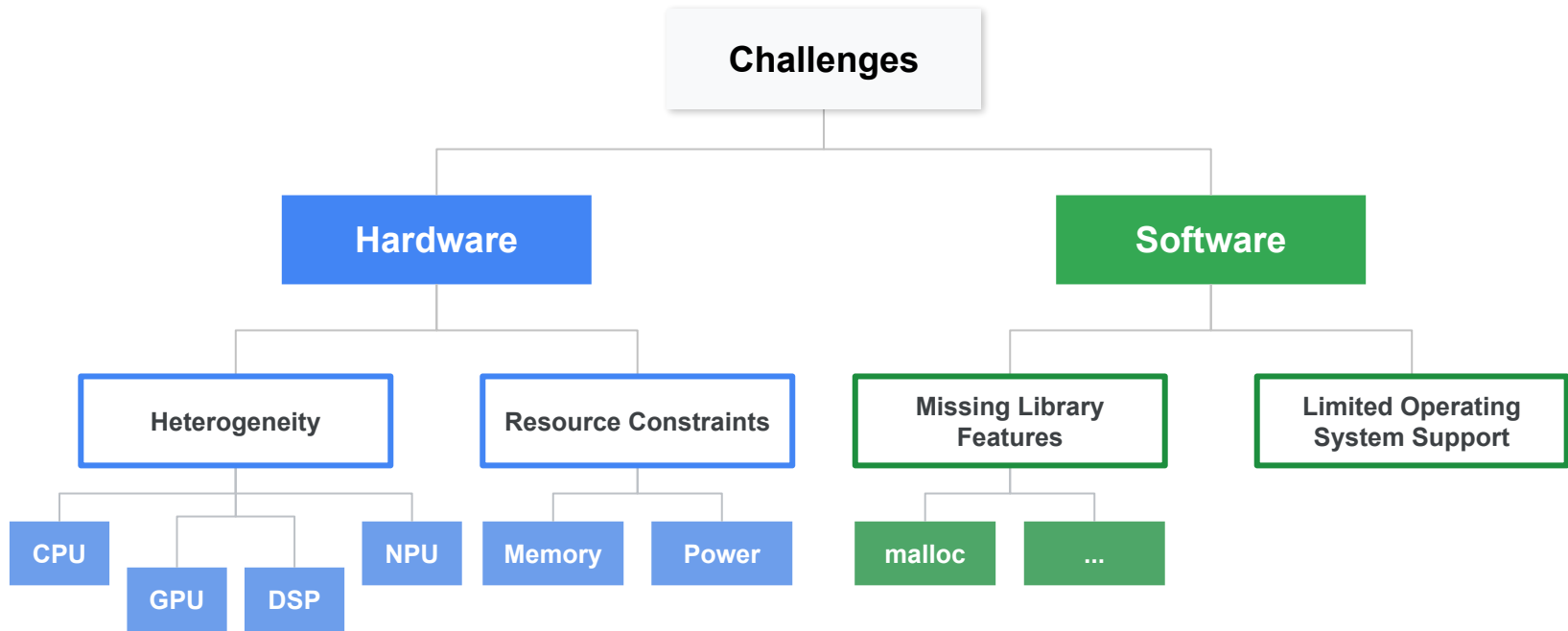


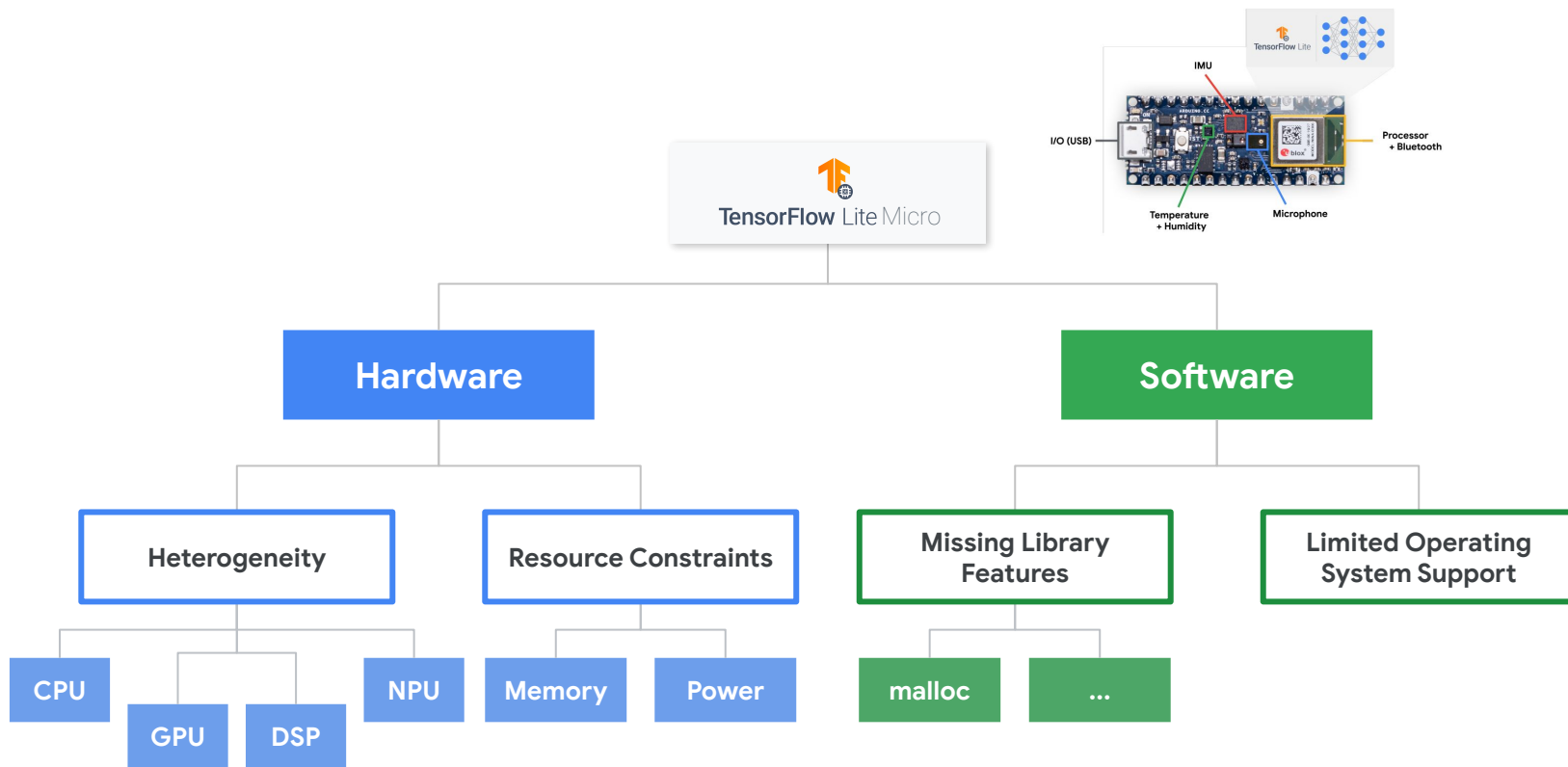


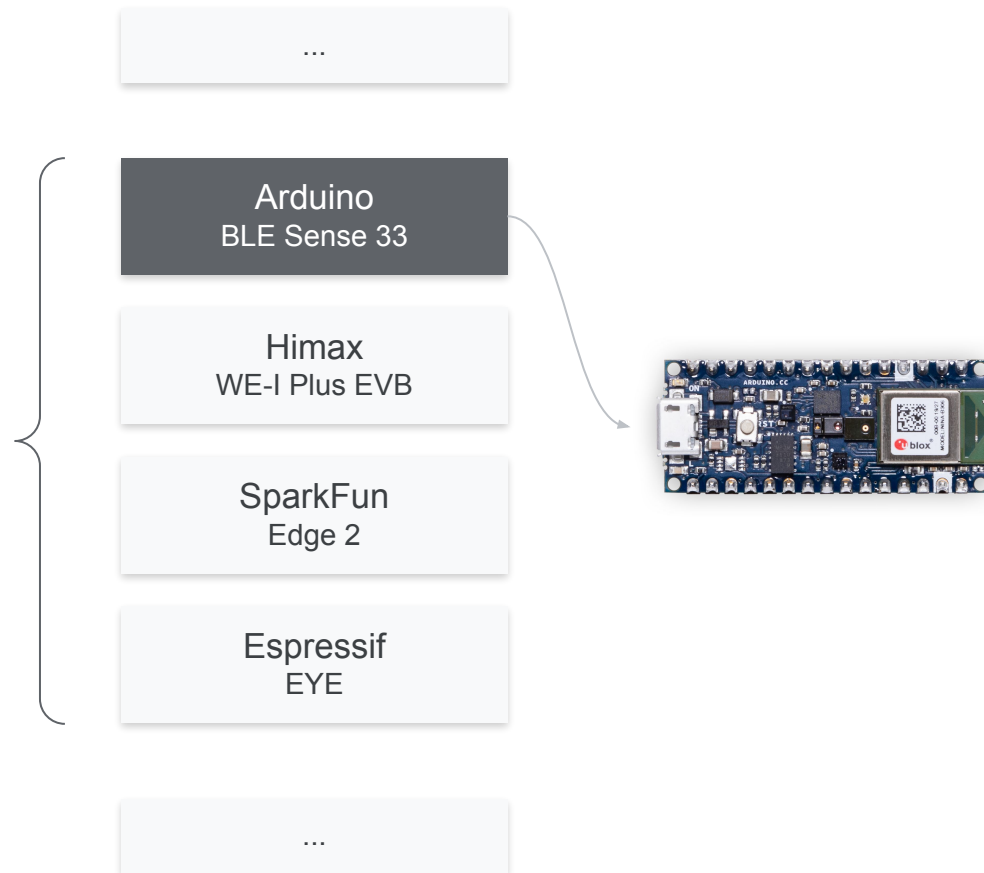
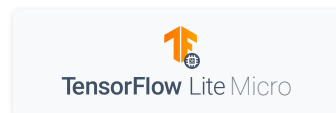






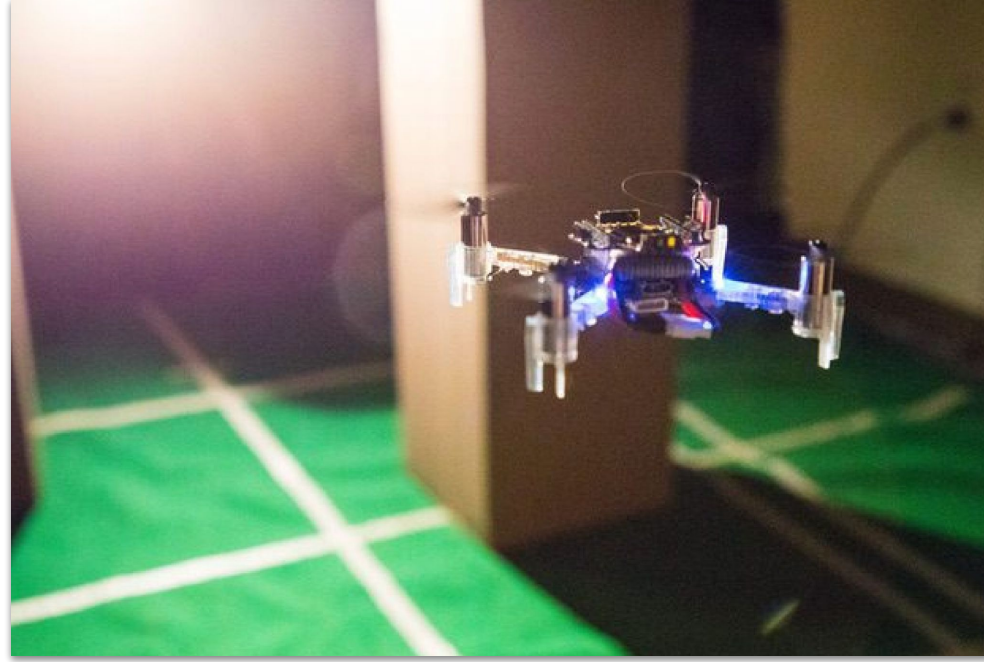
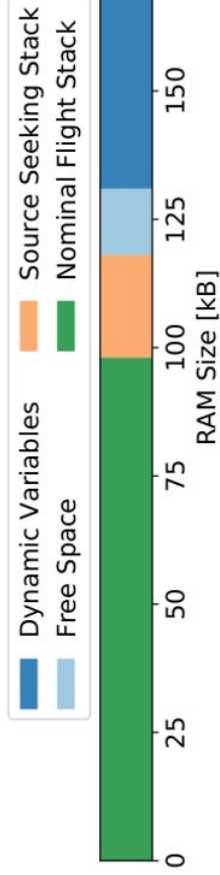








TensorFlow Lite Micro



Duisterhof, B.P., Krishnan, S., Cruz, J.J., Banbury, C.R., Fu, W., Faust, A., de Croon, G.C. and Reddi, V.J., 2019. Learning to seek: Autonomous source seeking with deep reinforcement learning onboard a nano drone microcontroller. arXiv preprint arXiv:1909.11236 and ICRA 2021.

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 David¹, Jared Duke¹, Advait Jain¹, Vijay Janapa Reddi^{1,2},
Nat Jeffries¹, Jian Li¹, Nick Kreeger¹, Ian Nappier¹, Meghna Natraj¹,
Shlomi Regev¹, Rocky Rhodes¹, Tiehen Wang¹, Pete Warden¹

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 botword 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, gyroscopes, PPG optical sensors, and other devices enable consumer and industrial applications, including predictive maintenance (Goebel et al., 2020; Saito 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 de-

¹Google ²Harvard 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. Copyright 2021 by the author(s).

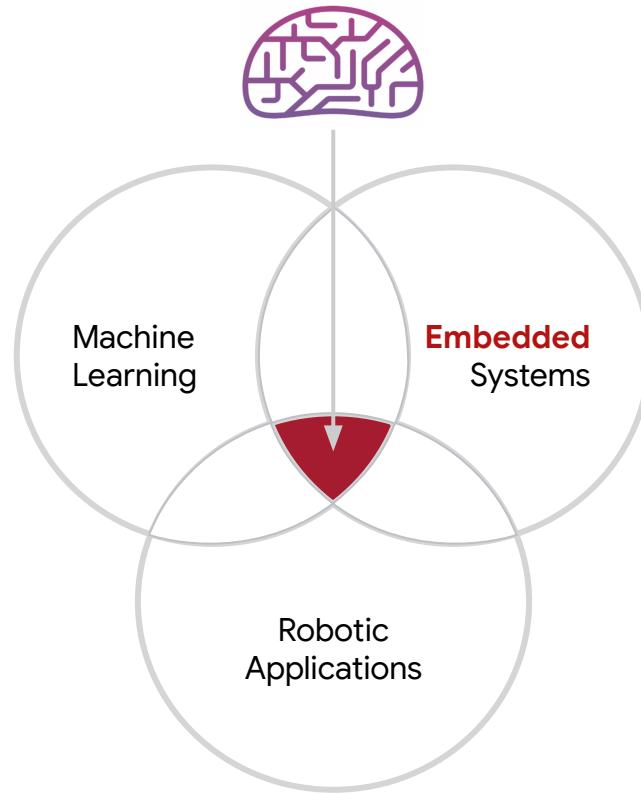
vices requires overcoming two crucial challenges. First and foremost, embedded systems have no unified TinyML framework. When engineers have deployed neural networks to such systems, they have built one-off frameworks that require manual optimization for each hardware platform. Such custom frameworks 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.

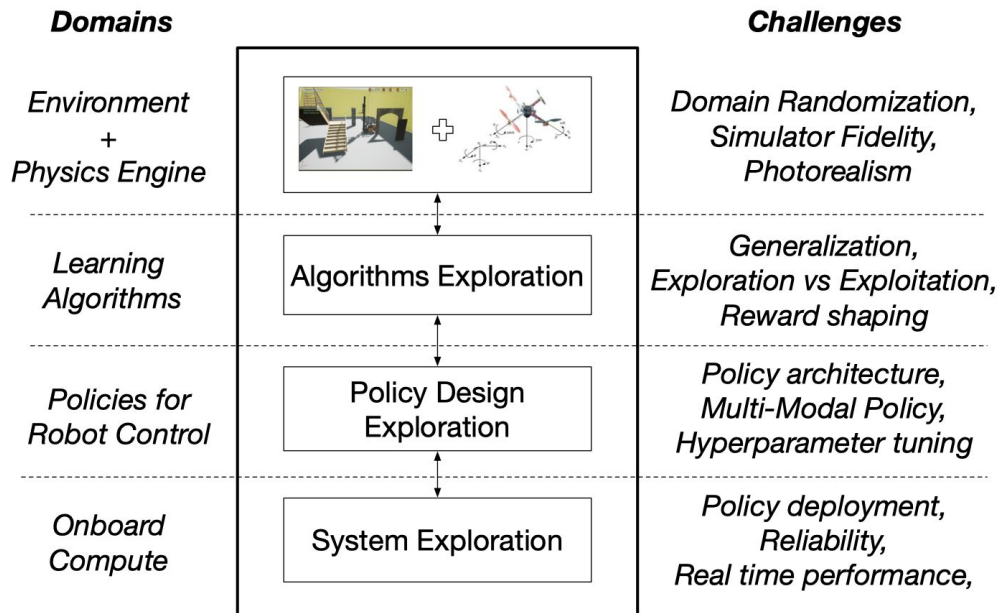
The lack of a proper framework has been a barrier to accelerating TinyML adoption and application in products. Beyond deploying a model to an embedded target, the framework must also have a means of training a model on a higher-compute platform. TinyML must exploit a broad ecosystem of tools for ML, 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:

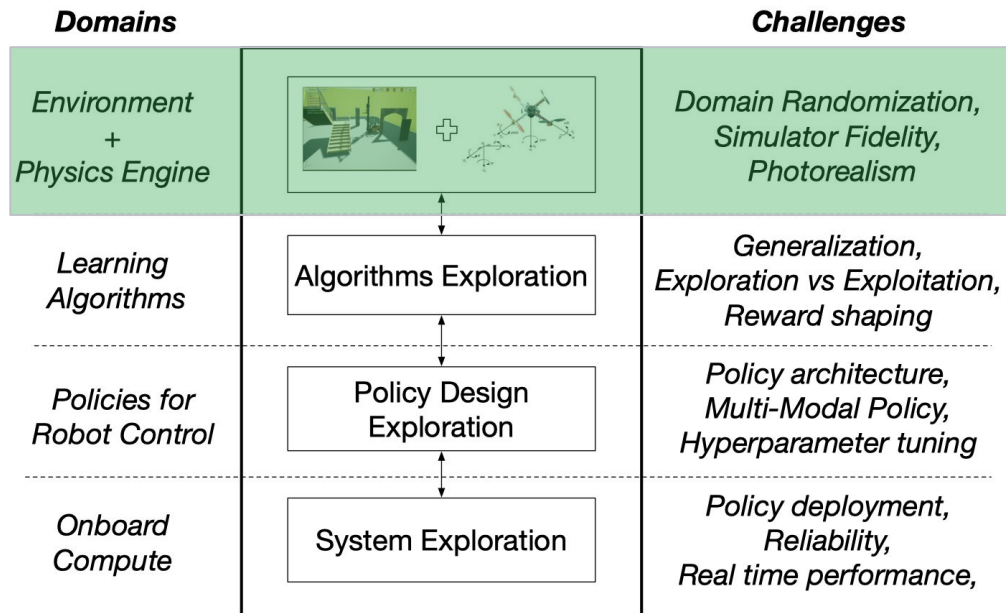
David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Wang, T. and Warden, P., 2021. TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems. Proceedings of Machine Learning and Systems, 3.



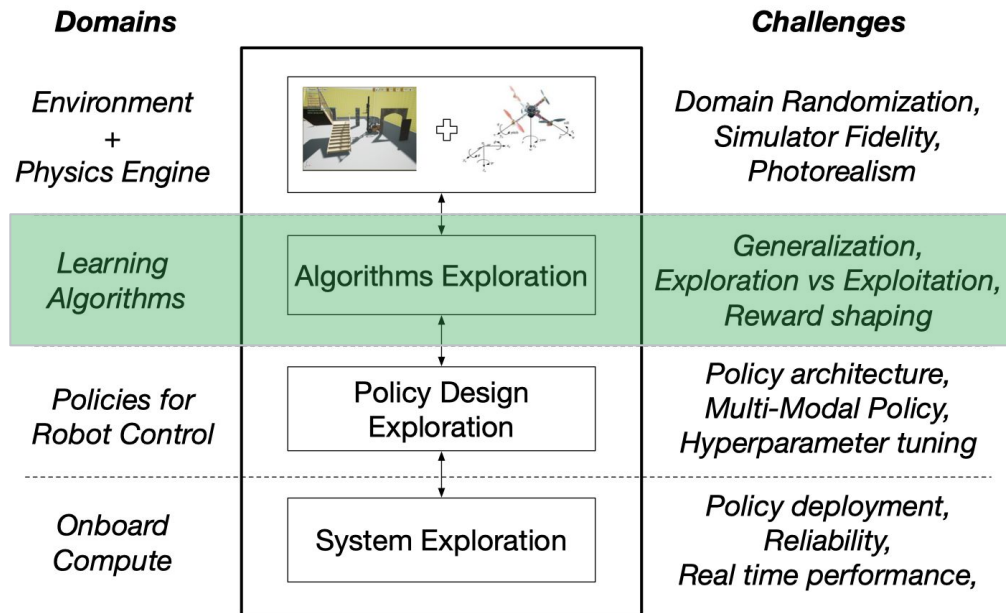
Understanding the Role of Computing



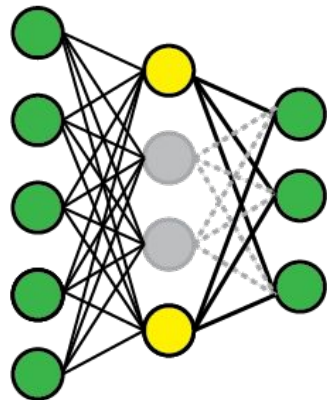
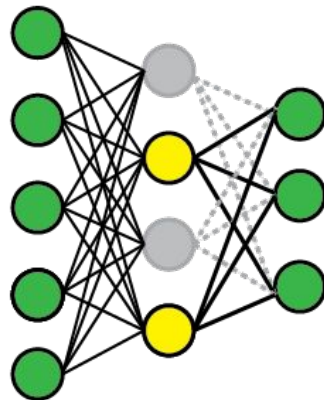
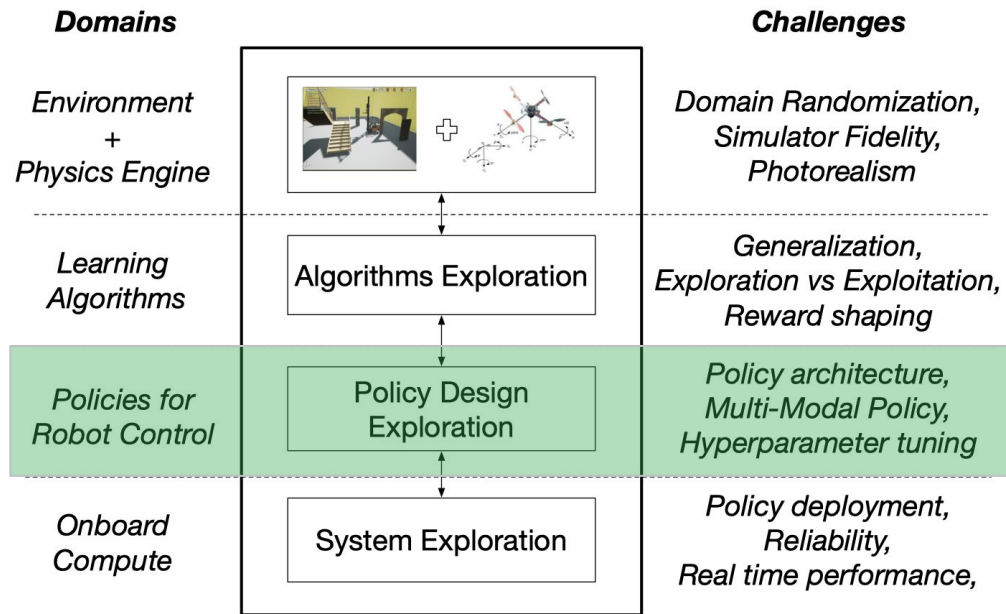
Understanding the Role of Computing



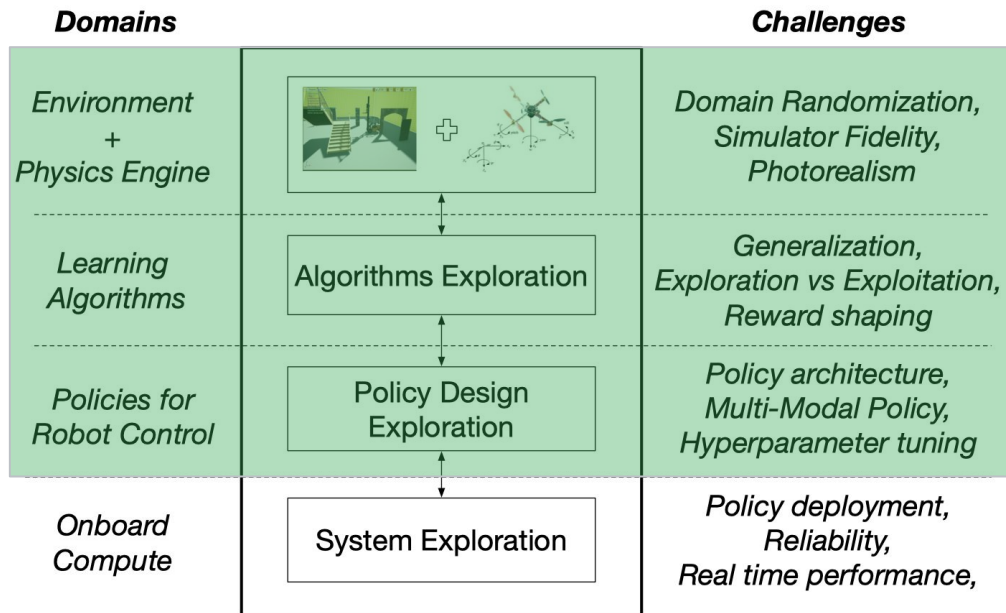
Understanding the Role of Computing



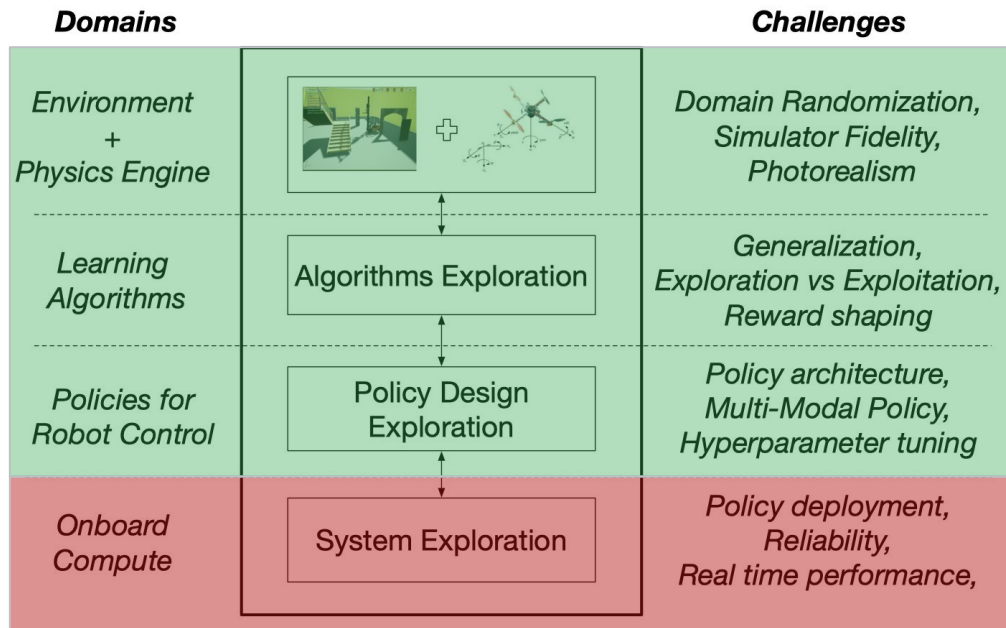
Understanding the Role of Computing



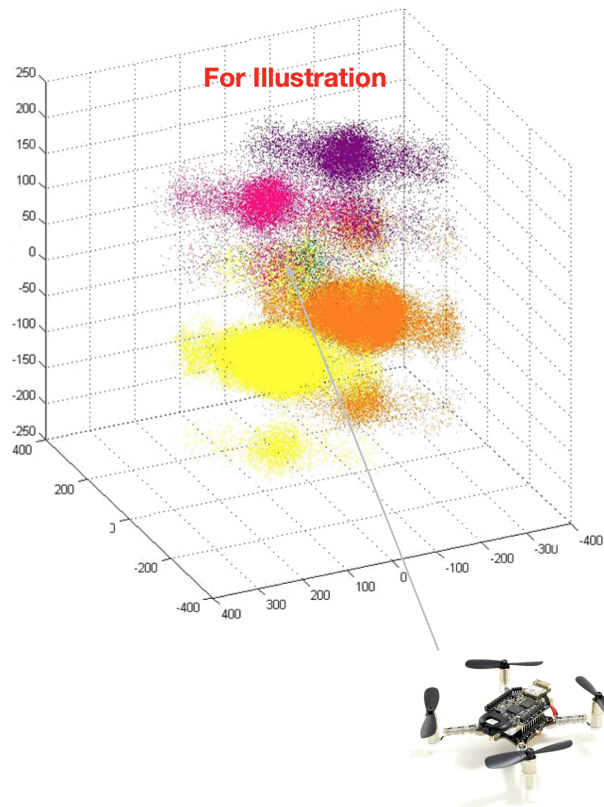
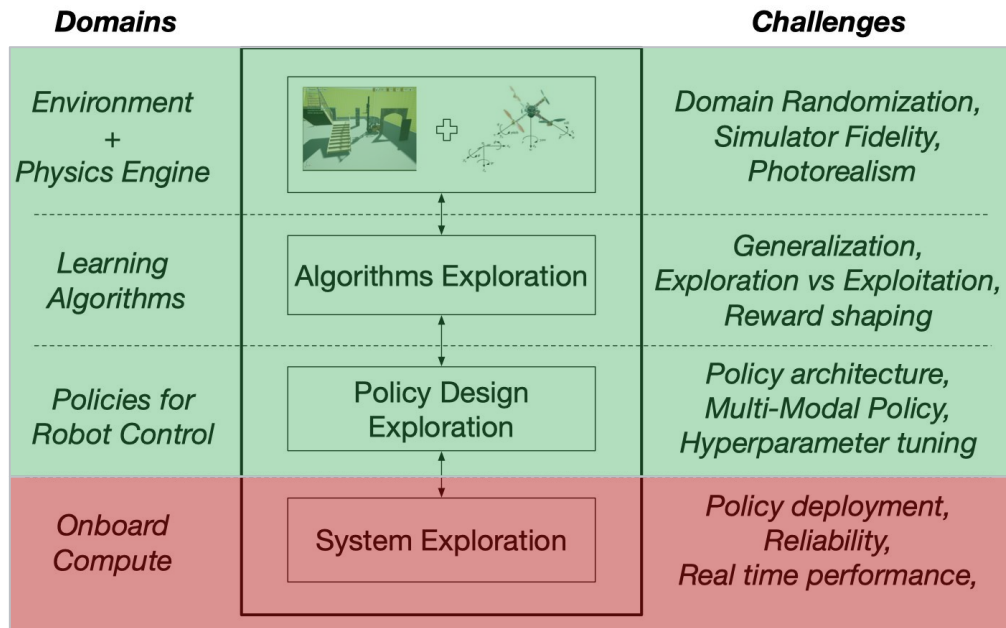
Understanding the Role of Computing



Understanding the Role of Computing

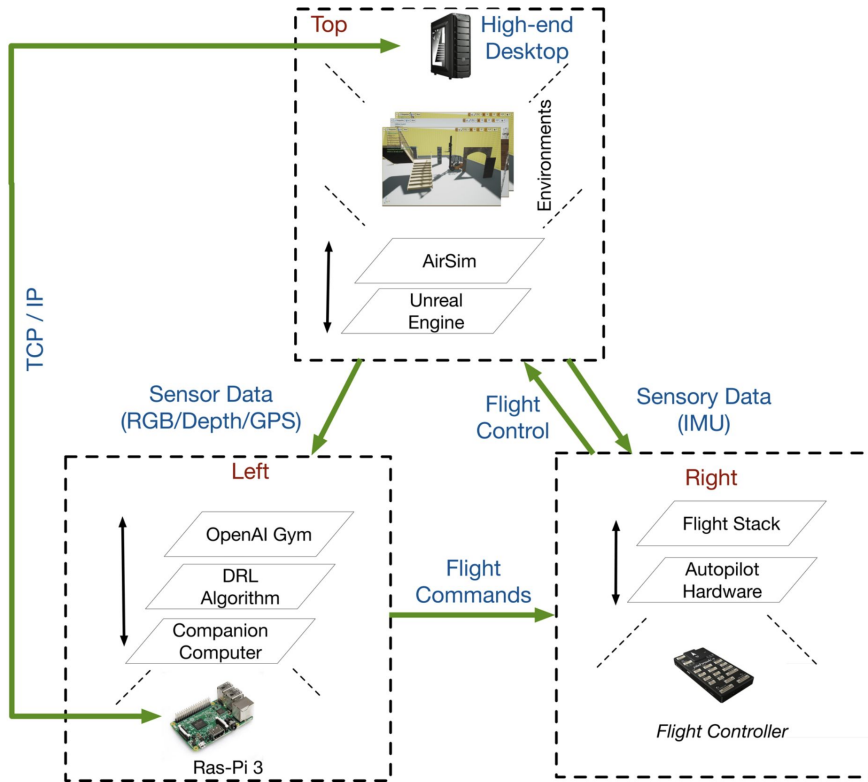


Understanding the Role of Computing



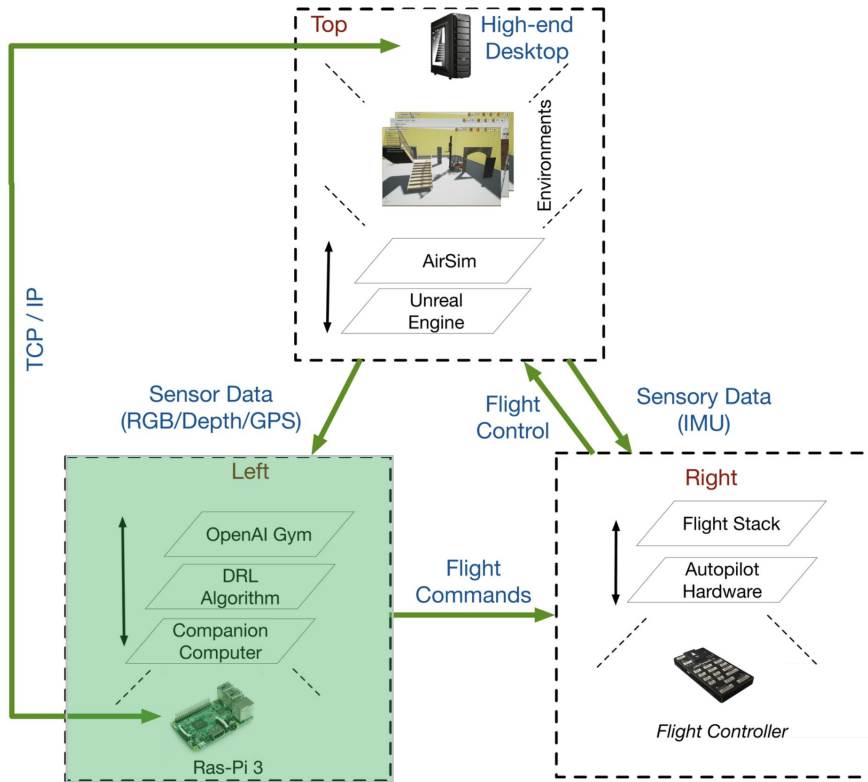
End-to-End Learning Simulation Engines

- Hardware-in-the-loop
 - Flight controller
 - Onboard compute (tinyML)

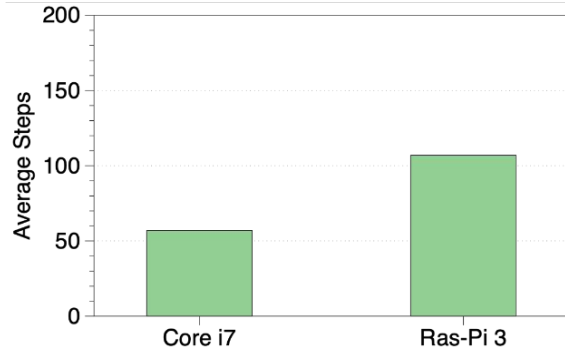
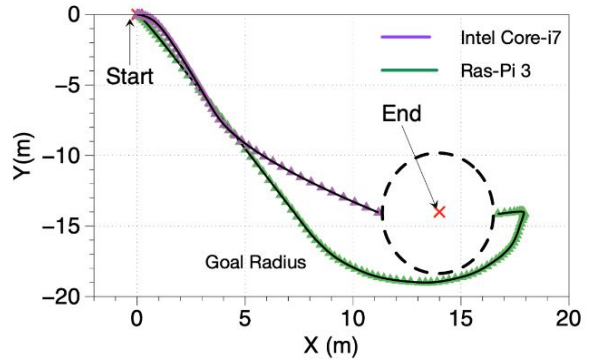
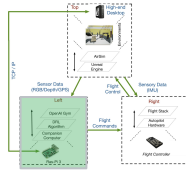


End-to-End Learning Simulation Engines

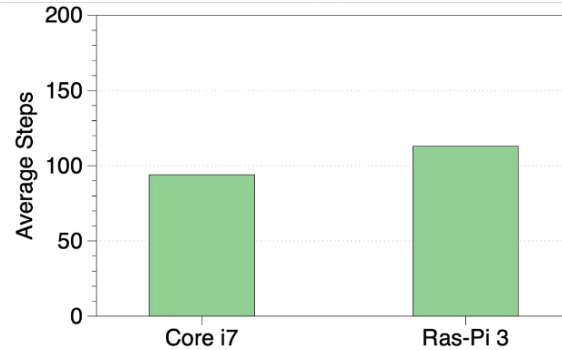
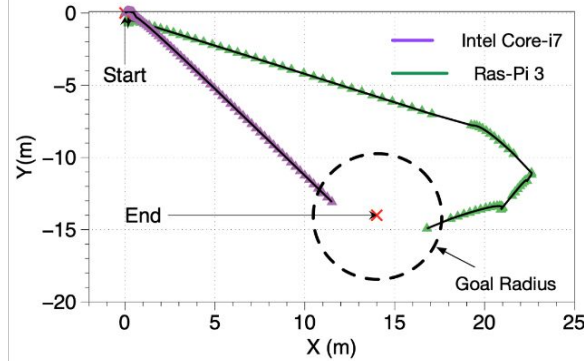
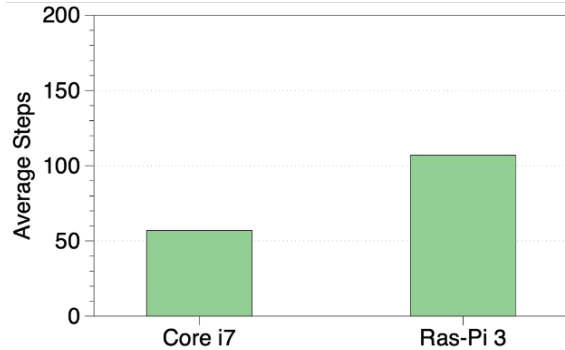
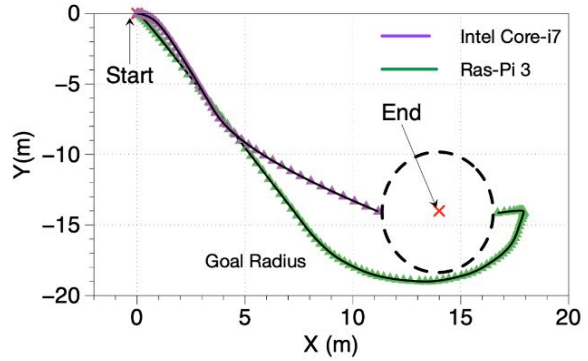
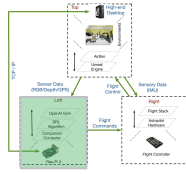
- Hardware-in-the-loop
 - Flight controller
 - Onboard compute (tinyML)



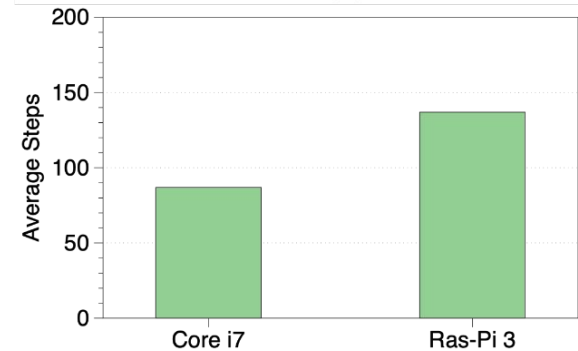
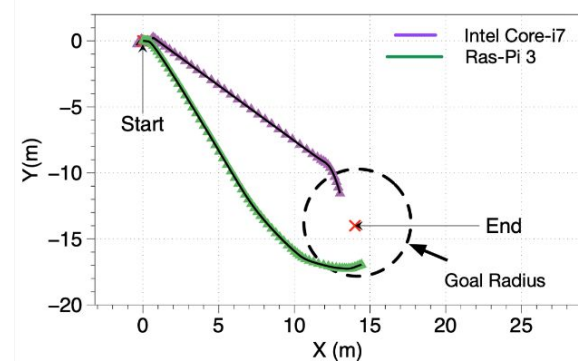
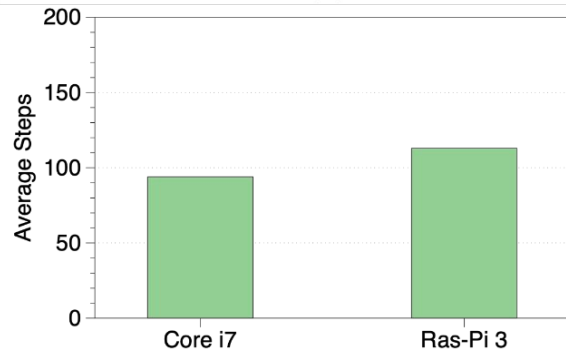
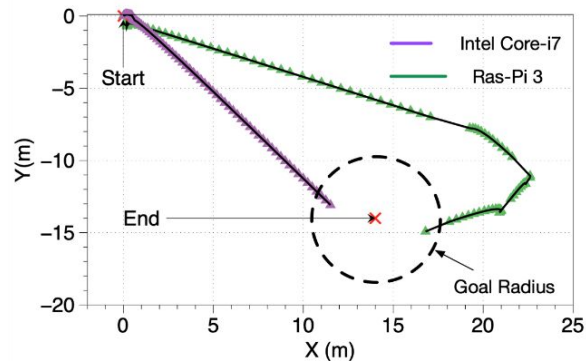
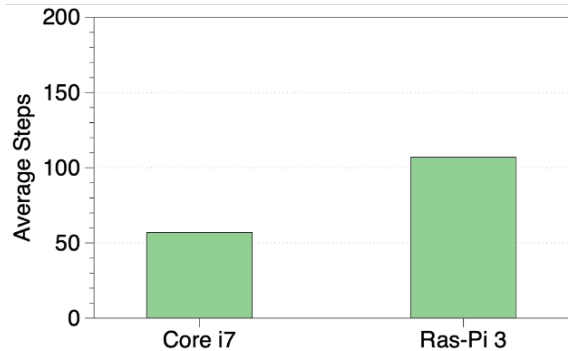
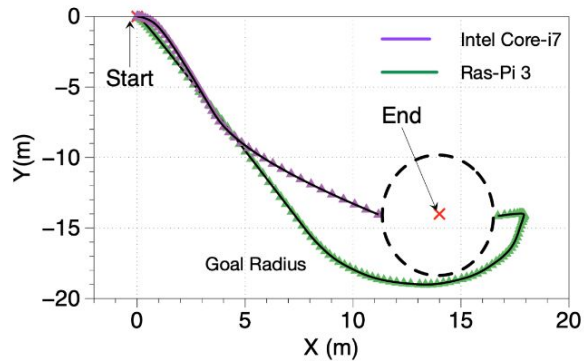
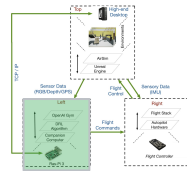
Understanding the “Blind Spots”



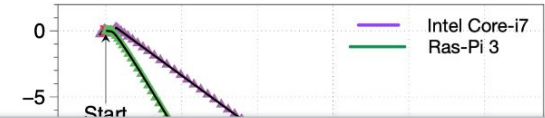
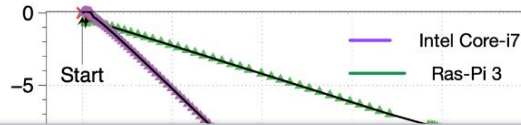
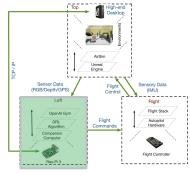
Understanding the “Blind Spots”



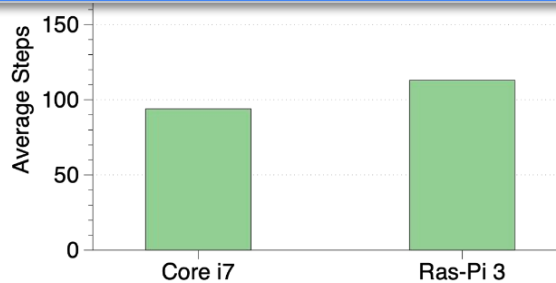
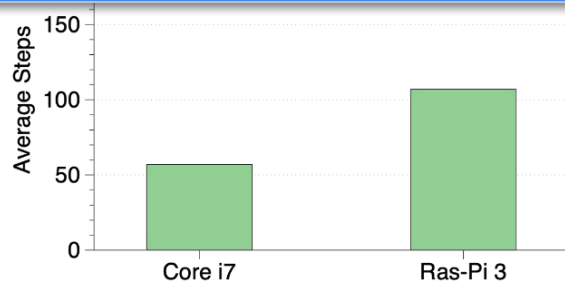
Understanding the “Blind Spots”



Understanding the “Blind Spots”



The deployment platform has a direct impact on the “performance” (success rate, latency, etc.) of the learning algorithm and so it must be taken into consistent consideration.



Air Learning: Deep RL Gym For Autonomous Navigation

Built to **consider the entire vertical co-design stack:**

- Random **environment generator** for domain randomization to enable RL generalization
- **Open source benchmark** to train RL algorithms, policies, and reward optimizations using regular and curriculum learning
- Demonstrate the “**hardware induced gap**”
- Describe the significance of **energy consumption** and the platform’s abilities when evaluating policy success rates

Machine Learning (2021) 110:2501–2540
<https://doi.org/10.1007/s10994-021-06006-6>



Air Learning: a deep reinforcement learning gym for autonomous aerial robot visual navigation

Srivatsan Krishnan¹ · Behzad Boroujerdian² · William Fu¹ · Aleksandra Faust³ · Vijay Janapa Reddi^{1,2}

Received: 16 March 2020 / Revised: 2 January 2021 / Accepted: 21 May 2021 /
Published online: 7 July 2021
© The Author(s) 2021

Abstract

We introduce Air Learning, an open-source simulator, and a gym environment for deep reinforcement learning research on resource-constrained aerial robots. Equipped with domain randomization, Air Learning exposes a UAV agent to a diverse set of challenging scenarios. We seed the toolset with point-to-point obstacle avoidance tasks in three different environments and Deep Q Networks (DQN) and Proximal Policy Optimization (PPO) trainers. Air Learning assesses the policies’ performance under various quality-of-flight (QoF) metrics, such as the energy consumed, endurance, and the average trajectory length, on resource-constrained embedded platforms like a Raspberry Pi. We find that the trajectories on an embedded Ras-Pi are vastly different from those predicted on a high-end desktop system, resulting in up to 40% longer trajectories in one of the environments. To understand the source of such discrepancies, we use Air Learning to artificially degrade high-end desktop performance to mimic what happens on a low-end embedded system. We then propose a mitigation technique that uses the hardware-in-the-loop to determine the latency distribution of running the policy on the target platform (onboard compute on aerial robot). A randomly sampled latency from the latency distribution is then added as an artificial delay within the training loop. Training the policy with artificial delays allows us to minimize the hardware gap (discrepancy in the flight time metric reduced from 37.73% to 0.5%). Thus, Air Learning with hardware-in-the-loop characterizes those differences and exposes how the onboard compute’s choice affects the aerial robot’s performance. We also conduct reliability studies to assess the effect of sensor failures on the learned policies. All put together, Air Learning enables a broad class of deep RL research on UAVs. The source code is available at: <https://github.com/harvard-edge/AirLearning>.

Keywords Deep reinforcement learning · Autonomous aerial robots · Resource-constrained deep RL · Robotics · Deep RL challenges · Sim2Real · Real life RL

Editors: Yuxi Li, Alborz Geraimifard, Lihong Li, Csaba Szepesvari, Tao Wang.







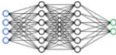







✉ Srivatsan Krishnan
krishnas@seas.harvard.edu

Extended author information available on the last page of the article



Components







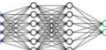







Design Space

Sensors	 RGB	 RGB-D $\sim O(10)$	 Lidar	
Autonomy Algorithms	 DroNet	 TrailNet	 CAD2RL	 Custom $\sim O(100 \text{ Billions})$
Onboard Compute	 NCS	 TX2	 Ras-Pi	 Custom Accelerator $\sim O(100 \text{ Millions})$
UAV Platform	 Mini-UAV	 Micro-UAV	 Nano-UAV $\sim O(10-100)$	

Off-the-shelf
components

Components







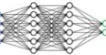







Design Space

Sensors	 RGB	 RGB-D $\sim O(10)$	 Lidar	
Autonomy Algorithms	 DroNet	 TrailNet	 CAD2RL	 Custom $\sim O(100 \text{ Billions})$
Onboard Compute	 NCS	 TX2	 Ras-Pi	 Custom Accelerator $\sim O(100 \text{ Millions})$
UAV Platform	 Mini-UAV	 Micro-UAV	 Nano-UAV $\sim O(10-100)$	



Components

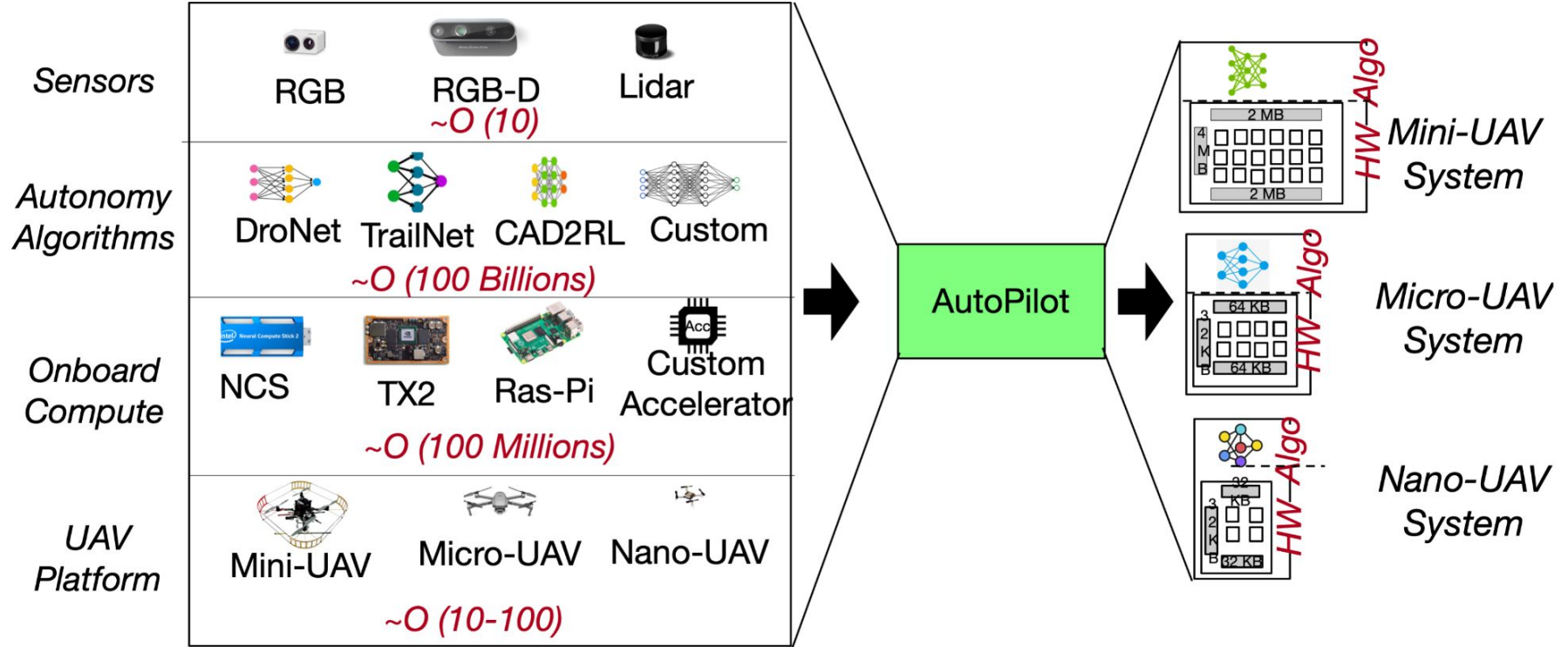
Design Space

Sensors	 RGB	 RGB-D $\sim O(10)$	 Lidar	
Autonomy Algorithms	 DroNet	 TrailNet	 CAD2RL	 Custom $\sim O(100 \text{ Billions})$
Onboard Compute	 NCS	 TX2	 Ras-Pi	 Custom Accelerator $\sim O(100 \text{ Millions})$
UAV Platform	 Mini-UAV	 Micro-UAV	 Nano-UAV $\sim O(10-100)$	

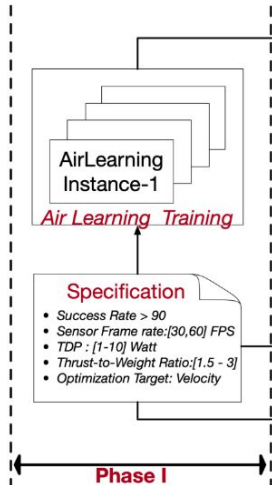


Components

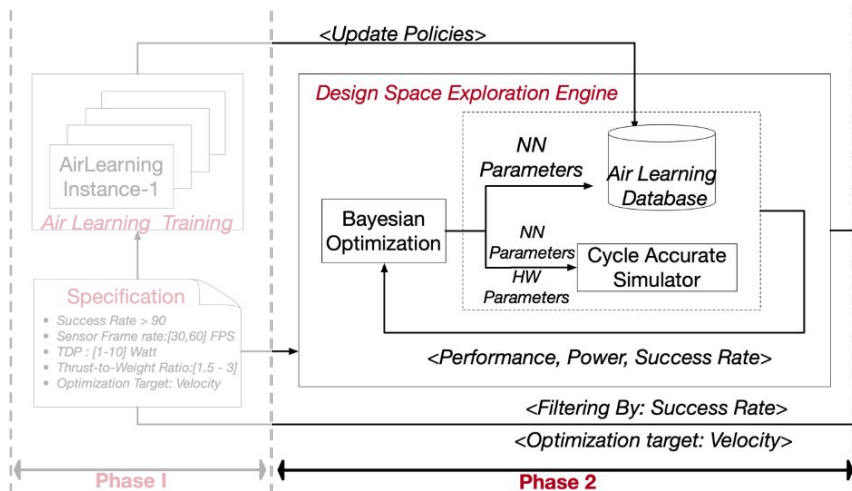
Design Space



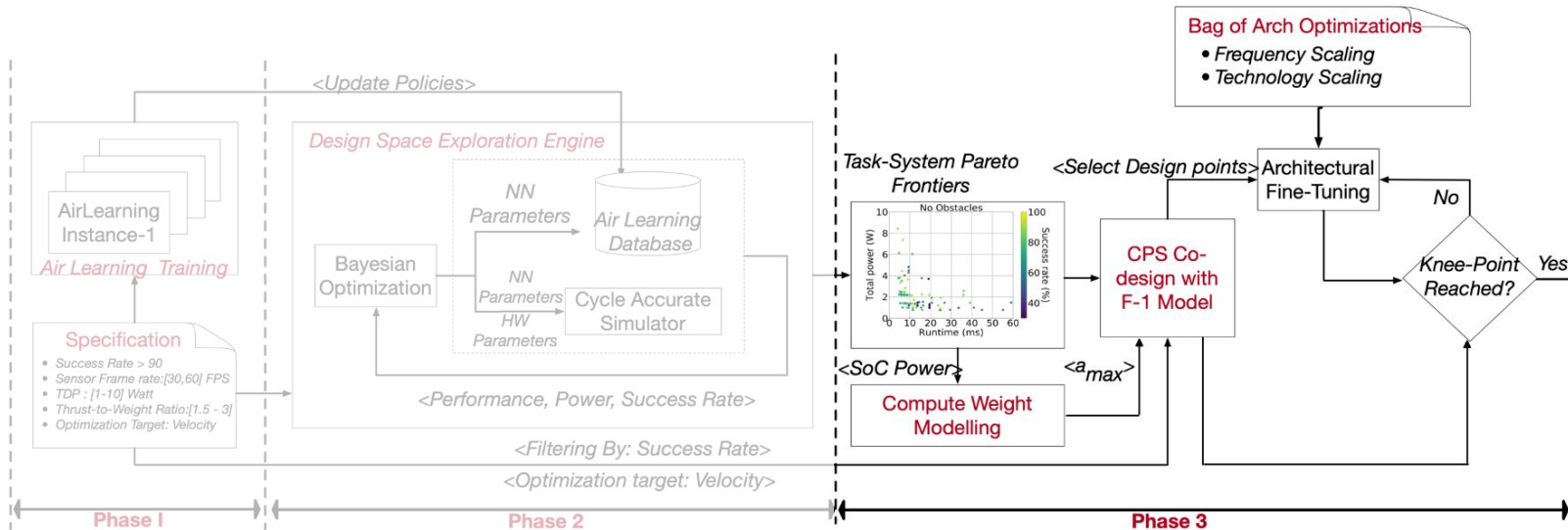
AutoPilot: An End-to-end Design Space Explorer



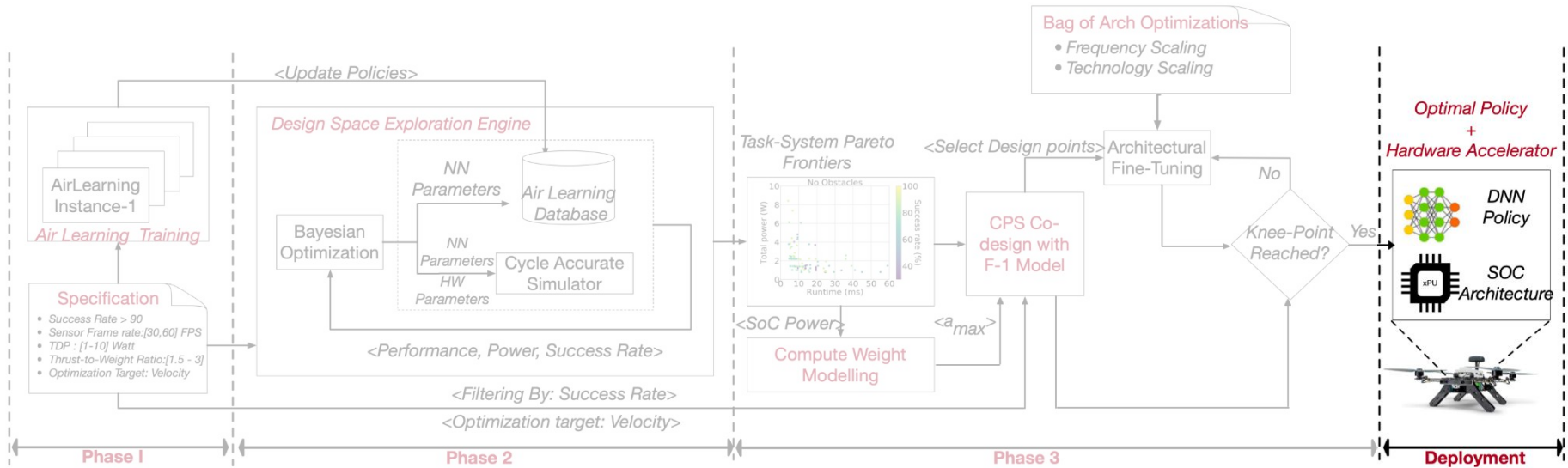
AutoPilot: An End-to-end Design Space Explorer

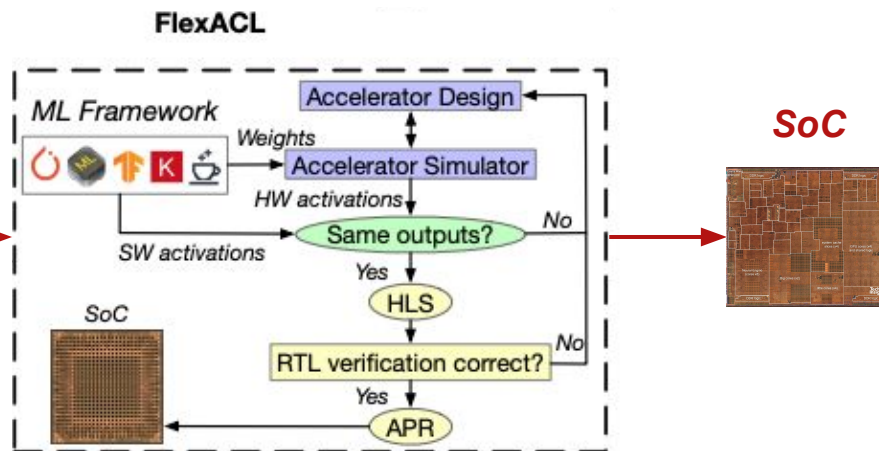
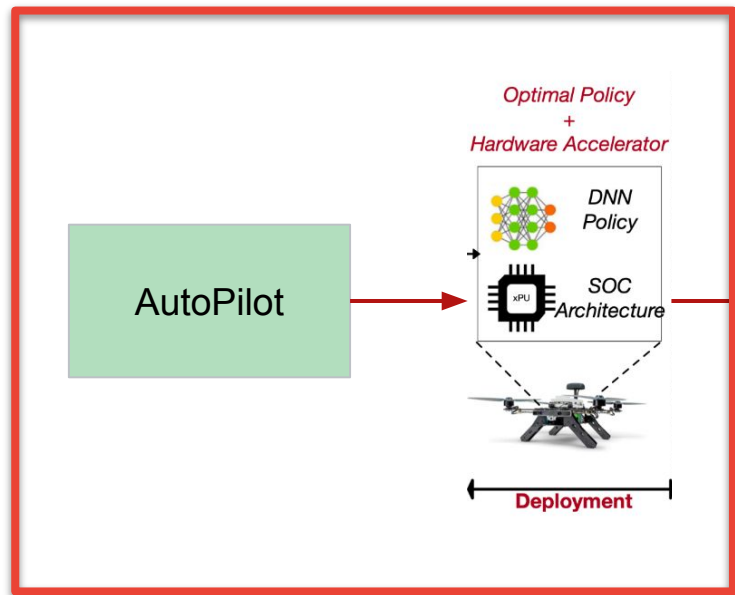


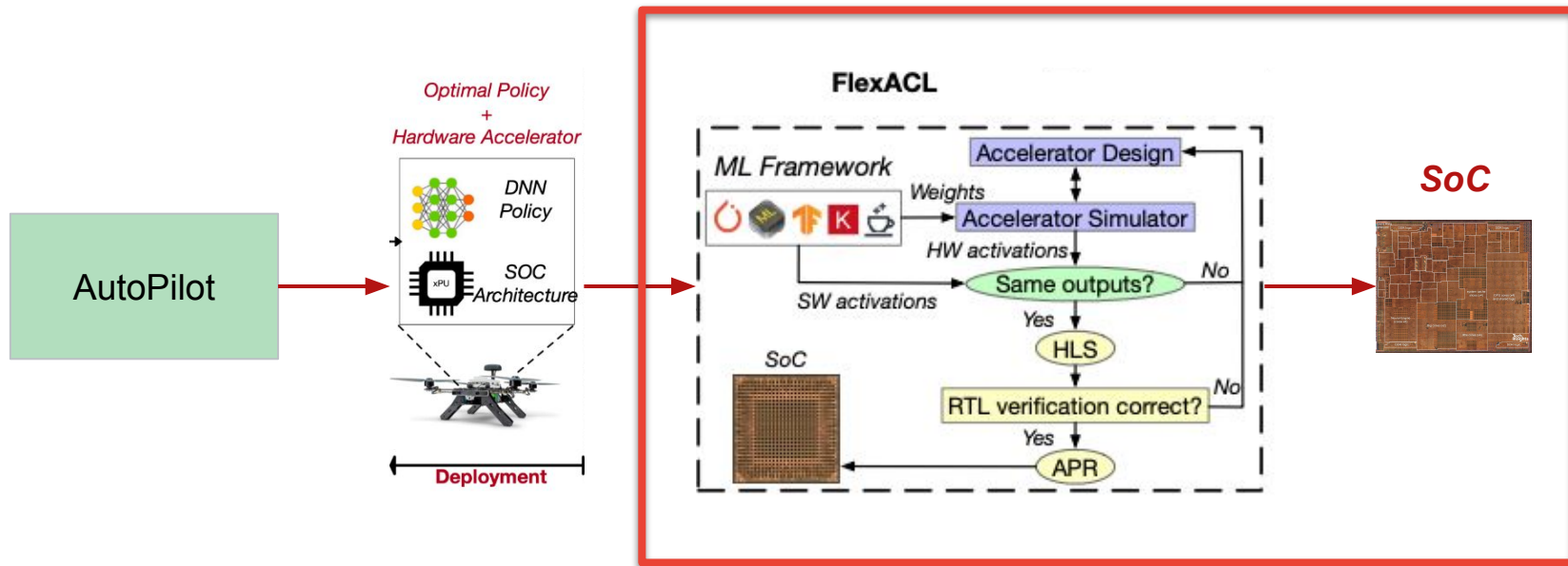
AutoPilot: An End-to-end Design Space Explorer



AutoPilot: An End-to-end Design Space Explorer







AutoPilot: Automating Co-Design Space Exploration for Autonomous UAVs

Automate the search for compute for autonomous robots:

- Explore the **cyber-physical design space**
- Design **custom computing solutions**, rather than existing off-the-shelf components for maximizing efficiency
- Collectively **optimize across a wide range of different parameters** that would not be possible without “AutoDSE”

AutoPilot: Automating Co-Design Space Exploration for Autonomous UAVs

Srivatsan Krishnan[†], Zishen Wan[†], Kshitij Bhardwaj[†], Paul Whatmough[‡], Aleksandra Faust[§], Sabrina M. Neuman[†], Gu-Yeon Wei[†], David Brooks[†], and Vijay Janapa Reddi[†]

[†]Harvard University

[‡]ARM Research

[§]Google Brain Research

Abstract

Building domain-specific accelerators for autonomous unmanned aerial vehicles (UAVs) is challenging due to a lack of systematic methodology for designing onboard compute. Balancing a computing system for a UAV requires considering both the cyber (e.g., sensor rate, compute performance) and physical (e.g., payload weight) characteristics that affect overall performance. Iterating over the many component choices results in a combinatorial explosion of the number of possible combinations: from 10s of thousands to billions, depending on implementation details. Manually selecting combinations of these components is tedious and expensive. To navigate the cyber-physical design space efficiently, we introduce AutoPilot, a framework that automates full-system UAV co-design. AutoPilot uses Bayesian optimization to navigate a large design space and automatically select a combination of autonomy algorithm and hardware accelerator while considering the cross-product effect of other cyber and physical UAV components. We show that the AutoPilot methodology consistently outperforms general-purpose hardware selections like Xavier NX and Jetson TX2, as well as dedicated hardware accelerators built for autonomous UAVs, across a range of representative scenarios (three different UAV types and three deployment environments). Designs generated by AutoPilot increase the number of missions on average by up to 2.25x, 1.62x, and 1.43x for nano, micro, and mini-UAVs respectively over baselines. Our work demonstrates the need for holistic full-UAV co-design to achieve maximum overall UAV performance and the need for automated flows to simplify the design process for autonomous cyber-physical systems.

1. Introduction

Unmanned aerial vehicles (UAVs) are on the rise in real-world deployments [75, 52, 15, 62], but building computing systems for these platforms remains challenging. They are complex systems in which the traditional computing platform is just one component among many others. To achieve overall performance, it is important to understand what implications other UAV components have on the design of onboard compute.

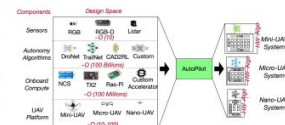


Figure 1: AutoPilot efficiently automates navigation of the large UAV component design space to co-design optimal onboard compute across a range of autonomous UAV systems.

Co-designing hardware accelerators with other UAV components requires navigating a large design space (see Fig. 1), e.g., 100's of UAVs [44] \times millions of HW accelerators [78] \times billions of autonomy algorithm neural network model parameters [22] \times 100's of sensors [45] $\approx 10^{18}$. Worse, this number is still conservative since each UAV type includes additional components such as a flight controller and a battery. Taming this large space can be expensive and tedious. Automating the co-design of the hardware accelerator and other UAV system components can optimize mission performance while keeping design overheads low as UAV systems evolve.

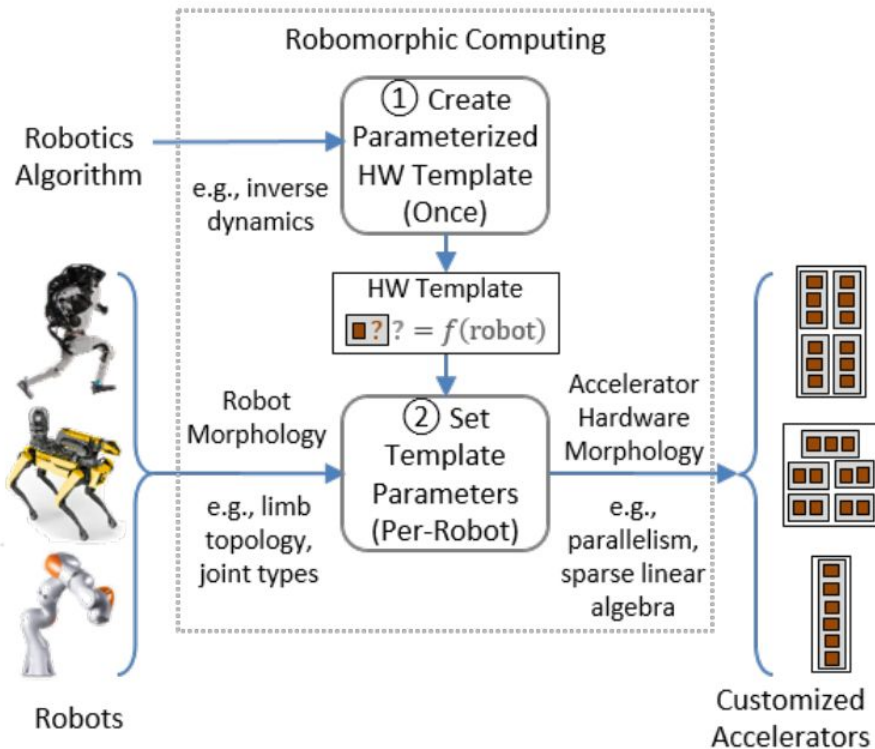
Key challenges in UAV design include the ability to systematically navigate the large design space of components, and understanding which combinations of these components maximize overall UAV performance. While specialized hardware is critical for compute efficiency, designing it is an expensive process. It is essential to establish automated design methodologies that remain agile as future autonomous systems evolve.

To address these challenges, we introduce AutoPilot: a cyber-physical co-design automation framework for autonomous UAVs. Given a high-level specification of autonomy task, UAV type, and mission goals, AutoPilot automatically navigates the large design space to perform full-system UAV co-design to generate a combination of autonomy algorithm and corresponding hardware accelerator to maximize overall UAV performance (e.g., number of missions).

The AutoPilot takes a high-level specification for the auton-

arXiv:2102.02988v3 [cs.RO] 10 Sep 2021

Beyond UAVs



Robomorphic Computing: A Design Methodology for Domain-Specific Accelerators Parameterized by Robot Morphology

Sabrina M. Neuman
sneuman@seas.harvard.edu
Harvard University
Cambridge, MA, USA

Brian Plancher
brian_plancher@g.harvard.edu
Harvard University
Cambridge, MA, USA

Thomas Bourgeat
bthom@csail.mit.edu
MIT
Cambridge, MA, USA

Thierry Tambe
ttambe@g.harvard.edu
Harvard University
Cambridge, MA, USA

Srinivas Devadas
devadas@mit.edu
MIT
Cambridge, MA, USA

Vijay Janapa Reddi
vj@eecs.harvard.edu
Harvard University
Cambridge, MA, USA

ABSTRACT

Robotics applications have hard time constraints and heavy computational burdens that can greatly benefit from domain-specific hardware accelerators. For the latency-critical problem of robot motion planning and control, there exists a performance gap of at least an order of magnitude between joint actuator response rates and state-of-the-art software solutions. Hardware acceleration can close this gap, but it is essential to define automated hardware design flows to keep the design process agile as applications and robot platforms evolve. To address this challenge, we introduce robomorphic computing: a methodology to transform robot morphology into a customized hardware accelerator morphology. We (i) present this design methodology, using robot topology and structure to exploit parallelism and matrix sparsity patterns in accelerator hardware; (ii) use the methodology to generate a parameterized accelerator design for the gradient of rigid body dynamics, a key kernel in motion planning; (iii) evaluate FPGA and synthesized ASIC implementations of this accelerator for an industrial manipulator robot; and (iv) describe how the design can be automatically customized for other robot models. Our FPGA accelerator achieves speedups of 8x and 46x over CPU and GPU when executing a single dynamics gradient computation. It maintains speedups of 1.9x to 2.9x over CPU and GPU, including computation and I/O round-trip latency, when deployed as a coprocessor to a host CPU for processing multiple dynamics gradient computations. ASIC synthesis indicates an additional 7.2x speedup for single computation latency. We describe how this principled approach generalizes to more complex robot platforms, such as quadrupeds and humanoids, as well as to other computational kernels in robotics, outlining a path forward for future robomorphic computing accelerators.

CCS CONCEPTS

• Hardware → Hardware accelerators; • Computer systems organization → Robotics.

KEYWORDS

robotics, hardware accelerators, dynamics, motion planning

ACM Reference Format:

Sabrina M. Neuman, Brian Plancher, Thomas Bourgeat, Thierry Tambe, Srinivas Devadas, and Vijay Janapa Reddi. 2021. Robomorphic Computing: A Design Methodology for Domain-Specific Accelerators Parameterized by Robot Morphology. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21)*, April 19–23, 2021, Virtual, USA. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3445814.3446746>

1 INTRODUCTION

Complex robots such as manipulators, quadrupeds, and humanoids that can safely interact with people in dynamic, unstructured, and unpredictable environments are a promising solution to address critical societal challenges, from elder care [24, 53] to the health and safety of humans in hazardous environments [34, 60]. A major obstacle to the deployment of complex robots is the need for high-performance computing in a portable form factor. Robot perception, localization, and motion planning applications must be run online at real-time rates and under strict power budgets [12, 26, 47, 55].

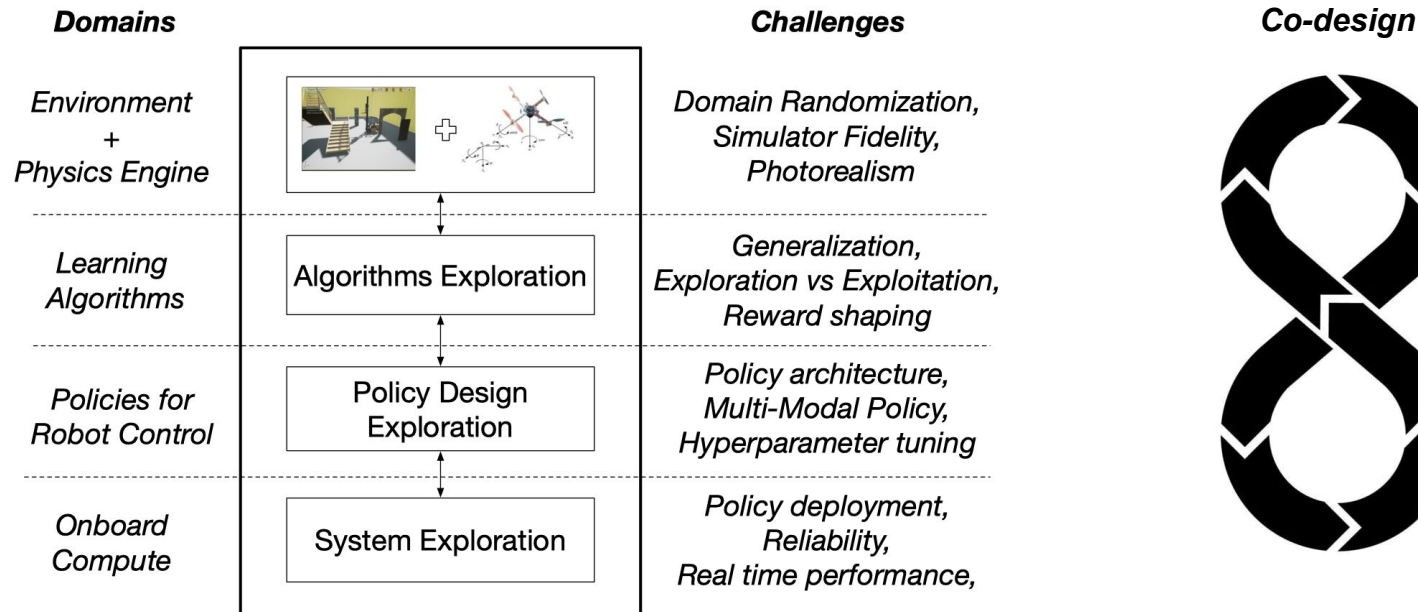
Domain-specific hardware acceleration is an emerging solution to this problem, building on the success of accelerators for other domains such as neural networks [7, 23, 49]. However, while accelerators have improved the power and performance of robot perception and localization [7, 49, 56], relatively little work has been done for motion planning [33, 38].

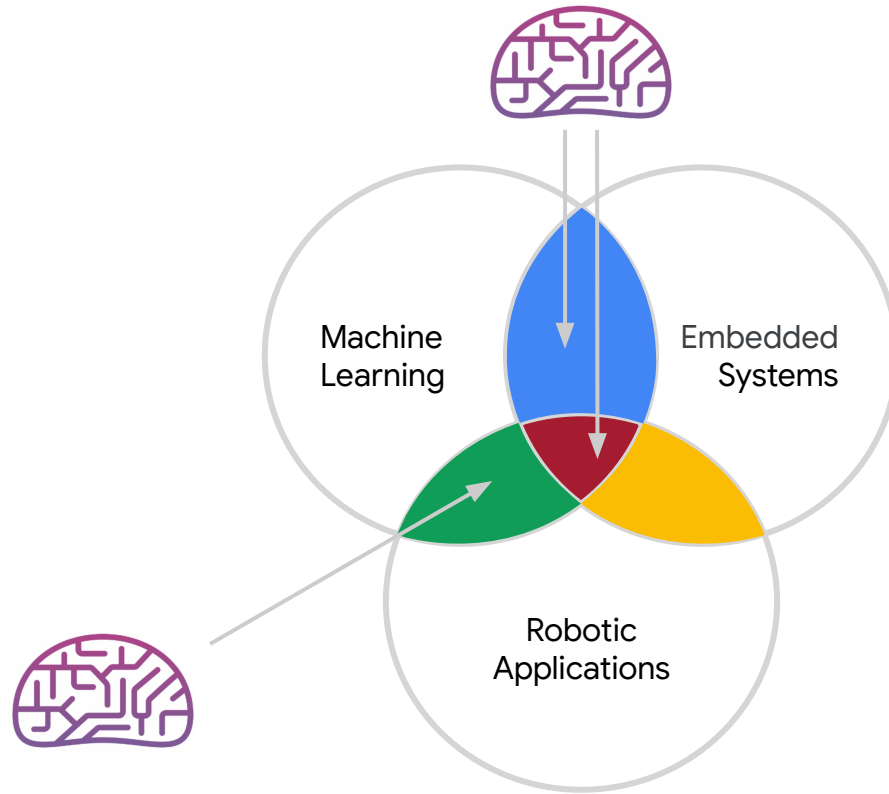
Motion planning algorithms calculate a valid motion path from a robot's initial position to a goal state. Online motion planning approaches [41, 57] rely heavily on latency-critical calculation of functions describing the underlying physics of the robot, e.g., rigid body dynamics and its gradient [5, 14, 18]. There exist several software implementations that are sufficient for use in traditional control approaches [6, 16, 22, 27, 36, 39], but emerging techniques such as whole-body nonlinear model predictive control (MPC) [9, 26] reveal a performance gap of at least an order of magnitude:

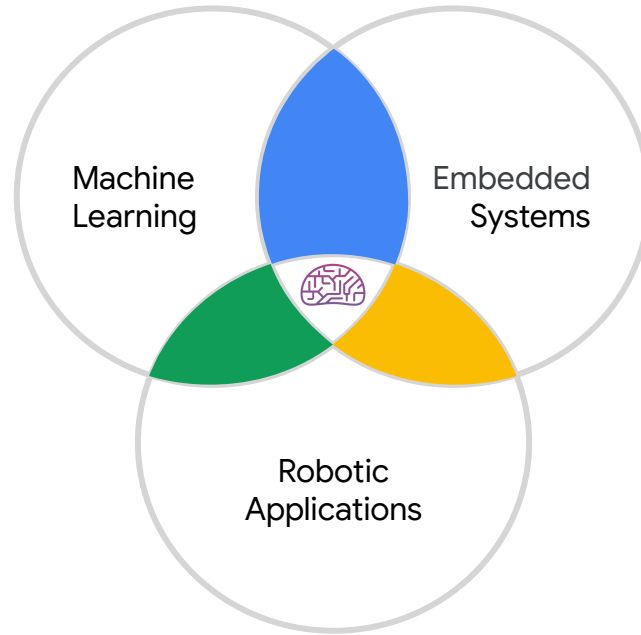
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner(s).

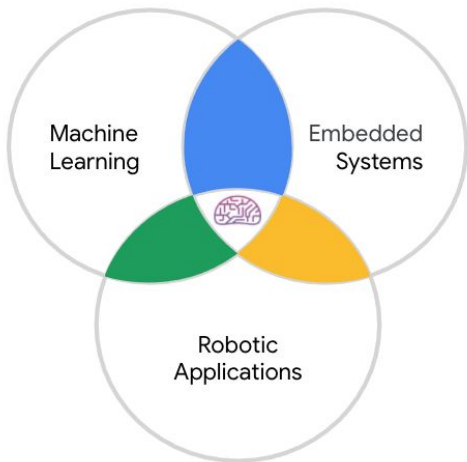
ASPLOS '21, April 19–23, 2021, Virtual, USA
© 2021 Copyright held by the owner(s).
ACM ISBN 978-1-4503-8317-2/21/04.
<https://doi.org/10.1145/3445814.3446746>

Neuman, S.M., Plancher, B., Bourgeat, T., Tambe, T., Devadas, S. and Reddi, V.J., 2021, April. Robomorphic computing: a design methodology for domain-specific accelerators parameterized by robot morphology. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

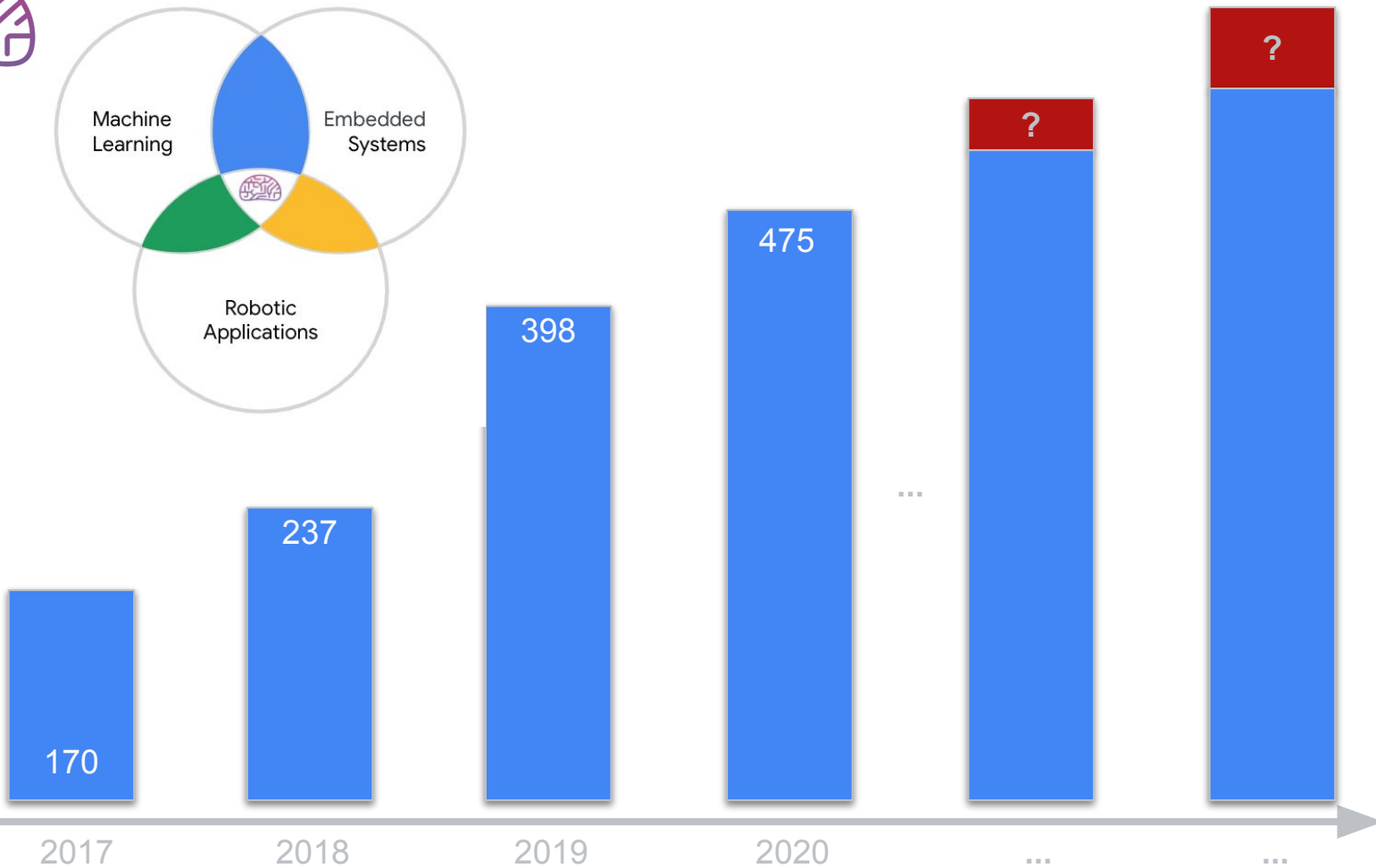


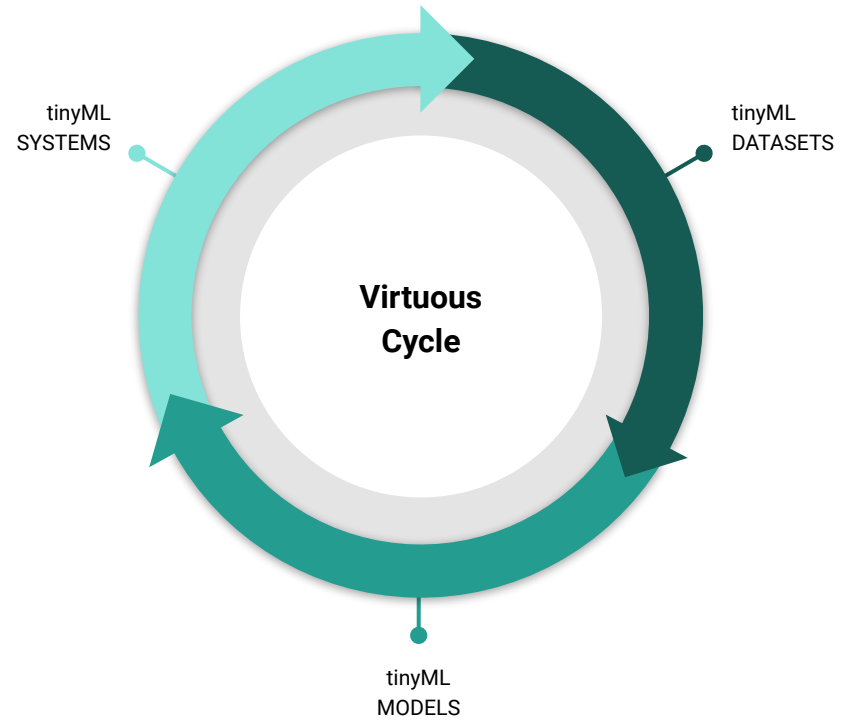


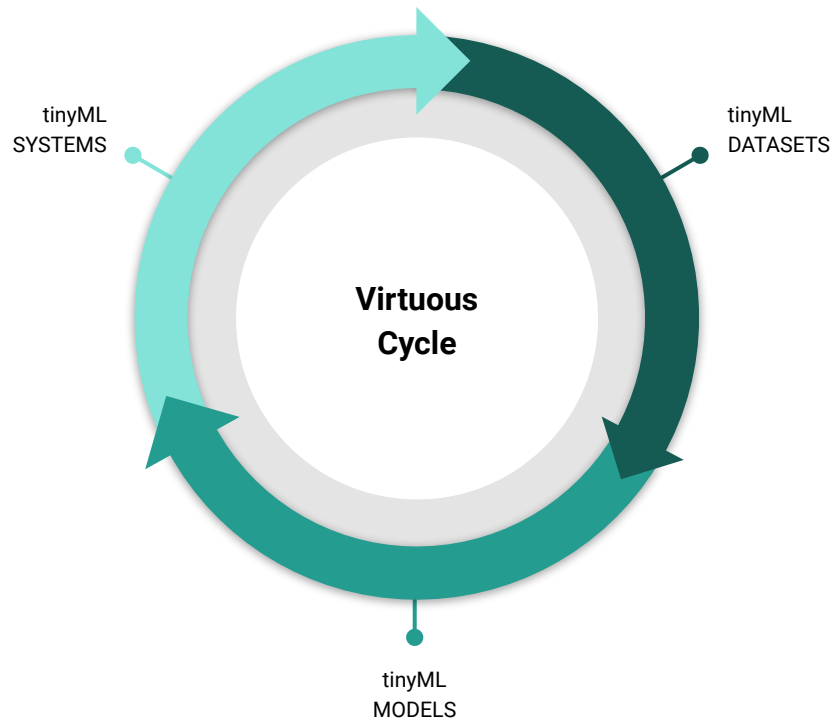


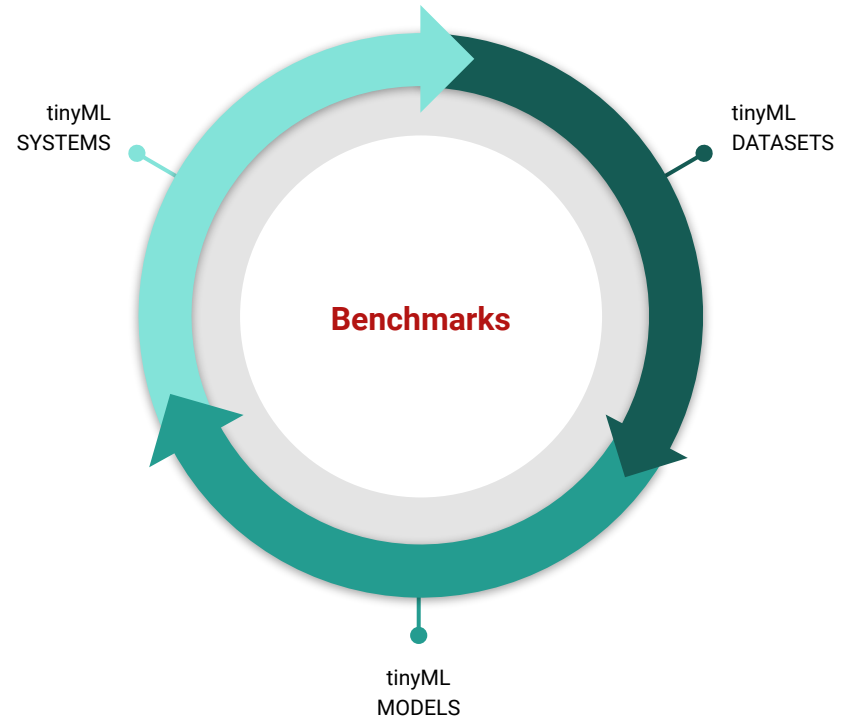


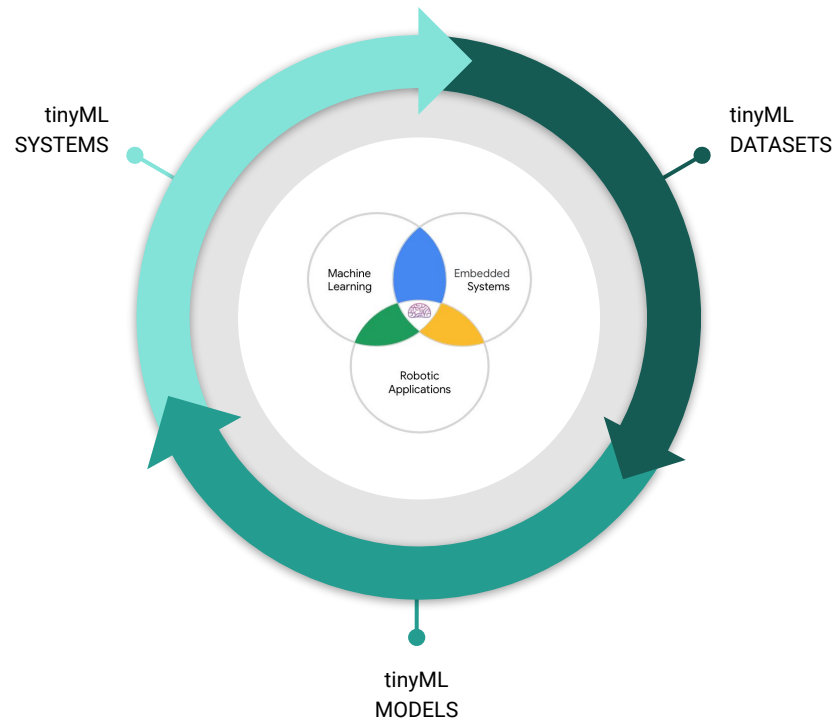
Number of Papers Submitted

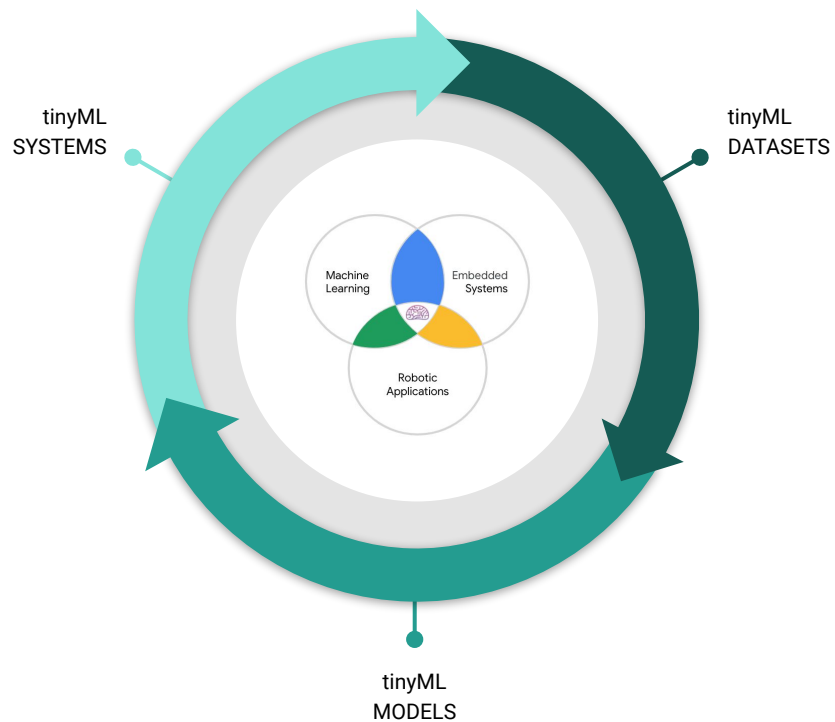














- Tiny machine learning (**tinyML**): ML applications on **low-power, cheap, commodity hardware**.
- Focus on **always-on machine learning use cases for robotics** with rich sensory input.
- **How can tinyML impact robotics?**



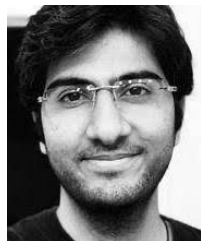
**Mark
Mazumder**



**Colby
Banbury**



**Brian
Plancher**



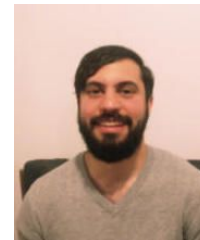
**Srivatsan
Krishnan**



**Bardienus
Duisterhof**



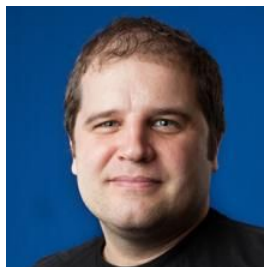
**Will
Fu**



**Behzad
Boroujerdian**



**Aleksandra
Faust**



**Pete
Warden**



**Laurence
Moreney**



The Future of
Robot Learning is
Tiny and Bright.