Squeezed Edge YOLO: Onboard Object Detection on Edge Devices

Edward Humes[1] **Mozhgan Navardi**[2] **Tinoosh Mohsenin**[2]
[1]University of Maryland, Baltimore County [2]Johns Hopkins University
ehumes2@umbc.edu {mnavard1, tinoosh}@jhu.edu

Abstract

Demand for efficient onboard object detection is increasing due to its key role in autonomous navigation. However, deploying object detection models such as YOLO on resource constrained edge devices is challenging due to the high computational requirements of such models. In this paper, an compressed object detection model named Squeezed Edge YOLO is examined. This model is compressed and optimized to kilobytes of parameters in order to fit onboard such edge devices. To evaluate Squeezed Edge YOLO, two use cases - human and shape detection - are used to show the model accuracy and performance. Moreover, the model is deployed onboard a GAP8 processor with 8 RISC-V cores and an NVIDIA Jetson Nano with 4GB of memory. Experimental results show Squeezed Edge YOLO model size is optimized by a factor of 8x which leads to 76% improvements in energy efficiency and 3.3x faster throughout.

1 Introduction

Interest in Machine Learning (ML) is dramatically increasing as it provides a promising solution for various applications such as autonomous navigation [1, 2]. Object detection models in particular can significantly assist in autonomous navigation by detecting obstacles and pre-defined objects of interest in the environment [3]. However, object detectors have high computational requirements due to the need for accuracy and the ability to detect various object categories. GPUs with significant computational capacity are often mandatory to train such complex models, yet onboard processing and edge computing necessitates low-power and low-computation algorithms as a result of the limited power and computational capacity available [4].

Object detectors are trained classifiers that can identify and locate multiple objects within an image. These detectors are trained on a set of annotated images, and their accuracy is evaluated on unseen datasets. There are two commonly used object detector paradigms: single-shot and two-shot. Single-shot-based methods such as You Only Look Once (YOLO) [5], Single Shot Detector (SSD) [6], etc., directly predict the class probabilities and Bounding Box (BBox) coordinates for objects in an image. In contrast, two-shot architectures such as R-CNN [7], Faster R-CNN [8], etc., generate a set of region proposals and then classify and refine them to output the final object detection. Moreover, two-shot object detection methods have several advantages over other methods, including robustness to scale and size variations, accurate localization, flexibility, and improved object recognition [9]. However, these advantages come at the expense of inference speed, with single-shot object detectors generally being faster than two-shot object detectors. Despite this, even single-shot objector models are difficult to deploy to resource constrained edge devices due to their high computational complexity. Therefore, it is important to improve object detection models to meet power consumption and real-time requirements on such devices [10, 11].

In recent years, researchers have presented optimized object detection models [10, 12, 13, 14, 15, 16, 17, 18] to enable onboard object detection on edge devices. Work in [13, 14, 15] proposed an

37th First Workshop on Machine Learning with New Compute Paradigms at NeurIPS 2023(MLNPCP 2023).

optimized model for collision avoidance and obstacle detection to enable autonomous navigation for a tiny drone with a GAP8 processor. An Automatic License Place Recognition (ALPR) is proposed on [18] which leverages a GAP8 processor. TinyissimoYOLO is proposed in [17] which is deployed on different microcontrollers, achieving an accuracy of 58.5% for three objects. In [10], an optimized YOLO model, EdgeYOLO, is presented that achieves a reduction in model size by about 10x from around 250 MB to 25 MB. They deployed their proposed model on an NVIDIA Jetson board for system-level verification. In [16] a metareasoning framework is proposed for an efficient goal-oriented autonomous navigation by switching between an optimized YOLO model, Squeezed Edge YOLO, and a lighter Convolutional Neural Network (CNN) based model.

In this work, the focus is on the Squeezed Edge YOLO [16] with detailed implementation and more experimental results. Then, we compare the model with the state-of-the-art work in [10] which have similar accuracy when we deployed both models on a JetBot equipped with an NVIDIA Jetson Nano board to show the latency and energy improvement. Moreover, we deploy the Squeezed Edge YOLO model on extremely resource constrained devices such as the GAP8 processor while meeting real-time requirements. The main contribution of this work is as follows:

- A detailed implementation of an energy efficient Squeezed Edge YOLO as a tiny Machine Learning (tinyML) model is provided in this work for tiny edge devices.
- Onboard YOLO-based model deployment and edge computing on resource constrained edge devices such as the GAP8 and NVIDIA Jetson Nano.
- Power and latency analysis on GAP8 processor and NVIDIA Jetson Nano as a result of the memory hierarchy and onboard data transmission overhead.

To evaluate the Squeezed Edge YOLO[16], we deployed it on the GAP8 processor. Moreover, we compared the throughput and energy efficiency of the model on an NVIDIA Jetson Nano board with state-of-the-art-works. Experimental results show a 3.3x faster throughput while achieving a 76% energy efficiency improvement.

2 Proposed Approach

In this section, we describe challenges in deploying object detection neural networks to resource-constraint edge devices as a motivation. Then, we propose a hardware-aware optimized object detection model named Squeezed Edge YOLO for onboard processing.

2.1 Motivation

While we were able to deploy EdgeYOLO [10] to the Jetson Nano, our attempt to deploy the base EdgeYOLO model onboard the GAP8 resulted in failure. Due to the model's size, it would overflow the limited stack space available on the GAP8 and crash. Therefore, we applied hardware-aware model optimizations by profiling EdgeYOLO, and working to reduce memory consumption while attempting to retain an acceptable level of inference accuracy. For this aim, in the next section we discussed relevant hardware features and presented an optimized Squeezed Edge YOLO.

2.2 GAP8 Hardware Architecture

The GAP8 is a microcontroller designed specifically with tinyML in mind [19], and as shown in Figure 1, it contains one main processor core capable of running at up to 250 MHz, and an octacore cluster capable of running at up to 175 MHz. In addition, it contains a variety of hardware accelerators including a convolution engine, as well as a 3-layer memory hierarchy composed of 64KB of L1 RAM, 512 KB of L2 RAM, and an off-chip L3 memory composed of many megabytes of RAM and flash. Past work with the GAP8 has demonstrated that it is primarily constrained by its memory hierarchy [20], rather than its computational capabilities. Most modern microprocessors go to great lengths to mask memory latency, such as by having multiple layers of caches on top of RAM, hardware-prefetching, out of order execution, etc. The GAP8's memory hierarchy is entirely software-managed, L1 memory is the fastest, needing only a single cycle to access, L2 memory takes multiple cycles to access, while L3 memory is the slowest - it is not memory mapped, requiring Direct Memory Access (DMA) transfers from the off-chip L3 RAM into L2 memory in order

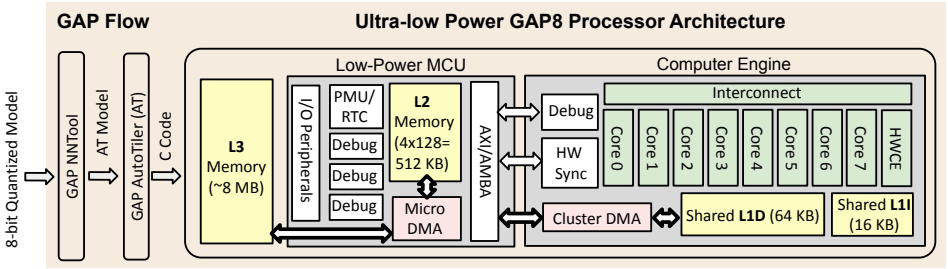


Figure 1: GAP8 processor architecture and custom GreenWaves toolchain. As the figure shows, the GAP8 has a variety of hardware features in order to make the most of its limited power budget and be able to run neural networks on the edge [19, 21, 16].

Table 1: Backbone feature extraction [16]

Layer	Type	Size/Stride	Filters	Output
0	Conv	3x3/2	16	64x64x16
1	Conv	3x3/2	32	32x32x32
2	SE	-	-	32x32x32
3	Conv	3x3/1	16	32x32x16
4	SE	-	-	32x32x16
5	Conv	3x3/1	16	32x32x16
6	SE	-	-	32x32x16
7	Route	Input: [6, 4]		32x32x32
8	Conv	3x3/1	32	32x32x32
9	SE	-	-	32x32x32
10	Route	Input: [2, 9]		32x32x64
11	SE	-	-	32x32x64
12	MaxPool	2x2/2		16x16x64

Table 2: Neck feature enhancement [16]

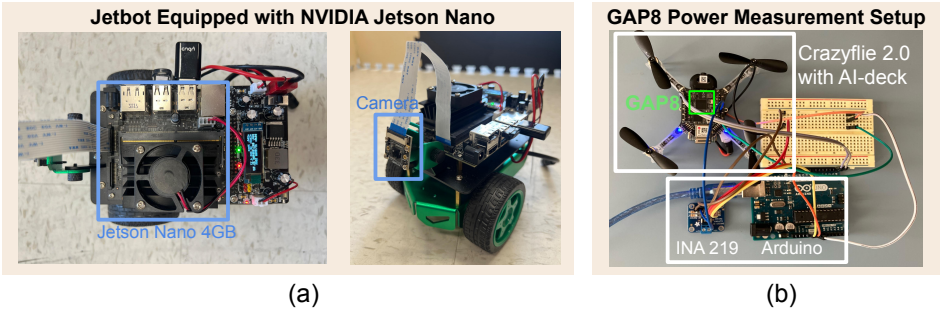
Layer	Type	Size/Stride	Filters	Output
0	Conv	3x3/1	64	16x16x64
1	Conv	3x3/2	128	8x8x128
2	SE	-	-	8x8x128
3	Conv	1x1/1	256	8x8x256
4	SE	-	-	8x8x256
5	Conv	1x1/1	256	8x8x256
6	Conv	3x3/1	128	8x8x128
7	Route	Input: [5,]		8x8x256
8	Conv	3x3/1	64	8x8x128
9	Upsample	2x2	-	16x16x128
10	Conv	1x1/1	64	16x16x64
11	Route	Input: [10, 0]		16x16x128
12	Conv	3x3/1	128	16x16x128
13	SE	-	-	16x16x128
14	Conv	1x1/1	128	16x16x128
15	SE	-	-	16x16x128
16	Conv	3x3/1	64	16x16x64
17	Detect	Input: [6, 16]		

to read and modify data stored within. Thus, it is the responsibility of applications to arrange their data in an optimal layout around the memory hierarchy to minimize stalls by the GAP8's in-order processor cores and accelerators.

2.3 Squeezed Edge YOLO

Given this information, our first step was to shrink EdgeYOLO's original input image size specifically when running on the GAP8. As the AI-deck expansion board (which includes the GAP8 processor) can only capture images at a resolution of 324x244, we shrunk the input image size from 416x416x3 to 128x128x3. Besides this though, 128x128x3 images only require 49152 bytes, or less than the whole of L1 memory. In practice, the image will almost never be stored in its entirety within L1 RAM - network parameters also need to be stored in L1 memory for efficient retrieval, although the smaller size does make moving various data tiles much easier. We briefly considered using a higher image resolution, albeit in gray scale, for object detection, however we ended up deciding against this, given that we wanted to identify specific colored objects for reinforcement learning purposes. Fortunately, the smaller size still manages to achieve a good trade-off between inference latency and accuracy for object detection.

More broadly, besides input image size, we focused our attention on other aspects contributing to high memory usage, which more often than not was a combination of convolutional layers and residual blocks. By default, the base Edge YOLO model produces many output channels for each convolutional layer, while this assists feature extraction in images, these output channels must be stored somewhere. Besides the large number of channels, another challenge for the GAP8 with these convolutional layers is the size of the weights required. While 1x1 convolutions are not as great for extracting features present within an image, they require 9x less weights than a 3x3 convolution. Thus, as depicted in Table 1 and 2, where possible, we attempted to either reduce the number of



(a) (b)

Figure 2: A JetBot used for experimentation as well as a power measurement setup for the GAP8. (a) The JetBot contains a Jetson Nano as well as various peripherals including a camera. (b) The INA 219 and Arduino are used in this paper to measure the GAP8 processor power consumption when the model is deployed onboard.

input and output channels present in EdgeYOLO convolutional layers, or if that was still not enough, switch to using 1x1 convolutional layers.

Residual blocks [22] are quite common in modern computer vision networks, as they both ameliorate the problem of vanishing gradients, and reduce total parameter count. However, despite using less parameters than a network with many layers stacked over each other, they are taxing on the GAP8's memory hierarchy. This is because in addition to the weights and activations of the current neural network layer being operated on, intermediate activations in skip connections need to be stored somewhere in memory until they are fed into another layer. The base EdgeYOLO model, and most modern YOLO implementations generally speaking, often contain deeply nested skip connections covering dozens of layers, some of which contain their own skip connections. Shuffling these around the memory hierarchy in tandem with the regular execution of the neural network is challenging, and so we removed many residual blocks, and simplified others to only covering a few layers. In addition, we removed one of the three detection heads; we figured that as the images we were operating on were already quite small, and the detection heads are primarily for performing object detection at different scales, it would be somewhat redundant. By extension, this also removed the longest skip connection, along with its associated inner skip connections.

3 Experimental Results

In this section, we describe our evaluation of Squeezed Edge YOLO deployed on two target edge devices: an NVIDIA Jetson Nano with 4GB of RAM shared between the CPU and GPU, and an AI-deck with a GAP8 microcontroller onboard. Moreover, we specifically focus on: inference latency on each processor, power consumption, and inference accuracy in comparison with the state-of-the-art-work in [10].

3.1 Experimental Setup

For experiments, we used a JetBot with an NVIDIA Jetson Nano containing 4GB of RAM as shown in Figure 2 (a), as well as a Crazyflie drone with an AI-deck expansion board, which contains a GAP8 microcontroller. The Jetson Nano contains hardware onboard for measuring the power draw of the Tegra X1's CPU and GPU cores. Meanwhile, the GAP8 required the use of external hardware. Figure 2 (b) shows an INA219 and an Arduino connected to the AI-deck's power pins in order to measure average power usage during the runtime of the network. As we were primarily interested in autonomous navigation, we created a custom dataset for testing, composed of 1500 images of various shapes captured from our deployment platforms. We additionally created a separate mixed dataset of human images using 3500 images taken from CrowdHuman [23], and 3500 more images we captured using the AI-deck.

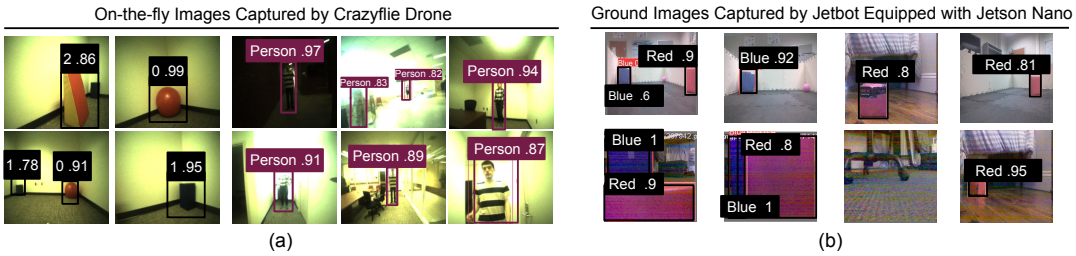


Figure 3: (a) Squeezed Edge YOLO object detection images (first two column), along with human detection images (last three column) captured using the Crazyflie's AI-deck expansion board. (b) Object detection images captured using the RGB camera on the JetBot. The overlaid labels show the classification and detected accuracy by Squeezed Edge YOLO model.

Table 3: For a fair comparison, models are re-implemented. The size, power consumption, and inference latency of YOLOv5s [9], EdgeYOLO [10], and the Squeezed Edge YOLO [16] is extracted. Squeezed Edge YOLO is successfully deployed to the GAP8 processor which is about 8x smaller than YOLOv5s. The shapes dataset is used for this experiments.

Model	Parameters	Accuracy	Model Size	Power	Inference Latency
YOLOv5s [9]	7.3 M	0.96 mAP	237 Mb (32-bit)	N/A	N/A
EdgeYOLO [10]	8.1 M	0.99 mAP	65 Mb (8-bit)	Failed due to memory limitations	
Squeezed Edge YOLO [16]	**931 K (∼8x smaller)**	**0.95 mAP**	**7.5 Mb (8-bit)**	**541 mW**	**130 ms**

3.2 Model Optimization and Onboard Object Detection

Once we had our final object detection model, it achieved an acceptable inference latency for our purposes aboard the AI-deck: roughly 8 inferences a second, displayed in Table 3, and as shown in Figure 3 achieving an acceptable level of accuracy for object detection tasks. Upon examining hardware utilization in the cycle-accurate GAP8 simulator, named GVSOC [24], we found it had relatively low micro-DMA usage as a percentage of total runtime, as well as few CPU stalls on the cluster cores. This indicates that the memory hierarchy is being used effectively, the cores are being kept busy with crunching through the neural network, as the data is where it needs to be when the processor begins operating on it.

3.3 Onboard Model Implementation

In this section, we describe the steps involved in deploying both EdgeYOLO and the Squeezed Edge YOLO model we developed to the target hardware platforms, the GAP8 and Jetson Nano. Additionally, the process of extracting power and latency measurements is described.

GAP8 Processor. GreenWaves Technologies [19], the company that developed the GAP8, publishes a custom toolchain [25] to ease deploying neural networks to the GAP8. This toolchain, depicted in Figure 4 consists of a modified GCC compiler with GAP8 support added in, the Neural Network Tool (NNTOOL), and AutoTiler. NNTOOL is responsible for converting adjusting ONNX and TFLite models to replace unsupported operations, as well as to export the weights into a ROM image. AutoTiler, takes care of converting the models execution graph into C code making use of optimized compute kernels, as well as tiling to manage memory transfers for memory locality.

The GAP8 does not contain any floating point units [20], and so we quantized the Squeezed Edge YOLO model to 8-bit integers, which also has the side effect of reducing RAM usage. We then used GreenWaves' toolchain to convert the model into C code which we then compiled and benchmarked both in GVSOC and on the AI-deck. As GVSOC is designed to be a cycle-accurate emulator of the GAP8, we were able to extract performance info, as well as hardware utilization in the form of VCD traces, whereas deployment to the AI-deck allowed extracting energy utilization. The VCD traces found in Figure 4 showed that Squeezed Edge YOLO made efficient use of the memory hierarchy. While the much larger, and much slower Squeezed Edge YOLO models that we designed, made heavy use of L3 RAM.

NVIDIA Jetson Nano 4GB. To show the efficiency of the Squeezed Edge YOLO in comparison with the proposed model in [10], we deployed both models on the NVIDIA Jetson Nano as EdgeY-

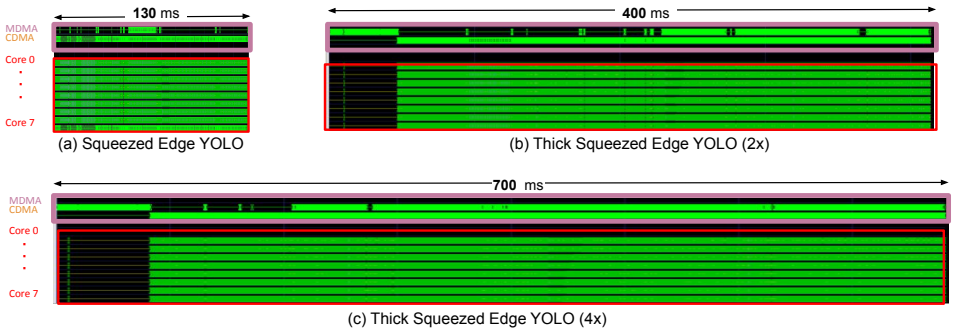


Figure 4: Inference phase recorded in VCD traces for different size of Squeezed Edge YOLO model on GAP8: (a) Squeezed Edge YOLO [16] (b) an enlarged Squeezed Edge YOLO (2x parameters) (c) an enlarged Squeezed Edge YOLO (4x parameters). The green portion represents hardware utilization. Cluster-DMA (CDMA) and Micro-DMA (MDMA) are responsible for data transfer between L1, L2 and L3 memories in GAP8, respectively.

Table 4: Hardware implementation results that compare Squeezed Edge YOLO with the work presented in [10]. As EdgeYOLO was unable to be deployed to the GAP8, we used the Jetson Nano as our baseline hardware for comparison purposes.

Metric/Approach	EdgeYOLO [10]	Squeezed Edge YOLO [16]
Inference Latency (ms)	231	70
Throughput (Inference/Sec)	4.2	**14.2 (3.3x faster)**
CPU Power Consumption (mW)	777	1158
GPU Power Consumption (mW)	3846	2434
Performance (GOPS)	198.2	94.3
Energy Efficiency (GOPS/J)	185.7	**788.5**
Energy/Inference (mJ)	1068	**251.5 (76% improvement)**

OLO does not meet GAP8 resource constraints. The Jetson Nano is much closer to the workstations and servers typically used for training and deploying machine learning models, given that it has an ARMv8 CPU, and NVIDIA Maxwell architecture GPU with CUDA support, along with supporting both FP16 and FP32 representations. Thus, it can directly execute PyTorch and TensorFlow models on both its CPU and GPU. However, NVIDIA provides a machine learning framework named TensorRT [26], which applies GPU-specific optimizations in order to boost the performance of models during execution. Thus, we converted our models to TensorRT before executing them on-device. Additionally, as the Jetson Nano does not face the same memory constraints as the GAP8, we tested all models aboard the Jetson Nano using a 3 channel 416x416 input image size. The Jetson Nano is able to monitor power going to both the CPU and GPU respectively, obviating the need for the same setup we used for measuring power on the GAP8 as reported in Table 4. Experimental results show Squeezed Edge YOLO used less CPU and GPU power consumption in comparison EdgeYOLO [10].

4 Conclusion

In this work, we examined an optimized YOLO model named Squeezed Edge YOLO for onboard processing on resource constrained edge devices containing limited sources of power and computational capacity. In order to achieve this, a baseline YOLO model was optimized in terms of its number of parameters and computations to reduce the model size. Evaluation was carried out on a drone known as the Crazyflie with a GAP8 processor containing 8 RISC-V cores, and a JetBot with an NVIDIA Jetson Nano containing 4 GB of memory. The experimental results show that Squeezed Edge YOLO is about 8x smaller than existing edge focused YOLO model. We measured power consumption and latency upon deploying Squeezed Edge YOLO to the target devices, and found that by optimizing the YOLO model, it achieves a 76% and 3.3x energy efficiency and throughput improvement on the Jetson Nano board.

Acknowledgments and Disclosure of Funding

This project was sponsored by the U.S. Army Research Laboratory under Cooperative Agreement Number W911NF2120076.

References

[1] Viktor Rausch, Andreas Hansen, Eugen Solowjow, Chang Liu, Edwin Kreuzer, and J Karl Hedrick. Learning a deep neural net policy for end-to-end control of autonomous vehicles. In *2017 American Control Conference (ACC)*, pages 4914–4919. IEEE, 2017.

[2] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, 22(2):712–733, 2020.

[3] Lorenzo Lamberti, Luca Bompani, Victor Javier Kartsch, Manuele Rusci, Daniele Palossi, and Luca Benini. Bio-inspired autonomous exploration policies with cnn-based object detection on nano-drones. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.

[4] Arnab Neelim Mazumder, Farshad Safavi, Maryam Rahnemoonfar, and Tinoosh Mohsenin. Reg-tune: A regression-focused fine-tuning approach for profiling low energy consumption and latency. *ACM Transactions on Embedded Computing Systems*, 2023.

[5] Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai, and Bo Ma. A review of yolo algorithm developments. *Procedia Computer Science*, 199:1066–1073, 2022.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[8] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[9] Tejaswini Manjunath, Mozhgan Navardi, Prakhar Dixit, Bharat Prakash, and Tinoosh Mohsenin. Reprohrl: Towards multi-goal navigation in the real world using hierarchical agents. *On 37th AAAI Conference on Artificial Intelligence, The 1st Reinforcement Learning Ready for Production workshop*, 2023.

[10] Siyuan Liang, Hao Wu, Li Zhen, Qiaozhi Hua, Sahil Garg, Georges Kaddoum, Mohammad Mehedi Hassan, and Keping Yu. Edge yolo: Real-time intelligent object detection system based on edge-cloud cooperation in autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):25345–25360, 2022.

[11] Arnab Neelim Mazumder and Tinoosh Mohsenin. Reg-tunev2: Hardware-aware and multi-objective regression-based fine-tuning approach for dnns on embedded platforms. *IEEE Micro*, 2023.

[12] Rachel Huang, Jonathan Pedoeem, and Cuixian Chen. Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers. In *2018 IEEE international conference on big data (big data)*, pages 2503–2510. IEEE, 2018.

[13] Mozhgan Navardi, Edward Humes, and Tinoosh Mohsenin. E2edgeai: Energy-efficient edge computing for deployment of vision-based dnns on autonomous tiny drones. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, pages 504–509. IEEE, 2022.

[14] Antonio Loquercio, Ana I Maqueda, Carlos R Del-Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018.

[15] Daniele Palossi et al. A 64-mw dnn-based visual navigation engine for autonomous nano-drones. *IEEE Internet of Things Journal*, 6(5):8357–8371, 2019.

[16] Mozhgan Navardi, Edward Humes, Tejaswini Manjunath, and Tinoosh Mohsenin. Metae2rl: Toward metareasoning for energy-efficient multi-goal reinforcement learning with squeezed edge yolo. *IEEE Micro*, 2023.

[17] Julian Moosmann, Marco Giordano, Christian Vogt, and Michele Magno. Tinyissimoyolo: A quantized, low-memory footprint, tinyml object detection network for low power microcontrollers. *arXiv preprint arXiv:2306.00001*, 2023.

[18] Lorenzo Lamberti, Manuele Rusci, Marco Fariselli, Francesco Paci, and Luca Benini. Low-power license plate detection and recognition on a risc-v multi-core mcu-based vision system. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[19] Eric Flamand, Davide Rossi, Francesco Conti, Igor Loi, Antonio Pullini, Florent Rotenberg, and Luca Benini. Gap-8: A risc-v soc for ai at the edge of the iot. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–4. IEEE, 2018.

[20] Edward Humes and Tinoosh Mohsenin. Optimizing deep neural network architectures for low-power autonomous tiny uav navigation. *UMBC Review*, 2023.

[21] GreenWavesTechnologies. Gap8 processor architecture. [online]. *Available: https://greenwaves-technologies.com/manuals/BUILD/HOME/html/index.html*.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Shuai Shao, Zijian Zhao, Boxun Li, Tete Xiao, Gang Yu, Xiangyu Zhang, and Jian Sun. Crowdhuman: A benchmark for detecting human in a crowd. *arXiv preprint arXiv:1805.00123*, 2018.

[24] GVSOC Virtual Platform. *Available: https://greenwaves-technologies.com/manuals/BUILD/GVSOC/html/index.html*.

[25] GAP8 Manual greenwaves-technologies.com. https://greenwaves-technologies.com/manuals/BUILD/HOME/html/index.html. [Accessed 10-04-2023].

[26] NVIDIA TensorRT developer.nvidia.com. https://developer.nvidia.com/tensorrt/. [Accessed 10-04-2023].