MetaTinyML: End-to-End Metareasoning Framework for TinyML Platforms

Mozhgan Navardi, Edward Humes, Tinoosh Mohsenin

***Abstract*—Efficiently deploying deep neural networks on resource-limited embedded systems is crucial to meet real-time and power consumption requirements. Utilizing metareasoning as a higher-level controller along with Tiny Machine Learning (TinyML) can enhance energy efficiency and reduce latency on such systems by overseeing available resources. This study introduces MetaTinyML, a comprehensive metareasoning framework for self-guided navigation on TinyML platforms. The framework adapts its decision-making process by factoring in environmental changes to select the most suitable algorithms for the current scenario. Implementation of MetaTinyML on an NVIDIA Jetson Nano 4GB system integrated with a Jetbot ground vehicle demonstrated up to 50% power consumption enhancement. View a video demonstration of the MetaTinyML framework at: Video.**

***Index Terms*—Metaresoning, TinyML, Embedded Systems, Edge Computing, Autonomous Systems.**

I. Introduction

TINY MACHINE LEARNING (TinyML), as an edge computing concept, is a rapidly expanding field that establishes a connection between embedded systems and Machine Learning (ML) by effectively optimizing hardware and software components. This optimization enables battery-powered smart devices to execute ML inferences via Deep Neural Networks (DNN) or Reinforcement Learning (RL) algorithms. The primary goal of TinyML is the deployment of ML inference on extremely low power devices with limited sources of onboard memory to process collected data from on-device sensors without necessitating communication with cloud devices [1]–[5]. By moving ML inference to the edge, there are multiple key challenges that must be addressed including latency and throughput to meet real-time requirements, power consumption and energy efficiency, all while maintaining an acceptable level of accuracy [1].

Figure 1 shows the TinyML design flow, which eliminates the necessity for any external processing resources, leading to both improved real-time decision-making capabilities and minimal latency: a critical attribute for applications characterized by real-time applications, such as self-driving automobiles and Unmanned Ground Vehicles (UGV). However, energy efficiency and latency in terms of hardware and accuracy in terms of software are key challenges that need to be addressed when applying TinyML to autonomous systems.

Mozhgan Navardi and Tinoosh Mohsenin are the Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD 21218 USA (e-mail: mnavard1@jhu.edu; tinoosh@jhu.edu).
Edward Humes is with the Department of Computer Science and Electrical Engineering, Baltimore, MD, 21250 USA (email: ehumes2@umbc.edu).

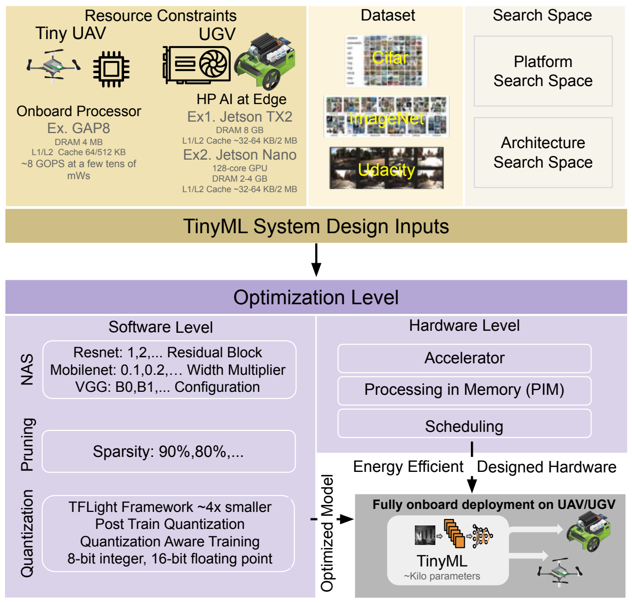
Resource Constraints
Tiny UAV
UGV
Onboard Processor
Ex. GAP8
DRAM 4 MB
L1/L2 Cache 64/512 KB
~8 GOPS at a few tens of mWs
HP AI at Edge
Ex1. Jetson TX2
DRAM 8 GB
L1/L2 Cache ~32-64 KB/2 MB
Ex2. Jetson Nano
128-core GPU
DRAM 2-4 GB
L1/L2 Cache ~32-64 KB/2 MB
Dataset
Cifar
ImageNet
Udacity
Search Space
Platform Search Space
Architecture Search Space
TinyML System Design Inputs
Optimization Level
Software Level
NAS
Resnet: 1,2,... Residual Block
Mobilenet: 0.1,0.2,… Width Multiplier
VGG: B0,B1,... Configuration
Pruning
Sparsity: 90%,80%,...
Quantization
TFLight Framework ~4x smaller
Post Train Quantization
Quantization Aware Training
8-bit integer, 16-bit floating point
Hardware Level
Accelerator
Processing in Memory (PIM)
Scheduling
Energy Efficient Designed Hardware
Optimized Model
Fully onboard deployment on UAV/UGV
TinyML
~Kilo parameters

Fig. 1. TinyML design flow overview includes TinyML system design inputs (resource constraints, dataset and search space) and optimization level in software and hardware [1].

Metareasoning [6]–[12] can be utilized for efficient deployment of ML algorithms by switching between different algorithms in real-time based on environmental changes. Thus, real-time tasks on embedded system can be scheduled in such a way that reliability target levels are met while reducing power/energy consumption. Metareasoning consists of a ground level, an object level, and a meta-level. The ground and object levels can be likened to the environment and agent components in reinforcement learning, with the meta-level monitoring the object level to guide real-time adjustments in the agent's behavior.

In this paper, we propose a novel approach to improve the latency and power consumption of RL and DNN models while maintaining acceptable accuracy levels. We build upon the existing state-of-the-art works that have suggested software and hardware level techniques to optimize these models. However, unlike previous approaches that focused solely on deploying highly computationally intensive models onto resource-constrained devices, the proposed approach takes into account environmental changes and leverages metareasoning to make online situationally-aware decisions. Metareasoning is an online decision-making approach that allows us monitoring of the agent and changes in the environment in real-time

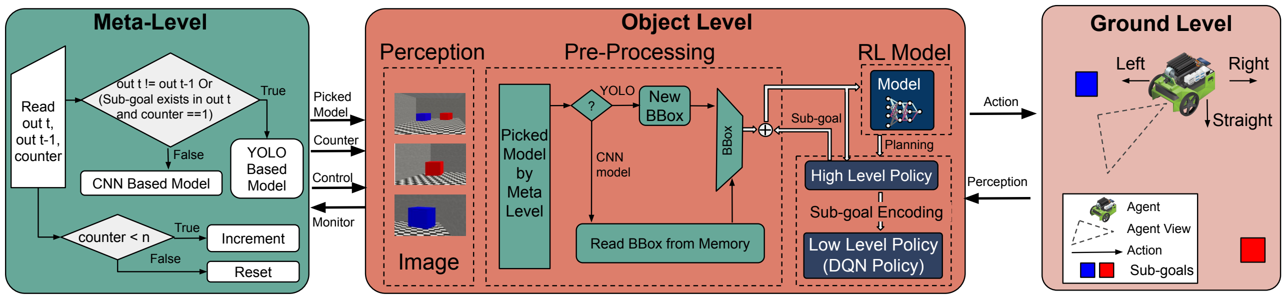

Fig. 2. A high-level diagram for the proposed energy-efficient (E2) **Meta**reasoning framework for **TinyML** platforms: MetaTinyML. Based on the proposed meta policy, a complex model, YOLO based, or a lighter model, CNN based, will be picked for energy efficient object detection and autonomous navigation.

and thus select the most suitable algorithm accordingly [13]. To the best of our knowledge, no prior work has explored the application of metareasoning for improving the power consumption of TinyML models in an end-to-end deployment scenario. To address this gap, we introduce a meta policy in this paper, leading to significant improvements in throughput (inferences per second) and energy efficiency (GOPS/J) in goal-oriented RL-based autonomous navigation systems.

II. Metareasoning in TinyML Platforms

A. Framework Overview

Fig. 2 depicts a high-level diagram for the proposed MetaTinyML: a **Meta**reasoning framework in **TinyML** platforms which has a k-goal Reinforcement Learning (RL) model for autonomous navigation and reaching k goals in a specific order within an environment. The proposed framework includes three main parts: (1) ground level (environment), object level (agent), and meta level (metareasoning). In the ground level, the agent performs its assigned mission and sends its state (captured image) in each step to the object level in order to determine the next action. Within the object level, an RL model with a high level and low level policy is responsible for agent navigation and giving the next action. To improve the RL model's performance, rather than feeding the whole image as input to the RL model, a simplified input of a 1xk+(4+k) vector will be passed to the RL model. In order to generate the RL model input, a pre-processing module including an object detection model and multi-label image classification model is used in this framework. In each step, one of these models will be picked by the meta-level module to process the captured image. If the YOLO object detection model processes the image, new generated Boundary Boxes (BBox) coordinates will be saved in memory, otherwise the previous BBox will be kept in memory. The RL model input is the BBox (4xk) concatenated with the generated sub-goal (1xk).

In each step, the meta-level monitors the agent and any environmental changes in order to switch between the YOLO and CNN models. To do this, we update four values in memory: the BBoxes, the new model output (out t), model output's previous step (out t-1) and a counter. The meta-level reads out t, out t-1, and the counter. Based on the proposed meta policy, it will then pick the YOLO or the CNN model. The goal is to maximize the number of times the CNN model is picked, as it is a lighter and faster model, resulting in increased energy-efficiency and higher throughput.

B. Proposed Meta Policy

For the meta-level, we proposed a meta policy to efficiently switch between a lighter CNN based object detection model, and a more intensive YOLO model. The CNN model provides a list of detected objects within the captured image, but lacks information on object locations or BBoxes. However, BBoxes are crucial for goal-oriented navigation in an RL model as they help determine the distance between the agent and the goal. Therefore, we cannot completely replace the YOLO model with the CNN model, however, we can decrease the number of times that the YOLO model must be run. To do this, we added a meta-level to the framework to monitor the agent and any environmental changes to determine whether we need a new BBox, or can simply rely on the previous generated BBox can be used. The meta-level will switch between these two models and try to pick the CNN model as much as possible based on the proposed meta policy.

Fig. 2 depicts the proposed MetaTinyML policy's meta-level. The intuition behind MetaTinyML is to utilize the previous BBox if there is a minor change in the agent's view, as the newly captured BBox would be similar to the previous one. By comparing the last two object detection model outputs saved in memory, MetaTinyML decides whether to use the CNN model with the previous BBox or to pick the YOLO model for generating a new BBox. On the other hand, if the detected objects in the last two outputs are not the same or the goal is in the detected objects in the last output we can pick the YOLO model to generate a new BBox. To further enhance the MetaTinyML policy, a counter n can be incorporated to optimize the frequency of running the YOLO model based on the detection of the goal within the model outputs. Therefore, we do not need to pick the YOLO model each time the goal is detected within the model outputs, we instead run YOLO every n steps. The selection of an optimal counter value is crucial to maintain a balance between efficiency and performance in reaching the goal. To achieve even further power consumption improvement, we added a small sleep time after each execution of the lighter CNN model. Additionally, introducing a small sleep time after executing the lighter CNN model results in a notable improvement in power consumption called as MetaTinyML-sleep.

TABLE I
END-TO-END POWER CONSUMPTION AND TIME TO COMPLETION FOR REACHING THE GOAL ON A JETBOT BY A NVIDIA JETSON NANO 4GB BOARD. TWO VERSIONS OF THE PROPOSED METATINYML ARE COMPARED WITH STATE-OF-THE-ART WORK NAMED REPROHRL [15] AND METAE2RL [16].

Model	Freq. (MHz)	Power Consumption (W)		Time (Sec)
		CPU	GPU	
ReProHRL [15]	High (GPU: 921, CPU: 1479)	1.3	2.4	95
MetaE2RL [16]		1.3	2.4	179
MetaTinyML		1.0	2.6	60
MetaTinyML-Sleep		**0.8 (40% impr.)**	**1.2 (50% impr.)**	151
ReProHRL [15]	Low (GPU: 640, CPU: 918)	0.7	1.2	133
MetaE2RL [16]		0.7	1.2	250
MetaTinyML		0.6	1.1	95
MetaTinyML-Sleep		**0.5 (26% impr.)**	**0.7 (44% impr.)**	173

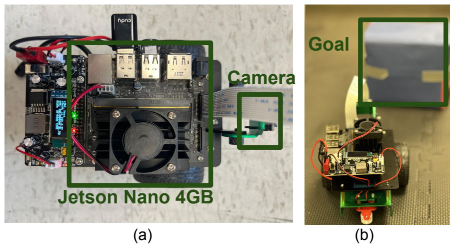

Fig. 3. (a) Experiment setup with a robotic Jetbot UGV equipped by NVIDIA Jetson Nano 4GB. (b) UGV agent in action trained to reach the blue box [14].

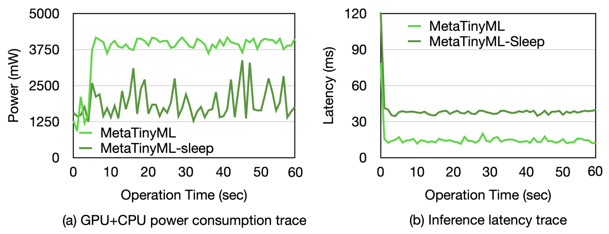

Fig. 5. Power consumption and inference latency trace while deploying MetaTinyML and MetaTinyML-sleep on Jetson Nano 4GB

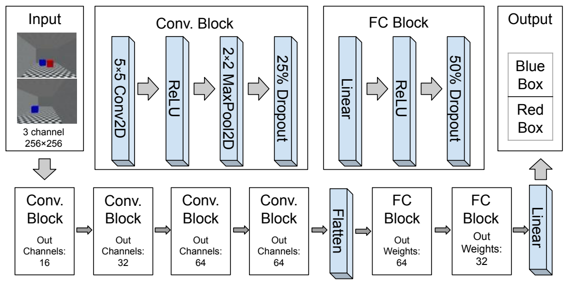

Fig. 4. Classification CNN-based light model diagram architecture.

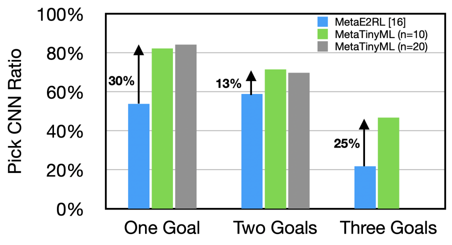

Fig. 6. Picked CNN model/YOLO model ratio during training the RL model. The proposed approach, MetaTinyML, picks the lighter model, up to 80% for processing the captured image which is 30% more than MetaE2RL [16].

III. EXPERIMENTAL RESULTS

A. Experimental Setup

A Jetbot equipped with a Jetson Nano featuring 4GB of memory, as depicted in Fig. 3, is utilized for real-world application. We have developed two models for detecting target objects within the environment: one utilizing YOLOv5n and the other a simple CNN-based light model. The architecture of the CNN-based light model is illustrated in Fig. 4. For the light model, which focuses on object presence detection, we utilized a 3-channel 256x256 image resolution. In contrast, the YOLOv5n model, designed to run less frequently and tasked with recording object bounding boxes, utilizes a larger input image size of 416x416. Both models were trained using a custom dataset comprising images of colored shapes captured in simulation and in real-world scenarios.

B. End-to-End MetaTinyML Result

Hardware Results. We deployed the end-to-end MetaTinyML on on a Jetbot equipped with a Jetson Nano containing 4GB of RAM and compared it with a state-of-the-art work named ReProHRL [15]. Table I results show power consumption and time to task completion for ReProHRL with no meta policy, MetaE2RL [16], MetaTinyML, MetaTinyML-sleep. Based on the provided results, MetaTinyML-sleep is the most energy-efficient approach, however task completion time is increased in comparison to other versions due to the added sleep time. Applying metareasoning in TinyML autonomous navigation leads to up to 50% energy improvement in comparison with the proposed approach in [15].

Switching Overhead. To analyse the model switching overhead, we measured power consumption and traced inference latency. Fig. 5 depicts the GPU+CPU power consumption and latency while deploying end-to-end MetaTinyML and MetaTinyML-sleep on Jetson Nano for 60 seconds. Based on the results, the switching point can be extracted when power consumption or latency drop as the lighter model will consume less power and is faster. On the other hand, there is no sharp line in the plot when switching happens meaning there is negligible overhead for switching. Moreover, Fig. 5 (a) depicts MetaTinyML-sleep consumes less power than MetaTinyML, however it has approximately twice the latency. Additionally,

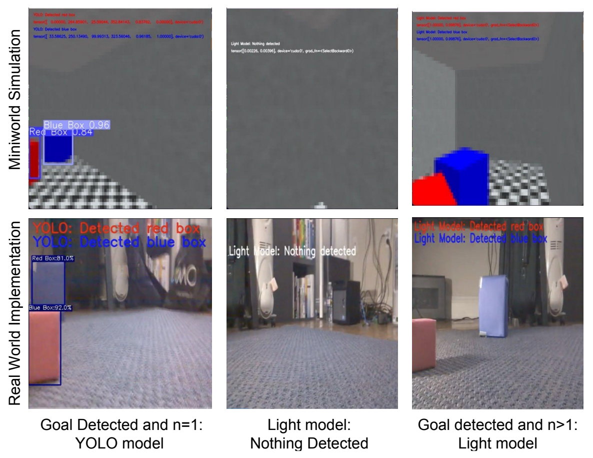
Miniworld Simulation
Real World Implementation
Blue Box 0.96
Red Box 0.84
YOLO: Detected red box
YOLO: Detected blue box
Red Box:81.0%
Blue Box:92.0%
Light Model: Nothing detected
Light Model: Detected red box
Light Model: Detected blue box
Goal Detected and n=1: YOLO model
Light model: Nothing Detected
Goal detected and n>1: Light model

Fig. 7. Sim to real implementation of MetaTinyML-sleep approach for reaching one goal.

we extracted the number of times we run CNN model and YOLO model and reported the CNN model/YOLO model ratio on Fig. 6. The results show that MetaTinyML picks the lighter model up to 80% of the time to process the captured image, which is 30% more than MetaE2RL [16].

Sim to Real Transfer. Fig. 7 depicts the sim to real transfer of the proposed MetaTinyML-sleep approach. The captured images show the implemented framework switches between the two models based on the presented meta policy.

IV. Related Work

There are surveys [1], [4], [5] addressing TinyML system challenges, and conducting state-of-the-art work within this domain. Work in [1] discusses the design flow of TinyML along with a variety of related works within TinyML for software and hardware level optimization.

Metareasoning approaches [8], [13], [16]–[20] can be considered as a promising solution for scheduling tasks and managing memory usage and power consumption at the edge and TinyML systems. Work in [18] proposed a meta policy to switch between cloud and onboard implementation to fulfill a mission in autonomous systems. Moreover, [16] applied a metareasoning approach for multi-goal reinforcement learning navigation while proposing squeezed edge YOLO on a Crazyflie drone with GAP8 Processor. However, in these work they did not deploy the metareasoning approach end-to-end on the platform and there is no evaluation result on the end-to-end efficiency of proposed metareasoning approach.

V. Conclusion

In this work, an end-to-end metareasoning framework, MetaTinyML, is proposed for goal-oriented autonomous navigation on TinyML platforms. In the proposed framework, a new meta policy is presented to make real-time decision for using CNN or YOLO based models to do the process. To evaluate the proposed MetaTinyML framework, the end-to-end framework is deployed on a Jetbot equipped by NVIDIA Jetson Nano 4GB board. A demo of the proposed approach which the jetbot could successfully reach the goal while switching between two models is provided in this link.

Acknowledgments

This project was sponsored by the Laboratory of Physical Sciences.

References

[1] M. Shafique, T. Theocharides, V. J. Reddy, and B. Murmann, "Tinyml: current progress, research challenges, and future roadmap," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1303–1306.

[2] H. Han and J. Siebert, "Tinyml: A systematic review and synthesis of existing research," in *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, 2022, pp. 269–274.

[3] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov *et al.*, "Benchmarking tinyml systems: Challenges and direction," *arXiv preprint arXiv:2003.04821*, 2020.

[4] L. Dutta and S. Bharali, "Tinyml meets iot: A comprehensive survey," *Internet of Things*, vol. 16, p. 100461, 2021.

[5] M. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–37, 2021.

[6] E. Horvitz, *Metareasoning: Thinking about thinking*. MIT Press, 2011.

[7] R. Ackerman and V. A. Thompson, "Meta-reasoning: Monitoring and control of thinking and reasoning," *Trends in cognitive sciences*, vol. 21, no. 8, pp. 607–617, 2017.

[8] S. T. Langlois, O. Akoroda, E. Carrillo, J. W. Herrmann, S. Azarm, H. Xu, and M. Otte, "Metareasoning structures, problems, and modes for multiagent systems: A survey," *IEEE Access*, vol. 8, pp. 183 080–183 089, 2020.

[9] M. K. Dawson Jr and J. W. Herrmann, "Metareasoning approaches for thermal management during image processing," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 86281. American Society of Mechanical Engineers, 2022, p. V007T07A036.

[10] J. Svegliato, C. Basich, S. Saisubramanian, and S. Zilberstein, "Using metareasoning to maintain and restore safety for reliably autonomy," in *Submission to the IJCAI Workshop on Robust and Reliable Autonomy in the Wild (R2AW)*, 2021.

[11] J. Caylor, J. W. Herrmann, C. Hung, A. Raglin, and J. Richardson, "Metareasoning for multi-criteria decision making using complex information sources," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications IV*, vol. 12113. SPIE, 2022, pp. 305–313.

[12] E. Carrillo, S. Yeotikar, S. Nayak, M. K. M. Jaffar, S. Azarm, J. W. Herrmann, M. Otte, and H. Xu, "Communication-aware multi-agent metareasoning for decentralized task allocation," *IEEE Access*, vol. 9, pp. 98 712–98 730, 2021.

[13] J. W. Herrmann, "Introduction to metareasoning," in *Metareasoning for Robots: Adapting in Dynamic and Uncertain Environments*. Springer, 2023, pp. 1–16.

[14] E. Humes, M. Navardi, and T. Mohsenin, "Squeezed edge yolo: Onboard object detection on edge devices," *ML with New Compute Paradigms (MLNCP) Workshop at NeurIPS*, 2023.

[15] T. Manjunath, M. Navardi, P. Dixit, B. Prakash, and T. 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.

[16] M. Navardi, E. Humes, T. Manjunath, and T. Mohsenin, "Metae2rl: Toward metareasoning for energy-efficient multi-goal reinforcement learning with squeezed edge yolo," *IEEE Micro*, pp. 1–9, 2023.

[17] J. W. Herrmann, K. Carey, and S. L. Molnar, "Using metareasoning to improve autonomous robot planning," in *Open Architecture/Open Business Model Net-Centric Systems and Defense Transformation 2023*, vol. 12544. SPIE, 2023, pp. 218–223.

[18] M. Navardi and T. Mohsenin, "Mlae2: Metareasoning for latency-aware energy-efficient autonomous nano-drones," *The IEEE International Symposium on Circuits and Systems (ISCAS)*, 2023.

[19] J. W. Herrmann, *Metareasoning for Robots: Adapting in Dynamic and Uncertain Environments*. Springer Nature, 2023.

[20] S. Molnar, M. Mueller, R. Macpherson, L. Rhoads, and J. W. Herrmann, "Using metareasoning on a mobile ground robot to recover from path planning failures," 2023.