An Optimization Framework for Efficient Vision-Based Autonomous Drone Navigation

Mozhgan Navardi†, Aidin Shiri†, Edward Humes† Nicholas R. Waytowich‡, Tinoosh Mohsenin†
†Department of Computer Science & Electrical Engineering, University of Maryland Baltimore County
‡US Army Research Laboratory

***Abstract*—Fully autonomous drones are a new emerging field that has enabled many applications such as gas source leakage localization, wild-fire detection, smart agriculture, and search and rescue missions in unknown limited communication and GPS denied environments. Artificial intelligence and deep Neural Networks (NN) have enabled applications such as visual perception and navigation which can be deployed to make drones smarter and more efficient. However, deploying such techniques on tiny drones is extremely challenging due to the limited computational resources and power envelope of edge devices. To achieve this goal, this paper proposes an efficient end-to-end optimization method for deploying deep NN models for vision-based autonomous drone navigation applications, such as obstacle avoidance and steering task. This paper formulates two different methods for implementing the NN inference phase onto tiny drones and analyzing the implementation results for each case: 1) a Cloud-IoT implementation and 2) Onboard Processing. Several models are trained with state-of-the-art scalable NN architectures and the most efficient cases in terms of computation complexity and accuracy are selected for implementation on a cloud server and several edge devices. By designing hardware-friendly NN models and optimal configuration of the implementation platforms, we were able to reach up to 97% accuracy, speed up the computation 2.3x, have 22x less complexity, and 53% energy reduction. Also, we achieve up to 25 fps on the GAP8 processor, which is enough for real-time drone navigation requirements, even when the model is running on a small IoT device.**

***Index Terms*—Autonomous Systems, Obstacle Avoidance, Drone Navigation**

I. Introduction and Related Works

Autonomous nano-scale Unmanned Aerial Vehicles (UAVs) are emerging technologies that have enabled countless indoor and outdoor applications like gas leakage source localization [1]. These devices with a form factor of a few centimeters typically weigh less than a few grams. Also, a tiny processor with power enveloping less than a few Watts collects information from its onboard sensors and either performs application computations locally or transmits the essential information to a centralized server to do the computation [2]. However, implementing fundamental perception tasks like robotic vision requires extremely careful design consideration due to their intensive computation, while transferring data back and forth to a server may be hard to accomplish due to bandwidth limitations, security concerns, or power consumption.

Recently, Neural Network (NN) based robotic perception models for autonomous drone navigation and obstacle avoidance have demonstrated the best accuracy and have shown better generalizability in more complex environments [3]. It also has been demonstrated that NN-based models are more resilient to permanent and transient faults in navigational systems [4]. But usually, this improvement comes with the cost of being the main computational bottleneck of the system [5]. Therefore, the task of selecting a proper NN architecture for the perception task and considering multi-objective optimization problems with parameters such as nn accuracy. model size and number of computations is of great importance. Things get even more complicated when taking application specific and hardware platform constraints such as inference speed, processor power envelop, and memory size into account. Different hardware platforms have different computational resources and power constraints. Therefore, a desirable NN model should be scalable to scale up/down the parameters and computations, so that it could be easily tailored based on the target implementation platform capabilities to meet the user's required constraints.

In this work, we propose an optimization framework for implementing scalable state-of-the-art NN models based on the hardware platform capabilities for autonomous drone navigation applications like obstacle avoidance and steering.

This paper makes the following major contributions:

- Efficient end-to-end method for deploying deep NNs for obstacle avoidance and steering tasks for autonomous drones which can be deployed in real world scenarios.
- Formulating a method for implementing NN inference for autonomous drones and proposing a detailed analysis for 1) Cloud-IoT and 2) Onboard Edge implementations.
- Extensive experiments with several state-of-the-art network architectures and implementation of the most efficient models on a desktop server, embedded CPU, GPU, and IoT platform.

II. Proposed Framework

This work proposes a framework that provides a method for generating optimized models for autonomous drone navigation for implementation on the cloud and edge. The efficiency of NN implementation is often reported with the number of operations per second per Watt (FLOPS/W or GOPS/W). However, for time and energy sensitive applications such as autonomous drone navigation, several factors such as accuracy, throughput, latency, power consumption, and energy efficiency must be considered for providing a comprehensive big picture to perform the trade-offs between different implementation techniques. This work investigates the process of training

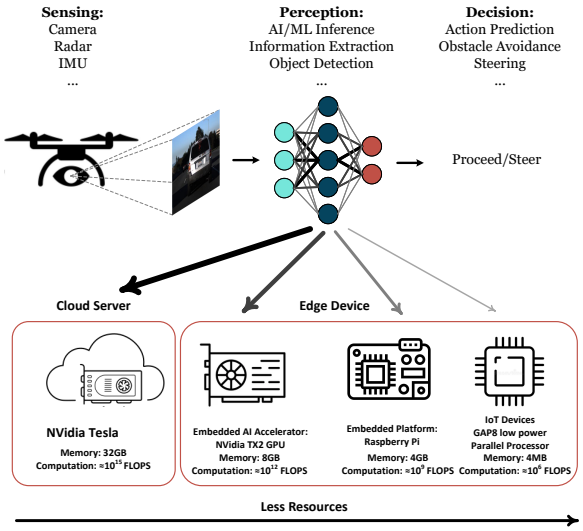

Fig. 1: A high level diagram of proposed framework: scalabe neural networks are trained for autonomous drone obstacle avoidance and steering task

several state-of-the-art scalable NN models and their efficient deployment for obstacle avoidance and steering task, to structure an optimization framework that takes different hardware implementation metrics and constraints into account.

A. Application

One of the main deployment platforms of NN-based autonomous drone navigation systems is tiny drones. They usually have limited processing power and capabilities. We used several state-of-the-art deep NNs to train a model which can predict the probability of collision. Moreover, it outputs a proper steering angle to avoid obstacles while the drone is flying autonomously based on a single view grayscale camera. The drone receives two outputs from the model and generates proper navigation output accordingly. If the probability of collision generated by the first output is higher than a threshold, the drone will change its direction and steer based on the angle generated by the second output.

B. Scalable Neural Network Models

Big network size and a large number of computations limit the capability of model inference for the real-time autonomous navigation application. Edge devices such as embedded and IoT processors have limited computation resources and server implementation is usually constrained by bandwidth limitations. Several recent works have proposed NNs that can be scaled with respect to the model size and computations to meet the application and hardware implementation requirements. In this work, we trained our NN model with five scalable models including: MobileNet V1-3 [7], [8], [9], EfficientNet [10], and ResNet [11]. We considered four different configurations for each model and trained models to measure their accuracy, size, and the number of computations. MobileNet complexity can be modified by assigning different values to its width multiplier (0.25, 0.5, 0.75, 1). For the EfficientNet, we experimented with the B0-B3 configuration. Also, we used the ResNet by serializing 1, 2, 3, and 4 residual blocks sequentially.

Algorithm 1 Latency and Energy Optimization Pseudo-Algorithm

Input: $\alpha_u,\ \alpha_d,\ \beta,\ upload\_speed,\ download\_speed, v,\ x,$ $edge\_power\_budget.$
Output: $t_{total},\ p_{total}.$

1: ▷ Train phase
2: $m_{cloud},\ m_{edge} \leftarrow train\_different\_models()$
3: $m_{edge} \leftarrow 8\text{-}bit\_ptq(m_{edge})$
4: ▷ Inference phase
5: $t_{cloud}, t_{edge} = computation\_latency(m_{edge},\ m_{cloud})$
6: $p_{cloud}, p_{edge} = computation\_power(m_{edge},\ m_{cloud})$
7: $p_{uplink}, t_{upload} \leftarrow upload\_power\_latency(\alpha_d, \alpha_u, \beta, upload\_speed)$
8: $p_{downlink}, t_{download} \leftarrow download\_power\_latency(\alpha_d, \alpha_u, \beta, download\_speed)$
9: $t_{limit} \leftarrow \frac{v}{x}$
10: **if** $t_{limit} > t_{edge}$ && $edge\_power\_budget > p_{edge}$ **then**
11: $\quad c \leftarrow 0,\ e \leftarrow 1$ ▷ Do computations on the edge
12: **else if** $t_{limit} > t_{cloud} + t_{upload} + t_{download}$ **then**
13: $\quad c \leftarrow 1,\ e \leftarrow 0$ ▷ Do computations on the cloud
14: **else**
15: $\quad$ **return** *failed*
16: **end if**
17: $t_{computation} \leftarrow c \times t_{cloud} + e \times t_{edge}$
18: $t_{communication} \leftarrow c \times (t_{upload} + t_{download})$
19: $t_{total} \leftarrow t_{computation} + t_{communication}$
20: $p_{computation} \leftarrow c \times p_{cloud} + e \times p_{edge}$
21: $p_{communication} \leftarrow c \times (p_{uplink} + p_{downlink})$
22: $p_{total} \leftarrow p_{computation} + p_{communication}$
23: **return** t_{total}, p_{total}

C. Dataset

Two datasets are used to train a model for obstacle avoidance and steering from raw grayscale image inputs. Firstly, a dataset [3] of nearly 32K images is annotated with collision/no collision labels. Secondly, a dataset of Udacity's projects [12] contains 70K images, captured while driving a car by left, center, and right view cameras. These datasets are used for training collision detection and steering model.

D. Framework Algorithmic Description

In general, there are two methods for implementing the drone perception: 1) Implementing the model on the cloud by streaming sensory inputs such as images to a server and decision results back to the drone 2) Implementing the model on the edge. Fig. 1 illustrates the common methods for implementation of the NN-based perception for UAVs. Previous works have demonstrated that NN is the main computation and power bottleneck of the drone perception [5]. In this regard, we propose an optimization algorithm for selecting appropriate models for implementation of the NN inference on the cloud or the edge, based on different application constraints such as accuracy, latency, and power consumption.

The proposed method, illustrated in Algorithm 1, includes two separate phases. The train phase (lines 1-3) is implemented offline on a server. Afterward, lines 4-23 will be run in the inference phase. The algorithm inputs are the application parameters and constraints such as the velocity of drone (v), distance from obstacle (x) and edge_power_budget along with wireless network communication parameters including $\alpha_u,\ \alpha_d,\ \beta,\ upload\_speed$ and $download\_speed$ which is

TABLE I: Average download and upload speed and power model parameters of wireless networks [6]

	Download/Upload Speed		Power Model		
Network Type	Download Speed (Mbps)	Upload Speed (Mbps)	α_u (mW/Mbps)	α_d (mW/Mbps)	β (mW)
3G	2.027	1.1	868.98	122.12	817.88
4G	13.76	5.85	438.39	51.97	1288.04
Wi-Fi	54.97	18.88	283.17	137.01	132.86

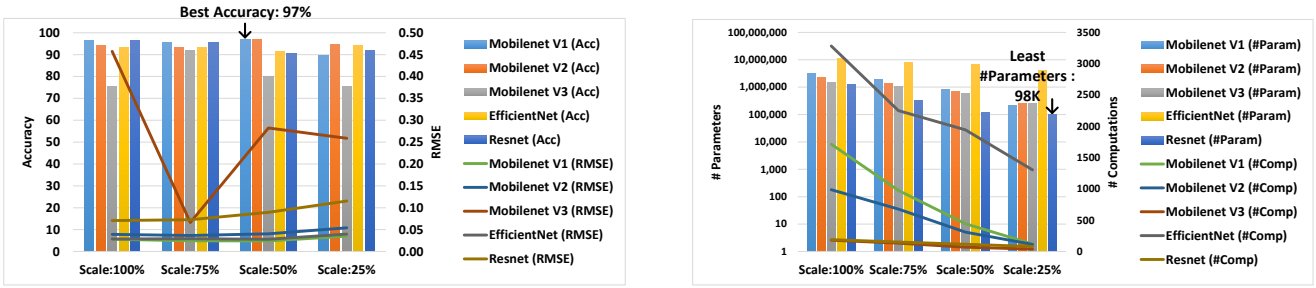

Fig. 2: Accuracy, model size, and number of computations for different configurations of state-of-the-art scalable neural networks

mentioned in Table I. Total power consumption p_{total} and total latency t_{total} are the outputs. In line 2, train_models() function trains collision and steering angle model with the state-of-the-art models and return two models which are trained with the goal of achieving the best accuracy or least latency. The high accuracy model m_{cloud} is originally considered for cloud implementation. On the other hand, due to the power constrain on the edge device a low-power model which means a model with lower computation complexity is chosen for the edge (m_{edge}). In order to reduce memory complexity on edge, $8\text{-}bit\_ptq()$ function applies a 8-bit post train quantization (PTQ) on m_{edge} (line 3). In the inference phase, line 5, we measure latency if we do computation on cloud (t_{cloud}) or edge (t_{edge}). The communication bandwidth and power consumption is a significant parameters in real-time inference for drone navigation applications. For measuring these parameters, we refer to standard download and upload speed in average and communication power consumption for wireless networks [6]. In order to calculate the power and latency of communication between edge and cloud, Table I parameters could be used along with equation 1, to calculate the theoretical results.

$$\begin{aligned} p_{uplink} &= \alpha_u \times upload\_speed + \beta, \\ p_{downlink} &= \alpha_d \times download\_speed + \beta \end{aligned} \tag{1}$$

Lines 7 and 8 use the aforementioned parameters as input of the framework. Since one of the important tasks of the drone is obstacle avoiding, so we have a limited time to give steering angle to the drone. This limited time t_{limit} is calculated in line 9. Based on the latency and power constraints, if we can run computations on edge (line 10), edge variable e will be set (line 11). If we cannot meet constraints by running computations on edge, we will do computation on cloud and set variable c (lines 12 and 13). In line 17, computation latency calculates which will be equal to t_{cloud} if c is set or t_{edge} if e is set. Also, in line 19 we will have $t_{communication}$ if we did communication on cloud ($c = 1$). Total power consumption is also calculated in the same way in lines 20-22. Algorithm will be return failed if it cannot meet requirements unless will be return total power consumption p_{total} and latency t_{total}.

III. Implementation and Experimental Results

A. Software Experimental Results

To find the most efficient model for collision detection and steering angle, we evaluate the models discussed in section II-B. We used 10% of the mentioned dataset in section II-C as evaluation set, trained each model for 100 epochs, and measured the task accuracy for correct label prediction, and Root Mean Squared Error (RMSE) of steering angle. The number of computations and model size are also calculated for each of four different configurations. Fig. 2 depicts models accuracy, size, RMSE and number of computations. As you can see in Fig. 2, we have a decreasing trend for the number of computations and parameters by decreasing the scale size.

We selected two efficient models for hardware implementation, one with the best accuracy and the other optimized for the least latency. For implementing the models on the edge devices, we performed PTQ to shrink the model size even further without significant loss in accuracy. As it is illustrated in Fig. 2, MobileNetV1 with the configuration of 50% has the highest accuracy of 97% and the lowest RMSE of 0.02. Therefore, we choose this model as the accuracy aware model. On the other hand, the ResNet 25% has the least computation complexity and therefore, the least latency, with the compromise of a few percent of accuracy and RMSE, which are 92% and 0.11, respectively. The number of computations of ResNet 25% is 3.8x less than MobileNetV1 50%. After applying 8-bit PTQ on ResNet 25%, we reach a 22x less complexity.

B. Off-the-Shelf Edge-Device Platforms Implementation Results and Analysis

To evaluate latency and power consumption of the trained models on edge devices, we implement them on NVIDIA jetson TX2 and GAP8 IoT processor [2], [13]. TX2 board has Quad-Core Arm A57 processor along with Pascal GPU cores with a maximum frequency 1300MHz. GAP8 processor also has a tiny fabric controller with 8 cluster cores for parallelization, which can be run at the maximum frequency of 150MHz. We configured both devices to achieve maximum

TABLE II: Hardware implementation results of both models on the commercial off-the-shelf devices

Metric/Platform	Accuracy Aware Model (m_{cloud}, MobileNetV1-50%)				Latency Aware Model (m_{edge}, ResNet-25%)			
	Server (M1 Pro)	CPU (ARM)	GPU (Jetson TX2)	Low Power Processor (GAP8)	Server (M1 Pro)	CPU (ARM)	GPU (Jetson TX2)	Low Power Processor (GAP8)
Computation Latency (ms)	30	250	40	95	22	21	13	40.6
Throughput (Inference/Sec)	33.3	4	25	**10.5**	45	47.6	76.9	**25**
Computation Power (W)	N/A	3.9	4.8	0.77	N/A	3.7	4.7	0.85
Performance (GOPS)	14.8	1.8	11.1	4.7	3.7	3.8	6.2	2
Energy/Inference (mJ)	N/A	969	192	**73**	N/A	79	70	**34**

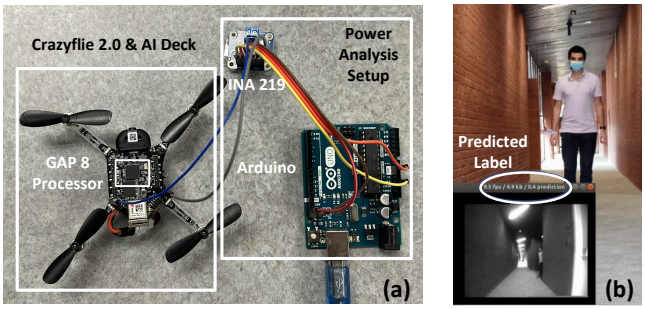

Fig. 3: a) GAP8 Processor residing the AI-Deck with power analysis setup b) Grayscale images captured form Crazyflie camera labeled in real-time

performance and measured the models' implementation results. Fig. 3(a) shows implemented setup for measuring the power consumption of GAP8 when we do onboard processing. We use INA 219 and Arduino to monitor power consumption.

For implementing the autonomous drone navigation applications, Fig. 3(b), we used CrazyFlie 2.0, an open-source flying development instrument with AI-Deck extension board [2]. The AI-Deck features a low power camera, Wi-Fi module, and ultra low power RISC-V processor GAP8 which lets CrazyFlie do onboard processing. Grayscale images captured from the drone camera are resized to 200x200 to feed to NN models either onboard or on the cloud. We implemented both models on hardware platforms and measured several metrics reported in Table II. The bold numbers in Table II indicate that the latency aware model has a 2.3x higher throughput than the accuracy aware model on the GAP8 processor for edge implementation by consuming almost the same power. Therefore, we reach 53% lower energy consumption in each inference by applying a tiny model on GAP8. Also, communication power and latency that we calculated based on Table I and equation 1 are 0.9W and 100ms, respectively. We can compare this information along with the results of Table II with two main constraints of the algorithm 1: $edge\_power\_budget$ and t_{limit} in order to make a decision for doing onboard processing or cloud-IoT implementation. The measured communication latency for transmitting the images between the drone and the cloud would be the upper bound for cloud-based implementation. Higher communication speed can be achieved by reconfiguring the Wi-Fi module, but with the cost of bandwidth and higher energy consumption, which can be calculated by using power in Table I and equation 1.

IV. Conclusion

In this paper, we proposed a framework for optimization of neural network based autonomous drone navigation using single vision input. We evaluate the performance and accuracy of several state-of-the-art scalable neural network models trained for obstacle avoidance and steering task. Our framework formulates the problem of selecting appropriate neural network architecture based on parameters such as accuracy, energy efficiency, latency, and throughput. We selected two accuracy aware and latency aware models for implementation and measured the aforementioned results for cloud and edge computing devices. Comparing to the accuracy aware model which has 97% accuracy, the latency optimized model achieves up to 2.3x speedup, 22x less complexity, and 53% lower energy consumption with a 5% accuracy penalty.

V. Acknowledgment

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

References

[1] B. P. Duisterhof *et al.*, "Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 9099–9106.

[2] D. Palossi *et al.*, "A 64-mw dnn-based visual navigation engine for autonomous nano-drones," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8357–8371, 2019.

[3] A. Loquercio *et al.*, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018.

[4] Z. Wan *et al.*, "Analyzing and improving fault tolerance of learning-based navigation systems," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 841–846.

[5] H. Genc *et al.*, "Flying iot: Toward low-power vision in the sky," *IEEE Micro*, vol. 37, no. 6, pp. 40–51, 2017.

[6] A. E. Eshratifar *et al.*, "Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, 2018, pp. 111–116.

[7] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[8] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[9] A. Howard *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[10] M. Tan *et al.*, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

[11] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[12] Udacity, "An open source self-driving car," in *https://www.udacity.com/self-driving-car*, 2016.

[13] E. Flamand *et al.*, "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*. IEEE, 2018, pp. 1–4.

[14] S. Liu *et al.*, "Stochastic dividers for low latency neural networks."