Enhancing UAV Systems via Task Offloading at the EDGE

Rajashekhar Reddy Tella*, Andrea Marotta †, Piero Castoldi ‡ Luca Valcarenghi ‡, Koteswararao Kondepu*
* Department of Computer Science and Engineering, IIT Dharwad, Dharwad, India
† Department of DISIM Science and Engineering University of L'Aquila
‡ Telecommunications, Computer Engineering,and Photonics Institute Scuola Superiore Sant'Anna
Email: {200030058, k.kondepu}@iitdh.ac.in

Abstract—**Mobile Edge Computing (MEC) is a pivotal driver of 5G and subsequent mobile cellular networks, enhancing various life aspects through advanced communication and computation. MEC evaluates the computational tasks and ensures ultra-low latency as well as higher bandwidths. It has a wide range of applications, such as mobile health, surveillance systems, road infrastructure sector, and smart factory setups. A notable application is MEC-assisted autonomous navigation, where camera-fitted Unmanned Aerial Vehicles (UAVs) send images to the MEC for object detection and use the inference for controlling the UAV navigation. The offloading prolongs UAV battery life and flight duration. However, the onboard computation reduces both the battery life and flight duration, which disrupts the applications. In this study, we offload the computational tasks to both the MEC and the Cloud infrastructures in order to preserve the battery life and prolong the flight time. We present the time required for object detection inference using both the EDGE and the Cloud in latency-sensitive scenarios. The results show that employing the EDGE could outperform the Cloud both in terms of latency and throughput. We also investigate the energy consumed by both CPU and GPU that are employed with the EDGE for object detection tasks.**

Index Terms—**Edge computing, Edge, MEC, CNN, Object Detection, Autonomous Navigation**

I. Introduction

The rise in population in many areas has led to increased road traffic, emphasizing the need for traffic surveillance systems. Surveillance involves monitoring and observing traffic using various technologies like cameras and sensors. However, traditional camera-based surveillance has certain limitations, including the high infrastructure costs and the requirement for extensive manpower for constant monitoring. However, aerial surveillance using unmanned aerial vehicles (UAVs) overcomes these challenges, and integrating artificial intelligence (AI) with it enhances its effectiveness in improving road safety and avoiding congestion.

Among the many components of aerial surveillance, UAVs, commonly referred to as drones, are critical. The growing popularity of UAVs is attributed to their cost-effectiveness, ease of deployment, and exceptional mobility. The remote control capabilities of UAVs play a pivotal role in delivering timely alerts and expediting rescue and recovery missions, especially during situations where communication networks go down [1]. Additionally, they play an important role in aerial surveillance by capturing images from elevated positions and transmitting them to base stations for continuous monitoring. This surveillance capability proves invaluable in monitoring crop quality across vast expanses and inspecting mining sites for potential issues [2]. Furthermore, UAVs can be controlled by commands generated based on their captured images, making the surveillance systems even more powerful and autonomous. The generated commands can be such that the UAV avoids the obstacle on its way, tracks the detected object, etc. So, a UAV can navigate autonomously by installing a camera and a central control unit where the former helps capture the snaps of the environment around it, and the latter controls the movement based on the inference made on the snaps. Here, the central control unit comprises a flight controller and a computational unit. The flight controller receives the direction commands transmitted by human operators in guided navigation and from the computational unit in the autonomous navigation system. Raspberry Pi, Jetson Nano, and Nvidia Jetson TX2 are a few examples of computational units that can be employed for the inference task. UAVs have scarce computational capabilities due to their limited weight, size, and power ability. The payload of a UAV is mainly contributed by the weight of the battery, frame, and flight controller, and it is limited. This restricts increasing the power of the battery, which in turn increases the weight of the battery. On the other hand, it is known that image processing is a CPU-intensive task, and it requires more power [3]. A possible solution is represented by utilizing onboard computation units, like Raspberry Pi, etc, which controls the UAV's movement. However, this represents a suitable approach only in use cases where flight time is not critical. Instead, most applications require a long flight time to achieve the desired goals. Offloading a part of the complete task would be a better alternative than carrying out the end-to-end computation onboard. In this case, communication overhead and delays are the major concerns. The communication delay generally includes the time taken to send the data to the desired location, data-processing time, and the time taken to obtain the response from the server to the user.

This calls for the exploitation of Mobile Edge Computing (MEC) as a suitable solution to support computation offloading and low-latency communication. The fifth generation of mobile communications (5G) natively offers processing through accelerated edge data centers or edge clouds to perform artificial intelligence/machine learning (AI/ML) tasks. To provide quick service response for low latency, edge computing seeks to move cloud resources and services to the network's edge — an intermediate layer between the end users and cloud data centers. The MECs have gained popularity because they can support latency-sensitive applications associated with computational tasks. Edge clouds are generally deployed with machine learning models like Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), etc., and these models are set up as a service.

This paper focuses on presenting a UAV system exploiting edge offload to detect and track target objects. The considered application exploits edge and cloud instances to perform inference on the captured frames using the custom-trained pre-existing algorithm. Results show that the use of edge in the taken application outperforms the cloud in latency and throughput. The experiments of object detection have been performed on both GPU and CPU. The results show that the GPU-based implementation is efficient compared to CPU-based implementation in terms of processing time and energy.

II. Related Work

Studies have been conducted to investigate the capabilities of UAVs, edge computing, and their combined applications in traffic surveillance. In [4] explored the utilization of UAVs for a multi-functional airborne traffic management system (Air-TMS) to address non-recurrent traffic congestion. Another work in [5], presented the applicability of UAVs in traffic monitoring and emergency identification, considering regulatory and safety concerns.

The authors in [6] experimentally demonstrated that UAV may suffer trajectory deviations from the intended path up to 5 $meters$ if the network latency exceeds 400 ms and more than 2 $meters$ if the packet loss probability exceeds 0.2. These findings underscore the necessity of an edge cloud infrastructure to ensure reliable and low-latency communication for UAV traffic management. In [3], the authors shed light on the potential of utilizing UAVs and MEC in various practical scenarios, such as pedestrian detection and object detection. In addition, the authors in [7] presented a work exploiting the MEC's low-latency and high bandwidth feature for firefighting in residential areas of future smart cities.

To the best of our knowledge, no previous work has demonstrated that specifically explores the utilization of UAV resources (i.e., battery and others) by offloading either to the MEC (or EDGE) or Cloud for image detection use case.

The following section gives an overview of the system architecture of Edge-assisted surveillance using UAVs.

III. Architecture Overview

Fig. 1 shows the architecture for autonomous surveillance using a UAV following a client-server model, where the computational unit on the UAV transmits captured images to the Edge and awaits a response containing the detected objects. The UAV and the edge are connected through a WiFi network for seamless communication. The UAV's computational unit,

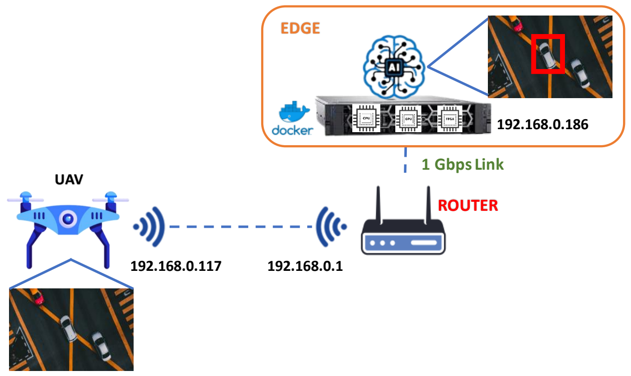

Fig. 1. Detection and Tracking Architecture

consisting of a Raspberry Pi and a flight controller, handles image transmission and flight control operations. On the edge side, high-performance computational units execute optimized object detection algorithms as a Docker service to achieve real-time object detection on the received images, enabling quick responses.

A. EDGE

The objective of object detection and tracking is a latency-sensitive application where the trajectory-related decisions should be taken in order of milliseconds (ms). In our setup, we have used the CPU and GPU for image processing at the EDGE. The YOLOv3 algorithm was used to perform the inference on the received images. We have used the idea of transfer learning to develop a model to satisfy our requirements. The YOLOv3 algorithm is generally trained on the COCO dataset with 80 object categories. As the objective in our case was to identify a specific car in the traffic and follow it, and the default algorithm does not differentiate between the target and the remaining cars, we have custom-trained the YOLOv3 model with the dataset made by capturing the images of the target object alongside another car as shown in Fig. 2b. We have captured 200 images and used the image augmentation technique to train the algorithm. After capturing the images from various angles, we drew the bounding boxes around the objects and labeled them using LabelImg [8] — an open-source graphical image annotation program. For custom training of YOLOv3, the dataset annotations should be in textual format, where each line corresponds to an object description. LabelImg generates the annotations in the required format for YOLOv3. Upon the dataset is ready, the training

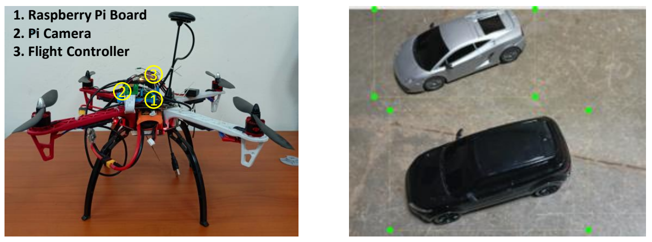

Fig. 2. (a) UAV components; (b) Snapshot of LabelImg

is performed using the YOLOv3 model with the modified configuration file, which includes (i) setting the number of classes; (ii) file path to the training and validation images folder; (iii) number of epochs; (iv) batch size for training; and (v) number of units in the last layer. We used the GPU server enabled with A100 card to train our model. The model was trained for 100 epochs, and the validation accuracy was around 85%.

B. Unmanned Aerial Vehicle

UAVs have gained popularity in recent years because of their low-cost deployment, which enables them to carry out experiments to show proof of concept of an idea. We developed an in-house quad-copter using all the required individual parts like the carbon frame, motors, ESCs, etc. As shown in Fig. 2a, the UAV has a flight controller who controls the speeds of the motors, yaw angles, etc. We installed a Raspberry Pi board on the built quad-copter to enable communications between the flight controller and the EDGE for capturing the images using the pi-camera. The flight controller and the Pi board are connected using a special type of chord 6-pin DF-13 connector on one side, which goes into the telemetry port, and the other side has 6 female Dupont connectors. This allows the Raspberry Pi to send commands to control the movement of the drone.

The drone's movement is controlled via custom-developed Python scripts that use PySerial, DroneKit, and Micro Air Vehicle (MAV) proxy. PySerial is a well-known Python package that allows for serial communication with external devices. This library is commonly used in projects involving serial port connection with microcontrollers, boards that run Arduino, GPS modules, sensors, and other hardware devices. DroneKit is a Python toolkit that provides a high-level API for drone communication and control. It enables Python developers to create apps for autonomous drone operations, monitoring, and management. DroneKit works with various autopilot systems, including ArduPilot [9] that support the MAVLink protocol. It offers an abstraction layer that makes dealing with drones and accessing flight data easier. MAVProxy is command-line Ground Control Station (GCS) software used to control and monitor unmanned vehicles that use the MAVLink protocol. It enables the use of a computer terminal or a remote shell to interface with and manage MAVLink-enabled vehicles such as drones. MAVProxy bridges the MAVLink-enabled vehicle and the user, providing a flexible and extensible interface for sending commands, receiving telemetry data, and performing various tasks. It supports multiple vehicle connections and can handle multiple MAVLink streams simultaneously.

The UAV is installed with an R-Pi board and a Pi camera in addition. The custom-trained object detection algorithm is containerized, and it is run as a Docker service that receives the images and responds with the detections in that image. The docker container runs as an API endpoint, and the R-Pi board sends the captured images using the HTTP protocol.

IV. Experimental Setup and Results

This section details the deployment of a custom-trained object detection algorithm at the EDGE with several hardware options, such as CPUs and GPUs, and with the remote Clouds.

Fig. 3 shows the considered evaluation scenario, which contains the UAV, A Wi-Fi router, EDGE, and two remote Clouds (1&2). The EDGE device utilizes an 11th Gen Intel Core i7-11700 CPU model running at a base frequency of 2.50 GHz and equipped with two processors, each processor with 8 cores. The EDGE device is also equipped with an Nvidia

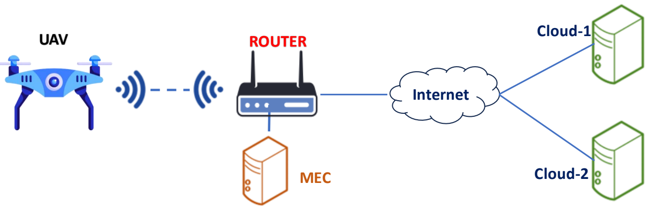

Fig. 3. Communication Architecture

A100 GPU for its computation. In contrast, the Cloud system is powered by an Intel Xeon Platinum 8272CL CPU, operating at a base frequency of 2.60 GHz, and this CPU configuration consists of a single processor with 2 cores per processor. The Cloud infrastructure is deployed via Microsoft Azure services in both Pune and Chennai locations in India, utilizing identical configurations, and the router is a D-Link DIR-615 with a bandwidth of 300Mbps. It brings both the EDGE and UAV into the same subnet so that they can communicate, and also it connects the UAV to the internet. Both the EDGE and the Cloud run the custom-trained YOLOv3 for object detection. The evaluation of the proposed UAV task offloading considers different network metrics (i.e., throughput and latency) as well as an enumeration of system performance factors like processing time and energy usage.

Latency and throughput of the network between the UAV, the EDGE, and both the Cloud instances are measured with the *ping* and *iperf3* tools widely used in network performance analysis. For *ping*, packets (Internet Control Message Protocol — ICMP) are sent every second, while *iperf3* employed a single UDP stream with a bitrate of 100M, surpassing the expected throughput to gauge network capacity.

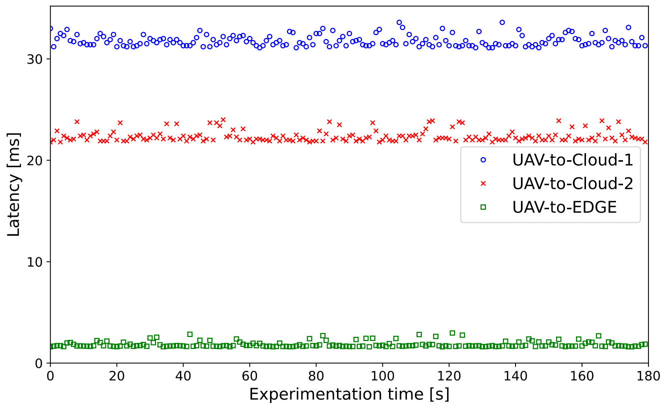

Fig. 4. Latency (EDGE vs Cloud-1/Cloud-2)

Fig. 4 illustrates the Round Trip Time (RTT) – namely Latency — defined as the time taken from the UAV to the EDGE/Cloud instances. We used the ping command to calculate the latency, and the experiment was carried out for

180 s, sending the ICMP packets every second. The results show that the geographical separation between the computing servers substantially impacts latency in the considered use case. As described earlier, the EDGE is set up near the user as the Cloud is deployed far away. As shown in Fig. 4, the UAV-to-EDGE average latency shows around 1.6 ms with a significant reduction. The average latency from the UAV to Cloud-1 (located in Pune, India) and Cloud-2 (located in Chennai, India) exceeds the latency to the EDGE server by 20 ms and 30 ms, respectively.

Fig. 5 depicts the uplink throughput to both the Cloud instances and the EDGE. It is clear that the throughput to the EDGE is higher compared to that of the Cloud, which is deployed far away from the user. Upon running the custom-trained object detection algorithm on both GPU and CPU, results show that the former can process images at 27.93 Frames Per Second (FPS), and the latter can process them at 21.73 FPS. Note that the processed FPS can be observed slightly higher due to the high-end processing device (NVIDIA A100). The custom-trained YOLOv3 model is optimized for both CPU and GPU processing by using *–include* and *–device* options while exporting the trained-model weights using Intel's OpenVino toolkit, the performance of the models has increased significantly. The optimized model achieves image processing speeds of 41.6 FPS on the CPU and 43.47 FPS on the GPU. By adding half of the RTT measured using the ping to the processing times of the original model, we get the complete response time for the CPU as 48 ms, corresponding to 21 FPS, and for the GPU as 37 ms, corresponding to 27 FPS.

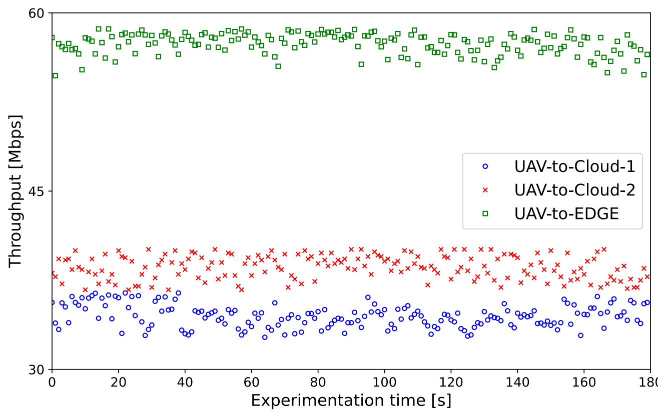

Fig. 5. Throughput (EDGE vs Cloud-1/Cloud-2)

It can be seen that the EDGE implementation for the object detection of surveillance gives better results in terms of latency and throughput compared to that of a Cloud. EDGE is an added advantage as it gives us the ability to use the hardware, which accelerates the inference. The above results clearly show that the EDGE equipped with a GPU performs better compared to a CPU in terms of the number of frames it can process per second, which is an important factor when it comes to surveillance.

Another important evaluation metric is the energy consumed by the hardware processing for one image. The CPU energy consumption is determined using the s-tui [10] tool, whereas the energy dissipated by the GPU is calculated through *nvidia-smi* command. The results show that the energy consumed by the CPU is around 11.8 J and that by the GPU is around 4.23 J. Thus the CPU-based implementation nearly needs 2.8 times more energy compared to that of the GPU, making the GPU-based implementation energy efficient.

V. Conclusion and Future work

This study presents the utilization of an MEC and a Cloud for applications that demand low latency, such as UAV-based surveillance. We analyzed the latency and throughput between MEC and the Cloud. Additionally, it showed the advantages of replacing CPUs with GPUs due to their ability to process a higher volume of images per second. The results demonstrate that the computational offloading to the EDGE would enable the efficient usage of the battery for the UAV flight instead of onboard computation. The potential future work of this paper is to build an Edge Cloud infrastructure with FPGAs and use the optimized models for inference.

VI. Acknowledgements

This work was supported by DST NM-ICPS, TiHAN at the Indian Institute of Technology (IIT) Hyderabad. This work was partially supported by the European Union - NextGenerationEU under the Italian Ministry of University and Research (MUR) National Innovation Ecosystem grant ECS00000041 - VITALITY - CUP E13C22001060006 and the KDT-JU project Collaborative edge-cLoud continuum and Embedded AI for a Visionary industry of thE futuRe (CLEVER) (grant agreement no. 101097560). KDT-JU receives funding from the Horizon Europe Research Framework and the National Authorities.

References

[1] L. Gupta, R. Jain, and G. Vaszkun, "Survey of important issues in uav communication networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1123–1152, 2016.
[2] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, pp. 460–466, May 2015. [Online]. Available: https://doi.org/10.1038/nature14542
[3] Y. C. Makkena, R. R. Tella, N. Parekh *et al.*, "Experience: Implementation of Edge-Cloud for Autonomous Navigation Applications," in *in Proc. 15th COMSNETS*, 2023, pp. 579–587.
[4] CAIT UTC NC8, "Final report," Tech. Rep., 2018. [Online]. Available: https://cait.rutgers.edu/wp-content/uploads/2018/05/cait-utc-nc8-final.pdf
[5] K. Ro, J.-S. Oh, and L. Dong, "Lessons Learned: Application of Small UAV for Urban Highway Traffic Monitoring," 01 2007.
[6] O. Bekkouche, T. Taleb, and M. Bagaa, "UAVs Traffic Control Based on Multi-Access Edge Computing," in *in Proc. of GLOBECOM*, 2018, pp. 1–6.
[7] V. Gudepu, B. Pappu, T. Javvadi *et al.*, "Edge Computing in Micro Data Centers for Firefighting in Residential Areas of Future Smart Cities," in *in Proc. of ICECCME*, 2022, pp. 1–6.
[8] Tzutalin, "labelimg: An open source graphical image annotation tool," https://github.com/tzutalin/labelImg, Year, accessed: August 31, 2023.
[9] "Ardupilot open source autopilot," https://ardupilot.org/, Year, accessed: August 31, 2023.
[10] amanusk, "s-tui: Terminal ui for monitoring your computer," https://github.com/amanusk/s-tui, Year, accessed: August 31, 2023.