Real-Time Edge Analytics of Video Feeds from a UAV Fleet

Suman Raj*, Harshil Gupta, Sougat Shome and Yogesh Simmhan

Department of Computational and Data Sciences
Indian Institute of Science, Bangalore 560012 INDIA
Email: sumanraj@iisc.ac.in, harshilgupta@iisc.ac.in, sougat.191ec152@nitk.edu.in, simmhan@iisc.ac.in

***Abstract*—Fleets of Unmanned Ariel Vehicles (UAVs), also called drones, are becoming common-place to support diverse applications in logistics and urban monitoring. These rely on advances in computer vision and artificial intelligence for navigation, traffic estimation, pollution monitoring, etc., carried out both using on-board computing devices and accelerated edge computing platforms connected to their base-stations. Drones can also be put to use for social good to enhance the active lifestyle of visually-challenged individuals through navigational assistance and situation awareness, enabled through visual analytics over video streams from the drone. Supporting such scenarios imposes time-critical requirements for completing the analytics as the safety of the person is of the utmost priority. When an accelerated edge device supports multiple such drones in their analytics, we need to balance the timelines of analytics and the priority of processing different streams based on the current situation. We propose a framework for performing video analytics on a GPU-accelerated edge device that supports video streams from multiple drones, and helps assess the current threat to the user they are tracking. It addresses the complexity of priority-based task processing of concurrent video streams, and helps prioritize the completion of time-critical tasks while supporting dynamic variation of the task importance based on evolving situations. Our experiments conducted using multiple video streams from real drones and using Nvidia Jetson Nano as the edge platform validates the effectiveness of our real-time video analytics framework.**

***Index Terms*—Distributed stream processing, Edge analytics, UAVs, Task Offloading**

I. Introduction

Recent advancements in artificial intelligence and computer vision has led to the wide-spread use of Unmanned Ariel Vehicles (UAVs), also called drones, assisted by autonomous navigation. There is a growing trend of fleets of drones being used for food delivery, flying ambulances, monitoring platforms for urban safety and disaster response [1]. UAVs have become popular due to their agility in urban spaces and their lightweight structure.

A single UAV can carry a variety of on-board sensors such as video cameras, gyroscope, GPS, LiDAR, etc. However, the payload that rotary-wing drones can carry is also more limited. As a result, their on-board compute capacity is limited to critical control and navigation requirements. Processing of the data collected by the drones is either done by off-loading it to a base-station and onward to fog and cloud computing resources, or deferring the execution to after the drone lands. Applications that consume the data generated from the drone can be divided into one of these two categories. While the latter is more easily managed when the drone docks for recharging, supporting the time-sensitive needs of the former presents unique challenges.

In-flight processing may be required for *immediate reaction*, such as avoiding an obstacle, *near-term planning*, such as changing the orientation to point to a user, or *medium-term planning* to change the waypoints being visited during a flight mission. Such processing includes computer vision and inferencing using Deep Neural Networks (DNNs), predictive analysis, network algorithms and solving optimization functions. This requires a light-weight yet intelligent system for distributed data analysis with latency guarantees, and sending the processed data to different components for actuation. Reliable distributed scheduling of analytics tasks is an open problem, and requires the knowledge of the capabilities of on-board edge compute devices, nearby UAVs, fog/cloud computing at the base station, battery levels, mission parameters, 4G/5G/WiFi connectivity, etc. With recent advancements in 5G, [2] discusses on what to offload from the UAV in order to reduce the compute power and the image processing time, based on the available data rates.

We focus on the need for such real-time processing for a more specific and socially-impactful problem of using a personal drone as a mobile valet to assist a visually impaired person (VIP) in helping lead an active yet safe outdoor lifestyle. According to 2021 WHO report [3], the number of people suffering from visual impairment is $\approx$ 2.2 billion. Vision impairment severely impacts quality of life among adult populations resulting in lower rates of workforce participation and higher rates of depression and anxiety.

Even after decades of research about navigation support for visually impaired people, moving independently still remains a major challenge.

Specifically, we consider a scenario where multiple VIPs are being assisted by multiple drones with the basic capabilities of tracking and guiding the user are offered through on-board analytics on the drone. However, additional video analytics on situation awareness and thread-perception that

* Student Author

require more complex DNN models need to be off-loaded to a GPU- accelerated edge device connected to the base-station. The challenge that we address is to schedule the competing analytics requirements of multiple drones by the edge device, that takes into account latency of processing, priority of the task based on current conditions of the user and fairness. This has to also adapt to a dynamic environment where some users may be in a higher-thread situation that requires faster responses (e.g., approaching a road with cross-traffic), with the outcome of the analytics itself feeding into the priority for that drone feed in future.

The rest of the paper is organized as follows: We briefly describe related work in Sec. II, present the domain and technical problem in Sec. III, propose our solution approach in Sec. IV, discuss the implementation details in Sec. V, offer our evaluation results and analysis in Sec. VI, and conclude in Sec. VII.

II. Related Work

Several approaches using smart technologies such as sensor based walking assists [4] and smart canes [5] that senses the environment and warns users about dangers have been developed to assist the visually impaired with outdoor navigation. However with the increasing popularity of computer vision, steps have been taken to provide a way to visualize the surroundings by visually impaired persons using cameras. [6] uses AI powered backpack to analyze the environment using computer vision and warns users about threats using voice assistant. Smart guiding glasses [7] uses depth cameras along with ultrasonic sensors and warns users using vibrations and voice. Recent apps like Lookout which is being developed by Google India [8] to help the visually impaired are also working towards this domain.

[9], [10] explores the use of drones for navigating visually impaired people. [11] have evaluated the adaptation of drone as navigation guidance by blind runners where the runners follow the drone using rotor sound. The experiment was conducted in a highly managed setting which is impractical to be adopted on a large scale. Integrating UAVs with computer vision algorithms helps achieve a 360°view of the runner's surrounding and leveraging the benefits of edge computing provides real-time analytics of the video feeds for navigation of the runner in a dynamic environment.

Intelligent offloading of tasks to accelerated edge server in a dynamic environment within hard latency requirements remains a challenging problem to solve. [12] reduces energy consumption within fog computing environments without violating service latency constraints by intelligently performing partial task offloading. [13] uses a novel graph min-cut algorithm to dynamically map microservices to the edge or the cloud to satisfy application-specific response times across a 5G network. [14] uses a service-oriented approach for task offloading, where the decision whether to invoke a service locally or on a nearby server is taken at runtime. [15] proposes a novel Mission Scheduling Problem that maximizes the utility from the activities while meeting activity deadlines as well as energy and computing constraints.

III. Problem Scenario

A Visually Impaired Person (VIP) is followed by a buddy drone from a fleet of such drones. Each such drone has an on-board monocular camera such that it can view the person as well as the path that lies ahead of them. The two key capabilities that the platform should offer are the identification, tracking and navigation guidance of the VIP using just the camera feed; and real-time threat-perception and warning for VIP based on situation awareness. We use vision-based analytics supported by scalable computing design to accomplish these.

a) Identifying the VIP: : In order to track the visually impaired person, the drone has to identify them within a public space. We leverage compute vision algorithms to make this identification by analyzing the video feed from the drone. Specifically, the VIP wears identification tags in the form of an accessory such as a distinctive cap or vest, a marker such as Hiro [16] or QR code, or an organization logo. Also, we need a method to easily distinguish the orientation of the person with respect to the drone, i.e., to determine if the drone is facing toward the front or the back of the VIP. In our current design, the VIP wears a hazard vest with the logo of our lab in the back. The buddy drone streams the videos to a mobile edge device, like a smart phone, that the VIP has and serves as a base-station. The video is analysed using a *light-weight object detection DNN model* [17] that is trained to detect the hazard vest within a frame, and subsequently the presence or absence of the logo to detect the orientation.

b) Tracking the VIP: : Here, the goal of the drone is to continue to keep the VIP in its field of view (FOV) with the appropriate orientation, i.e., facing the back of the VIP and with a view of the forward path they are walking along. Here again we use the *object detection model* [17] on the mobile device to detect the movement of the VIP within a frame. Based on this, we use a *PD control loop algorithm* to alter the drone's direction of motion along 6 direction (up/down/left/right/front/back) and the speed in that direction such that the VIP remains in the center of the camera's FOV and within a certain distance range from them. This can also give cues to the VIP based on on-board GPS location on which direction the person should move in based on a pre-planned path. Running the model and control algorithm on the mobile device reduces any latency for this fine-grained tracking and real-time decisions to avoid losing track of the VIP.

c) Threat detection: : Here, the goal is to examine the ambient environment of the video and determine if there are any threats that the VIP should be warned about. This can include upcoming road traffic or crowding, obstructions in the path, potholes or steps, etc. and estimating the distance from such threats. Such video analytics tends to require more sophisticated models and also a suite of models trained to detect different scenarios.

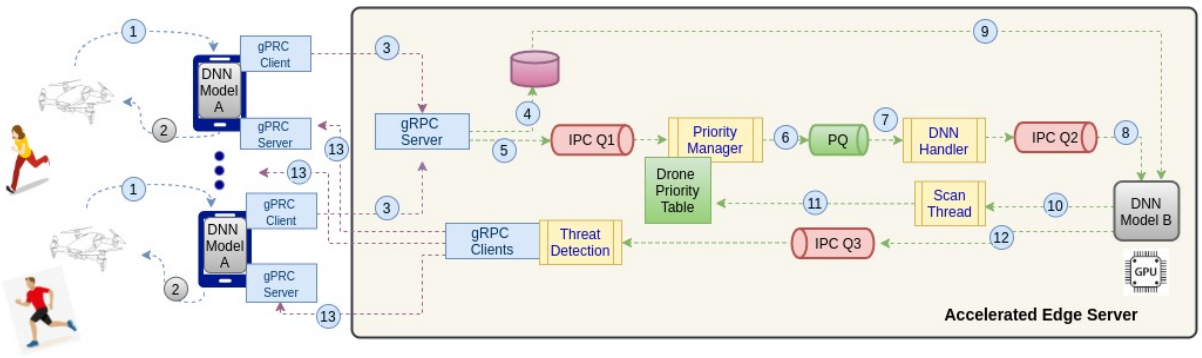

Fig. 1: Architecture Diagram

This becomes infeasible on the limited compute capacities of mobile devices and it is necessary to offload the video streams for processing onto a GPU-accelerated edge device and respond with a warning signal back to the VIP in case threats are detected. Further, there may be several such buddy drones operating at a time and all of them may be sharing the same GPU edge device. This opens up several requirements and challenges that we examine.

We assume that the mobile base-station chunks the video stream and sends it to the GPU-edge over a wireless network, such as WiFi or 4G. **(1)** The video chunks from each drone will need to be processed in a timely manner and in a FIFO order. Fairness should be ensured when processing chunks from different drones by the edge device. **(2)** Further, any delay in processing a chunk from a drone may cause that chunk to become stale, and it may be dropped, in favor of a more recent chunk from that drone. **(3)** Lastly, the video streams from different drones can have different priorities based on the current threat perception. E.g., drones that are navigating in a crowded region may require faster processing of videos while those in open spaces may accept higher latency. Further, the priorities may also change based on the outcome of the video analytics. A people-counting analytic on a chunk may detect an upcoming crowd and future chunks from that drone, including those that are already queued for processing, will need to increase their priority. Similarly, moving from a busy road to an empty street can cause future chunks to drop in their priority.

Our proposed platform design on the GPU edge device to help address these requirements is discussed next.

IV. Solution Approach

We propose a video analytics pipeline and scheduling algorithm on the GPU edge device to meet the different goals of timely and priority based task execution, while responding to dynamic conditions of the drones. Figure 1 details the architecture, using a mix of different concurrent and inter-process execution semantics. The blue, purple and green dotted arrows represent communication over network sockets, Remote Procedural Calls (RPC) and inter-process/thread communication respectively.

The streams are received from the mobile devices by a Google RPC (gRPC) service running on the edge. We denote the streams received by the edge as $S = \{S_1, S_2, S_3, ..., S_n\}$ where S_i represents the video stream received from the i^{th} drone. Each stream is partitioned into chunks of specific time duration, and saved in the local file system. These chunks are given by, $C = \{c_1^1, c_1^2, ...c_1^m, c_2^1, c_2^2, ...c_2^m, c_n^1, c_n^2, ..., c_n^m\}$ of duration d seconds and having f frames each, where c_i^j represents j^{th} chunk of i^{th} video stream (Fig. 1 ④).

Each chunk that is created is associated with a *metadata tuple* which includes the drone id, the file path to the video chunk and the last modified chunk timestamp, i.e. the timestamp of the last frame in the chunk, and puts it in the inter-process communication queue (IPC Q1) (Fig. 1 ⑤).

The *priority manager* process removes each chunk from the IPC Q1 and puts it in a priority queue (PQ) data structure after assigning a priority to the chunk. The chunk priority depends on the current priority assigned to each drone, which is based on the current threat perception for its surroundings and maintained in a *drone priority table* that contains the ⟨drone_id, priority, update_timestamp⟩ (Fig. 1 ⑥). Next, the *DNN Handler* pulls out the chunk with the next highest priority from the front of the queue and delegates it to a thread in its local thread pool which reads the video chunk from the file path and invokes the DNN model on the frames present in the chunk (Fig. 1 ⑧, ⑨).

The output from the DNN model forked to a *Threat Detection* process which maps the output values or classes from the DNN to corresponding warning messages, in case some threshold is crossed, and sends the message back to the mobile device to notify the user (Fig. 1 ⑫, ⑬).

The model output is also sent to the *scan thread* which has multiple roles, one of which is to update the drone priority table based on the DNN model's output (Fig. 1 ⑩, ⑪). Using a threat perception logic, the output of the DNN model is mapped to a priority for the drone matching the video chunk. If the new priority matches the existing priority for the drone in the table, then we just update the update_timestamp for the entry with the chunk's timestamp. This maintains the freshness of the priority. If the priority has changed, then the entry is updated with the new priority and the chunk timestamp.

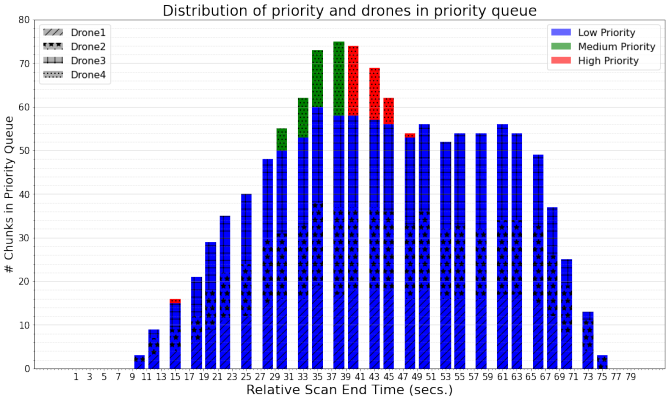

Fig. 2: Distribution of priority and drones in priority queue

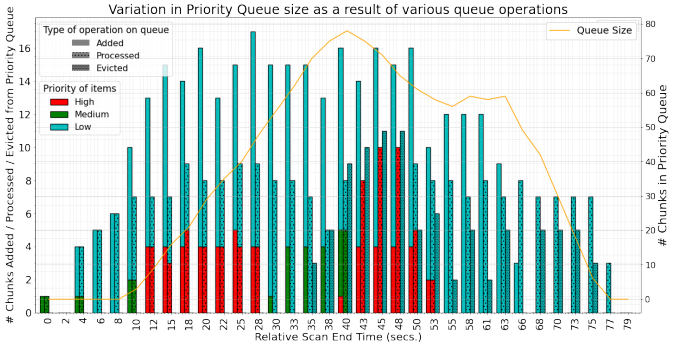

Fig. 3: Variation of priority queue size with queue operations

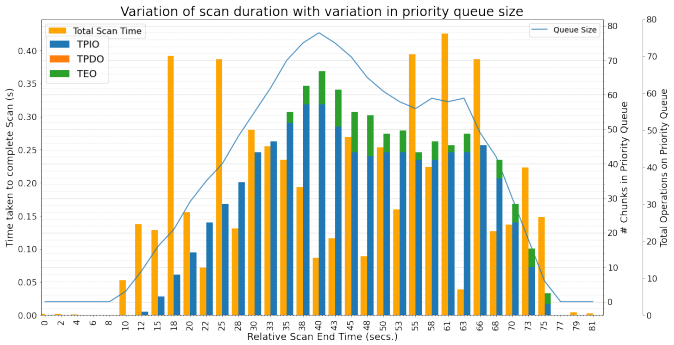

Fig. 4: Variation of scan duration with priority queue size. Legends in the plot are given by Total Priority Increase Operations (TPIO), Total Priority Decrease Operations (TPDO), Total Eviction Operations (TEO)

The priority entry is changed only if the chunk's timestamp is newer than the prior timestamp in the table. This ensures monotonicity of the priority updates even with out-of-order processing of chunks from the same drone due to multi-threading.

The Priority Queue (PQ) will cause chunks for drones with a higher priority to be returned for processing, while the chunks for drones with a low priority will get *starved*. The scan thread avoids this starvation by periodically scanning all entries in the PQ. At the start of each scan, it takes a snapshot of the drone priority table and increments the *scan index* of each chunk by 1. The scan index is used to update the priority of a chunk based on how long it has stayed in the queue without being processed. We define a *priority step*, which is the interval of scans after which the priority of a chunk should be increased. The current priority of the chunk is updated using equation:

$$cp = ip + \left(\frac{csi - ips}{ps}\right)$$

where cp is the current priority of the chunk, ip is the initial priority when the chunk was placed in the queue, csi is current scan index, ips is initial priority step as a baseline, and ps is the priority step.

Also, the scan index handles *stale chunks* that have stayed in the queue long enough that they are irrelevant and can be evicted to make space for fresher chunks. It drops the chunks after s scans, where this is a configurable parameter.

V. Implementation

We implement our framework in Python as it provides the flexibility to integrate with off-the-shelf DNN Models and also interface with buddy drones. Ryze Tello drones powered by DJI are used as the buddy drones for each VIP due to its light-weight and nano form-factor. The drone follows the VIP wearing a hazard vest and streams the video feed to the connected mobile device over a UDP socket connection. The mobile device uses a Python wrapper to interact with the drone using the Tello API and runs a light-weight YOLOv3 model to detect the hazard vest. We distinguish the front-view of the runner from the rear by using the Google Tesseract OCR model [18] to detect text printed on the back of the hazard vest. The output of the DNN Model triggers the navigation commands to control the movement of the drone, sent over the socket connection.

The mobile device simultaneously streams the received video over a gRPC connection to the accelerated edge device for further video processing. We use an Nvidia Jetson Xavier Nano developer kit as our GPU edge device, which has 128-core Maxwell GPU, quad-core ARM A57 CPU @ 1.43 GHz, 4GB RAM and SD-card storage. The Nano runs a YOLOv5 model for counting the number of vehicles and people in a frame [19], and based on the crowd density, updates the threat perception.

VI. Evaluation

A. Setup

We perform detailed experiments using videos streamed from drones to understand the operations on each video chunk throughout its lifespan in the pipeline. Specifically, evaluate the queuing policies, the dynamic updates to the priorities, the eviction policy, and the performance of DNN execution.

We replay and stream real-videos collected from 4 Tello drones with initial drone priorities set to low priority for three drones $\{d_1, d_2, d_3\}$ and drone d_4 set to high priority. Each mobile device sends these videos at 30 fps and 720p

resolution to the edge device over a 1 $Gbps$ LAN connection. We run the experiment over a 3 min recording period. The gRPC server on the Jetson Nano creates video chunks of 1 s duration and downsamples it to 1 fps. The scan interval is set to be 2 s and the eviction threshold is 10 s. The priority step is 1 second and we have three drone priority values of low, medium and high that map to values 2, 5 and 7 respectively. We run the experiment with 1 DNN thread.

B. Priority Queue Behavior

Fig. 2 visualises the priority distribution of chunks from different drones present in the Priority Queue (PQ) at end of every scan. The queue size at time t_1 is given as:

$$queuesize(t_1) = queuesize(t_0) + a(t_1) - p(t_1) - e(t_1)$$

where a, p and e are the number of chunks that were added, processed and evicted at that scan.

Fig. 3 details the variation of the PQ contents and operations across time, from the start of the execution. The *queue size* is shown by a yellow line on the right Y axis. The size of the PQ increases when chunks are added and the size reduces when either the chunks are removed for processing by the DNN Model or they are evicted due to staleness. The number of chunks affected by *different operations* in a scan interval are shown in the bar plot on the left Y axis using different textures, while the *priority of the affected chunks* are shown in different colors of the bar.

The PQ size linearly increases with time and reaches a maximum value of 78 at the end of the 40^{th} scan. The initial chunks added to the PQ are immediately processed since the DNN thread is idle and can schedule them on the DNN model, thus maintaining the PQ size at 0. However, as the input rate for chunk arrival exceeds the chunk processing rate by the DNN, the chunks start getting queued up. The *scan thread* starts increasing the priority of chunks by the end of 12^{th} scan. This correlates with the operations done by the scan thread in Fig. 4. At the end of 12^{th} scan, we can see priority update operations start. Similarly, there is a decline in the queue size when eviction kicks in after the 35^{th} scan. All the chunks that are discarded are always low priority chunks, giving room to relatively higher priority chunks to get processed. This validates the effectiveness of the starvation and eviction logic. Towards the end of the experiment, as new chunks are not added, the evictions are negligible and queue size decreases as the remaining chunks complete processing.

C. Analysis of Scan Duration

Fig. 4 studies the variation of scan duration, i.e., the cost of performing each periodic scan and applying the priority update and eviction logic, with the variation in the PQ size and the number of operations performed in a scan. As expected, the time to complete a scan increases with the queue size in the initial part of the experiment. However, it starts reducing after the 30^{th} scan and reaches a minimum duration of 0.09 seconds even as the queue size reaches its peak at the 40^{th} scan. However, if we consider the changes to the chunk priorities in Fig. 3 between the 30^{th}–40^{th} scan, there are no drone priority updates that happen. This reduces the processing time per scan since no changes to a drone priority means priorities change only due to the scan index increasing.

VII. Conclusion

In this paper, we have explored the use of UAVs as buddy drones that assist VIP in an active lifestyle. We have designed a framework that processes drone feeds on the mobile device for real-time tracking tasks and offloads the feeds to the GPU edge server for more complex video analytics such as threat detection. Our design aims to guarantee the processing of higher priority videos and intelligently enacts priority update for fairness and due to change in threat perception, and eviction policies. As a future work, we propose to optimize the solutions for more sophisticated task scheduling problems and explore dynamic drone routing to respond to current situations.

References

[1] A. Ollero, J. Ferruz, F. Caballero, S. Hurtado, and L. Merino, "Motion compensation and object detection for autonomous helicopter visual navigation in the comets system," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, 2004.

[2] S. Hayat, R. Jung, H. Hellwagner, C. Bettstetter, D. Emini, and D. Schnieders, "Edge computing in 5g for drone navigation: What to offload?" *IEEE Robotics and Automation Letters*, vol. 6, no. 2, 2021.

[3] "Blindness and visual impairment fact sheets," 2021.

[4] M. M. Islam, M. S. Sadi, K. Z. Zamli, and M. M. Ahmed, "Developing walking assistants for visually impaired people: A review," *IEEE Sensors Journal*, vol. 19, no. 8, 2019.

[5] "WeWalk: Enhancing the mobility of visually impaired people."

[6] D. Brown, "Researchers design an AI-powered backpack for the visually impaired," 2021.

[7] J. Bai, S. Lian, Z. Liu, K. Wang, and D. Liu, "Smart guiding glasses for visually impaired people in indoor environment," *IEEE Transactions on Consumer Electronics*, vol. 63, no. 3, 2017.

[8] A. Al-Heeti, "Google expands Lookout app for people who are blind or vision-impaired," 2020.

[9] M. Avila, M. Funk, and N. Henze, "Dronenavigator: Using drones for navigating visually impaired persons," in *ACM SIGACCESS Conference on Computers & Accessibility*, 2015.

[10] M. Avila Soto, M. Funk, M. Hoppe, R. Boldt, K. Wolf, and N. Henze, "Dronenavigator: Using leashed and free-floating quadcopters to navigate visually impaired travelers," in *ACM SIGACCESS Conference on Computers & Accessibility*, 2017.

[11] M. Al Zayer, S. Tregillus, J. Bhandari, D. Feil-Seifer, and E. Folmer, "Exploring the use of a drone to guide blind runners," in *ACM SIGACCESS Conference on Computers and Accessibility*, 2016.

[12] X. Zhang, A. Pal, and S. Debroy, "Effect: Energy-efficient fog computing framework for real-time video processing," in *IEEE/ACM CCGrid*, 2021.

[13] K. Rao, G. Coviello, W.-P. Hsiung, and S. Chakradhar, "Eco: Edge-cloud optimization of 5g applications," in *IEEE/ACM CCGrid*, 2021.

[14] T. Kasidakis, G. Polychronis, M. Koutsoubelias, and S. Lalis, "Reducing the mission time of drone applications through location-aware edge computing," in *IEEE International Conference on Fog and Edge Computing (ICFEC)*, 2021.

[15] A. Khochare, Y. Simmhan, F. B. Sorbelli, and S. K. Das, "Heuristic algorithms for co-scheduling of edge analytics and routes for uav fleet missions," in *IEEE INFOCOM*, 2021.

[16] A. Colley, D. Wolf, K. Kammerer, E. Rukzio, and J. Häkkilä, "Exploring the performance of graphically designed ar markers," in *International Conference on Mobile and Ubiquitous Multimedia*, 2020.

[17] "YOLOv3: Real-Time Object Detection Algorithm."

[18] "Tesseract OCR: An optical character recognition (OCR) engine."

[19] A. Agnihotri, "Implementing Real-time Object Detection System using PyTorch and OpenCV," 2021.