# A Trade-off Analysis of Latency, Accuracy, and Energy in Task Offloading Strategies for UAVs

Egemen Erbayat*, Rujia Zou*, Xianglin Wei†, Guru Venkataramani*, Suresh Subramaniam*
*The George Washington University, Washington, DC, USA, {erbayat,rjzou,guruv,suresh}@gwu.edu
†The 63rd Research Institute, National University of Defense Technology, Nanjing 210007, China, wei_xianglin@163.com

***Abstract*—Object detection is a critical aspect of computer vision, particularly for unmanned aerial vehicles, commonly known as drones. Drones depend on object detection to perform various tasks such as surveillance or search and rescue. However, drones face challenges such as limited computation and small object detection, which requires high accuracy within tight timeframes. The emergence of edge computing allows real-time processing, balancing accuracy and latency by offloading tasks to more powerful edge servers. However, efficient resource allocation remains a challenge. This paper presents results from real-world experiments that explore the trade-offs among latency, accuracy, and energy consumption in UAV object detection. Notably, the experiments demonstrate that offloading can reduce latency, particularly for tasks demanding high levels of accuracy. These results can inform offloading and resource allocation decisions.**

***Index Terms*—Edge computing, offloading, UAV**

## I. Introduction

Object detection has gained significant attention in recent years due to its diverse applications across various domains, making it a fundamental task in computer vision [1]. For example, the development of unmanned aerial vehicles (UAVs), commonly known as drones, has made object detection a vital aspect of enabling autonomous capabilities in aerial platforms. The integration of object detection techniques with drone technology has brought about significant changes in various fields such as surveillance, search and rescue, environmental monitoring, and infrastructure inspection, among others. However, drones have their own challenges, such as having low computational capacities, energy constraints, and having small objects in the images. Considering those constraints, higher accuracies are aimed at with limited time [2].

With the deployment of 5G networks and the introduction of edge computing, a new computing paradigm arises: making computing services closer to the user at the network. Although cloud servers have more powerful computing, offloading to a cloud server has higher latency, which makes some timely critical tasks that require very small latencies impossible [3]. The use of edge computing reduces the distance data is required to travel in case of offloading. This latency drop makes offloading an option in real-time processing besides onboard processing. The other advantage of offloading is the ability to use more complex models to achieve higher accuracies. This trade-off involves the need to efficiently solve resource allocation and decision-making problems. In the literature, various reinforcement learning approaches have been proposed, often relying on assumptions and simulation results (e.g. [4]–[10]). However, a notable gap exists in the lack of real-world analysis. To address this gap, in this work, we conduct real-world experiments to see the trade-offs between system metrics such as latency, accuracy and energy. The other contribution is that we evaluate each image in the dataset, one by one, and under the system conditions that exist when the image arrives. As such, these results are expected to be useful for allocating resources and making decisions on whether to offload to edge servers or to execute the task on-board.

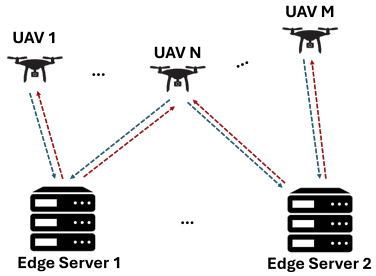



Fig. 1: System model.

The rest of the paper is organized as follows. We present the system model in Section II. Afterward, we present our experiment setup, experiment results and analysis in Section III and we present our conclusions in the last section, Section IV.

## II. System Model

Drones are used in edge computing in two ways. They can be mobile edge servers or mobile "users". If they are mobile edge servers, they need to have high computational power, and other users offload their tasks onto the drones. On the other hand, they can be users themselves and offload their tasks to another device with higher computational power. We consider a system where drones are users, and we have edge servers with fixed locations, as shown in Figure 1. Therefore, there are two options for drones: process onboard or offload. Each drone has to make a decision on each arriving task considering the system's current situation, computational power, and computational load. The decision-making algorithm considers the processing time, accuracy, or resource usage such as energy and makes its decision. Two decisions that can be made by a UAV are shown in Figure 2. We consider object detection as the task and use two different models

2024 IEEE Cloud Summit | 979-8-3503-7006-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/CLOUD-SUMMIT61220.2024.00015

979-8-3503-7006-5/24/$31.00 ©2024 IEEE
DOI 10.1109/Cloud-Summit61220.2024.00015

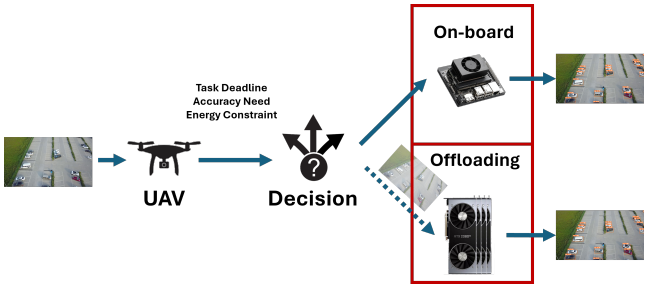



Fig. 2: Decision Process.

with different parameter sizes to evaluate the trade-offs among processing time, accuracy, and resource usage. In this work, we perform real-world experiments to see possible outcomes. Through these experiments, we figure out which method might be better for certain situations. For example, one method might be quicker but less accurate, while another might be slower but more accurate. We want to see how they compare in different situations so that when it comes time to make decisions, we will have a better understanding of what might work best.

## III. Experiments

### A. Experimental Setup

To obtain results, we employ a drone equipped with sensing capabilities and an edge server boasting considerable computational power. This setup allows for efficient data processing and analysis in real-time, enabling various applications such as surveillance, mapping, or environmental monitoring. To ensure that our results are applicable to a broader range, we utilize a real-world dataset. Details are as follows:

- **Dataset:** Visdrone test set [11] consists of aerial imagery captured by the drone. The dataset has 16 different video clips consisting of a total of 6,333 frames from various places with ground truths. Using these frames, we simulate a camera and feed a drone.
- **Devices**:
  - **UAV:** Jetson Orin Nano Developer Kit is used as a computation device that has a 1024-core NVIDIA Ampere architecture GPU with 32 Tensor Cores and a 6-core ARM CPU with 8 GB RAM. The developer kit has 40 TOPS AI performances and it provides significantly powerful AI capabilities at low cost.
  - **Edge Server:** 12-core Intel(R) Core(TM) i9-7920X CPU @ 2.90GHz, 128 GB RAM and 4 × NVIDIA GeForce RTX 2080 Ti 12 GB GPU.
- **Model Selection:** We utilize various You Only Look Once (YOLO) [12] models in this process. YOLO is a widely used system for detecting objects in real time, which employs a single neural network to directly predict bounding boxes and class probabilities from full images in one evaluation. YOLO version 8 has five different models with different parameter sizes and the following is the order of the parameters listed in increasing size: n, s, m, l, x. An increase in parameter size may increase inference time and accuracy. After testing five of them, we observe that the inference time for l and x is higher than for m, whereas their accuracy is similar. Likewise, we observe the same relationship between s and n. Therefore, we pick n and m as our two options. n represents the simple model, while m represents the complex model.
- **Offloading**: To overcome bandwidth limitations, we utilize resizing and JPEG compression [13] while offloading images from the UAV to the edge server. The reason behind resizing is that ML models mostly need a specific image size and do resizing automatically. Therefore, resizing before compression and transmission helps us to reduce compression/decompression latency and transmission latency without changing the output. By resizing data, the process of transmission is accelerated by approximately 70%. After resizing images to YOLO input size, we apply compression. Compression reduces the number of packets that are transmitted by around 1/6th of their original amount with only 5% compression loss. To achieve low latency, we also utilize The User Datagram Protocol (UDP) to offload images. Figure 3 depicts the system we use for offloading. We have an edge server and it is connected to a router using an Ethernet cable. The router receives packets from the UAV wirelessly and sends them to the edge server.

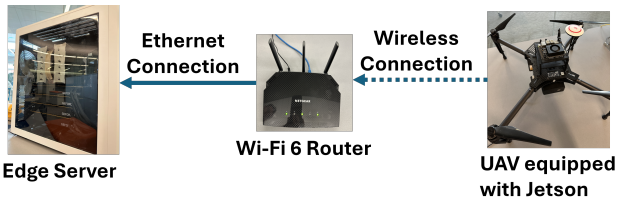



Fig. 3: Connections for offloading.

- **Performance Metrics:**
  - **Latency:** We consider latency for each image or video frame. The latency has two parts: inference time and offloading time if applicable. Inference time is the time required to execute an object detection model. Offloading time is the time spent on all the operations required for offloading. If the decision is onboard processing, then offloading time is zero. In the case of offloading, we have to consider the sum of resizing, compression/decompression, and transmission time when the image is offloaded to the edge server.In that case, operations start on UAV and end at the edge server. To have accurate time measurement, we synchronize all devices' clocks using Network Time Protocol (NTP). Using NTP, we are able to use timestamps from different devices while preserving measurement precision.
  - **Accuracy:** We evaluate accuracy performance using mean average precision (mAP) and mean average

recall (mAR) [14]. To better understand mAP and mAR, we first review some related concepts. Precision, Recall, and Intersection over Union (IoU) are fundamental metrics used in evaluating the performance of object detection systems.

* Precision measures the accuracy of the positive predictions made by the model. In object detection, precision represents the ratio of correctly detected objects to all detected objects:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

* Recall, also known as sensitivity or true positive rate, measures the completeness of the positive predictions made by the model. In object detection, recall represents the ratio of correctly detected objects to all ground truth objects:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

* IoU is a measure of the overlap between the predicted bounding box and the ground truth bounding box. It is calculated as the ratio of the intersection area to the union area:

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

  IoU is often used as a criterion for determining whether a predicted bounding box is considered a true positive detection.
* The Precision-Recall (PR) curve is a graphical representation of the trade-off between precision and recall for a given classification model.

After that, Average Precision (AP) and Average Recall (AR) are calculated for each object class. AP measures the area under the precision-recall curve. It is calculated by averaging the precision values obtained at different recall levels:

$$AP = \frac{\sum_{k=1}^{n}(P(k) \cdot \Delta R(k))}{R_{\text{max}}}$$

where:

* $P(k)$ is the precision at the $k$-th point,
* $\Delta R(k)$ is the change in recall from $k-1$ to $k$,
* $R_{\text{max}}$ is the maximum recall value.

Average Recall measures the average recall obtained at different precision levels. It is calculated by averaging the recall values at different precision levels:

$$AR = \frac{\sum_{k=1}^{n}(R(k) \cdot \Delta P(k))}{P_{\text{max}}}$$

where:

* $R(k)$ is the recall at the $k$-th point,
* $\Delta P(k)$ is the change in precision from $k-1$ to $k$,
* $P_{\text{max}}$ is the maximum precision value.

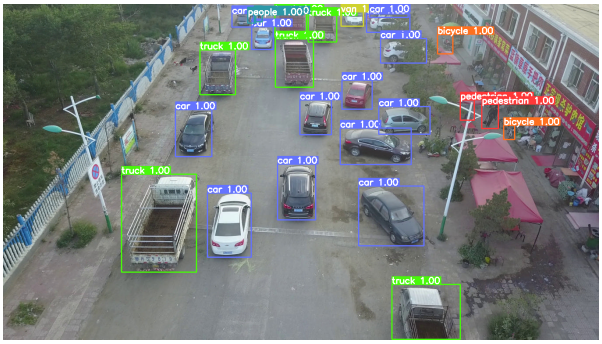
These metrics provide insights into a model's performance in terms of precision and recall, which are crucial for tasks like object detection and information retrieval. In this way, we can penalize the absence of object detection and redundant detections. After obtaining AP and AR, mAP and mAR are calculated by averaging AP and AR values obtained for each class, respectively. We use different thresholds for mAP and mAR. They can be explained as follows: $\text{mAP}_{\text{all}}$ is the mAP obtained by averaging over all IoU thresholds from 0.5 to 0.95 with step size 0.05. $\text{mAP}_{0.5}$ and $\text{mAP}_{0.75}$ are at thresholds 0.5 and 0.75, respectively. $\text{mAR}_{\text{max=k}}$ represents the average maximum recall score for k detections per image, calculated over various IoU thresholds and classes. We have scores for each $k$ in $1, 10, 100$.

– **Energy Consumption:** The third metric is the energy consumed. Similar to latency, we have two energy components – the energy consumption for offloading and the energy consumption for inference. We are able to measure the power consumption of a system in real-time with Python libraries as software. We measure instantaneous power consumption at regular intervals. We noticed that setting this interval too small caused instability in measurements. Therefore, we use $0.1$ second as our power measurement interval. We are interested in the active power consumed for task execution and do not want to include the standby power, which is the power consumed by the system when it is not being actively used. To measure the standby power, we observe the system for a dedicated period of time without executing any additional tasks. During this period, we measure the power consumption of the system at regular intervals for 2 minutes to determine the average standby power. We measure the average standby power for each model-dataset pair. After that, we continue to measure power periodically and asynchronously from model inference. After each inference is done, we use the last power measurement value as the power consumption of the corresponding inference. We estimate the energy used for inference by assuming that the power consumption is constant during inference, i.e., by multiplying the difference between standby and inference powers with the inference time.

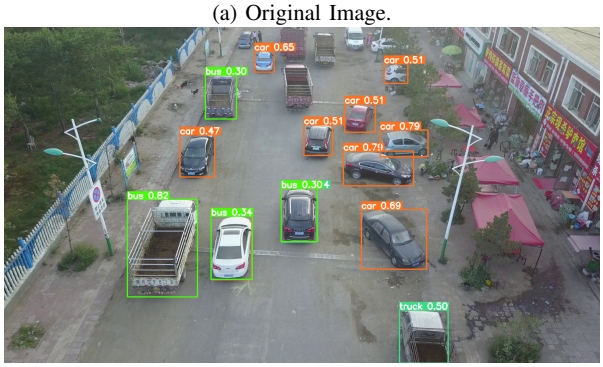
### *B. Experimental Results and Analysis*

We run our experiments 20 times to minimize the effect of noise. We calculated the mean and standard deviation of these 20 samples. For every frame, we have calculated the mean and standard deviation of each metric based on 20 samples of experimental output. To enhance the presentation of our

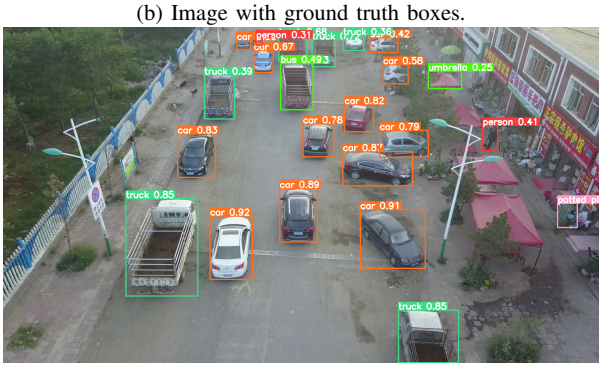
(a) Original Image.



(b) Image with ground truth boxes.



(c) Image with YOLOv8-n predictions.



(d) Image with YOLOv8-m predictions.

Fig. 4: Example frame from video uav0000077_00720_v.

TABLE I: Average results on set uav0000077_00720_v (780 images, image size ≈ 3MB).

| | Device | Model | $T_I$ | $E_I$ | $T_O$ | $E_O$ | $\text{mAP}_{\text{all}}$ | $\text{mAP}_{0.5}$ | $\text{mAP}_{0.75}$ | $\text{mAR}_{\text{max=1}}$ | $\text{mAR}_{\text{max=10}}$ | $\text{mAR}_{\text{max=100}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mean | UAV | YOLOv8n | 29.39 | 60.38 | 0 | 0 | 11.76 | 17.74 | 12.97 | 5.14 | 12.82 | 12.95 |
| | UAV | YOLOv8m | 67.36 | 237.27 | 0 | 0 | 20.82 | 32.12 | 22.58 | 7.82 | 21.95 | 22.24 |
| | Edge Server | YOLOv8n | 5.96 | 293.11 | 26.22 | 5.82 | 11.44 | 17.22 | 12.71 | 5.09 | 12.32 | 12.46 |
| | Edge Server | YOLOv8m | 9.62 | 579.21 | 26.15 | 5.64 | 20.10 | 30.92 | 21.75 | 7.24 | 21.24 | 21.55 |
| Std. Dev. | UAV | YOLOv8n | 3.63 | 11.25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | UAV | YOLOv8m | 4.84 | 21.38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Edge Server | YOLOv8n | 0.32 | 23.18 | 2.17 | 0.9 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Edge Server | YOLOv8m | 0.32 | 32.34 | 1.91 | 0.9 | 0 | 0 | 0 | 0 | 0 | 0 |

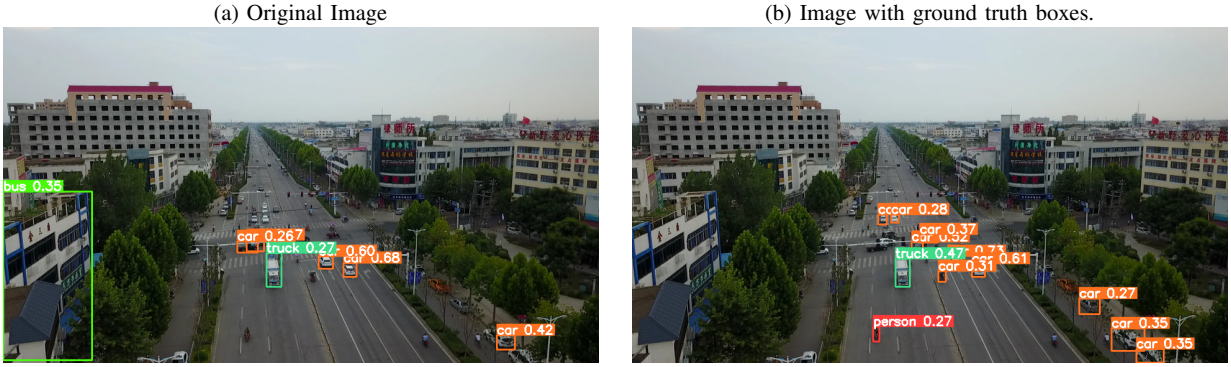
Device: The device on which the task is being run
$T_I$: Single image inference latency in milliseconds
$E_I$: Energy used for single image inference in millijoules
$T_O$: Single image offloading latency in milliseconds
$E_O$: Energy used for offloading a single image in millijoules

findings, we computed the average of our experimental results and verified our analysis using sample frames from it. The average results of our experiments for two sets are displayed in Table I and Table II. Sample frames are exhibited in Figure 4 and Figure 5, while Figure 4a and Figure 5a showcase the original images. Figure 5a presents smaller objects in pixels than Figure 4a.

Having smaller objects directly affects the output mAP and mAR scores because YOLOv8 is trained with relatively larger objects. It is visually seen that for both groups, YOLOv8n misses some detections that YOLOv8m is able to detect. Yolov8m misses some small objects, especially in Figure 5. However, for mid-size objects, YOLOv8m returns sufficient output, shown in Figure 4d. When the average results are evaluated, in all cases, YOLOv8m has an mAP score that is almost more than double that of YOLOv8n, but it has a relatively longer inference time and higher energy consumption. The set uav0000077_00720_v has mid-size objects and the set uav0000188_00000_v has smaller objects. When switching from YOLOv8n to YOLOv8m, mAP scores doubled and tripled for the set with midsize objects and the set with small-size objects, respectively. That is, as the object size decreases, the ratio of mAP scores between the complex and simple models increases. We see a similar but more sharp increase in the ratio of mAR scores between the complex and simple models. The reason behind that is mAP reflects

(a) Original Image

(b) Image with ground truth boxes.




(c) Image with YOLOv8-n predictions.

(d) Image with YOLOv8-m predictions.

Fig. 5: Example frame from video uav0000188_00000_v.

TABLE II: Average results on set uav0000188_00000_v (260 images, image size ≈ 1.5MB).

| | Device | Model | $T_I$ | $E_I$ | $T_O$ | $E_O$ | $\text{mAP}_{\text{all}}$ | $\text{mAP}_{0.5}$ | $\text{mAP}_{0.75}$ | $\text{mAR}_{\text{max=1}}$ | $\text{mAR}_{\text{max=10}}$ | $\text{mAR}_{\text{max=100}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mean | UAV | YOLOv8n | 28.53 | 63.61 | 0 | 0 | 1.72 | 3.92 | 1.13 | 0.63 | 1.99 | 2.00 |
| | UAV | YOLOv8m | 65.84 | 261.19 | 0 | 0 | 5.09 | 9.11 | 4.57 | 2.55 | 5.39 | 5.68 |
| | Edge Server | YOLOv8n | 5.74 | 291,46 | 25.23 | 5.13 | 1.82 | 3.94 | 1.32 | 0.68 | 2.09 | 2.10 |
| | Edge Server | YOLOv8m | 9.50 | 632.77 | 25.33 | 6.24 | 4.99 | 8.85 | 4.56 | 2.45 | 5.32 | 5.58 |
| Std. Dev. | UAV | YOLOv8n | 3.37 | 11.38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | UAV | YOLOv8m | 4.45 | 20.84 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Edge Server | YOLOv8n | 0.32 | 18.27 | 2.35 | 0.7 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Edge Server | YOLOv8m | 0.21 | 27.42 | 2.16 | 0.7 | 0 | 0 | 0 | 0 | 0 | 0 |

Device: The device on which the task is being run
$T_I$: Single image inference latency in milliseconds
Device: The device on which the task is being run
$E_I$: Energy used for single image inference in millijoules
$T_O$: Single image offloading latency in milliseconds
$E_O$: Energy used for offloading a single image in millijoules

the model's success in detecting objects, while mAR reflects the model's success in retrieving all the relevant objects. If the model is not powerful enough to perform well with small objects, it may miss some detections, which will penalize the mAR score more severely. Therefore, the use of a complex model is a must when the task is to detect small objects. In our case, if higher accuracy is needed, the YOLOv8m model is required but the rise in on-board latency causes a negative impact on the real-time processing capability of UAVs. To cope with that, offloading is necessary to continue real-time processing by sacrificing more energy. The other important point is that the UAV is busy for the whole time period for on-board processing, but it is busy only for about 10 milliseconds when offloading is chosen, which is for resizing, compression and transmission. That is, selectively offloading images may be an option to continue real-time processing when a UAV has more than one camera. It is important to note that the output of object detection remains the same for the same input image and model combination. However, lossy compression can impact object detection performance. We observe this effect on mAP and mAR scores when comparing UAV and edge servers for the same dataset and model combination. Although the difference is negligible, we do not notice any change in the detected object count. The difference in mAP

and mAR scores results from slight variations in bounding box coordinates. Note that $mAP_{0.75}$ is always lower than $mAP_{0.5}$ because as the IoU threshold increases, the criteria for a correct detection become stricter. This results in fewer true positives, lower precision, and lower mAP scores. Similarly, the mAR score increases with the maximum number of detections allowed since more detections lead to capturing more true positive instances, thereby increasing recall. However, due to the limited number of objects in each image, increasing the maximum detection has no effect after a certain point. This can be observed by comparing $mAR_{max=10}$ and $mAR_{max=100}$.

As a result, there are many decision possibilities. If the objects being detected are large in size, the task is the detection of only closer objects or the task deadline is very tight, running YOLOv8n on-board might be considered. However, running YOLOv8m on-board is not suggested because offloading the task has lower latency unless there is an energy constraint and time is generous. On the other hand, offloading for running YOLOv8m may be the decision when high-accuracy results are required; detecting farther objects or detecting objects moving away is desired, such as surveillance tasks. The reason is that there is a decrease in performance as the size of objects decreases. Similarly, running YOLOv8n on an edge server is not preferred because it has higher energy consumption and latency than running on-board unless the UAV needs to run more than one task at the same time and energy is constrained. Note that energy constraints on model usage can be negligible for UAVs since the energy consumption while inference or offloading for UAVs is significantly lower than the energy required for hovering [15]. However, it is possible that a single UAV may need to run multiple inferences at the same time. In such cases, it is advisable to run YOLOv8m on an edge server. The reason behind this is that even though running YOLOv8m on an edge server requires minimal additional resources, it provides high accuracy. When we consider the case with multiple UAVs connected to a single edge server, offloading may not be an option for all UAVs, even if the edge server may serve multiple UAVs, because of the edge server's capacity. It may be suggested to offload processing when the UAV is farther from the object and perform on-board processing when it is closer to the object, based on the relationship between object size and model size.

## IV. Conclusion

In conclusion, UAVs have many crucial tasks such as search and rescue, surveillance, and infrastructure inspection, but they face challenges such as limited computation and limited time. Edge computing offers higher computational power with low latency, which may make offloading an option considering the system's needs. This paper presents experiments exploring trade-offs in UAV object detection latency, accuracy, and energy consumption to see possible outcomes of decision-making. The experiments demonstrate that offloading can reduce latency when it comes to tasks that require higher accuracy. However, there is no general and precise decision between on-board processing and offloading. Algorithms that take into account the objective and the most recent information of a system are necessary for decision-making. Future work should focus on developing adaptive algorithms that can dynamically switch between on-board processing and offloading based on real-time analysis of the UAV's operational context and objectives. This includes investigating machine learning models that predict the optimal computation mode by considering factors such as network conditions, task complexity, and energy constraints.

## Acknowledgment

This work was supported in part by NSF grant CNS-2219180.

## References

[1] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object Detection in 20 Years: A Survey," *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257–276, 2023.

[2] Y. Akbari, N. Almaadeed, S. Al-ma'adeed, and O. Elharrouss, "Applications, databases and open computer vision research from drone videos and images: a survey," *Artificial Intelligence Review*, vol. 54, 02 2021.

[3] P. McEnroe, S. Wang, and M. Liyanage, "A Survey on the Convergence of Edge Computing and AI for UAVs: Opportunities and Challenges," *IEEE Internet of Things Journal*, vol. 9, no. 17, pp. 15435–15459, 2022.

[4] X. Dai, Z. Xiao, H. Jiang, and J. C. S. Lui, "UAV-Assisted Task Offloading in Vehicular Edge Computing Networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 2520–2534, 2024.

[5] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, and D. Niyato, "Multi-Agent Deep Reinforcement Learning for Task Offloading in UAV-Assisted Mobile Edge Computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 6949–6960, 2022.

[6] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, and D. Niyato, "Multi-Agent Deep Reinforcement Learning for Task Offloading in UAV-Assisted Mobile Edge Computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 6949–6960, 2022.

[7] J. Chen, S. Chen, S. Luo, Q. Wang, B. Cao, and X. Li, "An intelligent task offloading algorithm (iTOA) for UAV edge computing network," *Digital Communications and Networks*, vol. 6, no. 4, pp. 433–443, 2020.

[8] J. Almutairi, M. Aldossary, H. A. Alharbi, B. A. Yosuf, and J. M. H. Elmirghani, "Delay-Optimal Task Offloading for UAV-Enabled Edge-Cloud Computing Systems," *IEEE Access*, vol. 10, pp. 51575–51586, 2022.

[9] J. Xiong, H. Guo, and J. Liu, "Task Offloading in UAV-Aided Edge Computing: Bit Allocation and Trajectory Optimization," *IEEE Communications Letters*, vol. 23, no. 3, pp. 538–541, 2019.

[10] X. Zhang, L. Wang, X. Wu, L. Xu, and A. Fei, "Energy-Efficient Computation Offloading and Data Compression for UAV-Mounted MEC Networks," in *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, pp. 6904–6909, 2023.

[11] P. Zhu, L. Wen, D. Du, X. Bian, H. Fan, Q. Hu, and H. Ling, "Detection and tracking meet drones challenge," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7380–7399, 2021.

[12] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLO," Jan. 2023.

[13] G. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[14] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A Survey on Performance Metrics for Object-Detection Algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 237–242, 2020.

[15] H. V. Abeywickrama, B. A. Jayawickrama, Y. He, and E. Dutkiewicz, "Comprehensive Energy Consumption Model for Unmanned Aerial Vehicles, Based on Empirical Studies of Battery Performance," *IEEE Access*, vol. 6, pp. 58383–58394, 2018.