Multitask Scheduling of Computer Vision Workload on Edge Graphical Processing Units

Arani Bhattacharya
IIIT-Delhi
arani@iiitd.ac.in

Paritosh Shukla
IIIT-Delhi
paritosh18063@iiitd.ac.in

Ansuman Banerjee
Indian Statistical Institute
ansuman@isical.ac.in

Saumya Jaipuria
Indian Statistical Institute
saumyajaipuria@gmail.com

Nanjangud C Narendra
Ericsson Research
nanjangud.narendra@ericsson.com

Dhruv Garg
Ericsson Research
dhruv.s.garg@hotmail.com

***Abstract*—The increasing urbanization in developing countries calls for more efficient and safer transportation systems. A key technique used to enhance such efficiency and/or safety is to utilize running of computer vision algorithms to identify obstructions that may come up, and notify vehicles in real-time. Such real-time detection and notification requires sufficient computation resources located logically and physically close to the cameras. While utilization of edge compute devices has been proposed in the literature, it is unclear how such devices with heterogeneous processing units can handle real-time detection while multi-tasking. In this work, we profile the performance of a few devices with embedded and desktop-quality GPUs, and show that the performance while multi-tasking can be modeled as a submodular function. We utilize this observation to model load-balancing of camera videos as an instance of a submodular welfare problem, and solve it using a greedy algorithm. Our extensive trace-driven simulations show that our technique outperforms the baseline by over 40%.**

I. Introduction

With increasing urbanization in developing countries, a critical problem that has emerged is the rise in the number of traffic accidents. With weaker infrastructures and vehicles with lower safety features, developing countries generally have roads that are more vulnerable to traffic accidents. For example, the number of fatalities per capita is three times higher in Africa than in the European Union [1]. Recent studies [1], [2] show that this is due to (i) lower enforcement of speed limits as compared to that in developed countries, (ii) simultaneous utilization of road space by non-motorized entities, such as bicycles, pedestrians and animals, and (iii) presence of potholes and unexpected obstructions. Current solutions to these problems involve substantial investment in traffic enforcement, which is usually done manually and is expensive in practice.

To mitigate this problem of traffic accidents, a number of studies have proposed for monitoring of city traffic by extensive deployment of cameras. Currently, low-cost cameras costing around $25 can take video streams at a resolution of 1080p [3], which is sufficient to monitor traffic. An automated system of enforcing traffic rules faces two challenges. First, running the machine learning models to identify the obstructions is compute-intensive. This challenge is significant because of the large number of cameras involved, and limited availability of GPU memory. For example, the Indian city of Hyderabad already has installed over 500,000 cameras to monitor traffic [4]. Currently running a single instance of one of the most common computer vision algorithms, YOLO [5], requires over 1GB of GPU memory per video. The largest GPU server currently available commercially, nVidia A100, only has 80GB of GPU memory. Thus, managing the video processing requirements for a city like Hyderabad would require 6250 such GPU servers, costing around $\$312,000$, which is clearly not affordable.

A prominent solution to resolve this challenge is to utilize distributed video analytics [6], [7]. Such distributed video analytics utilizes deployment of multiple compute devices closer to the cameras in locations such as cell towers. This reduces the amount of data needed to be sent to the cloud server. Many modern edge devices, such as Jetson Nano [8] and Jetson AGX [9], usually contain a heterogeneous mix of CPUs and GPUs. Since the number of cameras is usually larger than the number of deployed edge devices, execution of video analytics needs to happen in parallel. A key challenge of such parallel execution is to provide reliable latency while utilizing such edge devices.

Providing such reliable latency is challenging for a number of reasons. First, the wireless network used to send the data has a significant probability of losing the packets [10]. Second, multi-tasking on an edge device with heterogeneous processors of GPUs and CPUs leads to additional uncertainty in execution time. The problem of multi-tasking on edge devices while providing reliable latency is further exacerbated by the fact that unlike server GPUs, edge device GPUs do not have the scope of allocating resources via virtualization [11].

In this work, we first use a statistical measurement of the wireless channels to identify the latency required for reliable

©IEEE. This article has been accepted for publication at IEEE COMSNETS 2023. The authentic version of this article can be accessed at https://dx.doi.org/10.1109/COMSNETS56262.2023.10041358. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

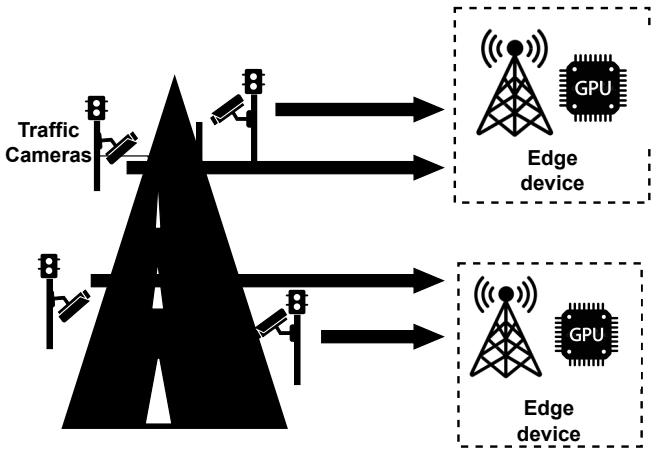

Fig. 1: An illustration of the multi-tasking typically seen in edge devices to process the video frames coming in from traffic surveillance cameras.

delivery of video packets. We then schedule the videos of the different cameras on the edge devices. The goal of this scheduling is to maximize the overall processing throughput. However, such scheduling is challenging as the execution time on parallel execution of tasks on GPUs needs to be accurately modeled. The amount of execution time spent when running parallel jobs on GPUs is known to be non-linear [12]. Maximizing the throughput in such a general case is a non-convex problem, making it difficult to solve.

We further handle the non-linear execution time using a major observation. We observe that the throughput of edge devices on running multiple tasks in parallel, is *submodular* in nature. In other words, there is a diminishing return in terms of throughput with increase in the number of videos processed in parallel. This enables us to model the scheduling of videos as a *submodular welfare problem* [13]. For submodular welfare problems, a greedy allocation of tasks leads to a 0.5 approximation algorithm. We leverage this observation to design our approximation algorithm to handle this problem.

We evaluate our approaches via extensive trace-driven simulations. Our traces consist of instances of actual execution of YOLO-v5 [14] on the edge device Jetson Nano. We also have the locations of the cell towers and the traffic intersections of San Francisco city. Using this data, we compare the performance of our algorithm with other baseline techniques. Our simulations show that our greedy algorithm outperforms the baseline techniques.

Our contributions may be summarized as follows:

- We observe that the throughput of edge devices with GPUs for video processing follows a submodular property.
- Using the submodular observation, we model the load balancing problem on edge devices as a submodular welfare problem, and use a greedy algorithm to solve.
- We evaluate our approach on traffic data traces of San Francisco city using a Jetson Nano, and show that our technique is effective in practice.

The rest of this paper is organized as follows. Section II describes the problem statement, and formally models it. Section III explains the greedy approach. Section IV describes our implementation along with experimental details. Related work is discussed in Section V, and finally, Section VI concludes the paper with suggestions for future work.

II. Formal Model

We consider the following problem. Our goal is to automatically identify *Vulnerable Road Users* (VRU), whose presence on roads and highways has the potential to cause chaos and disruption. As illustrated in Fig. 1, we assume that stretches of the highway are under the jurisdiction of an edge server, which is being served by multiple cameras positioned at specific spots off the highway. The cameras capture video feeds of the highway and transmit them periodically to one or more edge servers, where the feeds are analyzed. The result of the analysis could highlight the presence of obstacles on the road that could potentially hinder motorists. Examples of such obstructions could include accidents, potholes, children playing on the road, animals, etc. Note that identifying such VRUs could potentially require their identification by observing video streams from multiple cameras.

The video streams are sent to an edge device connected to the cell towers over a cellular network. After the video feeds are analyzed, the presence of any such VRU is then detected. The information on the VRUs is then transmitted to oncoming vehicles that could be affected by the presence of the VRUs. The determination of which are the "affected" vehicles is made by the edge server in real time based on locations and trajectories of the vehicles which it detects, and which are approaching the VRU in question.

The total latency to alert the vehicles is, therefore, the sum of the following components:

1) The network latency taken to send the video feeds to the edge device
2) The processing time required to analyze the video on the edge device, which has GPUs in it
3) The time required for cloud processing (if necessary). If there is a complex video query, requiring coordination among multiple video feeds, and at least one of the videos is executing on a different edge device, then the output is sent to the cloud. The cloud server in turn alerts the closest edge device.
4) The latency incurred when the edge device sends alerts to one or more vehicles about the danger ahead.

Note that such an hierarchical edge/cloud model of traffic surveillance has been utilized by prior works [15].

A. A Formal Model of the Vehicle Monitoring System

Camera and Edge Devices: Let $\mathbf{C}$ be the set of cameras used to monitor the roads, and $\mathbf{D}$ be the set of edge devices. A single edge device $D_j \in \mathbf{D}$ receives videos transmitted from one or more cameras, denoted by the subset $\mathbf{T}_j \subseteq \mathbf{C}$. These received videos are then processed by the edge devices

to identify any obstructions on the roads by running a machine learning model. If the required identification requires additional cooperation from multiple edge devices, then the output is sent to the cloud server.

Once the obstructions are identified after completing the processing, warning messages are transmitted from the edge devices to the vehicles. Since the videos of every camera must be processed on some edge device, we have $\cup_{j=1}^{|\mathbf{C}|}\mathbf{T}_j = \mathbf{C}$, i.e. each camera must be reachable to some edge. We assume that the warning messages are sent through a separate reliable protocol such as URLLC [16], as it requires very low throughput but provides high reliability.

Video Transmission: We note that videos are transmitted from the set of cameras $\mathbf{T}_j$ to the edge device $D_j, \forall D_j \in \mathbf{D}$. This video transmission naturally happens over a wireless network through discrete packets of data. Since this is a safety critical application, the wireless network must provide some reliability guarantees. Since wireless networks are inherently lossy in nature, only *stochastic* guarantees are possible [17]. Such guarantees are usually characterized by ϵ_{ij}, where ϵ_{ij} is the probability of losing each packet from camera C_i to edge device D_j. We also assume that the wireless network channels are independent in nature, i.e. transmissions in one channel do not affect the value of ϵ_{ij} for any other channels. This is in general true for cases where data is sent using frequency-division multiplexing, as in cellular networks.

We now model the total time to receive a video frame. As in most video streaming systems, a single frame is said to be received only when all its packets reach the edge device. Since we assume that the wireless network channels are independent of one another, the time t_{ij}^p to transmit a packet is constant for a given camera $C_i \in \mathbf{C}$ and edge device $D_j \in \mathbf{D}$. If a packet is lost, the camera has to retransmit it. We denote the total time required to transmit the packet by T_{ij}^p. Note that since the wireless network is stochastic in nature, T_{ij}^p is a random variable. Running the ML model is possible only after every packet is received, i.e. only after T_{ij}^p time has elapsed after video transmission.

Execution of ML Model: Once the video frames are received, the edge device runs the ML model to identify any obstructions. We assume that running the machine learning model takes time t_{ij}^c depending on the parameters set in the camera, and the compute capability of the edge device. Note that this execution time t_{ij}^c is a function of $\mathbf{T}_j$, since executing other models adds an overhead to the overall execution cycle.

B. Our Problem Formulation

Our broad goal is to assign the processing of each camera $C_i \in \mathbf{C}$ to edge device $D_j \in \mathbf{D}$. As mentioned above, the wireless network is inherently stochastic in nature. Thus, we allocate sufficient time so that the video reaches with a high probability, defined by $1-\delta$, where we set $\delta = 0.05$. We say that a single segment is successfully delivered if a total of k_s packets are transmitted with probability $1-\delta$. If the random variable Y_{ij} denotes the number of successful transmissions, the probability of success is denoted by $P(Y_{ij} = k_s)$. Thus, our constraint is given by:

$$P(Y_{ij} = k_s) \geq 1 - \delta \tag{1}$$

Note that Y_{ij} is a random variable denoting the number of attempts, where the total number of transmission attempts is equal to z_{ij}. Thus, the total time allocated for transmission is given by:

$$T_{ij}^p = z_{ij} t_{ij}^b, \tag{2}$$

where t_{ij}^b denotes the actual time to transmit a packet.

Let x_{ij} be a decision variable such that $x_{ij} = 1(0)$ if data of camera C_i is (not) processed on edge device D_j. The total time to process the data also includes the time to send alerts t_j^m, which we assume is a constant. Then, the total time T_i^s to process the data of C_i is given by:

$$T_i^s = x_{ij} \sum_{j=1}^{n} T_{ij}^p + t_{ij}^c + t_j^m. \tag{3}$$

Note that since videos from every camera must execute on some edge device, we have:

$$\sum_{j=0}^{|\mathbf{D}|} x_{ij} = 1, \quad \forall C_i \in \mathbf{C}. \tag{4}$$

Our objective is to maximize the overall throughput, which is the reciprocal of the total latency. Formally, we define this objective as:

$$\textbf{Minimize} \ \sum_{i=1}^{m} T_i^s. \tag{5}$$

Constraints (1)–(4) and the objective function in Expression (5) together formally define our problem as a case of stochastic optimization. In the next section, we discuss our solution to solve this problem.

III. Our Solution to the Optimization Problem

We solve the stochastic optimization problem defined in the previous section in two steps. In the first step, we reduce the stochastic form to a standard deterministic form. In the next step, we make some additional observations, which enable us to propose an approximate algorithm.

A. Reduction to Deterministic Form

We first look at the stochastic constraint. We note that the random variable Y_{ij} only depends on two parameters – the quality of channel ϵ_{ij} and the number of packets that need to be successfully transmitted, k_s. Suppose we set a variable z_{ij} as the number of transmission attempts. Note that each of these transmission attempts are independent of each other, and are successful with the probability $1-\epsilon_{ij}$. This gives us the following observation:

Observation 1. *The random variable z_{ij} follows the negative binomial distribution with parameters k_s and $1-\epsilon_{ij}$, i.e. $Z_{ij} \sim NB(k_s, 1-\epsilon_{ij})$.*

We note that as long as the variable $z_{ij} \geq Z_{ij}$, the stochastic constraint is satisfied. This event can be written as:

$$\sum_{k=0}^{\infty} P(z_{ij} \geq k_s + k) = 1 - F(z_{ij}) = I_{1-\epsilon_{ij}}(k_s + 1, k_s), \quad (6)$$

where the function $I_{1-\epsilon_{ij}}$ represents the Cumulative Distribution Function (CDF) of the binomial distribution, also known as regularized incomplete beta function [18].

B. Algorithm to Solve the Simplified Deterministic Problem

We first note that we now have a non-linear objective with a set of linear partition constraints. This is a form of non-linear bin packing problem [19], which is known to be NP-Hard. Thus, as we show in the following discussion, we design a greedy algorithm to solve this problem. Because of some additional properties of the objective, we also show that this greedy algorithm provides an approximate result.

Our solution depends on a couple of key observations about the objective function. Intuitively, if a scheduler sends the jobs (video streams) to a particular GPU, then increasing the number of jobs should never reduce the total number of frames processed per unit time. When the number of jobs is relatively low (say 1 or 2), it is also possible to exploit more parallelism leading to more number of frames processed per unit time. If the number of jobs is higher, then the scheduler uses time-sharing of jobs to ensure that the number of frames processed per unit time does not fall. This gives us the following observation:

Observation 2. *The function $1/T_i^s$ is monotone in nature, i.e. adding any element C_j to $\mathbf{T}_i$ can never reduce the value of $1/T_i^s$. Formally,*

$$\frac{1}{T_i^s(\mathbf{T}_i \cup C_j)} \geq \frac{1}{(\mathbf{T}_i)} \quad (7)$$

Our next observation is that although it is possible to increase the number of frames processed per unit time, the amount of improvement achieved gradually diminishes with an increase in the number of jobs. This is also intuitive, as initially the processor lies under-utilized, and so there is more scope of jobs running in parallel. As the scope of running parallel jobs reduces, the amount of improvement also reduces. At some point, there is no more improvement possible, and at this point, the GPU scheduler stops sending jobs for parallel processing, but sends them in sequential batches. This property of diminishing returns is called submodularity. We therefore observe the following:

Observation 3. *The function $1/T_i^s$ is submodular in nature, i.e. for $\mathbf{T}_i \subseteq \mathbf{T}_k$, and $C_j \notin \mathbf{T}_k$, we have:*

$$\frac{1}{(\mathbf{T}_i^s \cup C_j)} - \frac{1}{\mathbf{T}_i^s} \geq \frac{1}{(\mathbf{T}_k^s \cup C_j)} - \frac{1}{\mathbf{T}_k^s} \quad (8)$$

It is well-known that functions which are monotone and submodular have constant-factor approximations for the problem of maximization with matroid constraints [13]. The key idea behind the algorithm is as follows. Each camera device is allocated to the edge device on which its execution would run the fastest. This goes on, until each camera's video is scheduled. Note that this already takes into account the packet losses due to the uncertainties in the wireless network. The algorithm, formally defined in Algorithm 1, is therefore a greedy one, where we assign each job to the edge device that has the lowest latency one-by-one. From [13], we get the following theorem:

Algorithm 1 Algorithm to select the edge devices where the videos should be sent for processing.

INPUT: Values $T_i^s, \forall i = 1, \ldots, |\mathbf{C}|$,
OUTPUT: $x_{ij}, \forall i = 1, \ldots, |\mathbf{C}|, j = 1, \ldots, |\mathbf{D}|$

```
x_ij ← 0, i, j
F ← φ
Compute T_ij(C_i), ∀i, j {Compute latency of network and execution of each individual camera to all edge devices}
S ← T_ij, ∀i, j in ascending order
for i = 1 to |C| do
    k ← arg min_j T_ij
    x_ik ← 1
return x
```

Theorem 1. *Algorithm 1 gives an approximation ratio of 0.5, i.e. it gives an objective value not less than 0.5 times the optimal.*

We now analyze the time complexity of Algorithm 1. We first note that for each camera, it computes the time taken to execute the video stream on all possible reachable edge devices. We denote the number of reachable devices of a camera C_j by d_j. The time taken to execute can be computed in constant time. Thus, the total time taken to iterate over all cameras is $|\mathbf{C}| \times \sum_j d_j \leq |\mathbf{C}|^3$, which is equal to $O(|\mathbf{C}|^3)$.

IV. Experimental Evaluation

Evaluation Setup: We perform trace-driven simulation, where we first find the traces by running YOLO-v5 [14] on a Jetson Nano. We use the Python multithreading library available by default to create multiple threads and process the videos in parallel. We use the time command to record the amount of time taken to run the entire program. We perform the experiments on two different datasets. The first dataset, called San Francisco, consists of cell towers in a part of San Francisco city along with the road intersections. We show the location of the roads and cellular towers in Figure 2. The second dataset, called City Flow Grid, uses the tool City Flow [20] to synthetically generate a grid of roads. We then add random cell towers to be able to simulate the presence of traffic.

Baseline Techniques: We compare our approach with two baseline techniques. The first technique, referred to as "Random", sends the videos to any random edge device that is reachable from the camera. The second technique, referred

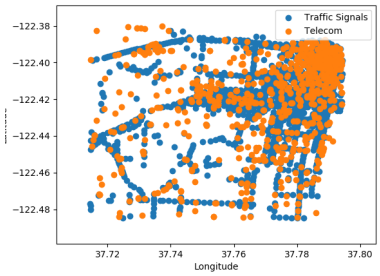

Fig. 2: Configuration of San Francisco grid

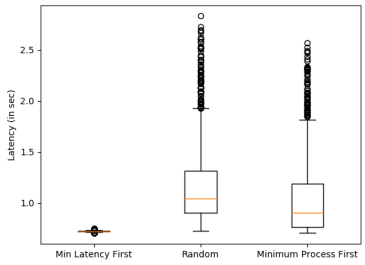

Fig. 3: Result on San Francisco dataset

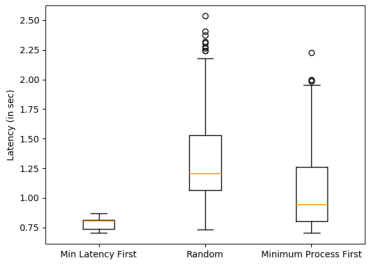

Fig. 4: Result on City Flow grid dataset

to as "Minimum Process", sends the videos to the closest edge device, i.e. the one that has the minimum network latency. We choose these techniques as baselines as they are the most intuitive techniques.

Performance of our Algorithms: We note in Figure 3 and Figure 4 that our technique has significantly lower latency than the baseline approaches. For example, our approach has a median latency of $0.35s$, whereas the "Minimum Process" has a latency of $0.61s$ (an improvement of 42.7%) on the San Francisco dataset. Similarly, on the city flow grid dataset, the latency values seen are equal to $0.78s$ and $0.97s$ (an improvement of 19.6%) for our approach and "Minimum Process" respectively. This shows that utilizing our approach gives a major reduction in latency.

We further note that our approach also leads to a major reduction in both the variance and the number of outliers. This is because, the load balancing seen in our approach leads to the presence of fewer edge devices with very high number of jobs. Thus, apart from reducing the median latency, we also see a major improvement in the reliability of processing. This is very crucial in our application, as it is essential to send alert messages with high reliability.

V. Related Work

A number of recent works have focused on traffic surveillance. We classify these works into two categories. The first category of works focus on filtering the data sent to edge devices, whereas other works focus on scheduling tasks on the edge.

Data Filtering: A number of works reduce the latency by implementing light-weight filtering techniques on the camera or a compute device connected to the camera. For example, Vigil [21] implements a lightweight computer vision technique and removes frames with no activity. Chameleon [22] picks the optimal configuration of videos to send, such as resolution, bitrate and frames per second to avoid sending redundant data. The work [23] presents a method for dynamic bandwidth allocation for edge-based real-time video analytics. DDS [24] shows a technique of fetching only the required parts of the video at high resolution, while allowing other parts at a lower resolution. Reducto [25] utilizes a few lightweight computer vision techniques such as presence of motion within an area and along the edges to filter out unnecessary frames. SmartFilter [26] has the same objective, but with a simpler approach of using a binary classifier. These studies complement our approach of scheduling tasks on edge devices.

Scheduling of Computer Vision Workloads on Edge: The requirement of real-time traffic surveillance and autonomous vehicles has spawned a significant amount of research on running computer vision workloads with low latency. For example, VideoStorm [27] identifies the latency requirement of the queries and schedules them accordingly. DeepRT [28] schedules convolutional neural networks on GPU machines. However, unlike our work, the focus of both VideoStorm and DeepRT are on scheduling for server-quality GPUs. VideoEdge [29] improves upon VideoStorm by incorporating hierarchical edge devices. Llama [30] similarly tunes the video analytics pipeline to optimize cost and reduce both bandwidth and compute resource usages. However, the impact of multitasking on edge GPUs is not considered by either VideoStorm or VideoEdge. The work [31] uses a technique of prioritizing areas of the video for processing that have more stringent real-time constraints. These works complement our approach of multi-tasking the running of computer vision workloads on edge devices.

VI. Conclusion

In this work, we solve the problem of scheduling computer vision based traffic surveillance on edge devices connected to cell towers. Unlike prior works, we assume that the edge devices contain GPUs as they are commercially available. Since such edge GPUs do not have hardware virtualization or support for inbuilt parallelism, multitasking is used to run multiple jobs concurrently. We first observe that the execution time follows a submodular pattern, and utilize this observation to design a greedy approximation algorithm. We then evaluate this algorithm with a few reasonable baselines via both synthetic and real data and show that our algorithm performs much better in practice.

References

[1] K. Jadaan, E. Al-Braizat, S. Al-Rafayah, H. Gammoh, and Y. Abukahlil, "Traffic safety in developed and developing countries: A comparative

analysis," *Journal of Traffic and Logistics Engineering Vol*, vol. 6, no. 1, 2018.

[2] S. Heydari, A. Hickford, R. McIlroy, J. Turner, and A. M. Bachani, "Road safety in low-income countries: state of knowledge and future directions," *Sustainability*, vol. 11, no. 22, p. 6249, 2019.

[3] "Raspberry pi camera module v2-8 megapixel,1080p (rpi-cam-v2)," accessed on July 22, 2022. [Online]. Available: https://www.amazon.com/Raspberry-Pi-Camera-Module-Megapixel/dp/B01ER2SKFS

[4] "Why hyderabad became india's surveillance capital," published on Dec 7, 2021; Accessed on July 22, 2022. [Online]. Available: https://www.thenewsminute.com/article/why-hyderabad-became-india-s-surveillance-capital-158466

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[6] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[7] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[8] "Jetson nano developer kit," https://developer.nvidia.com/embedded/jetson-nano-developer-kit, accesssed on Nov 18, 2022.

[9] "Jetson agx xavier series," https://www.nvidia.com/en-in/autonomous-machines/embedded-systems/jetson-agx-xavier/, accessed on Nov 18, 2022.

[10] A. A. Süzen, B. Duman, and B. Şen, "Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. IEEE, 2020, pp. 1–5.

[11] https://www.nvidia.com/en-in/data-center/graphics-cards-for-virtualization/.

[12] S. P. Panda, A. Banerjee, and A. Bhattacharya, "User allocation in mobile edge computing: A deep reinforcement learning approach," in *2021 IEEE International Conference on Web Services (ICWS)*, 2021, pp. 447–458.

[13] J. Vondrák, "Optimal approximation for the submodular welfare problem in the value oracle model," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 2008, pp. 67–74.

[14] "Yolo-v5." [Online]. Available: https://github.com/ultralytics/yolov5

[15] H. Yao, C. Bai, M. Xiong, D. Zeng, and Z. Fu, "Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 16, p. e3975, 2017.

[16] Z. Li, M. A. Uusitalo, H. Shariatmadari, and B. Singh, "5g urllc: Design challenges and system concepts," in *2018 15th international symposium on wireless communication systems (ISWCS)*. IEEE, 2018, pp. 1–6.

[17] K. K. Leung, W. A. Massey, and W. Whitt, "Traffic models for wireless communication networks," *IEEE Journal on selected areas in Communications*, vol. 12, no. 8, pp. 1353–1364, 1994.

[18] E. Özçağ, I. Ege, and H. Gürçay, "An extension of the incomplete beta function for negative integers," *Journal of mathematical analysis and applications*, vol. 338, no. 2, pp. 984–992, 2008.

[19] U. Hoffmann, "A class of simple stochastic online bin packing algorithms," *Computing*, vol. 29, no. 3, pp. 227–239, 1982.

[20] H. Zhang, S. Feng, C. Liu, Y. Ding, Y. Zhu, Z. Zhou, W. Zhang, Y. Yu, H. Jin, and Z. Li, "CityFlow: A multi-agent reinforcement learning environment for large scale city traffic scenario," in *The World Wide Web Conference*. ACM, may 2019. [Online]. Available: https://doi.org/10.1145\%2F3308558.3314139

[21] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 426–438. [Online]. Available: https://doi.org/10.1145/2789168.2790123

[22] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 253–266. [Online]. Available: https://doi.org/10.1145/3230543.3230574

[23] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 257–266.

[24] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 557–570. [Online]. Available: https://doi.org/10.1145/3387514.3405887

[25] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 359–376. [Online]. Available: https://doi.org/10.1145/3387514.3405874

[26] J. Tchaye-Kondi, Y. Zhai, J. Shen, D. Lu, and L. Zhu, "Smartfilter: An edge system for real-time application-guided video frames filtering," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[27] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and Delay-Tolerance," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 377–392. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang

[28] Z. Yang, K. Nahrstedt, H. Guo, and Q. Zhou, "Deeprt: A soft real time scheduler for computer vision applications on the edge," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, 2021, pp. 271–284.

[29] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 115–131.

[30] F. Romero, M. Zhao, N. J. Yadwadkar, and C. Kozyrakis, "Llama: A heterogeneous and serverless framework for auto-tuning video analytics pipelines," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1–17. [Online]. Available: https://doi.org/10.1145/3472883.3486972

[31] S. Liu, X. Fu, M. Wigness, P. David, S. Yao, L. Sha, and T. Abdelzaher, "Self-cueing real-time attention scheduling in criticality-aware visual machine perception," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022, pp. 173–186.