Decentralized Modular Architecture for Live Video Analytics at the Edge

Sri Pramodh Rachuri
srachuri@cs.stonybrook.edu
Stony Brook University

Francesco Bronzino
francesco.bronzino@univ-smb.fr
Université Savoie Mont Blanc

Shubham Jain
jain@cs.stonybrook.edu
Stony Brook University

ABSTRACT

Live video analytics have become a key technology to support surveillance, security, traffic control, and even consumer multimedia applications in real time. The continuous growth in number of networked video cameras will further increase their widespread adoption. Yet, until now, developments in video analytics have largely focused on using fixed cameras, omitting the ever-growing presence of mobile cameras such as car dash-cams, drones, and smartphones. Edge computing, coupled with centralized clouds, has helped alleviate the network traffic and processing load, reducing latency and data transmissions. However, the current approach of processing video feeds through a hierarchy of clusters across a somewhat predictable path in the network will not be sufficient to support the integration of mobile feeds into the video analytics architecture. In this paper, we argue that a crucial step towards supporting heterogeneous camera sources is the adoption of a flat edge computing architecture. Such architecture should enable the dynamic distribution of processing loads through distributed computing points of presence, rapidly adapting to sudden changes in traffic conditions. In support of this hypothesis, we present exploratory results that show that smartly distributing and processing vision modules in parallel across available edge compute nodes can ultimately lead to better resource utilization and improved performance.

CCS CONCEPTS

• **Computer systems organization** → **Distributed architectures**; **Pipeline computing**; • **Computing methodologies** → *Computer vision problems*; *Object detection.*

KEYWORDS

Edge computing, Computer Vision, Video Analytics, Distributed Computing

1 INTRODUCTION

Extracting information from cameras is increasingly becoming vital across a number of applications. For example, cities have been deploying a large number of closed-circuit cameras that are used for safety, security, and traffic control applications. The police department of New York City uses over 18 thousand cameras spread all over the city [10]. Similarly, some estimates suggest that there are about 691 thousand cameras in the city of London [11]. Live video analytics solutions [1] rely on distributed cloud resources to implement pipelines of operations that process incoming video frames to progressively extract relevant information.

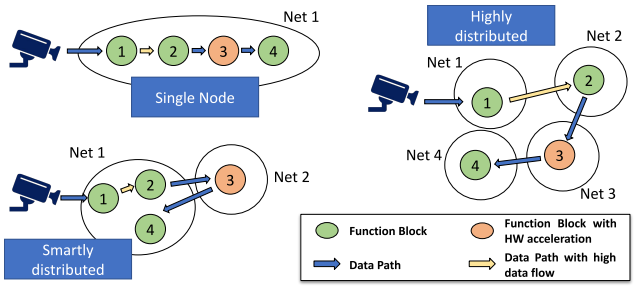

Figure 1: Distributions of processing blocks across different networks

Until now, advances in video analytics have largely focused on fixed cameras, particularly in the domain of traffic surveillance, that observe the same scene during their lifetime (e.g., [4, 6, 15]). Analytics architectures process video streams from these cameras through a hierarchy of clusters and a somewhat predictable path in the network. Often, the edge node in these clusters is the camera, with some compute abilities, connected via a wired link to the next cluster in the hierarchy. They are part of an organized and well-understood deployment that provides a lot of control knobs at each tier within the defined hierarchy. Yet, with the easier-than-ever access to and rising penetration of mobile cameras, standard video analytics architectures should adapt to incorporate their video feeds. These cameras, mounted on devices such as smartphones, dashboard cameras, and drones, often offer the unique advantage of being in the right place at the right time. However, in the absence of real-time support for gathering analytics from them, we are missing an opportunity for truly pervasive intelligent analytics.

Edge computing frameworks are rapidly evolving to include processing accelerators—i.e., GPUs [12] and TPUs [13]—at the extreme edge of the network. Such solutions aim to offer an extremely scalable approach to support low latency computation services through a pervasive deployment approach. Unfortunately, these architectures are inherently underpowered and relying on them to support video analytics requires to carefully decide how to distribute the processing load. This becomes particularly challenging when processing videos generated by mobile cameras as incoming traffic loads can rapidly shift over time.

HotEdgeVideo '21, January 31–February 4, 2022, New Orleans, LA, USA

ACM ISBN 978-1-4503-8700-2/22/01…$15.00
https://doi.org/10.1145/3477083.3480153

In this paper, we argue that, to move to the next step in live video analytics and support mobile video sources, compute architectures should maximize how they distribute their processing pipelines. Taking advantage of ultra localized computing nodes, orchestrators gain the ability to split processing pipelines not only hierarchically but horizontally too. As shown in Figure 1, multiple levels of distribution are possible, depending on the nodes' geographical distribution, leading to wonder how different deployment strategies could affect the application's performance.

To support our hypothesis and answer the aforementioned question, we explore how distributing vision functions across networks impacts the performance of video analytics pipelines. We implement two sample applications in the context of traffic control: volume monitoring and car color detection (for responding to amber alerts). We deploy these applications using three different deployment strategies: 1) A fully centralized solution that uses a single node for the entire processing; 2) A fully distributed solution that instantiates pipeline functions across multiple network regions; Finally, 3) a distributed solution that attempts to smartly select how to partition the pipeline based on each step performance. Our results suggest that smartly distributing the functions is an efficient way to utilize the resources at the edge.

The rest of the paper is structured as follows: Section 2 discusses related work. Section 3 introduces our system design guidelines. Section 4 presents the experimental platform we use to deploy the two sample applications. Section 5 presents the obtained experimental results. Finally, Section 6 concludes the paper.

2 RELATED WORK

In recent years, video analytics have become a key to supporting surveillance, security, traffic control, and multimedia use cases with processing at the edge of the network [1]. Different solutions have been proposed to support dynamic extraction of useful information from live camera feeds.

Microsoft Rocket [2] is a platform for performing live video analytics. It creates a pipeline for the processing of video frames with pluggable models. It also offers a method to offload the computationally heavy ML-based processing to the Azure cloud. Building on this platform, Spatula [4] uses temporal and spatial correlations between video streams from large camera networks to create a cross-camera analytics platform. It uses a pre-learnt model of cross-camera correlations to reduce the time taken for spatio-temporal search among the video streams. Similarly, Chameleon [6] adapts its video pipeline parameters like the NN-model to the current scene of a video stream to find the best balance between accuracy and speed. The authors show that systems can perform adaptation using cross-camera inference to reduce the cost of adaptation. JCAB [15] proposes an online algorithm that uses Lyapunov optimization and Markov approximation to jointly optimize the configuration and bandwidth allocation to the edge nodes. It optimizes based on network conditions, energy utilization, processing latency, and video scene.

Hetero-Edge [16] first proposed an architecture that distributes computing tasks across nodes exploiting the inherent parallelism of tasks belonging to the analytics pipelines. The authors created a platform for computer vision for edge computing using Apache Storm. The platform divides input tasks among slave nodes using the task's Directed Acyclic Graph (DAG). Further, they propose a scheduler that can make parts of frames independently processed by different slave nodes concurrently in favourable conditions. VideoEdge [3] proposes an optimization framework to determine the video resolution of the stream and placement of functions in different clusters. However, VideoEdge proposes a hierarchical distribution only, which does not adapt to the dynamic needs of mobile cameras. Handling mobility becomes relevant when integrating mobile devices in the processing pipeline. Follow Me at Edge [9] uses Lyapunov optimization and Markov approximation to tackle the task migration due to mobility problem in edge computing. It proposed a scheme that can be employed by nodes in the distributed system.

Our work aims to build on the insights presented in previous solutions to further push video analytics architectures towards exploiting all available resources to better address the challenges introduced by the integration of mobile cameras into the video analytics pipeline. We investigate the impact that horizontal distribution can have on resource consumption.

3 SYSTEM DESIGN

The overall objective of our exploration is to investigate a dynamic distribution of processing load. To this end, we identify three design characteristics: (1) Split-process execution, (2) Improved latency and resource utilization, and (3) Adaptability. We discuss these characteristics and how our implementation aims to achieve these.

Split-process execution. We adopt a split-process execution for functions in video analytics pipelines. Each pipeline consists of functions that are processed sequentially. To support applications such as vehicle recognition and counting on incoming video frames, the application pipeline includes decoding, object detection, cropping, and counter increments. This is only one of several ways in which a counting pipeline can be implemented. We make these subtasks independently processable by splitting them into separate blocks. Every block can independently work as a microservice and can have multiple inputs and outputs. For example, in Figure 2, a cropping function can receive video frames and bounding boxes of cars from different blocks and process them to output cropped images of cars. The video frames that the cropping block receives need to be cached the memory till it receives the bounding box position (from, say, an object detection block). Although cropping is not computationally intensive, it can very much affect the overall time taken by the pipeline and its accuracy, if not strategically placed in the network. Our primary motivation in performing split-process execution is to enable on-demand application pipeline construction for mobile cameras. Depending on where the processing is required in the network and the application to be supported, more blocks can be designed and integrated into the system. Example applications and their constituent modules are described in Section 4.

Improved latency and resource utilization. A primary objective in breaking apart vision pipelines and distributing them over a flat edge computing architecture is to enable parallel utilization of resources within the network. This approach is particularly beneficial in mobile scenarios where the camera's movement determines where frames have to be processed—i.e., avoid overloading of certain nodes while leaving others under-utilized. We conduct a

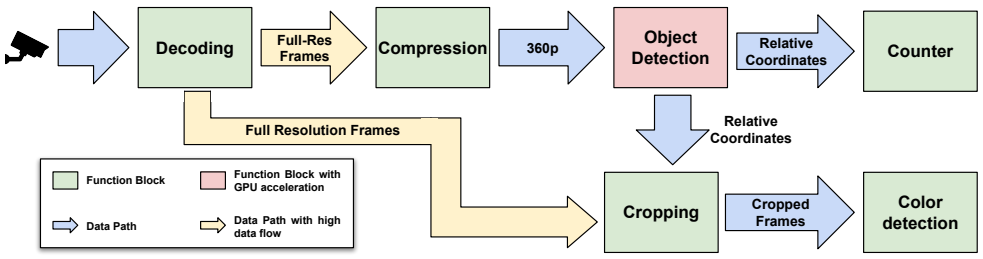

Figure 2: Pipeline in the experimental setup containing modular function blocks and data flow paths

feasibility study to understand how resource consumption and latency are affected in a parallel computational model. While parallel computations have been long studied in the context of operating systems and process scheduling, our focus is on processing vision functions that are generally executed sequentially. There are two key steps in achieving this. First, once the pipelines have been broken up, it enables applications to share common modules. For example, object detection is one of most common computational module in vision pipelines. Deploying the entire pipeline on a single machine in siloed processes can lead to wasted resources due to the inability to share common modules among applications. With a decentralized approach, applications are not siloed and can improve resource utilization by encouraging resource sharing among modules. Second, once the pipelines are broken up, modules can be assembled on-demand for new video analytics applications. Thus preventing the overhead of installing a new application.

Adaptability. Another desirable characteristic of the flat edge computing architecture is adaptability. Adaptability refers to the self-adjusting nature of the system. Adaptability can be addressed at two levels. At the micro level, computational blocks shared among multiple applications must adapt to variations in frame from difference sources. These variations can range from changes in lighting conditions to amount of information in the frame. For example, an increase in number of vehicles in the scene will increase the load on the modules involved in volume monitoring. This increase in the load and type of computation will require the system to scale up (or down to save energy) in the order of seconds. At the macro level, adaptability refers to the ability of the network in supporting multiple computational modules, preferably closer to mobile cameras even as they move around. In this regard, the placement of computational blocks can be scaled to meet the analytics demands from mobile cameras.

4 EXPERIMENTAL SETUP

We implement our test setup using Python-Flask and Gunicorn on top of an NVIDIA Jetson Nano [5]. The Jetson Nano consists of a Quad-core ARM A57 CPU, coupled with a 128-core NVIDIA Maxwell-based GPU and 4 GB of LPDDR4 memory. All our experiments on Jetson Nano are run using Jetpack 4.5.1 based developer kit image.

In order to isolate the modules and bind them to CPU cores, we port our test bench to docker. Each module runs on a separate docker container and compute-heavy modules are bound to specific CPU cores to minimize the effect of context switching on our evaluation. This will also help us to deploy the pipeline over container orchestration applications, such as kubernetes [7], in future. To emulate the communication delay, we use Traffic Control (`tc`) [14] and its Network Emulation (`netem`) [8] module to emulate latency and bandwidth configurations. For the evaluation we consider that the bandwidth and latency in a Local Area Network (LAN) are $1ms$ and $100Mbps$, and in a Wide Area Network (WAN) they are $40ms$ and $50Mbps$. We implement the following modules with the characteristics mentioned in Section 3 for our evaluation. The flow of the frames and messages between the modules is shown in Figure 2.

- Decoding - The decoding module is needed to receive the video stream from cameras in different formats like H264 or HEVC and generate individual frames. These frames are in the highest possible resolution and are sent to the next blocks. In our test setup, the decoder module uses a prerecorded traffic surveillance video to generate image frames.
- Compression - An image compression module is used to reduce the number of bytes being sent to modules located far away in the network. Lossy compression of images tends to affect the accuracy of object detection. So the adaptation of compression rate according to the image context is also an important problem for the future. We use OpenCV2 to perform image compression.
- Object Detection - Object detection is computationally intensive in general and can be accelerated by executing it on GPUs or TPUs. This module processes incoming image frames to generate bounding boxes of vehicles. As the images could have been compressed before being sent to this block, the output bounding boxes containing normalized coordinates. Depending on the image size and network conditions, it can be beneficial to send frames to a far away edge node equipped with a GPU rather than locally processing on CPU.
- Vehicle Counter - The number of vehicles passing through intersections is an important statistic a city administrator would want from a video analytics pipeline. Vehicle counting is performed by using bounding boxes from Object Detection and operator-specified boundaries for each camera.
- Cropping - The image cropper caches the incoming images and crops them once it receives the bounding boxes. It uses the image resolution and the normalized bounding boxes to get the exact pixel locations. The cropped images are significantly smaller in size compared to the full images as they contain only the areas of interest.

Metric	480P	360P	240P
Percent Decrement in #objects	0.97%	1.08%	1.31%
Intersection over Union (IoU)	0.93	0.91	0.87

Table 1: Change in accuracy of bounding boxes with change in input resolution

- Recognition - The cropped images can be processed to get the car's color and used for logging vehicle activity.

Every module needs to use a common language for messages and should be compatible with all possible platforms and architectures. We choose to use HTTP based communications to 1) simplify the building of our test bench, and 2) favour future adaptation to existing frameworks that rely on HTTP microservices. Gunicorn hosts an HTTP server for the application written in Python-Flask. All modules are run as Gunicorn web servers in separate containers. When a module needs to send data like image frames and bounding boxes of objects to another module, it makes an HTTP POST request to the destination module. The modules implemented in our test setup contain only stateless functions. Due to this, scaling up during high load scenarios is easy and only requires replication of modules and load balancing. Scaling up stateful functions like object tracking between the frames is not straight forward and we leave it for future work.

5 EVALUATION

In this section, we aim to understand the impact that distribution has on pipelines' performance, and answer three questions: (1) What is the baseline performance of an analytics pipeline deployed on our experimental setup, (2) What resources does it consume, and (3) How much does distribution of nodes in network affects performance. We then finally see how it compares to running this pipeline on a single node. For understanding resource utilization, we look at CPU utilization, memory in use, and GPU utilization. We also look at the time taken by the block and overall time taken per frame.

We implement the two sample applications shown in Figure 2: volume monitoring and car color detection. We deploy these applications using three different deployment strategies: 1) A fully centralized solution that uses a single Jetson Nano for processing the entire pipeline; 2) A fully distributed solution that places all functions across multiple network regions; and 3) A smartly distributed solution that assumes multiple devices are available for CPU processing over a local LAN, while a GPU equipped device is available traversing the WAN.

Since frames are compressed before being sent to object detection module in our implementation, having a change in the accuracy is inevitable. Table 1 shows how the accuracy of object detection using Yolo V4 changes with change in input resolution. Yolo's output for 720P version of the same video is taken as the reference.

In Figure 3, we see that the utilization of memory by the modules changes in the order of 25 MB. The memory utilization of the Object Detection module is significantly higher than that of the other modules. This could be explained by the fact that machine learning models and libraries are large in size. Load to a few modules like cropping and vehicle counting depends on the number of objects detected by the object detection module. Change in the traffic in the video stream causes small changes in the usage of memory by these modules. We can also notice that running two complete pipelines

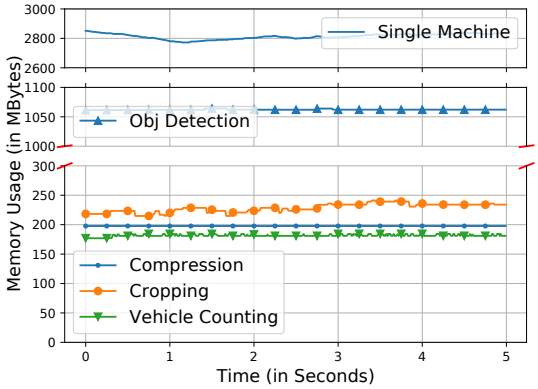

Figure 3: Change in Memory utilization of blocks in distributed pipeline over time compared to that on single machine

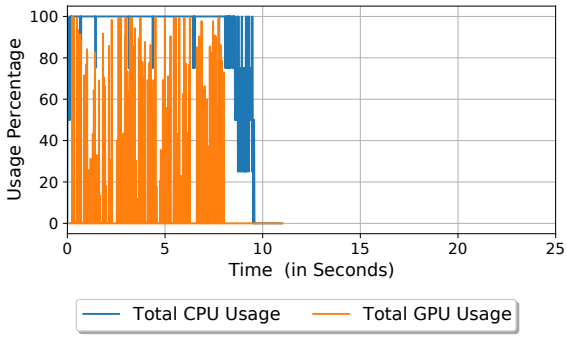

Figure 4: Change in processor utilization in distributed pipeline over time

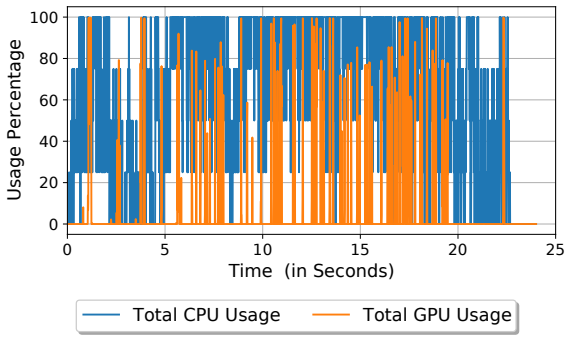

Figure 5: Change in processor utilization in single node pipeline over time

on a single machine occupies significantly high amount of memory. This can be attributed to the running of multiple instances of the same functions. Running separate instances of memory intensive functions like Object Detection for each pipeline leads to large memory consumption.

In Figure 4, we can see that total CPU usage by the distributed pipeline stays at 100% most of the time till the end of the execution. During this time, the system processed a total of 100 frames from 2 different videos. Running the same in a single machine mode took

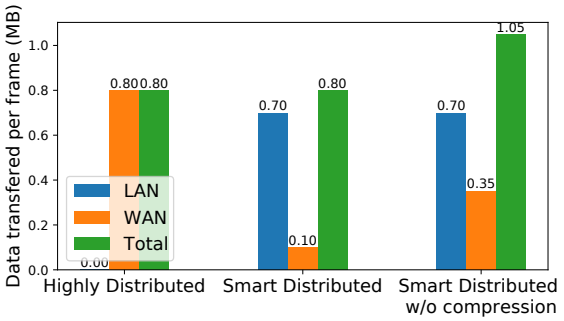

Figure 6: Amount of data transferred over Local Area Network and Wide Area Network in different distributions

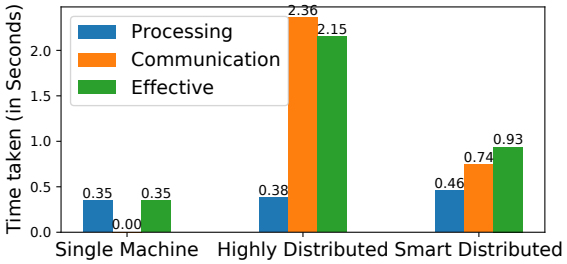

Figure 7: Time taken per frame in different stages

more time to execute. Figure 5 shows the CPU utilization over time for this configuration. We observe that utilization reduces possibly due to frequent context switching. Further, we note that in both Figure 4 and 5 utilization of GPU halts while the system executes rest of the modules.

In Figure 6 we see the number of bytes transferred on an average per frame in different type of networks. When we choose to have only the Object Detection module out of the LAN, we redirect most of the traffic from Wide Area Network (WAN) to Local Area Network (LAN). We can also note that compressing the video frames before sending them over WAN will reduce the load on WAN significantly.

From Figure 7 we note that in distributed pipelines, the communication overhead adds up a lot of delay to the time taken to process each frame. The communication overhead in highly distributed pipeline is very large as it is sending every message over WAN. Although the time taken in each frame in smartly distributed pipelines is approximately 1 second, the frames are concurrently processed. The cores can also process during the network IO wait time. From Figure 4 and 7, we can approximate that the smartly distributed pipeline is capable of processing frames at 10 FPS with a latency of 1 second.

6 CONCLUSION AND DISCUSSION

In this paper, we proposed a modular decentralized architecture for performing video analytics at edge. We argue that operations in video analytics pipeline can, and should, be split into independent functional modules and horizontally distributed throughout the network on an on-demand basis. This approach enables applications to scale up dynamically to support mobile cameras and their movement. We present a feasibility study of the proposed approach across different deployment strategies. We find that, if intelligently used, the distributed approach can improve overall system performance by taking better advantage of all computing resources, while compensating for the additional latency introduced by network transmissions.

In future work, we aim to build on these initial results by automating pipeline deployment, implementing strategies for selecting the location of processing modules across multiple applications. We also intend to build a programming construct which can be used to easily create video analytics functions as modules and build applications on our platform. Further, we will integrate mobile sources to validate how the architecture can dynamically adapt to changing conditions over time.

ACKNOWLEDGMENTS

This material is based in part upon work supported by the National Science Foundation under Award No. 2055520.

REFERENCES

[1] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodik, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-time video analytics: The killer app for edge computing. *computer* 50, 10 (2017), 58–67.

[2] Ganesh Ananthanarayanan, Yuanchao Shu, Landon Cox, and Victor Bahl. 2020. Project Rocket platform—designed for easy, customizable live video analytics—is open source. Microsoft Research Blog. https://www.microsoft.com/en-us/research/publication/project-rocket-platform-designed-for-easy-customizable-live-video-analytics-is-open-source/

[3] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. VideoEdge: Processing Camera Streams using Hierarchical Clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. 115–131. https://doi.org/10.1109/SEC.2018.00016

[4] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Victor Bahl, and Joseph Gonzalez. 2020. Spatula: Efficient cross-camera video analytics on large camera networks. In *ACM/IEEE Symposium on Edge Computing (SEC 2020)*. https://www.microsoft.com/en-us/research/publication/spatula-efficient-cross-camera-video-analytics-on-large-camera-networks/

[5] jetson 2021. Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[6] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: Scalable Adaptation of Video Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 253–266. https://doi.org/10.1145/3230543.3230574

[7] kubernetes 2021. Kubernetes — Production-Grade Container Orchestration. https://kubernetes.io/.

[8] netem 2021. NetEm - Network Emulator. https://man7.org/linux/man-pages/man8/tc-netem.8.html.

[9] Tao Ouyang, Zhi Zhou, and Xu Chen. 2018. Follow Me at the Edge: Mobility-Aware Dynamic Service Placement for Mobile Edge Computing. *IEEE Journal on Selected Areas in Communications* 36, 10 (2018), 2333–2345. https://doi.org/10.1109/JSAC.2018.2869954

[10] Rocco Parascandola. 2018. New NYPD surveillance cameras to cover stretch of Upper East Side not easily reached by patrol cars. https://www.nydailynews.com/new-york/nyc-crime/ny-metro-argus-cameras-east-20181024-story.html

[11] Jonathan Ratcliffe. 2020. How many CCTV Cameras are there in London? (Update for 2020/21). https://www.cctv.co.uk/how-many-cctv-cameras-are-there-in-london/

[12] Sam Sterckval. 2021. Performance comparison : Coral Edge TPU vs Jetson Nano. https://blog.raccoons.be/en/coral-tpu-jetson-nano-performance

[13] Manu Suryavansh. 2020. Google Coral Edge TPU Board Vs NVIDIA Jetson Nano Dev board Hardware Comparison. https://towardsdatascience.com/google-coral-edge-tpu-board-vs-nvidia-jetson-nano-dev-board-hardware-comparison-31660a8bda88

[14] tc 2021. tc - show / manipulate traffic control settings. https://linux.die.net/man/8/tc.

[15] Can Wang, Sheng Zhang, Yu Chen, Zhuzhong Qian, Jie Wu, and Mingjun Xiao. 2020. Joint Configuration Adaptation and Bandwidth Allocation for Edge-based Real-time Video Analytics. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 257–266. https://doi.org/10.1109/INFOCOM41043.2020.9155524

[16] Wuyang Zhang, Sugang Li, Luyang Liu, Zhenhua Jia, Yanyong Zhang, and Dipankar Raychaudhuri. 2019. Hetero-Edge: Orchestration of Real-time Vision Applications on Heterogeneous Edge Clouds. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 1270–1278. https://doi.org/10.1109/INFOCOM.2019.8737478