Roadside Traffic Monitoring using Video Processing on the Edge

Saumya Jaipuria
Indian Statistical Institute
saumyajaipuria@gmail.com

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

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

***Abstract*—Roadside traffic monitoring is increasingly performed by deploying roadside high-resolution video cameras and then running computer vision (CV) models on the video data. Since computer vision models are compute-intensive as they utilize deep neural networks (DNNs), the data is usually sent to one or more edge servers located adjacent to mobile base stations, thereby keeping the in-situ (on camera) processing load as less as possible. Recent techniques propose running CV models on tiles of videos separately to detect and track small objects. Several CV models exist, each with different requirements of compute and memory. Since more compute and memory-intensive CV models provide higher accuracy, a key challenge of such techniques is to determine which vision model should be used on which tile. This becomes even more challenging if multiple videos are processed by the same edge server. In this paper, we first formulate this problem of model selection and tile allocation as an Integer Linear Programming (ILP) instance, and then propose an approximation algorithm based on linear relaxation followed by randomized rounding to solve it. We present experimental results of our methods on an open source dataset based on trace-driven simulation to show that it gives result fast enough while also reducing execution time in a variety of scenarios.**

I. Introduction

Traffic monitoring via deployment of cameras is often used to reduce the number of traffic accidents, identify crimes and accidents on streets and to analyze traffic patterns for better planning. Such traffic monitoring depends on executing object detection algorithms on the video streams captured by the cameras. Executing object detection algorithms is possible on the compute resources available at the camera (though limited) [1] and on additional edge servers connected to cell towers [2].

A major challenge of such traffic monitoring is that accurate object detection is compute-intensive in nature. The most common strategy of object detection is to utilize a deep neural network (DNN) like YOLO [3] that has been pre-trained specifically for traffic surveillance. Furthermore, higher accuracy is possible by splitting the videos into rectangular blocks called tiles and running DNN on each tile separately [4]. However, such separate running of DNNs can lead to an added problem of scalability, as execution of CV models on individual tiles becomes an even more compute-intensive task.

To address this challenge, modern video analytic systems for traffic surveillance apply a very lightweight mechanism to identify the tiles that are likely to have any objects. Once the likelihood of objects being present are obtained, the object recognition algorithms with compute-intensive DNNs are used to obtain additional details such as the object type or size. For high accuracy, it is essential to run object recognition on as many tiles as possible, in the descending order of the likelihoods of the objects being present.

A second challenge comes from the fact that a large variety of object recognition algorithms exists today, with differences in amount of computation necessary and accuracy values. For example, a public version of YOLO [5], has around 5 different models. There are also other off-the-shelf models that could be used, with different computation-accuracy tradeoff. While smart cameras themselves often have the ability to run DNNs, they are usually limited to running only one or two simple models due to memory and compute limitations. Since both the cameras and the edge devices have memory and computation capability constraints, it is essential to judiciously decide (i) which devices (among multiple edge devices and/or on-camera compute) to utilize for execution, and (ii) which models to use, if any, for the individual tiles. These decisions further need to be taken on the network controller at the edge, making it challenging to use compute-intensive mechanisms to solve this joint model selection and tile allocation task. Further, the execution of CV models on the tiles needs to be completed quickly, preferably, with a guaranteed response time, so that the necessary alerts can be sent out.

Videos are usually sent over the network in temporal chunks called segments lasting for 0.5-2s of content. Each such video segment is split into a fixed number of tiles. In this paper, we address the problem of tile processing using CV models, where the objective is to maximize for each segment, the number of tiles processed within a stipulated time, subject to other constraints such as memory and compute available at both the camera and the edge nodes. We first show that this problem is NP-hard, and formulate an integer-linear programming (ILP) task. However, for large scale instances, using an ILP to derive an optimal solution in real-time often becomes infeasible, and this may affect the end to end response time, and delay the needed alerts. To circumvent this, we first solve a relaxed linear programming version of the formulation using a standard LP-solver. Standard LP-solvers are known to be highly optimized, and can solve problems in the order of milliseconds [6]. However, the solution given by an LP-solver is often not feasible, as it often provides fractional solutions to the problem of allocation of tiles to edge servers. We make the solution feasible by using a randomized rounding approach, where the fractional values of the decision variables are rounded to integer values with a probability distribution. We show that

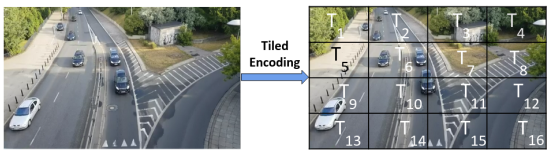


Fig. 1: An illustration of tiled encoding, with tiles numbered for notational convenience. Note that objects in T_2 look significantly smaller than those in T_6 or T_{10}, thus allowing smaller CV models to be run on T_6 / T_{10} than on T_2.

our constraints are satisfied with a high probability, making it possible for us to repeat the process of randomized rounding until we obtain a feasible solution. We further show that this solution is bounded by a constant times the optimal solution.

Our experiment uses a trace-driven simulation on the UA-DETRAC dataset [7]. We obtain traces of execution for 5 different YOLO models and use it to compare against a number of baseline approaches in terms of both the number of tiles scheduled and running time. We also compare it with the ILP, to compare against the best possible performance.

II. Background and Related Work

A. Working of Video Surveillance Systems

We first explain the functioning of a standard traffic surveillance system. We then explain the utility of using tiles.

A video surveillance system consists of a set of roadside cameras connected to edge servers over a cellular network [8], [9]. The edge servers have multiple computer vision (CV) models that can be utilized depending on necessity. Each computer vision model, such as YOLOv5 [5] and Fast-RCNN [10] comes with different levels of execution time, memory consumption and accuracy. Running such compute-intensive CV models is usually done on specialized vector processors, or graphical processing units (GPUs). Most of today's GPUs can run multiple CV models in parallel. In such cases of parallel execution, each CV model can start executing the video chunk received at different times depending on their arrival and CV model requirements. The video streams captured by cameras are first encoded into a standard form called *codec*. The most common codecs used are H.265 and AV1. Usually, these videos are divided into temporal segments of 0.5-2s, and then transmitted over the cellular network. However, a key challenge faced by CV models on running the videos on these chunks of videos is that it is often difficult to identify small objects [4]. A strategy proposed by a number of recent works is to split the video into smaller spatial blocks called tiles, and then run the CV models on each individual tile. Encoding in the form of tiles is also supported by the codecs and have been exploited by multimedia applications for a number of other applications, such as parallel encoding [11] and efficient allocation of bitrates to individual spatial portions [12]. Tiles also make it easier to apply lightweight techniques like frame differencing to determine where objects are more likely to be located. Then, it is possible to apply CV models on tiles in decreasing order of their likelihood of having objects.

Potential of Separate Tile Scheduling: We now motivate the advantages of separate tile scheduling using an example. Figure 1 shows a snapshot of a video segment. Conventional techniques send the entire segment together to the same edge device and apply a single CV algorithm on it. However, as visible in Fig. 1, often the camera is positioned in a way that objects in some specific parts of the frame (in this case, say Tile T_2) are farther from the camera lens than other objects (Tile T_9). Detecting objects from tile T_2, therefore, requires a larger CV model than detecting an object from tile T_{10}. This opens up the possibility of processing at a tile level granularity. Indeed, some of the tiles may need lightweight processing, which can be carried out at the camera itself, instead of sending them to the edge, thereby improving latency.

B. Technique of Randomized Rounding

A major challenge of solving scheduling problems like the one addressed in this paper, is that they are often NP-Hard. While such NP-Hard scheduling problems can usually be formulated as an instance of integer-linear programming (ILP) and solved optimally using optimization solvers, they are often time-consuming in practice. A common approach is to design an approximation algorithm, that provides a bound on its performance with respect to the optimal while running much faster than the ILP. One commonly used method of generating an approximation algorithm, is to relax the problem into a linear programming problem (with fractional and thus non-feasible outputs), and then apply rounding with a probability equal to the fractional value to either 0 or 1 to get an integral solution. By showing that this solution so obtained satisfies the constraints with a high probability, it is possible to show that we have an efficient approximation algorithm.

We utilize Chernoff bounds to show that the solution is feasible with high probability. Chernoff Bound [13][14] provides a bound on the tail of the distribution of the sum of n independent 0-1 random variables, which are also known as Poisson trials. It shows that the deviation of random variable X, where $X = X_1 + X_2 + ... + X_n$ (X_i's are Poisson trials), from its expected behaviour is low. Formally, let $X = \sum_{i=1}^{n} X_i$ where $X_i = 1$ with probability p_i and $X_i = 0$ with probability (1-p_i), and the X_is are independent. Let $\mu = \sum_{i=1}^{n} p_i$. Then,

- Upper tail (bound on the deviation above the mean): Inequality (1) shows that, for a real number $\gamma > 0$, the probability that X goes above its expectation μ, by $\gamma\mu$ or more, is low.

$$P(X \geq (1+\gamma)\mu) \leq e^{-\frac{\gamma^2}{\gamma+2}\mu} \tag{1}$$

- Lower tail (bound on the deviation below the mean): Inequality (2) shows that, for a real number $0 < \gamma < 1$, the probability that X goes below its expectation μ, by $\gamma\mu$ or more, is low.

$$P(X \leq (1-\gamma)\mu) \leq e^{-\frac{\gamma^2}{2}\mu} \tag{2}$$

C. Related Work

Selection of Appropriate CV Model: A number of studies address this problem, some of them being [15], [16], [17], [18]. Focus [15] runs a lightweight convolutional neural network (CNN) at the time of video ingestion, and then utilizes a deeper CNN if the video contains objects of interest. VideoStorm [16] utilizes the least compute-intensive DNN that also satisfies the accuracy constraint. VideoEdge [17] identifies the tradeoff between execution platform and accuracy of inference. JCAB [18] considers network bandwidth availability to identify the right configuration of video quality. These techniques all focus on selecting the right models, without considering tiling.

Use of Tiling for Video Optimization: A number of different video applications utilize tiles to optimize their bandwidth usage. For example, ClusTile [19], Mosaic [12] and Flare [20] all utilize tiles at different bitrate/resolutions depending on the user's viewport to optimize quality of experience of streaming 360 videos. EdgeDuet [4] shows the power of tiling for small object detection. Our work builds on these studies to further study the allocation of these tiles to edge devices.

User Edge Allocation: Multiple studies have formulated the problem of allocation of workloads to edge devices [21], [22], [23]. For example, Heteroedge [21] allocates computer vision tasks on edge servers by modeling and then scheduling directed acyclic graphs. Unlike our work, they do not focus on traffic surveillance, but on tasks given by smartphones of users. The work [22] allocates tasks to edge devices based on their levels of criticality, i.e. tasks that help prevent accidents are assumed to have higher probabilities than the rest. Our work considers these approaches, but identifies the right model as well as the allocation jointly. Another orthogonal approach is to use reinforcement learning for task allocation [23]. Since in our case the state and requirements are known to the controller, we take a deterministic version of the problem.

III. System Model and Problem Formulation

In the following discussion, we explain the system model assumed in our work. We assume that there is a network controller connected to all the base stations through a wired backbone network. The network controller can take intelligent decisions about which models to run on different edge servers, and data could be requested and fetched accordingly. This architecture has been widely used for task allocation in the context of edge computing [24], [25].

We now formally define the system of traffic surveillance. A set of cameras, denoted by $\mathcal{C} = \{C_1, \ldots, C_r\}$ are deployed on the roads for traffic surveillance. Each camera may transmit data to one or more among potentially multiple reachable cellular base stations (denoted by using the cellular network), which is then forwarded by the base station to the edge server for processing on the wired network connecting them. The camera itself has very limited memory and compute capability, that permits to have only one or two lightweight CV models.

Since each base station is directly connected to the edge servers (denoted by $\mathcal{S} = \{S_1, \ldots, S_n\}$), utilizing any edge server requires transmitting the required data to the corresponding base station. This is typically done by sending a control message from the cellular base station to the camera (as seen in modern cellular networks). However, such a request-and-fetch approach incurs a latency equal to the round-trip-time, which is typically less than 20 ms [26]. Note that the latency of communication between the cellular tower and the network controller is less than 1ms [27], as it is directly handled by the wired network connecting them.

The traffic surveillance system works as follows. We consider a system where the controller decides based on past calibration and/or knowledge, as in [28] and [29]. The system is bootstrapped by sending all the tiles of each camera (denoted by $\mathcal{T} = \{T_1, \ldots, T_p\}$) to the edge server closest to it. The edge server runs a number of CV algorithms to analyze the tile statistics, such as the size of objects, background content and speed of movement, as in [29]. These tile statistics are sent to the network controller. The network controller identifies the characteristics of the tile, and also has details of the capabilities of the CV models (denoted by $\mathcal{V} = \{V_1, \ldots, V_l\}$). Accordingly, it decides which CV models are suitable for execution on each tile. It utilizes this decision to further allocate each tile either to an appropriate model on the camera itself, or transfer it to some nearby edge server via the cellular base station. Allocating tiles to edge servers incurs an additional communication latency, equal to the time needed to send the tile to the appropriate base station. This decision is communicated to the cellular base station via the wired network, so that the requests can be further propagated to the appropriate cameras. The allocation for the set of tiles for the next segment is performed accordingly. This process repeats across the sequence of tiles, thus allowing the network controller to dynamically change the task allocation strategy if the video content changes due to changes in light/weather conditions, speed of the vehicles, etc. Once the system is online, the cameras send the tiles to the edge servers via the cellular network. The cellular network controller, connected to the cell towers by the wired network, gathers statistics such as object size and background content about the tiles, identifies the accuracy thresholds for each such tile, and then decides to allocate specific tiles to the edge devices or to the camera itself. The cellular base stations, in turn, request the required video tiles from the cameras according to the allocation decision and pass the tile data to the edge server connected to it. Fig. 2 shows the system.

Let $\mathcal{T} = \{T_1, T_2, \ldots, T_p\}$ be the set of tiles generated for a segment. Each tile T_i has an associated set of CV models by which it can be processed, considering accuracy requirements, denoted by δ_i, where $\delta_i \subseteq \mathcal{V}$. To maximize the number of tiles processed per segment, the network controller generates a strategy that serves the following:

- for each edge server, decide the set of CV models to be instantiated, depending on their total memory capacity, and the requirements of the tiles.
- for each tile, decide whether (i) it can be processed by a model on the camera itself, or (ii) needs to be sent to some

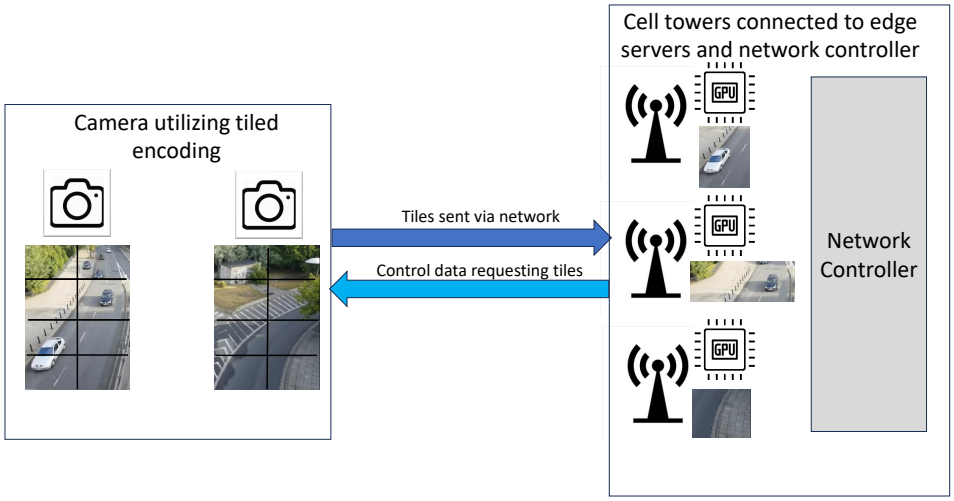


Fig. 2: Our system model showing how tiles are scheduled in the context of edge computing. Each cell tower has an edge server with a GPU. The network controller decides which tiles to schedule on which server, and sends it to the cameras as control data. The tiles are then sent over a cellular network.

edge server where the required CV model is available, or (iii) cannot be processed anywhere.

The task of tile generation, model selection and tile allocation is independent across segments. For each segment, the set of tiles to be processed is available. Depending on the tile requirements, the controller selects a set of models to run on each server and assigns the tiles to appropriate (model, camera / server) pair such that the total completion time for processing the contents of the segment stays within a pre-specified bound. For some segments, not all tiles may be possible to be processed completely within the bound due to available memory limits and the memory needs of the CV models needed by the tiles. Thus, we aim to maximize the set of tiles processed per segment, while keeping the completion time within a bound. This calls for a joint optimization for model choice and tile allocation that we address in this work. On a server $S_i \in \mathcal{S}$, the instantiated models run in parallel. The CV models running on the servers are not known apriori. Depending on the available memory and set of tiles to be processed for a segment, the models are selected for each edge server, while a fixed set of models runs on each camera. The models are assumed to be present in the secondary storage of each device, so that fetching them for execution in memory does not lead to additional latency. Each camera is in the coverage range of at least one server. The tiles generated by $C_l \in \mathcal{C}$ can be assigned to a CV model on C_l itself, or on any server S_k that covers C_l. The latter incurs an additional communication latency.

A. Problem Hardness

In this section, we show the NP-hardness of our joint tile allocation and model selection (JTAMS) problem with a reduction from the known NP-hard problem of Multiple Knapsack with assignment restrictions (*MKARP*) [30] [31]. In *MKARP*, a set of items, say Y, and a set of knapsacks, say X, are given as input. Each item $y_i \in Y$ has a non-negative weight, w_i, and a non-negative profit, p_i. Similarly, each knapsack $x_k \in X$ has a non-negative capacity. For each item $y_i \in Y$, the set of knapsacks that can hold them is defined ($A_i \subseteq X$). The objective is to find an assignment of items to knapsacks with maximum total profit, such that every element is assigned to at most one knapsack, satisfying the assignment restrictions and capacity constraint. We define the set of tiles T and set of models V, where each tile $T_i \in T$ can only be processed by model V_i, $V_i \in V$. We associate the set of models, V, with the set of items, Y; and the set of processing units, $\mathcal{P}$, with the set of knapsacks, X. The weight and profit for each element of Y is the memory capacity of the corresponding model and 1, respectively. For each item $y_i \in Y$, we associate A_i with the set of processing units that can process the corresponding tile T_i, that is $(g_i \cup \mathcal{N}_i)$. From this construction it now follows that a tile allocation and model selection satisfying the constraints, maximizes the number of assigned tiles *iff* the corresponding set of items, Y, can be assigned to the knapsacks satisfying the assignment and capacity constraints, maximizing the total profit. This allows us to conclude JTAMS is NP-Hard. □

B. ILP Formulation

Let $\mathcal{P} = \mathcal{C} \cup \mathcal{S} = \{P_1, P_2, \ldots, P_r, P_{r+1}, \ldots, P_{n+r}\}$ be the set of processing units, where $\mathcal{C}$ and $\mathcal{S}$ is the set of cameras and edge servers respectively. For each $P_i \in \mathcal{P}$, m_i represents its memory capacity and $\mathcal{V}(P_i)$ ($\subseteq \mathcal{V}$) denotes the set of CV models instantiated on it for a specific segment. Let α_{jk} denote the time to process a tile by CV model V_j on P_k. Let c_{ik} denote the communication latency of sending a tile T_i from camera C_l to processing unit P_k. Note that c_{ik} = 0 if the tile is processed at the camera itself (that is, $l = k$), otherwise it is a positive value which depends upon the distance between camera C_l that generated tile T_i, and server S_k. Let g_i denote

the camera that generated tile T_i, where $T_i \in \mathcal{T}$. Similarly, for each tile $T_i \in \mathcal{T}$, let $\mathcal{N}_i$ denote the set of nearby edge servers. Let y_{jk} be a binary decision variable that is set to 1 to indicate that CV model V_j is running on processing unit P_k. Also, let x_{ijk} be a binary decision variable that is set to 1 if tile T_i is assigned to CV model V_j on processing unit P_k.

$$\begin{aligned} x_{ijk} &\in \{0,1\} \quad \forall T_i \in \mathcal{T}, V_j \in \mathcal{V}, P_k \in \mathcal{P} \\ y_{jk} &\in \{0,1\} \quad \forall V_j \in \mathcal{V}, P_k \in \mathcal{P} \end{aligned} \tag{3}$$

We first model the memory constraints. We note that on each server, depending on its memory capacity, different combinations of CV models can be selected. Therefore, for each server S_k, the sum of the memory required by selected models should not exceed the available memory. Formally,

$$\sum_{V_j \in \mathcal{V}} [y_{jk} \times m(V_j)] \leq m_k, \quad \forall S_k \in \mathcal{S} \tag{4}$$

For each camera, a set of lightweight CV models C^w (where $C^w \subseteq \mathcal{V}$), satisfying the memory constraint, is pre-loaded.

$$y_{jl} = 1, \ \forall C_l \in \mathcal{C}, \forall V_j \in C^w, \ \& \ y_{jl} = 0, \ \forall C_l \in \mathcal{C}, \forall V_j \notin C^w \tag{5}$$

We further note that, to process a tile T_i by model V_j on P_k, model V_j should be loaded on processing unit P_k. Formally,

$$x_{ijk} \leq y_{jk} \ , \forall T_i \in \mathcal{T}, \forall V_j \in \mathcal{V}, \forall P_k \in \mathcal{P} \tag{6}$$

Furthermore, every tile should be assigned to at most one CV model on some server / camera, i.e:

$$\sum_{P_k \in \mathcal{P}} \sum_{V_j \in \mathcal{V}} x_{ijk} \leq 1 \quad \forall T_i \in \mathcal{T} \tag{7}$$

A tile T_i can only be assigned to a model that satisfies its accuracy constraint, δ_i. We set the remaining possibilities to 0.

$$x_{ijk} = 0 \ \forall T_i \in \mathcal{T}, \forall P_k \in \mathcal{P}, \ \forall V_j \notin \delta_i \tag{8}$$

Similarly, considering coverage constraints, any edge device that is not in the vicinity of a camera cannot be used, i.e:

$$x_{ijk} = 0, \quad \forall T_i \in \mathcal{T}, \forall V_j \in \mathcal{V}, \ \forall S_k \in (\mathcal{S} - \mathcal{N}_i) \tag{9}$$

Since a tile T_i, generated at camera C_l cannot be assigned to any other camera, we set the corresponding x variables to 0, i.e:

$$x_{ijl} = 0 \ \forall T_i \in \mathcal{T}, \forall V_j \in \mathcal{V}, \forall C_l \in \mathcal{C}, \text{ where } g_i \neq l \tag{10}$$

The time to process the tiles for a segment should not exceed a pre-specified bound $\mathcal{L}$. This has to account for the communication latency (may be 0 for a tile processed in-situ) and the completion time of the models running on an edge server for the specific set of tiles assigned for that segment. In other words, any model on any edge server / camera should finish processing the set of tiles assigned to it, within $\mathcal{L}$. Note that, the models on a processing unit run in parallel, while tiles assigned to a specific model on a server are processed one after another. To express this, we sum over all tiles assigned to a model on a processing unit, and for each such tile, account for the communication latency it may have incurred, along with the processing time necessary to process it on that model on that server, as expressed in the following.

$$\sum_{T_i \in \mathcal{T}} x_{ijk}(\alpha_{jk} + c_{ik}) \leq \mathcal{L}, \quad \forall V_j \in \mathcal{V}, \forall P_k \in \mathcal{P} \tag{11}$$

Our objective is to maximize the number of tiles that can be processed for each segment.

$$\text{Maximize:} \sum_{P_k \in \mathcal{P}} \sum_{V_j \in \mathcal{V}(P_k)} \sum_{T_i \in \mathcal{T}} x_{ijk} \tag{12}$$

We note that the constraints (4)–(12) are all linear in nature. A feasible solution to this ILP gives the set of models to run on each server, the maximum number of tiles that can be processed for this segment and an optimal allocation of the tiles to appropriate models on the server / camera while honoring the processing accuracy, memory and time bound constraints. It is worth noting that all models cannot be trivially instantiated on each edge server, considering the runtime memory needs of the CV models and the total memory capacity available on the edge server for executing the CV workloads. Each camera runs a fixed set of pre-loaded models.

Algorithm 1 Model selection and Tile allocation

```
procedure ALLOCATION(C, S, D, L, m_{|P|}, V, m(V), α, c)
    while t = 1 to ... do
        T ← Set of tiles generated for segment t
        δ_{|T|} ← Set of valid models for each tile
        g ← Camera index for each tile
        (e, p, x, y) ← ILP(T, δ, g)
        t ← t + 1
```

C. The Overall Approach

The above presents an ILP for the model instantiation and tile allocation problem for handling the set of tiles originating for a segment. Algorithm 1 summarizes the overall approach. At the beginning of each segment t, we get a set of tiles $\mathcal{T}$ and their respective accuracy ensuring models, $\delta_{|\mathcal{T}|}$. The function *ILP()* contains the ILP formulation, defined in Section III-B. A call to *ILP()* for a specific segment returns the following:

- Model selection for each server for that segment.
- Tile assignment for that segment
- Time e to produce the allocation for that segment. This is typically the time incurred for solving the ILP.

It may be noted that for any server, the set of models selected to be instantianted may change across segments, depending on the tile requirements and the resulting allocations.

IV. Approximation Algorithm

The ILP formulation given in Section III-B gives an optimal solution for JTAMS for each segment, however, the time it takes to generate a solution increases with an increase in problem size. Thus, we now present an approximation algorithm [32] [33] to use in place of the ILP. The approximation algorithm (Algorithm 2) is obtained by relaxing the integrality constraint (Eq (3)) on the variables and then using randomized rounding on the fractional solution to get an integral feasible

Algorithm 2 Approximation Algorithm (RR)

1: **procedure** APPROX($\mathcal{C}, \mathcal{S}, \mathcal{P}, \mathcal{N}, m_{|\mathcal{P}|}, \mathcal{V}, m(\mathcal{V}), \alpha, c$)
2: $(\widetilde{x}_{ijk}, \widetilde{y}_{jk}) \leftarrow CALLLP(\mathcal{T}, \delta, g)$
3: **repeat**
4: **for** $(j,k) \in (\mathcal{V} \times \mathcal{P})$ **do**
5: $\widehat{y}_{jk} \leftarrow 1$ with probability $\widetilde{y}_{jk}$
6: **for** $i \in \mathcal{T}$ **do**
7: **for** $(j,k) \in (\mathcal{V} \times \mathcal{P})$ **do**
8: **if** $j \in \delta_i, k \in (\mathcal{N}_i \cup g_i), \widehat{y}_{jk} = 1$ **then**
9: $\widehat{x}_{ijk} \leftarrow 1$ with probability $\frac{\widetilde{x}_{ijk}}{\widetilde{y}_{jk}}$
10: **else**
11: $\widehat{x}_{ijk} \leftarrow 0$
12: Among all (j,k)'s where $\widehat{x}_{ijk}$ is set to 1, select one at random and set others to 0
13: **until** $(\widehat{x}, \widehat{y})$ defines a feasible solution
14: **return** $\widehat{x}, \widehat{y}$

solution to the problem. The first step of the approximation algorithm is LP Relaxation, i.e., the variables x_{ijk} and y_{jk} can have any real value in the range $[0, 1]$. The optimal solution obtained from the above relaxed LP might be fractional, and is denoted by $\widetilde{x}_{ijk}$ and $\widetilde{y}_{jk}$. To get an integer solution, these fractional values are rounded to 0 or 1. Let $\widehat{x}_{ijk}$ and $\widehat{y}_{jk}$ denote the integer solution obtained after rounding. For each (j,k) pair, where $j \in \mathcal{V}$ and $k \in (\mathcal{P})$, variable $\widehat{y}_{jk}$ is set to 1 with probability $\widetilde{y}_{jk}$ (in Lines 4-5). Let g_i denote the camera that generated tile $T_i \in \mathcal{T}$. A tile can be assigned to model j on unit k only if j is available on k, that is, the corresponding $\widehat{y}_{jk}$ is set to 1. In lines (6-11), $\widehat{x}_{ijk}$ is set to 1 with probability $\frac{\widetilde{x}_{ijk}}{\widetilde{y}_{jk}}$, otherwise it is set to 0. In lines (7-11), a tile $i \in \mathcal{T}$ can be assigned to multiple (V_j, P_k) pairs by setting the corresponding $\widehat{x}_{ijk}$s to 1. However, from inequality (7), a tile can be assigned to at most one CV model on the camera or some server. To ensure that inequality (7) is satisfied, 1 out of the multiple possible allocations for tile i is selected at random, and the remaining $\widehat{x}_{ijk}$s set back to 0. Note that a solution is considered feasible (Line 13 in Algorithm 2) if the model selection satisfies the memory constraint on each server and the tile processing time does not exceed $\mathcal{L}$. We now derive some bounds related to our solution approach below.

The probability that decision variable $\widehat{y}_{jk}$ is set to 1 is, $P[\widehat{y}_{jk} = 1] = \widetilde{y}_{jk}$. $\widehat{x}_{ijk}$ can be 1 only when $\widehat{y}_{jk} = 1$,

$$\begin{aligned} P[\widehat{x}_{ijk} = 1] &= P[\widehat{x}_{ijk} = 1 | \widehat{y}_{jk} = 1] P[\widehat{y}_{jk} = 1] \\ &= \frac{\widetilde{x}_{ijk}}{\widetilde{y}_{jk}} \times \widetilde{y}_{jk} = \widetilde{x}_{ijk} \end{aligned} \tag{13}$$

Lemma 1. *The solution returned by Algorithm 2 satisfies the memory constraint on each server, in expectation.*

Proof. The expected amount of memory required by the models selected by Algorithm 2 on a server k is written as,

$$\begin{aligned} E[\sum_{j \in \mathcal{V}(k)} (\widehat{y}_{jk} m(V_j))] &= \sum_{j \in \mathcal{V}(k)} P[\widehat{y}_{jk} = 1] m(V_j) \\ &= \sum_{j \in \mathcal{V}(k)} (\widetilde{y}_{jk} m(V_j)) \leq m_k \end{aligned} \tag{14}$$

where the second equation holds because $\widehat{y}_{jk}$ is set to 1 with probability $\widetilde{y}_{jk}$. The last inequality is due to the memory constraint in the relaxed LP. □

Lemma 2. *The maximum time to process the tiles allocated by Algorithm 2 does not exceed $\mathcal{L}$, in expectation.*

Proof. The expected time to process the tiles by model V_j on processing unit P_k is,

$$\begin{aligned} E[\sum_{T_i \in \mathcal{T}} \widehat{x}_{ijk}(\alpha_{jk} + c_{ik})] &= \sum_{T_i \in \mathcal{T}} P[\widehat{x}_{ijk} = 1](\alpha_{jk} + c_{ik}) \\ &= \sum_{T_i \in \mathcal{T}} \widetilde{x}_{ijk}(\alpha_{jk} + c_{ik}) \leq \mathcal{L} \end{aligned} \tag{15}$$

where, the second equation holds because $\widehat{x}_{ijk}$ is 1 with probability $\widetilde{x}_{ijk}$. The last inequality is due to the presence of the bound constraint ($\leq \mathcal{L}$) in the relaxed LP. □

Theorem 1. *JTAMS is solved using Algorithm 2 with an approximation ratio of $(1 - \sqrt{\frac{2r}{\mathbf{Z}^*_{LP}}})$, where $\mathbf{Z}^*_{LP}$ is the optimal value of the relaxed problem in LP form.*

Proof. Algorithm 2 returns the expectation of the objective value as,

$$\begin{aligned} E[\sum_{P_k \in \mathcal{P}} \sum_{V_j \in \mathcal{V}(P_k)} \sum_{T_i \in \mathcal{T}} \widehat{x}_{ijk}] &= \sum_{P_k \in \mathcal{P}} \sum_{V_j \in \mathcal{V}(P_k)} \sum_{T_i \in \mathcal{T}} P[\widehat{x}_{ijk} = 1] \\ &= \sum_{P_k \in \mathcal{P}} \sum_{V_j \in \mathcal{V}(P_k)} \sum_{T_i \in \mathcal{T}} \widetilde{x}_{ijk} \end{aligned} \tag{16}$$

Note that each term $\widehat{x}_{ijk}$ is an independent random variable. Let $\mathbf{Z}^*_{LP}$ represent the optimal value of the relaxed *JTAMS*. Using the Chernoff Bound theorem [13],

$$P[\sum_{P_k \in \mathcal{P}} \sum_{V_j \in \mathcal{V}(P_k)} \sum_{T_i \in \mathcal{T}} \widetilde{x}_{ijk} \leq (1-\gamma)\mathbf{Z}^*_{LP}] \leq e^{-\frac{\gamma^2}{2}\mathbf{Z}^*_{LP}} \tag{17}$$

where $0 < \gamma < 1$. Let $\mathbf{Z}^*_{ILP}$ represent the optimal solution of the *JTAMS* problem. As this is a maximization problem, relaxed LP can assign all the tiles that ILP does, as well as a few more in fractions, hence $\mathbf{Z}^*_{ILP} \leq \mathbf{Z}^*_{LP}$, based on which we can further derive that,

$$\begin{aligned} &P[\sum_{P_k \in \mathcal{P}} \sum_{V_j \in \mathcal{V}(P_k)} \sum_{T_i \in \mathcal{T}} \widetilde{x}_{ijk} \leq (1-\gamma)\mathbf{Z}^*_{ILP}] \\ &\leq P[\sum_{P_k \in \mathcal{P}} \sum_{V_j \in \mathcal{V}(P_k)} \sum_{T_i \in \mathcal{T}} \widetilde{x}_{ijk} \leq (1-\gamma)\mathbf{Z}^*_{LP}] \leq e^{-\frac{\gamma^2}{2}\mathbf{Z}^*_{LP}} \end{aligned} \tag{18}$$

We minimize the upper bound of the probability by taking:

$$e^{-\frac{\gamma^2}{2}\mathbf{Z}^*_{LP}} \leq \frac{1}{e^r} \tag{19}$$

where, r is the number of cameras. This implies that the upper bound quickly converges to 0 as the number of cameras grows. Accordingly, γ should be,

$$e^{-\frac{\gamma^2}{2}\mathbf{Z}^*_{LP}} \leq e^{-r} or, \gamma \geq \sqrt{\frac{2r}{\mathbf{Z}^*_{LP}}} \tag{20}$$

In practise, $\mathbf{Z}^*_{LP}$ tends to be much bigger than r (that is, the number of tiles assigned by the relaxed LP is much larger than the number of cameras), and therefore $0 < \gamma < 1$ always holds when $\gamma \geq \sqrt{\frac{2r}{\mathbf{Z}^*_{LP}}}$. From Eq (17) and (20), we get the approximation ratio as $(1-\gamma) = (1-\sqrt{\frac{2r}{\mathbf{Z}^*_{LP}}})$. □

Theorem 2. *Algorithm 2 ensures that the memory used in any processing unit $P_k \in \mathcal{P}$ never exceeds $\frac{1}{2}(\sqrt{\frac{M}{m_k}}+1)(\sqrt{\frac{M}{m_k}}+2)$ times than its memory capacity with high probability, where M is the memory requirement of the largest model.*

Proof. In Algorithm 2, the models are selected for each processing unit, satisfying the memory constraint in expectation. Each term is an independent variable and therefore by applying the Chernoff Bound theorem, we have

$$P[\sum_{V_j \in \mathcal{V}(P_k)} \widehat{y}_{jk} m(V_j) \geq (1+\gamma) m_k] \leq e^{-\frac{\gamma^2}{2+\gamma} m_k} \tag{21}$$

where $\gamma > 0$. To make the upper bound as small as possible, we take,

$$e^{-\frac{\gamma^2}{2+\gamma} m_k} \leq \frac{1}{e^M} \tag{22}$$

where $M = max(m(V_j), \forall V_j \in \mathcal{V})$ represents the memory required by the largest model. M makes the upper bound close to 0. Accordingly, γ should satisfy,

$$\begin{aligned} & e^{-\frac{\gamma^2}{2+\gamma} m_k} \leq e^{-M} \\ \implies & \gamma \geq \frac{M + \sqrt{M^2 + 8Mm_k}}{2m_k} \end{aligned} \tag{23}$$

In practice, m_k should be much larger than M (i.e., $Mm_k >> M^2$) hence we set,

$$\gamma = \frac{M + 3\sqrt{Mm_k}}{2m_k} \tag{24}$$

Solving this, we get,

$$1 + \gamma = \frac{1}{2}(\sqrt{\frac{M}{m_k}} + 1)(\sqrt{\frac{M}{m_k}} + 2) \tag{25}$$

The memory used in P_k does not exceed $\frac{1}{2}(\sqrt{\frac{M}{m_k}}+1)(\sqrt{\frac{M}{m_k}}+2)$ times its memory capacity, with high probability. □

Thus, we have proved in Thoerem 1 that Algorithm 2 provides an objective value close to the optimal, whereas in Theorem 2 that it also requires few number of iterations to get a feasible solution.

V. Experiments

We now present experimental results to show the performances of the methods in practice. We compare the accuracy obtained and the number of tiles processed within the time bound.

Yolo model (V_j)	$m(V_j)$ (in GB)	$\alpha_{j,k}$ on 4GB Camera C_k (in s)	$\alpha_{j,k}$ on 8GB Server S_k (in s)
n	1.313	0.735	0.090
s	1.367	1.680	0.105
m	1.479	3.765	0.210
l	1.715	6.750	0.360
x	2.075	12.180	0.615

TABLE I: Memory requirement and per tile processing time of 5 YOLO models on different servers

A. Baseline Approaches

We compare Algorithm 2 with the following baselines:

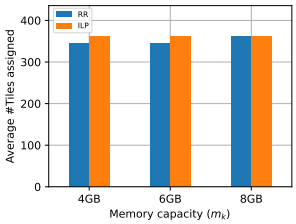
- *B-RMS:* Chooses a random subset of models in the first segment, and keeps it unchanged over the other segments.
- *B-LA:* Assigns each tile T_i to the CV model $M \in \delta_i$ with lowest accuracy value.
- *B-HA:* Assigns each tile T_i to the CV model $M \in \delta_i$ with highest accuracy value.
- *B-CAM:* Assigns each tile to the camera itself.

B. Experimental Setup

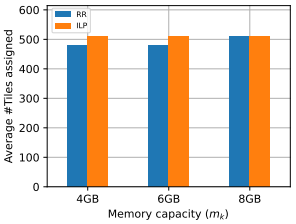
We have 25 edge servers and 125-200 camera locations from the Lumos dataset [34]. For each server, the coverage radius is defined as a random value in the range of 100-200m. All the cameras and servers are assumed to be 4GB and 8GB processing units, respectively. Yolo-v5 models nano (n) and small (s) are assumed to be pre-loaded on each of the cameras. For the videos, we use the DETRAC data-set, which consists of 20 videos [7]. Each camera is randomly assigned a video from this set. Videos are split into segments of 0.5s each, and using ffmpeg these segments are split into 4 tiles. We run the 5 yolo-v5 models (n, s, m, l and x) [5] on each of the tiled videos and record the accuracy value with 0.55 as the accuracy threshold. We also run the yolo-v5 models on a desktop with nVidia RTX 2080Ti GPU, and record the time taken and memory consumed to run each of the CV models on it (TABLE I). We confirmed that the execution time of the CV models on a GPU is independent of the content of the videos. We consider GPU memory capacities of 4GB and 8GB respectively. We conduct all experiments on a machine with AMD Ryzen5 processor with 125 GB RAM. We solve the LP and ILP formulations using the Python PULP library [35]. We assume that each camera segment generates 4 tiles. For each tile T_i, the set of CV model(s) which satisfy the accuracy threshold define the set δ_i. The number of slots is fixed as 167 (number of segments/chunks of $0.5s$ of the longest video). For each slot, we record (a) the number of tiles assigned in a segment by the ILP, approximation algorithm (RR) and the baselines; (b) the Execution Time (ET) incurred by these approaches to produce a solution; and (c) the average accuracy for tile allocation made by each approach.

C. Comparison of Performance

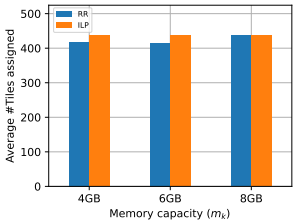
Comparison of Tiles Assigned: Figure 3 depicts the result of varying the memory capacity of the edge servers from 4GB to 8GB, in steps of 2GB, with the memory capacity of the



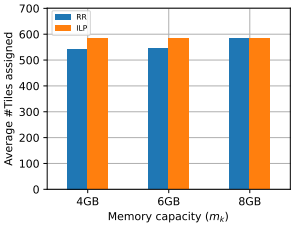
(a) 125 cameras



(c) 175 cameras

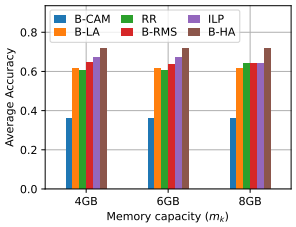


(b) 150 cameras

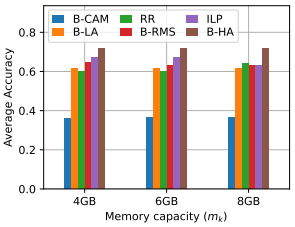


(d) 200 cameras

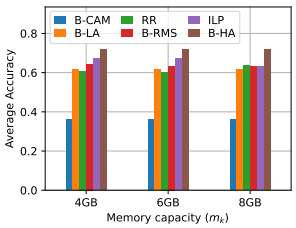
Fig. 3: Average number of tiles assigned in a segment



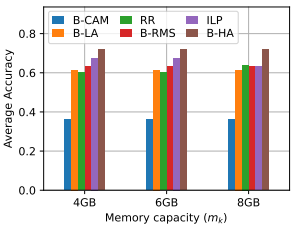
(a) 125 cameras



(c) 175 cameras



(b) 150 cameras



(d) 200 cameras

Fig. 4: Average Accuracy over 167 segments

cameras fixed at 4GB. The number of tiles assigned using Algorithm 2 is close to the ILP solution for each configuration. When the memory capacity increases from 4GB to 6GB, the number of tiles assigned using Algorithm 2 does not necessarily increase, however, when the memory capacity of the edge servers is increased to 8GB, then it is able to assign all the tiles. This is because, when the memory capacity of the edge servers is 8GB, all models can be loaded simultaneously on each server (refer Table I) and hence any model required by a tile is available on some nearby edge server.

Comparison of Average Accuracy: Figure 4 shows the average accuracy over 167 slots for ILP, Algorithm 2 and the baselines. For the first two cases, when all servers have 4GB and 6GB memory capacities, the average accuracy for Algorithm 2 is close to that of ILP. When all servers have 8GB memory capacity, Algorithm 2 has slightly better average accuracy than ILP in some situations, as both of these approaches successfully assign all the tiles however, the models to which a tile is assigned differ in their solutions. Compared to baselines, average accuracy of Algorithm 2 is close to Baseline *B-LA*. Baseline *B-HA* gives a solution with maximum average accuracy, whereas *B-CAM* gives a solution with average accuracy lower than the accuracy threshold (0.55).

Comparison on number of segments: Table II shows the number of segments (out of the total 167) for which different approaches were able to give a solution within a specified response time specified in Column 2 (time to produce a solution + time to process the set of assigned tiles). The time to process the assigned tiles is fixed to $\mathcal{L} = 16s$ for all cases as in prior systems like Chameleon [28], the execution time varies with the number of cameras. Algorithm 2 is able to give a solution, in time, in at least 96% slots. Although the ILP technically gives a feasible solution, its execution time becomes very high, leading to fewer number of instances completing in time. Among the baselines, none of the techniques except B-CAM finish most of the executions on time. Although B-CAM finishes them on time always, this comes at the cost of accuracy, as discussed earlier.

Cam.	Resp. Time (s)	GPU Mem. (GB)	ILP	RR	B-RMS	B-HA	B-LA	B-CAM
125	$28s$	4GB	145	167	79	0	2	167
		6GB	135	167	52	0	1	167
		8GB	132	167	0	1	7	167
150	$33s$	4GB	90	167	52	0	1	167
		6GB	143	167	12	0	0	167
		8GB	128	167	1	0	3	167
175	$41s$	4GB	101	167	116	51	53	167
		6GB	135	167	167	127	158	167
		8GB	144	167	126	65	112	167
200	$50s$	4GB	131	163	1	0	0	167
		6GB	150	161	1	0	0	167
		8GB	162	167	2	0	0	167

TABLE II: Number of segments completed in time

VI. Conclusion

We address the problem of model selection and tile allocation in the context of video processing for roadside surveillance. We formulate an ILP instance for solving the same and further propose a linear relaxation randomized rounding based approximation algorithm. We further conduct trace-driven simulation on an open-source traffic surveillance dataset, and show via extensive experiments that our algorithm gives accuracy close to the ILP. We believe that our proposal of processing at tile level granularity using edge servers for roadside surveillance will have important practical utilizations going forward. As future work, we wish to explore learning based approaches for model selection and tile allocation.

Acknowledgments

This work is (partially) funded by the Cisco University Research Program Fund, under Cisco Grant Number 76417363.

REFERENCES

[1] K. Garg, N. Ramakrishnan, A. Prakash, and T. Srikanthan, "Rapid and robust background modeling technique for low-cost road traffic surveillance systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 5, pp. 2204–2215, 2020. doi: 10.1109/TITS.2019.2917560

[2] Y. Nan, S. Jiang, and M. Li, "Large-scale video analytics with cloud–edge collaborative continuous learning," *ACM Transactions on Sensor Networks*, vol. 20, no. 1, oct 2023. doi: 10.1145/3624478

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. doi: 10.1109/CVPR.2016.91 pp. 779–788.

[4] Z. Yang, X. Wang, J. Wu, Y. Zhao, Q. Ma, X. Miao, L. Zhang, and Z. Zhou, "Edgeduet: Tiling small object detection for edge assisted autonomous mobile vision," *IEEE/ACM Transactions on Networking*, vol. 31, no. 4, pp. 1765–1778, 2023. doi: 10.1109/TNET.2022.3223412

[5] T.-H. Wu, T.-W. Wang, and Y.-Q. Liu, "Real-time vehicle and distance detection based on improved yolo v5 network," in *2021 3rd World Symposium on Artificial Intelligence (WSAI)*, 2021. doi: 10.1109/WSAI51899.2021.9486316 pp. 24–28.

[6] B. Meindl and M. Templ, "Analysis of commercial and free and open source solvers for linear optimization problems," *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*, vol. 20, 2012.

[7] L. Wen, D. Du, Z. Cai, Z. Lei, M. Chang, H. Qi, J. Lim, M. Yang, and S. Lyu, "UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking," *Computer Vision and Image Understanding*, vol. 193, p. 102907, 2020. doi: 10.1016/j.cviu.2020.102907

[8] S. Paul, K. Rao, G. Coviello, M. Sankaradas, O. Po, Y. C. Hu, and S. Chakradhar, "Enhancing video analytics accuracy via real-time automated camera parameter tuning," in *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '22, 2023. doi: 10.1145/3560905.3568527 p. 291–304.

[9] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, N. Karianakis, Y. Shu, K. Hsieh, V. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2022. doi: 10.48550/arXiv.2012.10557 pp. 119–135.

[10] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, dec 2015. doi: 10.1109/ICCV.2015.169 pp. 1440–1448.

[11] F. Ünel, B. O. Özkalayci, and C. Çiğla, "The power of tiling for small object detection," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019. doi: 10.1109/CVPRW.2019.00084 pp. 582–591.

[12] S. Park, A. Bhattacharya, Z. Yang, S. R. Das, and D. Samaras, "Mosaic: Advancing user quality of experience in 360-degree video streaming with machine learning," *IEEE Transactions on Network and Service Management*, vol. 18(1), pp. 1000–1015, 2021. doi: 10.1109/TNSM.2021.3053183

[13] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[14] M. Goemans, "Chernoff bounds, and some applications," February 2015.

[15] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu, "Focus: Querying large video datasets with low latency and low cost," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018. doi: 10.48550/arXiv.1801.03493 pp. 269–286.

[16] 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, Mar. 2017, pp. 377–392.

[17] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *Proceedings of IEEE/ACM Symposium on Edge Computing (SEC)*, 2018. doi: 10.1109/SEC.2018.00016 pp. 115–131.

[18] 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 – IEEE Conference on Computer Communications*, 2020. doi: 10.1109/INFOCOM41043.2020.9155524 pp. 257–266.

[19] C. Zhou, M. Xiao, and Y. Liu, "Clustile: Toward minimizing bandwidth in 360-degree video streaming," in *IEEE INFOCOM – - IEEE Conference on Computer Communications*, 2018. doi: 10.1109/INFOCOM.2018.8486282 pp. 962–970.

[20] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan, "Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices," in *MobiCom*, 2018. doi: 10.1145/3241539.3241565 p. 99–114.

[21] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," in *IEEE INFOCOM – IEEE Conference on Computer Communications*, 2019. doi: 10.1109/INFOCOM.2019.8737478 pp. 1270–1278.

[22] E. Liu, L. Zheng, Q. He, B. Xu, and G. Zhang, "Criticality-awareness edge user allocation for public safety," *IEEE Transactions on Services Computing*, vol. 16, no. 1, pp. 221–234, 2023. doi: 10.1109/TSC.2021.3131348

[23] S. P. Panda, K. Ray, and A. Banerjee, "Dynamic edge user allocation with user specified qos preferences," in *ICSOC*, 2020. doi: 10.1007/978-3-030-65310-1_15 pp. 187–197.

[24] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1529–1541, 2021. doi: 10.1109/TETC.2019.2902661

[25] C. Tang, C. Zhu, N. Zhang, M. Guizani, and J. J. P. C. Rodrigues, "Sdn-assisted mobile edge computing for collaborative computation offloading in industrial internet of things," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 24 253–24 263, 2022. doi: 10.1109/JIOT.2022.3190281

[26] A. Narayanan, X. Zhang, R. Zhu, A. Hassan, S. Jin, X. Zhu, X. Zhang, D. Rybkin, Z. Yang, Z. M. Mao, F. Qian, and Z.-L. Zhang, "A variegated look at 5g in the wild: Performance, power, and qoe implications," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021. doi: 10.1145/3452296.3472923 p. 610–625.

[27] J. Li and J. Chen, "Passive optical network based mobile backhaul enabling ultra-low latency for communications among base stations," *Journal of Optical Communications and Networking*, vol. 9, no. 10, pp. 855–863, 2017. doi: 10.1364/JOCN.9.000855

[28] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proceedings of the 2018 ACM SIGCOMM Conference*, 2018. doi: 10.1145/3230543.3230574 p. 253–266.

[29] 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 ACM SIGCOMM 2020 Conference*, 2020. doi: 10.1145/3387514.3405887 p. 557–570.

[30] G. Dahl and N. Foldnes, "Lp based heuristics for the multiple knapsack problem with assignment restrictions," *Annals of Operations Research*, vol. 146, no. 1, pp. 91–104, Sep 2006. doi: 10.1007/s10479-006-0048-1

[31] M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, and R. Ravi, "Approximation algorithms for the multiple knapsack problem with assignment restrictions," *Journal of Combinatorial Optimization*, vol. 4, no. 2, pp. 171–186, Jun 2000. doi: 10.1023/A:1009894503716

[32] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Service placement and request routing in mec networks with storage, computation, and communication constraints," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1047–1060, 2020. doi: 10.1109/TNET.2020.2980175

[33] L. Gu, Z. Chen, H. Xu, D. Zeng, B. Li, and H. Jin, "Layer-aware collaborative microservice deployment toward maximal edge throughput," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022. doi: 10.1109/INFOCOM48880.2022.9796670 pp. 71–79.

[34] A. Narayanan, E. Ramadan, R. Mehta, X. Hu, Q. Liu, R. A. K. Fezeu, U. K. Dayalan, S. Verma, P. Ji, T. Li, F. Qian, and Z.-L. Zhang, "Lumos5g: Mapping and predicting commercial mmwave 5g throughput," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20, 2020. doi: 10.1145/3419394.3423629 p. 176–193.

[35] "Optimization with pulp," accessed on Feb 10, 2023. [Online]. Available: https://coin-or.github.io/pulp/