RAVAS: Interference-Aware Model Selection and Resource Allocation for Live Edge Video Analytics

RAVAS: Interference-Aware Model Selection and Resource Allocation for Live Edge Video Analytics

Ali Rahmanian
Umeå University
Umeå, Sweden
ali.rahmanian@umu.se

Ahmed Ali-Eldin
Chalmers University of Technology
Gothenburg, Sweden
ahmed.hassan@chalmers.se

Selome Kostentinos Tesfatsion
Ericsson Research
Stockholm, Sweden
selome.kostentinos.tesfatsion@ericsson.com

Björn Skubic
Ericsson Research
Stockholm, Sweden
bjorn.skubic@ericsson.com

Harald Gustafsson
Ericsson Research, Ericsson AB
Lund, Sweden
harald.gustafsson@ericsson.com

Prashant Shenoy
University of Massachusetts
Massachusetts, USA
shenoy@cs.umass.edu

Erik Elmroth
Umeå University
Umeå, Sweden
elmroth@cs.umu.se

Abstract

Numerous edge applications that rely on video analytics demand precise, low-latency processing of multiple video streams from cameras. When these cameras are mobile, such as when mounted on a car or a robot, the processing load on the shared edge GPU can vary considerably. Provisioning the edge with GPUs for the worst-case load can be expensive and, for many applications, not feasible.

In this paper, we introduce RAVAS, a Real-time Adaptive stream Video Analytics System that enables efficient edge GPU sharing for processing streams from various mobile cameras. RAVAS uses Q-Learning to choose between a set of Deep Neural Network (DNN) models with varying accuracy and processing requirements based on the current GPU utilization and workload. RAVAS employs an innovative resource allocation strategy to mitigate interference during concurrent GPU execution. Compared to state-of-the-art approaches, our results show that RAVAS incurs 57% less compute overhead, achieves 41% improvement in latency, and 43% savings in total GPU usage for a single video stream. Processing multiple concurrent video streams results in up to 99% and 40% reductions in latency and overall GPU usage, respectively, while meeting the accuracy constraints.

Keywords: Edge Video Analytics, Model Selection, Resource Allocation, Interference-aware GPU Multiplexing

ACM Reference Format:
Ali Rahmanian, Ahmed Ali-Eldin, Selome Kostentinos Tesfatsion, Björn Skubic, Harald Gustafsson, Prashant Shenoy, and Erik Elmroth. 2023. RAVAS: Interference-Aware Model Selection and Resource Allocation for Live Edge Video Analytics. In *The Eighth ACM/IEEE Symposium on Edge Computing (SEC '23), December 6–9, 2023, Wilmington, DE, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3583740.3628443

1 Introduction

Millions of video cameras are deployed worldwide for diverse applications, ranging from urban mobility to factory automation. Real-time video analytics is central to these applications. DNNs models, such as Yolo [1], ResNet [2], and EfficientNet [3], are widely used for object detection and classification, with an increasing focus on edge deployment [4–6]. Edge video analytics allows the analysis to be done with less—or no—data transmission to remote clouds while also increasing the privacy of computations.

Running video analytics at the edge presents challenges regarding GPU deployment constraints compared to large-scale cloud data centers. Edge GPUs span a range of computational power and capabilities, from devices like NVIDIA Jetson Nanos [7] to high-performance GPUs like the A100s [8]. A single-edge server can process from one video stream to tens of streams, depending on the GPU capacity and the running application. Therefore, many edge GPUs need to multiplex the processing of video feeds from multiple cameras to ensure efficiency.

Multiplexing carries additional challenges for many edge applications where the number of video streams can vary due to mobility. Two examples of such applications are intelligent traffic systems, where vehicles offload video processing to the edge, and edge-robotics and drone-based applications, where mobile robots/drones offload video processing from on-device cameras to the edge for processing. Generally, for many applications, video workload can vary because of mobility, and hence, the number of streams that need processing on an edge GPU can change with time as the number of cameras close to the edge GPUs change, e.g., with moving edge robots or autonomous vehicles with cameras [9]. Another source of variability in the workloads comes from applications where on-camera or cross-video filtration techniques are used [10, 11], varying the number of frames sent to the edge-GPU for processing. Hence, the total number of frames that require processing on the GPU at any given time can vary drastically.

Motivation. Existing approaches in video analytics often prioritize either application-level aspects to meet Quality-of-Service (QoS) requirements or system-level techniques to optimize resource utilization [6, 12–14]. However, these approaches overlook the complex interplay between application requirements, system constraints, and the specific challenges posed by concurrent video feeds on the same GPU server.

Solely focusing on application-level needs can result in excessive resource allocation, leading to resource inefficiency [15]. Moreover, concurrent processing of multiple video feeds on the same GPU server can introduce inference interference, causing performance degradation [16]. Conversely, an exclusive emphasis on system-level resource management may not adequately address the unique demands of concurrent video feeds. The dynamic nature of video workloads, coupled with the varying capabilities of GPUs, necessitates a more comprehensive approach.

To overcome these challenges, an integrated approach that considers both application-level requirements and system-level constraints is required, considering the impact of concurrent video feeds on the same GPU server. Combining adaptive model selection for DNN-based video processing, efficient resource allocation, and strategies to mitigate inference interference, we can optimize GPU utilization, maintain accuracy, and minimize latency for concurrent video analytics on GPU-enabled edge servers.

The solution aims to select a model for each stream video feed to meet certain accuracy where the accuracy of the inference is what we measure and care about. On the other side, there are multiple video feeds that need to be fit and processed concurrently on the same GPU resource together. In this case, the resource allocator comes into play. If all models selected for feeds can be served and fit on the resource, the resource allocator mechanism is straightforward and allocates what they asked. Otherwise, the resource allocator may be unable to fit models selected for the resource sometimes. In this case, the resource allocator employs a dependent mechanism to revise the decision made for individual stream video feeds based on the constrained resource capacity.

The model selection focuses on the accuracy of models to choose the lightest model that provides certain accuracy. At the same time, resource allocation cares more about how it fits multiple concurrent workloads on a single GPU resource.

Model selection is input-dependent, and the decision is made based on the situation of a camera. By keeping model selection separate from resource allocation, Ravas enables the system to adapt more effectively to changing conditions. The resource allocation execution does not have to consider all possible and dependent model selections for all cameras together to make a dependent decision. In contrast, most of the time, the models independently selected for each camera can be fit on the GPU, and there is no need to make extensive decision-making. At the same time, it is a bottleneck to make the system scalable and make the decision-making process heavier.

A dependent model selection fails to work in such a situation where the action would be a list of models selected for all cameras, and considering the feedback of such model must demonstrate the state of all cameras. Such problem modeling with Reinforcement Learning (RL) would not be feasible with many states and actions.

Our contribution. In this paper, we introduce RAVAS, a real-time adaptive stream video analytics system that enables a higher level of multiplexing on edge GPUs with variable video workloads on the edge. This work aims to enable efficient optimization of object detection in an edge environment by also considering the overhead costs of re-configuration. RAVAS deploys a number of DNN inference models on edge GPUs, selecting the best model with the lowest resource requirement and inference time that fulfills the accuracy requirement of an application while adapting to variations in video workloads. Our main contributions to this work are:

- We propose an optimization algorithm based on RL to automatically determine a configuration that optimizes GPU resource usage and enhances the performance of an edge-based video analytics system.
- We present a resource allocation strategy to mitigate inference interference when concurrently executing video feeds on the same GPU.
- We evaluate our system with concurrent videos on multiple different GPUs with varying capabilities and show the efficacy in the system when other State-of-the-Art techniques fail.

2 Background

Edge Computing. Edge computing aims to provide applications with lower latency compared to the back-end clouds by utilizing a set of small and distributed computing resources near the end users. Video analytics applications are especially suited to run on edge resources as they decrease the bandwidth and latency without transferring large amounts of video data across the Internet to remote cloud data centers. Instead, data can be processed at nearby edge resources. The main downside of offloading computations to the edge is the constrained resources available, unlike large-scale cloud environment [17]. This calls for an efficient edge resource management that addresses the edge capacity limitation and application performance requirements.

Configuration Management for Video Analytics on Edge. Many existing techniques in the literature focus on optimizing the limited edge compute capacity by adjusting computation parameters such as input resolution and DNN model to achieve the lightest configuration that meets their constraints. Using a lighter model reduces the computational demands, whereas models with more network layers or higher input resolutions provide higher accuracy at the cost of increased computational complexity. Therefore, it is crucial to carefully select the appropriate model that minimizes resource usage while still meeting accuracy requirements. It should be noted that the accuracy of each model depends significantly on the content of the inferencing frame. Thus, lighter models can achieve high accuracy for frames with clear and sharp content.

Furthermore, maximizing concurrency on the constrained GPU compute capacity at the edge is essential to enhance GPU utilization. GPU multiplexing allows the sharing of the GPU among multiple applications to achieve concurrency [18]. Spatial multiplexing is a technique that divides the GPU among different models, enabling them to execute simultaneously. However, this approach can introduce application interference and potentially reduce overall throughput [19]. To mitigate interference and prevent GPU oversubscription, controlled spatial sharing allocates a specific percentage of the GPU compute capacity (i.e., GPU%) to each application, ensuring a particular number of GPU streaming multiprocessors (SMs) are allocated [18].

In the literature, extensive research has been conducted on selecting optimal DNN configurations for inference to meet the performance requirements of various applications [6, 20–22]. Additionally, efforts have been made to enhance GPU utilization through techniques like GPU multiplexing [18, 23]. RAVAS selects the most optimized model that satisfies latency and accuracy constraints in

this context. Furthermore, it incorporates an interference-aware resource allocator to utilize the GPU compute capacity effectively.
Why RAVAS? In our survey of the field, we found two main shortcomings. First, the problem with variable frame-load on edge GPUs has been mostly overlooked. While some existing solutions can handle this problem [6], as we show later in our evaluations, many of these solutions have a substantial overhead. Existing work does not consider the overhead of their solution and its effect on other co-located processes in the edge servers. RAVAS uses a lightweight RL-based model selection algorithm that selects the best object detection model out of a set of limited object detection models to meet the accuracy constraint with minimum compute requirements, allowing for much higher levels of concurrency on an Edge GPU. In building the actual system, we designed RAVAS to be modular, scalable, and compatible with multiple network transportation protocols, camera models, and video encoding/decoding formats.

In our evaluations, we compare RAVAS to both video analytics systems, including AWStream [13] and OTIF [12], and model selection baselines, which encompass Chameleon [6], BLEU [24], and LW [25]. AWStream [13], is a stream processing system that optimizes the trade-off between accuracy and bandwidth consumption for wide-area network streaming analytics. It combines offline and online training to create an accurate model that captures the relationship between application accuracy and bandwidth consumption, adapting to network conditions. OTIF [12], is another relevant system that focuses on tuning machine learning pipelines. It introduces an object detection and tracking architecture with five tunable parameters, enabling a balance between speed and accuracy specific to a given video dataset. Our evaluations show how the above two shortcomings severely affect the performance of these three systems. Chameleon [6] focuses on fast object detection and employs a profiling window to identify the best configuration. This configuration consists of a selected inference model, frames-per-second rate, and frame resolution, with the objective of meeting accuracy requirements while minimizing compute demands within a predefined interval. BLEU [24] iteratively builds a model from a set of DNNs based on accuracy or a hybrid approach. It starts with the most optimal DNN, adds those with the best improvement in the chosen metric, and terminates when accuracy improvement falls below a specified threshold, ensuring a balance between accuracy and runtime efficiency. This approach adaptively selects DNN models, making it suitable for real-world applications. LW [25] employs profiling and adaptation to enhance model selection and configuration, minimizing accuracy degradation and improving resource utilization—a pivotal baseline in our research.

3 Problem Statement

We define a model library consisting of a set of pre-selected inference models. The indexes of the models range from 1 (representing the lightest model) to g (representing the heaviest model), with the g^{th} index corresponding to the most accurate model. The computational requirements of the models vary, with g being the model that demands the most resources. The set of video feeds is represented as $v \in V$, where the video feed v undergoes dynamic variations in the number of frames requiring processing over time due to camera-specific frame filtration techniques.

The frame captured from video feed v at time t, denoted as f_t^v, undergoes processing by a single assigned model instance. The model instance analyzes the frame and generates a set of detected objects as output. Each output object consists of an object class, a bounding box representing the location of the object within the frame, and a confidence score indicating the probability of the presence of the object presence within the bounding box.

Since the inputs are live video feeds, measuring real-time accuracy is not feasible. To assess the accuracy of selected models, we adopt the approach employed in prior studies [6, 14, 22, 26] by considering the output of the most accurate model, referred to as the base model (g), as the reference. The accuracy measurement involves calculating the relative difference between the output of the selected model for a given frame f_t^v and the output of the base model for the same frame. The detected objects from the base model serve as the ground-truth objects for this comparison.

We employ the Intersection over Union (IoU) measure to evaluate the similarity between the detected objects of the selected and base models. This measure calculates the overlap of their bounding boxes, comparing the intersection area to the union area. It quantitatively assesses the similarity between objects. To be classified as a true positive, a detected object must satisfy two conditions: (1) a significant overlap with a ground-truth object and (2) a confidence score surpassing a predefined threshold. False positives occur when detected objects fail to meet both conditions, while false negatives arise when ground-truth objects have no corresponding detected objects meeting the specified criteria. Each ground-truth object is associated with a unique detected object.

We employ the widely used F1-score for accuracy assessment, which computes the harmonic mean of precision and recall. Precision measures the ratio of correctly detected objects to the total number of detected objects, while recall measures the ratio of correctly detected objects to the total number of ground-truth objects. The equation provided calculates the accuracy (c_t^v) of the processing frame f_t^v.

$$c_t^v = 2 \times \frac{\text{precision}(f_t^v) \times \text{recall}(f_t^v)}{\text{precision}(f_t^v) + \text{recall}(f_t^v)} \tag{1}$$

To tackle the objective of maximizing overall throughput on the edge server while achieving a predefined accuracy target (c_{thr}) and satisfying the latency requirement of processing each frame within 100 ms, we frame the problem as a constrained optimization problem:

$$\begin{aligned} &\max \ \ \textit{throughput}, \\ s.t. \quad & c_t^v \geq c_{thr}, \quad \forall\, v \in V \ \&\ t \in T \\ \text{latency}(f_t^v) &\leq 100, \quad \forall\, v \in V \ \&\ t \in T \end{aligned} \tag{2}$$

4 RAVAS Design

In this section, we introduce the modules in the RAVAS design and discuss our RL-based inference model selection and interference resource allocation algorithms. Ravas aims to select the most suitable model while meeting latency and accuracy requirements. We propose an independent model selection strategy and an interdependent resource allocation strategy using spatial multiplexing of GPU compute capacity among DNN models. Based on independent selections, models are assigned to camera groups with similar temporal characteristics. This approach prevents interference and optimizes system performance. Independent model selection is characterized

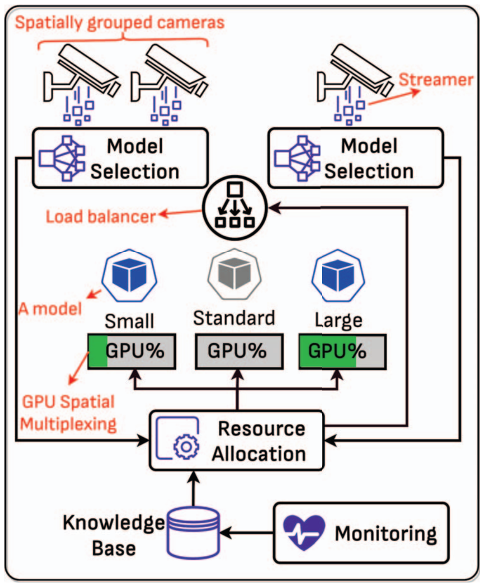


Figure 1. RAVAS framework architecture.

by speed and scalability. It enables rapid adaptation to the unique characteristics of each video feed, ensuring the prompt selection of the most suitable model. This efficiency is paramount in real-time applications where latency and accuracy are critical.

It is essential to clarify the role of RL model selection to suggest a model that achieves a predefined level of accuracy based on historical feed activity. However, it is not employed to solve the resource allocation problem, as integrating both model selection and resource allocation tasks into a single RL problem introduces a high degree of complexity. We emphasize on this in Section 4.3.

Given these considerations, we propose a two-fold strategy: independent model selection tailored to each feed and an interdependent resource allocation mechanism. This integrated resource allocation approach assigns models to individual feeds based on the suggestions made by their RL model selection agents and also allocates resources to the models. This strategy balances efficiency, scalability, and practicality, making it ideal for real-time applications with GPU resource constraints.

4.1 RAVAS System Architecture

In this section, we introduce the overall RAVAS system architecture. Figure 1 depicts all main modules of our framework, namely, the Profiler, the Streamer, the Inference, the Monitoring, and the Manager modules.

The RAVAS profiler captures the maximum throughput by allocating different GPU% to individual DNN models using GPU spatial multiplexing. It quantifies the rate at which each model processes video frames per second while ensuring a predefined latency constraint.

The Streamer module enables connectivity with video feed sources and relays frames to run inference models on the edge server. It also receives and stores the analytics output and application-level metrics.

The Inference module is responsible for executing inference models on a GPU-enabled server. It concurrently processes incoming frames from stream video feeds using multiple inference models. It returns the inference model output to the Streamer module for each frame.

The RAVAS Monitoring module collects infrastructure-level metrics (e.g., GPU utilization, power usage) and application-level metrics (e.g., latency) from Streamer instances. It stores telemetry information and maintains a historical record of decisions and their impact.

The Manager module in RAVAS encompasses model selection and resource allocation components. An independent model selection algorithm is employed to choose the most lightweight model that meets the required accuracy constraint for the frames captured by each camera. The resource allocation component employs dependent decision-making to select the most suitable model for processing frames from each camera, aiming to avoid GPU oversubscription and inference interference. Additionally, it allocates appropriate computing capacity for individual inference models using GPU spatial multiplexing.

4.2 RAVAS Profiler

The RAVAS profiler extracts throughput measurements for varying GPU percentages using spatial multiplexing. These measurements are crucial for the resource allocator, facilitating efficient resource allocation in the system. The profiler enables informed decision-making in resource allocation by assessing the overall throughput achieved by different models under diverse GPU percentage configurations. This data-driven approach optimizes GPU resource utilization while meeting performance objectives. Figure 2 visually presents the output of the RAVAS profiler, illustrating the achieved throughput for different GPU percentages. This information serves as an essential input for effective resource allocation strategies.

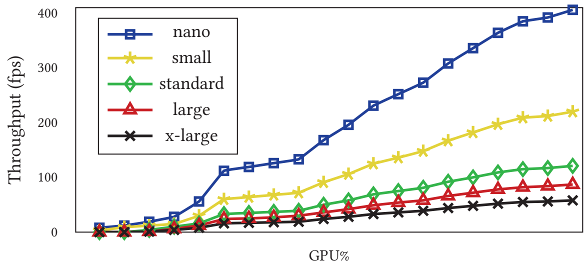


Figure 2. Profiler output depicting overall throughput meeting latency requirements for individual models at different GPU% using GPU spatial multiplexing.

4.3 Model Selection using RL

In this section, we propose an RL-based technique to suggest a model for individual video feeds while meeting a predefined accuracy constraint. RL systems include agents interacting with an environment to make decisions and take actions to achieve specific goals or maximize cumulative rewards. RL agents fundamentally work based on a strategy in pursuit of a goal called a policy. A policy represents a set of actions an agent employs to maximize its reward. Generally, there are two different RL methods: off-policy and on-policy methods. The agent directly learns and updates by following its current policy and action to make updates in the on-policy method. In contrast, the off-policy method diverges by allowing the agent to learn from experiences gathered using a different policy from its current policy. In this paper, we propose a Q-learning technique as an off-policy RL method, illustrated in Figure 3.

Ravas advocates for adopting the off-policy Q-learning RL method in this context for model selection, offering several compelling arguments. Firstly, the urgency of making immediate decisions in model selection aligns seamlessly with the capability of off-policy RL strategies to immediately select actions based on estimated values without rigidly adhering to the current policy. This approach permits the immediate selection of a model by utilizing the action with the highest estimated Q-value, regardless of the current policy. Secondly, off-policy methods are known for their adaptability and work well in situations that require quick adjustments to changing environments or varying model accuracy. In contrast, on-policy RL strategies offer stability and may exhibit slower adaptation due to their policy-centric exploration. Furthermore, the exploration strategies of on-policy RL, often involving random actions, can result in the exploration of sub-optimal models even when superior alternatives are evident. It potentially leads to inefficiencies and resource misallocations. These considerations collectively underscore the suitability of off-policy RL for the model selection without excessive concern for future actions and rewards.

Q-learning agent interacts with the environment by continuously learning action and state spaces. Q-learning is off-policy learning, meaning its behavior policy differs from its target policy.

The behavior policy defines how the agent must interact with the environment. The target policy learns to take the best possible actions according to the different states of the environment. We consider an independent Q-learning agent for individual video feeds to select the best model based on the content. The agent for video feed v takes an action a_t^v that refers to the index of the selected model for processing incoming frames from video feed v at time t. Table 1 showcases the complete set of models utilized in this paper, encompassing `nano` (n), `small` (s), `standard` (d), `large` (l), and `x-large` (xl). Since there is one unique model to select for each action, the number of actions in the action space equals the number of models. Action a_t^v, when applied to the environment, will result in a new state s_{t+1}^v, based on the feedback k_{t+1}^v received from the environment. The feedback is determined based on the accuracy of the selected model (refer to Equation (1)).

For feedback, we define four different categories: `very bad`, `bad`, `ideal`, and `excellent`. Each category describes how good or bad the latest action a_t^v was in the environment concerning the accuracy. Table 2 describes the rules for determining the feedback category k_{t+1}. The feedback category is determined according to the received accuracy (i.e., c_t^v), the accuracy constraint (i.e., c_{thr}), and the type of the selected model. If $a_t^v \neq g$, the feedback category received ranges from `very bad` to `excellent`. If the model g (i.e., the heaviest model) is used, the feedback category is `excellent` if and only if $c_t^v \geq c_{thr}$; otherwise, the category is `ideal`.

The feedback determines the next state and the reward based on the last action.

A state space S is defined as a finite set of all possible states for the environment. The state transits from state s_t^v to state s_{t+1}^v according to the selected model for video feed v at time t and the received feedback from the environment at time $t+1$. For instance, s_{t+1}^v ='`d very bad`' describes that action a_t^v leads to the selection of the `standard` model for video feed v and the feedback k_{t+1}^v was

Model	GPU Mem.(%)	GPU Util.(%)	Accuracy (mAP)	latency (ms)
nano	2.14	1.3	0.344	7.05
small	3.21	2.1	0.387	7.37
standard	6.42	7.13	0.591	9.51
large	7.49	11.21	0.7	10.95
x-large	10.7	15.65	0.736	14.23

Table 1. The average output of results of processing frames by different models on a Tesla V100 GPU accelerator: `nano` (yolov4-tiny-288), small (yolov4-tiny-416), standard (yolov4-288), large (yolov4-416), x-large (yolov4-608).

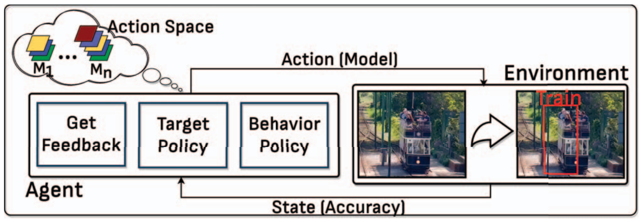


Figure 3. RL-based model selection.

k_{t+1}^v	$a_t^v \neq g$	$a_t^v == g$
`excellent`	$c_t^v > c_{thr} + .1$	$c_t^v \geq c_{thr} + .1$
`ideal`	$c_{thr} - .1 \leq c_t^v \leq c_{thr} + .1$	$c_t^v < c_{thr} + .1$
`bad`	$c_{thr}/2 \leq c_t^v < c_{thr} - .1$	*nan*
`very bad`	$c_t^v < c_{thr}/2$	*nan*

Table 2. Determining feedback according to the relative accuracy of the selected model at time slot t.

`very bad`. Since selecting action a_t^v would lead to receiving one of the four feedback categories, we have $4 \times g$ possible states, where g is the number of actions/models in the system.

Initially, the agent seeks to implicitly find a policy for model selection with the highest possible return by trial and error. The behavior policy is used to explore and generate actions based on the state of the environment. The agent has to make a balance between exploration and exploitation in decision-making. The behavior policy uses a two-dimensional matrix called Q table to store total reward values for all possible state-action pairs. Target policy learns to take the best possible actions by updating the policy (i.e., values in the Q table) through an action-value function to reach the optimal policy.

Algorithm 1 explains the proposed Q-learning based model selection solution. `RAVAS` function is the main function of Algorithm 1 (lines 1-14). It first initializes the Q table (line 2). At the beginning of each time slot, `behaviorPolicy` function selects model a_t^v for video feed v (line 4). The first frame at the beginning of each time slot is captured, denoted by f_t^v from the video feed v (line 5). The `Inference` function receives a frame and a model as input and returns all objects detected within the frame as the output of processing the frame by the specified model. Lines 6-9 process the f_t^v frame by the base model g if the selected model is not the best; otherwise, it processes the frame by the second best model (i.e. $g-1$) as the base model. Line 10 processes the f_t^v frame by the selected model a_t^v. The getFeedback function returns the feedback of using the selected model within the current time slot (line 11), and the getReward function calculates the immediate reward based on the feedback category (line 12). The targetPolicy function is used to update the Q table for the selected <state,action> pair (line 13). Finally, the agent transitions to the new state s_{t+1}^v, which is determined by the action taken in the last time slot and the received feedback category (line 14).

In the Q table, zeros are saved as initial values for all state-action pairs that are meaningful transition policies. Otherwise, $-\inf$ is considered as the default value for the state-action pairs that are not meaningful transition policies, and the RL agent should never use them. Figure 5 shows part of the initial values of the defined Q table. As an illustration, let us assume that the agent takes the action a_t^v for video feed v. Later, if the agent receives very bad as feedback, it should not take any action that leads to selecting the same or lighter model for the next time slot. If the agent receives bad as feedback, it should not take any action that leads to the selection of lighter models, but the same model may still be selected. If the agent receives ideal as feedback, the agent must continue to use the last model for the next time slot and avoid using heavier models to minimize latency and GPU usage. Otherwise, if the agent receives excellent as feedback, it should not take any action that leads to the selection of heavier models, but the same model still has a chance to be selected.

The selected model a_t^v processes all frames within time slot t. The last frame must also be processed by an additional model considered as the base model to measure the accuracy of the selected model. For the last frame within each time slot, t, the getFeedback function (lines 15-19) is used to calculate the relative accuracy of the selected model in comparison to the best model g, which its result is considered as the ground-truth. The accuracy performance of the selected model is measured using the F-Score function, which evaluates the relative accuracy of a list of detected objects by a model in comparison to the ground truth (refer to Equation (1)). If $a_t^v \neq g$, the relative accuracy describes how good or bad this model is compared to the ground truth (i.e., the output of the model g for the f_t^v frame) (line 19). Otherwise, the selected model is model g, and the relative accuracy describes how good or bad model g is compared to the second heaviest model $g-1$ (line 17). Line 20 calculates and returns the feedback category k_{t+1}^v based on the relative accuracy of the last frame, the accuracy threshold, and the type of used model (refer to the rules in Table 2).

The behaviorPolicy function selects and returns model a_t^v to process all the frames of video fed v within time slot t. The agent uses ϵ-Greedy strategy to make exploration-versus-exploitation decisions with a uniform probability. The function works based on ϵ constraint that describes the probability of exploration (ϵ) versus the probability of exploitation $(1-\epsilon)$ where $0 \leq \epsilon \leq 1$. For the exploitation, the agent takes the action with maximum value in the Q table given the current state according to the behavior policy (line 26). The agent defines two strategies for the exploration, each for the training and testing phases. During the training phase, it randomly takes an action from the action space (line 24). However, a complete random exploration after the training phase decreases accuracy because it does not consider any temporal features when taking an action. To handle this, we propose a rank-based probabilistic exploration strategy (line 29). Given the current state, it assigns a normalized weight in the 0–1 for each action.

feedback (k_{t+1}^v)	reward (r)
very bad	-0.1
bad	-0.05
ideal	$+0.1$
excellent	0.0

Table 3. Immediate rewards according to the received feedback category from the environment.

The targetPolicy function updates the values for state-action pairs in the Q table and is given as follows:

$$Q_v[s_t^v, a_t^v] = Q_v[s_t^v, a_t^v] + l \times \left(r + \gamma \max_{a_{t+1}^v} Q_v[s_{t+1}^v, a_{t+1}^v] - Q_v[s_t^v, a_t^v]\right) \quad (3)$$

where l,r, and γ represent the learning rate, immediate reward after completion of the current action, and the discount factor to make a trade-off between immediate and future rewards, respectively. Equation (3) sums the old value with temporal difference (TD) error where TD is the result of subtracting the new Q value (i.e. $\max_{a_{t+1}^v}(Q_v[s_{t+1}^v, a_{t+1}^v])$) from the old Q_v value (i.e. $Q_v[s_t^v, a_t^v]$) for a given state-action pair. The new value is calculated based on the immediate and the future rewards. $\max_{a_{t+1}^v}(\ Q_v[s_{t+1}^v, a_{t+1}^v)]$ indicates the maximum estimation of the future reward after transition to state s_{t+1}^v by choosing action a_{t+1}^v. The immediate reward is determined based on feedback from the environment and calculated according to Table 3.

4.4 Interference-Aware Resource Allocation

The resource allocation algorithm optimizes system performance by efficiently assigning models to video feeds and allocating computing resources. However, limited GPU computing capacity on the edge server may prevent assigning all selected models to every video feed. To address this, the resource allocator employs a dependent decision-making approach based on the models suggested by individual model selection agents for all video feeds, GPU compute capacity, and a heuristic prioritization strategy for video feeds. It also utilizes GPU spatial multiplexing to ensure the efficient utilization of just enough compute capacity on the GPU for individual models, thereby avoiding GPU oversubscription.

Algorithm 2 outlines the resource allocator algorithm designed to optimize resource allocation in the system. It operates on the models a_t^v and their previous versions a_{t-1}^v for all video feeds in the set V.

Initially, the resource allocation algorithm considers utilizing models suggested by the RL model selection agents of individual video feeds. If the total GPU% allocated to all models exceeds the GPU compute capacity, the system downgrades models for some video feeds recommended initially to use heavier models. The resource allocator decides to assign lighter-weight models compared to the initial recommended models to video feeds with higher experienced accuracy. This process repeats in a loop until the GPU compute capacity is no longer oversubscribed (lines 2 to 4). Following the frame rate update, the algorithm updates the spatial GPU allocation for all models (line 5).

The algorithm sorts the models in ascending order of accuracy to prioritize models with lower accuracy (line 6). The algorithm then iterates over the sorted video feeds and checks if the current model a_t^v can be upgraded to a heavier model. At the same time,

Algorithm 1 Q-learning based model selection

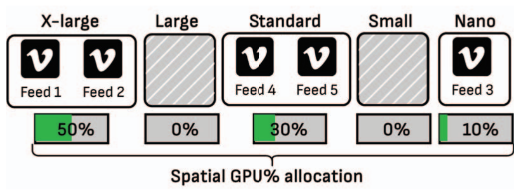
```
1:  function RAVAS($v$,Q-table,training)
2:    Initialize state $s_t$.
3:    for each time slot $t \in T$ do
4:      $a_t^v \leftarrow$ behaviorPolicy($s_t^v$,false)
5:      $f_t^v \leftarrow$ the first received frame from $v$.
6:      if $a_t^v \neq g$ then
7:        ground-truth $\leftarrow$ Inference($g$,$f_t^v$)
8:      else
9:        ground-truth $\leftarrow$ Inference($g-1$,$f_t^v$)
10:     detected$\leftarrow$ Inference($a_t^v$,$f_t^v$)
11:     $k_{t+1}^v \leftarrow$ getFeedback($a_t^v$,ground-truth,detected)
12:     $r \leftarrow$getReward($k_{t+1}^v$) by Table 3.
13:     Calculate targetPolicy($Q_v, r$) by Eq. (3).
14:     $s_{t+1}^v \leftarrow$ getState($a_t^v, k_{t+1}^v$)
15: function GETFEEDBACK($a_t^v$,ground-truth,detected)
16:   if $a_t^v == g$ then
17:     accuracy$\leftarrow$ F-Score(ground-truth,detected)
18:   else
19:     accuracy$\leftarrow$ F-Score(detected,ground-truth)
20:   return feedback category of the accuracy by Table 2.
21: function BEHAVIORPOLICY($s_t$,$training$)
22:   if $training == true$ then
23:     With $\epsilon$ probability:
24:       $a_t^v \leftarrow$ a random action.
25:     Otherwise:
26:       $a_t^v \leftarrow$ action with the best reward.
27:   else
28:     With $\epsilon$ probability:
29:       $a_t^v \leftarrow$ a random action by normalized weight.
30:     Otherwise:
31:       $a_t^v \leftarrow$ action with the best reward.
32:   return specified model for action $a_t^v$.
```

the GPU is not oversubscribed for video feed v. If this condition is satisfied, the algorithm selects a heavier model. It updates the spatial GPU% and the frame rate for individual models according to this upgrade so that it does not lead to GPU oversubscription (line 7). Finally, the algorithm returns the final selected models for each video feed, and the respective GPU% allocated to each model.

Algorithm 2 Interdependent Resource Allocation

```
1: function RESOURCEALLOCATOR( $a_t^v, a_{t-1}^v \; \forall v \in V$ )
   /* utilize lighter models if needed to fit in the GPU */
2:   while models oversubscribe the GPU capacity do
3:     choose $v \in V$ with maximum accuracy, where $a_t^v$ is heavier
       than $a_{t-1}^v$
4:     downgrade $a_t^v$ to a lighter model for feed $v$
5:     update spatial GPU% allocation for models
   /* utilize heavier models if the GPU capacity allows */
6:   for each $v \in$ sort($V$, increasing accuracy) do
7:     upgrade $a_t^v$ to a heavier model if GPU% allows
8:   return models for video feeds and GPU% for models
```

(a) Before resource allocation (time slot $t-1$)



(b) Accuracy output and selected model (time slot t)

Stream	Feed1	Feed2	Feed3	Feed4	Feed5
c_{t-1}^v	0.98	0.97	0.63	0.60	0.69
a_t^v	large	large	standard	x-large	x-large

(c) After resource allocation (time slot $t+1$)

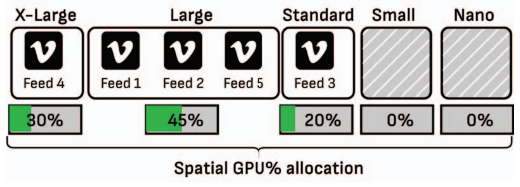


Figure 4. Illustrative example of resource allocation in action.

Figure 4 demonstrates the RAVAS resource allocator with an illustrative example. The figure shows five input stream video feeds labeled as feed 1 to feed 5. In Figure 4a, initial resource allocation and model selection for each video feed are shown. The first two feeds use the `x-large` model, utilizing 50% of the spatial compute capacity of a V100 GPU.

Figure 4b showcases the accuracy output of the models used at time slot t and the models selected by independent RL agents for the next time slot. However, concurrent processing of incoming frames from the video feeds using the newly selected models on a shared edge server with a V100 GPU results in GPU oversubscription and service-level agreement violations.

To tackle this issue, the RAVAS resource allocator employs a prioritization mechanism for video feeds, taking into account their output accuracy. Specifically, the feed with the highest level of accuracy is given the highest priority when allocating a lighter-weight model, surpassing its initially recommended model as determined by the RL model selection agent. This choice is based on the understanding that a lighter model stands a better chance of meeting the accuracy requirement for this particular feed, owing to its history of delivering superior accuracy with the previous (i.e., lighter) model.

Figure 4c shows that the resource allocator assigns the selected model by RL agents to video feeds 3 and 4 since there is available capacity. However, it cannot assign the `x-large` model to video feed 4 without oversubscription due to limited computing capacity. Instead, it allows an upgrade to a heavier model (i.e. `large`), although not the exact one chosen by the RL agent.

$$\begin{array}{r|ccccc|}
 & a_1 & a_2 & a_3 & a_4 & a_5 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
d \text{ very bad} & -\text{inf} & -\text{inf} & -\text{inf} & 0 & 0 \\
d \text{ bad} & -\text{inf} & -\text{inf} & 0 & 0 & 0 \\
d \text{ ideal} & -\text{inf} & -\text{inf} & 0 & -\text{inf} & -\text{inf} \\
d \text{ excellent} & 0 & 0 & 0 & -\text{inf} & -\text{inf} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots
\end{array}$$

Figure 5. A demonstration of Q-table with initial values only for the third model

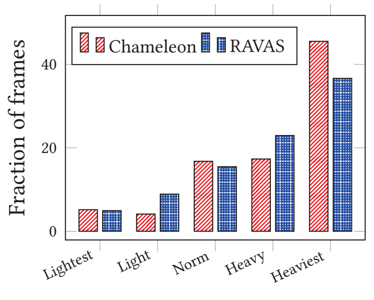


Figure 6. Fraction of individual model usage for processing frames for RAVAS and baseline algorithms.

5 Evaluation

Implementation Details. We built the RAVAS framework on a Kubernetes cluster v1.22.2. The framework is implemented in Python along with a set of scripts in Bash with a total size of around 3000 LoC. We containerized all the modules of the framework in Docker. We also provided the framework deployment for Kubernetes in various YAML files. Object detection models execute on TensorRt 8.0.1 framework [27, 28]. The connection between video feeds and the Ravas components is implemented using FFMPEG [29] that supports a wide range of protocols, muxers, demuxers, and filters. The monitoring module of the RAVAS framework uses Prometheus [30] as a standard, scalable cloud-native solution to collect application- and infrastructure-level metrics. All infrastructure-level GPU metrics are monitored using GPU node exporter [31]. A custom *pod-monitor* is implemented in Python as a custom Kubernetes resource [32, 33] to scrape application-level metrics from all Streamer modules, with a 1s time interval. The Optimizer queries metrics through Prometheus HTTP API [34] and Prometheus Query Language (PromQL) [35].

5.1 Experimental Setup

Hardware Setup. We used six physical machines to run the Streamer modules with 2.0 GHz CPU, 8 GB of physical memory, and 20 GB HDD. Each Streamer module is run in a separate physical machine to facilitate isolation. We also used an additional physical machine with a GPU as the edge server. The machine has 8X 2.0 GHz CPU cores, 16 GB of physical memory, 100 GB HDD, and $1X$ GPU accelerator. We experiment with four different types of GPU accelerators as edge servers, namely: Nvidia K80 with 12 GB GPU memory, Nvidia Tesla P4 with 8 GB GPU memory, Nvidia Tesla T4 with 16 GB GPU memory, and Nvidia V100 with 16 GB GPU memory.

Configurations. The frame rate for all video feeds is 10 frames per second. The optimization time interval is 500 *ms*. We chose this interval since it is long enough to observe the effects of the re-configuration and short enough to adapt more quickly to the dynamic execution requirement of the application. Different constraint values for C_{acc}, C_{iou}, and C_{conf} in the problem are 0.8, 0.5, and 0.7 respectively. The values for parameters γ and l in the target policy are set to 0.99 and 1.0 respectively. Also, we set β as random exploration probability in the testing phase equal to 0.01. We use the five algorithms in Table 1 in our experiments, with yolov4-608 being the base algorithm.

Dataset. In our evaluations, we conduct experiments using videos from the VIRAT dataset release 2.0 as the content for the streamed video feeds. We use all long ground videos of the VIRAT dataset, which are captured from surveillance cameras[36]. These videos have large spatial and temporal differences in their content, allowing us to make experiments under varying content. In our experiment, each Streamer module receives 10 frames per second from one of these videos. The first ten percent of frames of each video is used to train the Q table for RAVAS algorithm.

Comparison Scenarios and Baselines. We evaluate the performance of RAVAS in three different experiment scenarios. In the first two scenarios, we examine the performance of the proposed algorithm in processing the frames of a video feed and multiple concurrent video feeds on the edge server, respectively. In the last scenario, we evaluate the performance of RAVAS with different edge GPUs. We use two different sets of baselines to make a fair comparison in the evaluations: (1) baselines specifically focusing on model selection: Chameleon [6], BLEU [24], and LW [25]; (2) baselines focusing on video analytics systems: AWStream [13], and OTIF [12]. All model selection baselines are specifically used for evaluation in the first experiment. Additionally, some model selection and video analytics baselines are used to evaluate all experiments.

5.2 Evaluation with a Single Video Feed

Our first experiment aims to evaluate the performance of the RAVAS without considering the effects of concurrent processing of video feeds on the edge. To do so, only a single Streamer module receives frames from one video feed. We repeat the experiments for all videos in the VIRAT dataset. To show RAVAS in operation, Figure 6 shows the fraction of frames processed using each model from all the videos by both RAVAS and Chameleon. The fraction of frames processed by the heaviest model is 10% lower using RAVAS compared to Chameleon.

In Figure 7, we assess the performance of RL-based model selection compared to baseline selection models. To show how the selection of lighter models affects performance, figures 7-a and 7-b illustrate fractions of frames with accuracy constraints higher or equal to 0.7 and 0.8, respectively, on the x-axis as well as total frame processing latency in *ms* on the y-axis. Each point represents the average result of processing all frames of one of the ten video feeds in the data set. The figures reveal that the inference time (latency in *ms*) when employing the RAVAS model selection algorithm is nearly 50% better than Chameleon, with no significant difference in accuracy. Also, Ravas demonstrates significant improvement in both latency and accuracy compared to BLEU and LW. Additionally, Figure 7-c presents the average precision, recall, and F1-Score for all video feeds, comparing RAVAS with the three baseline models. The figure illustrates that Chameleon outperforms all other baselines in precision while it ranks as the worst-performing algorithm in terms of latency. Both RAVAS and Chameleon yield higher F1 scores and better overall accuracy. The primary reason for the low recall in

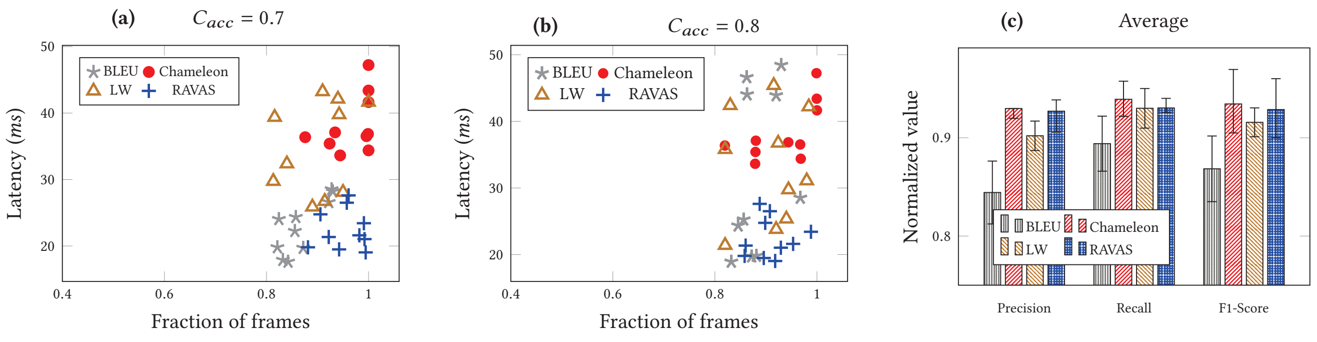


Figure 7. (a) and (b) fraction of frames with accuracy constraint C_{acc} = 0.7 and C_{acc} = 0.8 over latency. Each point represents the output of one video feed in the dataset. (c) Average precision, recall, and F1-Score.

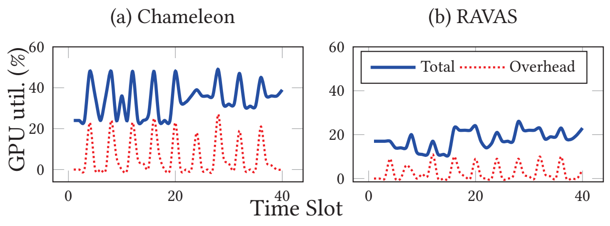


Figure 8. An illustration of GPU utilization over time for Chameleon and the RAVAS approach. The red dotted line represents the GPU overhead of the model selection algorithm and the solid blue line represents total GPU utilization of the model selection over time.

BLEU lies in its propensity to produce considerably more false-negative errors due to the omission of processing entire segments of incoming frames.

There are two reasons for RAVAS to have better latency performance. First, there is a lower fraction of heavier models used by RAVAS for processing frames, as illustrated in Figure 6. Second and foremost is the lower overhead of RAVAS in measuring the accuracy of individual models and in selecting a model according to the temporal effects of the environment (as discussed in Section 3). Figure 8 illustrates the total GPU utilization and the GPU usage overhead over time for RAVAS versus Chameleon to show the overheads of the different approaches. Figure 8-a shows that Chameleon has higher GPU utilization overhead peaks, resulting in sharp peaks in total GPU utilization. In contrast, Figure 8-b illustrates that the overhead of RAVAS caused smaller peaks, and as a result, the total GPU utilization is lower and smoother over time. Chameleon can use up to 48% of the GPU in processing a single video feed, while the RAVAS uses the GPU up to 26% at peak.

We also look at the average power and GPU utilization for the four baselines, including model selection and video analytics systems, along with the heaviest model (denoted as Golden) using the Nvidia toolkit. Figures 9 and 10 illustrate the average GPU power consumption and GPU utilization for all the videos for a single video feed. According to these two Figures, The Golden algorithm has no overhead on the GPU for model selection because it always uses a single model to process all incoming frames. Hence, it does not do model selection and evaluation. However, this algorithm requires more computational power than RAVAS and the other

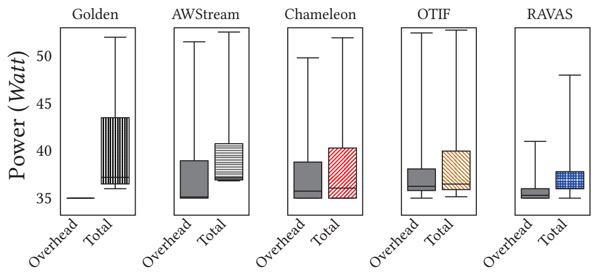


Figure 9. GPU power consumption of individual frames in all videos in the dataset for single video processing experiments.

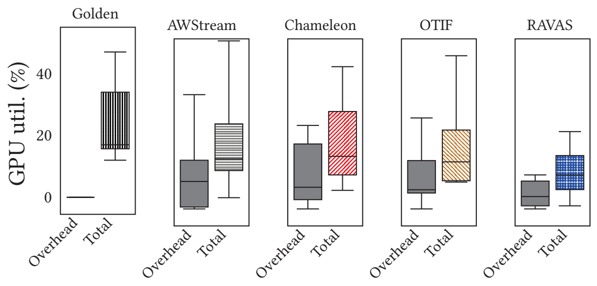


Figure 10. GPU utilization of individual frames in all videos in the dataset for single video processing experiments.

baseline algorithms since it always uses the heaviest model to process the frames. As a result, it uses higher total GPU utilization and power usage compared to other algorithms. The main difference in the GPU utilization of the proposed RAVAS and the four baselines lies in the overhead GPU utilization and power usage. RAVAS algorithm results in a significant improvement in total GPU usage by 43.4%.

Finally, to give more insights into the performance of RAVAS, Figure 12 illustrates the cumulative distribution function (CDF) for the fraction of frames processed with different latency values and accuracy thresholds. Figure 12-a shows that RAVAS processes a higher fraction of frames with lower latency than baseline algorithms. For example, 90% of the frames processed by RAVAS had a latency within 40 *ms*, while Golden, Chameleon, AWStream, and OTIF algorithms were respectively able to process only 63%, 70%, 74%, and 76% of the frames within the same period. Similar to Figure 7, Figure 12-b shows no substantial difference in the fraction of

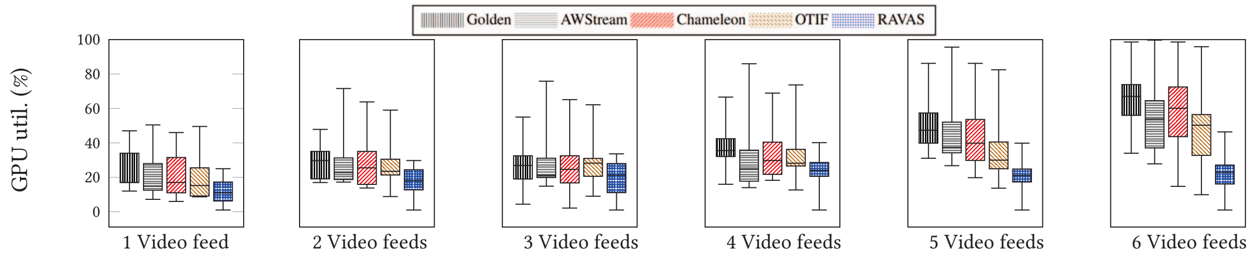


Figure 11. Average GPU utilization over different number of incoming stream video feeds.

frames processed with different accuracy constraints between the proposed and the baseline algorithms.

Key takeaways: RAVAS reduces GPU overhead and achieves lower latency with its lightweight interference-aware model selection approach compared to baselines.

5.3 Evaluation with Concurrent Video Feeds

In this experiment, we evaluate the effect of concurrent frame processing of multiple video feeds multiplexed on the same GPU on the performance of RAVAS. In this scenario, there are multiple Streamer modules; each receives frames from a specific video feed and transmits them to a unique model instance on the edge server. There are multiple model instances on the edge server to perform concurrent frame processing. We performed different experiments with one to six concurrent video feeds. Figure 11 shows the GPU utilization of the edge server while it processes the frames of concurrent video feeds. If the number of concurrent video feeds increases, the total amount of computation on the GPU increases, and as a result, the GPU utilization increases. As described previously, the maximum GPU utilization is affected by the overhead of the model selection algorithm and total computation to process concurrent frames. The maximum GPU utilization with concurrent video feeds in the baseline systems is significantly higher than RAVAS. Given that the computational overheads of Chameleon, AWStream, and OTIF are more than RAVAS (refer to Figure 10). The three baselines pose higher total GPU utilization with higher and more aggressive computation fluctuations over time than RAVAS. Although the Golden algorithm does not lead to computational overheads due to model selection, its heavy computation to process the incoming frames

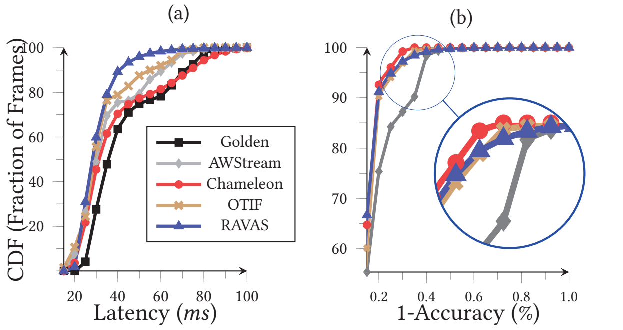


Figure 12. The cumulative distribution function (CDF) of the (a) latency and (b) accuracy.

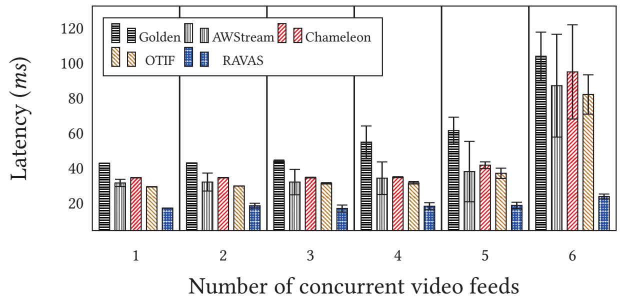


Figure 13. Average latency over different number of concurrent video feeds.

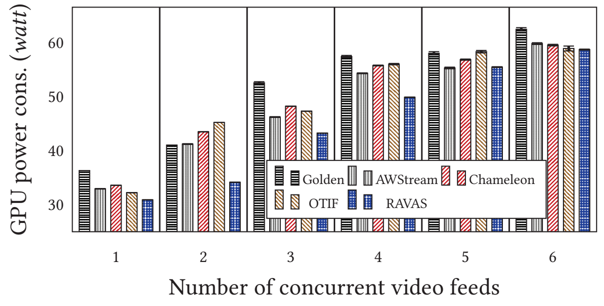


Figure 14. Average GPU power consumption over different number of concurrent video feeds.

significantly increases GPU utilization. In contrast, RAVAS causes a smoother increase in GPU utilization compared to baseline algorithms in concurrent video feed processing. For example, the GPU is fully utilized in all baselines when there are six concurrent video feeds, while RAVAS uses up to 60% of the GPU compute capacity.

Figures 13 and 14 illustrate the average frame processing latency and GPU power consumption for different numbers of concurrent video feeds, respectively. These figures show that the latency and the GPU power consumption are correlated with the GPU utilization. The increase in concurrent video feeds leads to increased latency and GPU power consumption for the baselines compared to RAVAS. Specifically, there is a considerable increase in latency for all baseline algorithms when there are six concurrent video feeds because the GPU is highly or fully utilized most of the time.

Key takeaways. RAVAS enables higher concurrency for stream video feeds with smoother GPU utilization, effectively mitigates interferences, and ensures end-to-end latency guarantees.

5.4 Impact of Different GPU models

In our final set of experiments, we wanted to evaluate how RAVAS and the different baselines perform with different GPU accelerators to understand performance variation across GPU architectures. We use the four GPUs described earlier with different computing capacities and power consumption levels. The edge server runs only one of the four GPU accelerators in each experiment. Before running our experiments, we profiled the four GPUs for the maximum number of feeds it can support. The experiments were performed with the maximum number of concurrent video feeds that a GPU accelerator is capable of processing: 1 video feed for Nvidia K80; 2 concurrent video feeds for Tesla P4; and six concurrent video feeds for Tesla T4 and Nvidia V100 GPU accelerators.

Figures 15 and 16 show the total GPU utilization and the average latency of the edge server with the different GPU models resepctively. Golden, Chameleon, and OTIF over-utilized K80, Tesla P4, and Tesla T4 accelerators. Also, AWStream over-utilized K80 and Tesla P4 accelerators. On the contrary, RAVAS performed significantly better in GPU utilization and did not over-utilize any GPU models. To better understand what happens when the GPUs are over-utilized, we plot the fraction of frames processed by each framework in Figure 17-b. None of the baselines is capable of supporting the processing of the video stream on the K80, with frame drop rates above 90%. On the contrary, RAVAS can process around 80% of the frames. Figure 17-a also shows the accuracy of the processed frames of the four frameworks. Note that even though Chameleon shows higher accuracy, it dropped more than 90% of the frames.

Figure 16 shows the frame processing latency for different algorithms on different GPU models. The figure shows that the average frame processing latency of all baseline algorithms has increased drastically due to the GPU over-utilization. For example, the average frame processing latency for Golden and Chameleon algorithms is 75, 700 *ms* and 3, 880 *ms*, respectively, while the frame processing latency of RAVAS algorithm is around 70 *ms* on Nvidia K80 GPU.

Figure 18 illustrates the fraction of violated frames in meeting the deadline for completing the frame processing, which is particularly beneficial for real-time applications. For this experiment, we consider a frame as violated if its processing takes longer than the deadline, as shown in the x-axis. We consider different values for the deadline from 80 to 1000 *ms* to see the performance of different algorithms on various latency constraints and GPU accelerators. As shown in the figure, RAVAS achieves an outstanding performance

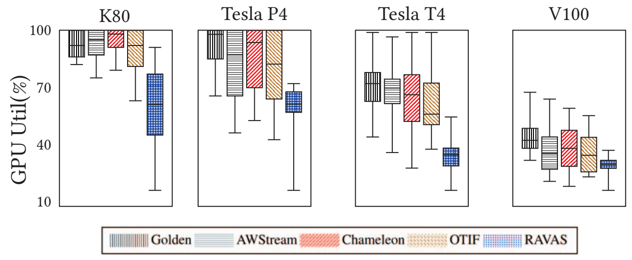


Figure 15. Average GPU utilization on different types of GPUs: (a) Nvidia K80 with one video feed; (b) Tesla P4 with two concurrent Stream video feeds; (c) Tesla T4 with six concurrent stream video feeds; and (d) Nvidia V100 with six concurrent stream video feeds.

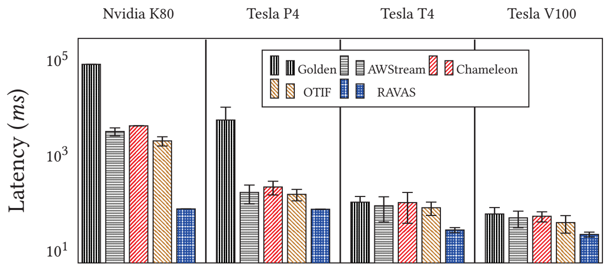


Figure 16. Average total latency on different GPU models.

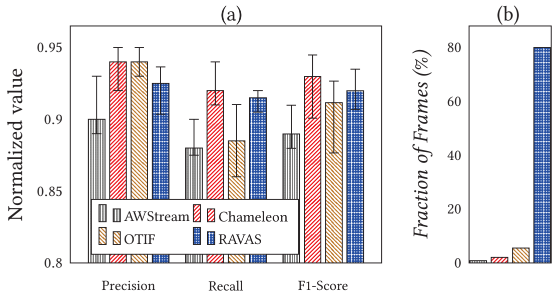


Figure 17. (a) Average accuracy metrics on K80 GPU; (b) Fraction of frames succeeded processing within 100 *ms* on K80 GPU.

where it secured the processing of most of the frames within the deadline compared to the baseline algorithms. For example, Golden, AWStream, Chameleon, and OTIF resulted in 100%, 98.4%, 95.9%, and 82.8% latency violation, respectively, when the deadline is 200 *ms* while RAVAS resulted only in a 0.3% latency violation. The proposed algorithm has no violation for a deadline higher or equal to 200 *ms*. This is attributed to the efficient usage of the GPU by the RAVAS algorithm.

Key takeaways. RAVAS avoids GPU over-utilization and ensures end-to-end latency within latency deadlines across different GPU models.

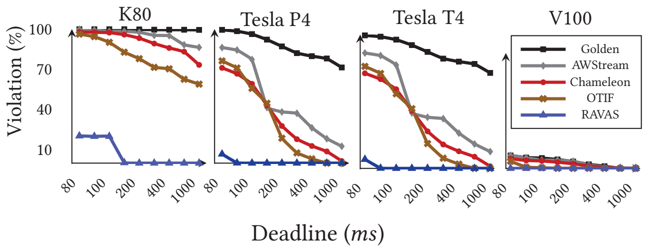


Figure 18. Fraction of violated frames on (a) Nvidia K80, (b) Nvidia Tesla P4, (c) Nvidia Tesla T4, (d) Nvidia Tesla V100 GPUs with different latency contraint treshold values.

6 Related Work

Edge video analytics applications are projected to be the most prominent use-case for edge computing [4]. Model configuration techniques for wide-area video network analytics have been suggested as a possible way to manage the resource usage, bandwidth, and performance tradeoffs in these systems. Different efforts have been made to reduce computing resources for video analytics.

Model Selection. Some recent works aim to choose the best configurations for the object detection model and frame input resolution as the main knobs that impact both resource usage level and prediction accuracy[6]. Zhang et al. [37] design a profiling-based video analytics system to maximize resource usage over quality. It profiles a large number of initial queries for each video. It then estimates resource quality and approximates the best configuration to optimize accuracy and resource usage. Nigade et al. [38] introduce an adaptation mechanism by considering service-level objectives for timely edge analytics. A naive feedback control makes adaptation decisions. Zhang et al. [39] introduce a rule-based model selection algorithm to select the best DNN model from a set of light to heavy DNN models for fluctuating workloads. Their algorithm decided to choose lighter models in the presence of load spikes. Kim et al. [25] introduce LW as a lightweight online profiling and configuration adaptation to dynamically optimize resource-accuracy trade-offs for multiple video streams on GPU-enabled edge servers with limited resources. LW enhances the model selection process through lightweight online profiling and adaptation, ultimately minimizing accuracy degradation while improving resource utilization within their system. Marco et al. [24] present a dynamic DNN model selection approach that balances accuracy and inference time based on input characteristics. Their model selection method, BLEW, leverages machine learning for swift model choice, enhancing execution optimization on embedded devices.

Infernce Serving Systems. Crankshaw et al. [40] design a general-purpose prediction serving system, Clipper, that supports customizing model selection policies in a containerized environment so that applications use isolated compute resources. Clipper also supports adaptive batching to maximize throughput and prediction caching to reduce latency. Ran et al. [41] design a framework for video analytics on front-end devices (i.e., smartphones), edge, and cloud resources. The framework uses only one convolutional neural network for object detection. The authors propose an offloading strategy that decides where to compute based on the estimation of the network condition while considering the application's requirements. Romero et al. [42] design an automated model-less system to provide inference-as-a-service. Their main contribution is to provide an inference system that does not require developers to specify various performance and accuracy requirements. Developers only submit their tasks and their application requirements. Then, the system automatically deploys a specific object detection model on a specific compute resource (e.g., CPUs and GPUs) and the scaling configurations to process the developer tasks.

7 Conclusion

The paper presents RAVAS as a lightweight, scalable, real-time edge video analytics framework with an RL-based model selection technique. The RAVAS framework handles the communication between video feed sources and object detection model instances, monitors, analyzes, and makes online optimized decisions. The proposed ML-based optimization algorithm automatically determines a configuration that optimizes GPU resource usage while considering the performance requirement of an edge-based video analytic application. Our experimental evaluation in a real environment demonstrates that RAVAS outperforms the baseline approaches in processing latency, GPU utilization, and power consumption while meeting the desired accuracy threshold. The experiments were made using various GPUs in an edge server for individual and concurrent videos to evaluate the performance of the proposed RAVAS framework under various conditions. Our analysis using a diverse data set shows that RAVAS incurs 57% less compute overhead and achieves 41.29% improvement in latency and 43.4% savings in total GPU usage for a single video feed and up to 99% and 40.0% reduction in latency and total GPU usage respectively for multiple concurrent video feeds while still meeting accuracy constraints.

Acknowledgments. This work was partially supported by Ericsson Research and the Industrial Doctoral School at Umeå University. Ahmed Ali-Eldin is supported by an SSF Future Research Leader grant. Prashant Shenoy is supported by NSF grants 2211302, 2211888, 2105494, and an Adobe grant.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.
[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
[3] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.
[4] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. Real-time video analytics: The killer app for edge computing. *Computer*, 50(10):58–67, 2017.
[5] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 119–135, 2022.
[6] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 253–266, 2018.
[7] Qianlin Liang, Prashant Shenoy, and David Irwin. Ai on the edge: Characterizing ai-based iot applications using specialized edge architectures. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pages 145–156, 2020.
[8] Jack Choquette, Edward Lee, Ronny Krashinsky, Vishnu Balan, and Brucek Khailany. 3.2 the a100 datacenter gpu and ampere architecture. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 48–50. IEEE, 2021.
[9] Nan Tian, Ajay Kummar Tanwani, Jinfa Chen, Mas Ma, Robert Zhang, Bill Huang, Ken Goldberg, and Somayeh Sojoudi. A fog robotic system for dynamic visual servoing. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1982–1988, 2019.
[10] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi 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*, SIGCOMM '20, page 359–376, New York, NY, USA, 2020. Association for Computing Machinery.
[11] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya Dulloor. Scaling video analytics on constrained edge nodes. *Proceedings of Machine Learning and Systems*, 1:406–417, 2019.
[12] Favyen Bastani and Samuel Madden. Otif: Efficient tracker pre-processing over large video datasets. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 2091–2104, New York, NY, USA, 2022. Association for Computing Machinery.
[13] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 236–252, 2018.
[14] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529*, 2017.
[15] Aditya Dhakal, Sameer G Kulkarni, and K. K. Ramakrishnan. Machine learning at the edge: Efficient utilization of limited cpu/gpu resources by multiplexing. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–6,

2020.

[16] Aditya Dhakal, Sameer G Kulkarni, and K. K. Ramakrishnan. Primitives enhancing gpu runtime support for improved dnn performance. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 53–64, 2021.

[17] Ahmed Ali-Eldin, Bin Wang, and Prashant Shenoy. The hidden cost of the edge: a performance comparison of edge and cloud latencies. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2021.

[18] Aditya Dhakal, K. K. Ramakrishnan, Sameer G. Kulkarni, Puneet Sharma, and Junguk Cho. Slice-tune: A system for high performance dnn autotuning. In *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, Middleware '22, page 228–240, New York, NY, USA, 2022. Association for Computing Machinery.

[19] Aditya Dhakal, Sameer G Kulkarni, and K. K. Ramakrishnan. Gslice: Controlled spatial sharing of gpus for a scalable inference platform. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, SoCC '20, page 492–506, New York, NY, USA, 2020. Association for Computing Machinery.

[20] Can Wang, Sheng Zhang, Yu Chen, Zhuzhong Qian, Jie Wu, and Mingjun Xiao. Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 257–266. IEEE, 2020.

[21] Chengcheng Wan, Muhammad Santriaji, Eri Rogers, Henry Hoffmann, Michael Maire, and Shan Lu. ALERT: Accurate learning for energy and timeliness. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 353–369. USENIX Association, July 2020.

[22] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 377–392, 2017.

[23] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Tvm: An automated end-to-end optimizing compiler for deep learning. OSDI'18, page 579–594, USA, 2018. USENIX Association.

[24] Vicent Sanz Marco, Ben Taylor, Zheng Wang, and Yehia Elkhatib. Optimizing deep learning inference on embedded systems through adaptive model selection. *ACM Trans. Embed. Comput. Syst.*, 19(1), feb 2020.

[25] Woo-Joong Kim and Chan-Hyun Youn. Lightweight online profiling-based configuration adaptation for video analytics system in edge computing. *IEEE Access*, 8:116881–116899, 2020.

[26] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 269–286, 2018.

[27] Nvidia tensorrt: Programmable inference accelerator, 2018. https://developer.nvidia.com/tensorrt/.

[28] Tensorrt demos. https://github.com/jkjung-avt/tensorrt_demos/.

[29] Ffmpeg. https://ffmpeg.org/ffmpeg-formats.html/.

[30] Prometheus. https://prometheus.io/.

[31] Nvidia gpu prometheus exporter. https://github.com/mindprince/nvidia_gpu_prometheus_exporter/.

[32] Prometheus operator. https://github.com/prometheus-operator/prometheus-operator.

[33] Prometheus client library. https://github.com/prometheus/client_python/.

[34] Prometheus http api. https://prometheus.io/docs/prometheus/latest/querying/api/.

[35] Prometheus query language. https://prometheus.io/docs/prometheus/latest/querying/basics/.

[36] The virat video dataset. https://viratdata.org/.

[37] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. Live video analytics at scale with approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 377–392, Boston, MA, March 2017. USENIX Association.

[38] Vinod Nigade, Ramon Winder, Henri Bal, and Lin Wang. Better never than late: Timely edge video analytics over the air. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, page 426–432, New York, NY, USA, 2021. Association for Computing Machinery.

[39] Jeff Zhang, Sameh Elnikety, Shuayb Zarar, Atul Gupta, and Siddharth Garg. Model-Switching: Dealing with fluctuating workloads in Machine-Learning-as-a-Service systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association, July 2020.

[40] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A Low-Latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, Boston, MA, March 2017. USENIX Association.

[41] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1421–1429, 2018.

[42] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. INFaaS: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 397–411. USENIX Association, July 2021.