

Short Paper: ASTRA – Low-Overhead Runtime Architecture for STReam Adaptation in Video Analytics

Mahshid Ghasemi, Zoran Kostic, Javad Ghaderi, and Gil Zussman
Columbia University, USA
{mahshid.ghasemi, zk2172, jghaderi, gil.zussman}@columbia.edu

Abstract—Real-time video analytics is crucial for smart city applications and cloud-connected vehicle control. To improve analytics accuracy, it is desirable to process the video at the highest resolution and frame rate. However, due to limited resources, streaming and processing video at the highest resolution and frame rate from all cameras is not feasible and adversely affects the analytics latency. Intelligent adaptation of cameras’ resolutions and frame rates based on network conditions and the video content is crucial in order to optimize the performance. In this paper, we present ASTRA, a low-overhead runtime architecture for online adaptation of live camera analytics at the edge. ASTRA can execute various online algorithms as a black box. We deployed ASTRA in the realistic NSF COSMOS testbed and uniquely assessed its real-time performance using COSMOS’ street-level cameras. The results indicate that ASTRA can provide system reliability (i.e., the probability of meeting accuracy and latency requirements) of more than 90% while maintaining performance within a deviation of less than 10% from optimal offline performance with average GPU utilization overhead of around 2% per camera.

Index Terms—Video analytics, edge computing, online adaptation, testbed evaluation

I. INTRODUCTION

Video analytics, powered by Deep Neural Networks (DNNs), has gained importance in a wide range of applications, including smart city infrastructure, traffic signal optimization, and cloud-connected vehicle control [1], [2], [3]. The recent proliferation of traffic cameras, coupled with the availability of advanced edge/cloud computing resources, is expected to facilitate large-scale video analytics.

Consider a system of geo-distributed cameras streaming to edge/cloud servers for real-time object detection. These cameras continuously generate a large volume of data that must be encoded, transmitted, decoded, and processed in real-time while adhering to network and computational capacity constraints. A microcosm of such a system is deployed in the NSF COSMOS wireless edge-cloud testbed in New York City (NYC) (see Fig. 1) [4], [2]. As part of this testbed, several traffic cameras are installed at busy streets and intersections in Manhattan, and their real-time video streams are processed at COSMOS’ edge servers to extract traffic- and crowd-related information such as pedestrian and vehicle density.

Two key performance metrics in such systems are **accuracy** and **(end-to-end) latency**. *Accuracy* is determined by evaluating how closely the output of the object detection (i.e.,



Figure 1: The NSF COSMOS testbed provides cameras and edge servers for piloting real-time video analytics systems.

bounding boxes and labels) aligns with manually annotated ground-truth data. *Latency* comprises encoding latency, network latency, decoding latency and DNN inference latency. Accuracy and latency are significantly impacted by network conditions, computational capacity, and video content.

System and environmental conditions. Accuracy and latency are significantly affected by dynamic *network conditions*, available *computational resources*, and *video content*. Reduced network throughput or packet loss can increase transmission latency or degrade video quality, while limited GPU capacity can increase inference latency when processing multiple high-resolution streams. Meanwhile, the resolution and frame rate required for sufficient accuracy depend on scene content: for example, lower frame rates may suffice when vehicles move slowly, while lower resolutions may be adequate when objects are sparse or close to the camera. Therefore, dynamically adapting resolution and frame rate can substantially reduce bandwidth and computational requirements while maintaining analytics performance. Therefore, optimizing parameters such as resolution and frame rate, can significantly save on bandwidth and computational power.

Iterative adaptation algorithm. The objective is to dynamically select video-stream resolutions and frame rates that maximize overall performance (a function of accuracy and end-to-end latency), subject to application-specific accuracy and latency constraints. We refer to each resolution, frame-rate combination as a *configuration*. Our measurements, consistent with prior work [5], show that accuracy and latency are *time-varying and unknown* functions of the configuration, affected by video content, network and computing resources, and DNN models. Thus, deterministic or offline optimization based on empirical models cannot reliably capture system behavior. Instead, iterative online adaptation evaluates configurations during real-time operation and selects each new configuration

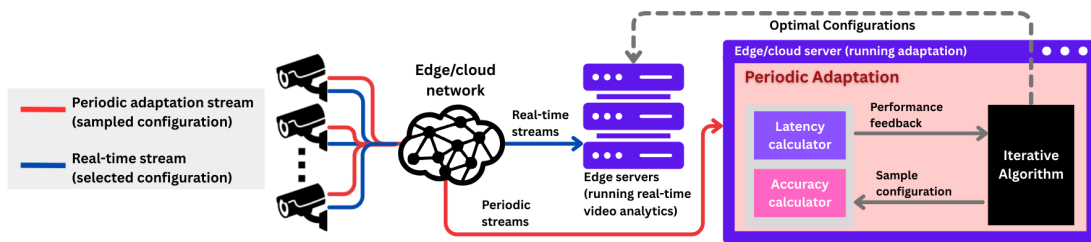


Figure 2: Deployment of online adaptation in a multi-camera setup connected to edge servers.

based on previously observed performance. We refer to such methods as *iterative algorithms*.

Key idea. The primary goal of online adaptation is to maximize performance under constrained resources. Achieving this requires that: (i) the adaptation runtime architecture incurs minimal overhead when executing the iterative algorithm, and (ii) given the rapid evolution of online adaptation algorithms, it is crucial that the architecture allows for easy integration of new algorithms, requiring only minimal changes. To meet these requirements, we introduce **ASTRA**, a low-overhead runtime Architecture for STREAM Adaptation in real-time video analytics that enables execution of existing and future iterative algorithms as a black-box (as shown in Fig. 2). A patent based on ASTRA is pending [6]. A more comprehensive version of this paper is presented in [7].

A. Challenges

Architecture challenges. Two main obstacles in executing online adaptation algorithms are as follows:

- (i) **Performance measurement cost:** *Iterative algorithms* sequentially measure each configuration’s performance. For object detection, measuring true accuracy requires manually annotated ground truth, which cannot be obtained in real-time. Instead, we use a **proxy ground truth** obtained via the most resource-intensive configuration (highest resolution and frame rate). Generating this proxy and thus measuring configurations’ accuracy/latency incurs significant overhead.
- (ii) **Switching delay:** *Iterative adaptation* evaluates configurations sequentially on short video segments, requiring frequent switching between video streams and analytics pipelines. These switches incur overhead from repeated decoding/encoding, buffering, pipeline reconfiguration, and data transfer.

Algorithmic challenges. Due to the incurred overhead of each iteration, the iterative algorithm must attain a near-optimal configuration in as few iterations as possible.

B. Contributions

Identifying and measuring runtime overheads in online video adaptation. We performed extensive profiling across memory, CPU, and GPU resources, and identified hidden bottlenecks and overheads in online adaptation systems. Quantifying these overheads and bottlenecks is crucial for verifying the feasibility of deployment.

Architecture that mitigates runtime overheads. We designed and implemented ASTRA as an online adaptation framework that mitigates the identified overheads.



Figure 3: COSMOS cameras deployed on the 1st and 2nd floor of a building and their view an intersection in NYC.

Executing representative adaptation algorithms as interchangeable black-box components. To compare different algorithmic choices, we used ASTRA to execute two different iterative algorithms (the “content” of the black-box in Fig. 2). We then compared their performance against two baselines: (a) the offline optimal (upper bound on achievable performance) and (b) a ground-truth-free adaptation method that estimates accuracy via confidence scores instead of ground-truth.

Realistic evaluation: We evaluated ASTRA in two complementary settings designed to capture both real-world conditions and fully controlled, reproducible scenarios.

(i) **Deployment on the COSMOS testbed.** We deployed ASTRA in the COSMOS testbed network and performed *real-time* adaptation for two live street-level cameras (depicted in Fig. 3) viewing an intersection in NYC [8]. Use of COSMOS cameras and their live and recorded video streams was designated IRB-exempt by Columbia University.

(ii) **Controlled and reproducible evaluation.** For fair comparison under identical conditions, we deployed ASTRA on Google Cloud virtual machines (VMs) and evaluated eight emulated cameras under realistic network latency. Cameras were emulated by streaming videos via the Real-time Transport Protocol (RTP). This setup preserves key characteristics of live deployments while enabling reproducibility across algorithms.

Results. Compared to baseline architectures, ASTRA reduces the per-iteration execution overhead of the iterative adaptation algorithm by an order of magnitude (e.g., from seconds to hundreds of milliseconds). ASTRA can achieve performance within **10%** of the optimal performance. It ensures system reliability (i.e., probability of meeting the accuracy and latency requirements) of more than **90%**. ASTRA incurs low adaptation overhead, averaging around **2%** GPU utilization per camera over time.

II. RELATED WORK

Several studies presented various methods to enhance the performance of real-time video analytics, despite environmental dynamics, that primarily fall into the following categories based on their optimization approaches.

Configuration adaptation. These studies employ periodic adaptation through continuous DNN updates [9], [5], [10], dynamic model switching [11], [12], and adjusting resolution, frame rate, or encoding [13], [14], [15], [16], [17]. Other systems target specific conditions such as packet loss [18], satellite links [19], and camera orientation [20], while RIVA [21] and OAVS [22] adapt bitrate allocation under dynamic network conditions. The benchmark in [23] studies the impact of decoder and model-size on cloudlet-offloaded drone video analytics. Other studies focus on on-camera frame filtering [24], region of interest determination [25], or cross-camera overlap [26].

Distributed inference/learning. These studies leverage distributed computing via DNN partitioning and edge-cloud load balancing [27], [28]. PIB [29] compresses feature maps on edge cameras. In another study [30], smart cameras offload tasks to an edge server.

Previous studies primarily focus on adaptation algorithms (the “content” of the black-box in Fig. 2) and are often evaluated using recorded videos rather than real-time streams. In contrast, our work focuses on the *system challenges* of executing configuration adaptation in real-time.

III. ONLINE CONFIGURATION ADAPTATION

Consider a video analytics system where real-time streams from several cameras are transmitted to edge servers for object detection using DNN models (see Fig. 2). The key performance metrics of such a system are accuracy, (end-to-end) latency, and bandwidth consumption. **Bandwidth** manifests itself in accuracy and network latency from the camera to the edge server when streaming over Transmission Control Protocol (TCP) connections.

Adaptation objective. The objective is to dynamically find the configurations that maximize a value function of accuracy and latency, subject to constraints on maximum latency and minimum accuracy. The unpredictable and dynamic nature of the problem limits the effectiveness of empirical modeling and deterministic optimization in identifying near-optimal configurations. Hence, an iterative algorithm is required to estimate the system behavior in an online manner. At each iteration t of such an algorithm, one *configuration* x_t is sampled. Here, $x_t \in \mathcal{X}$, where $\mathcal{X}$ is a discrete set consisting of a finite number of supported resolutions and frame rates. Then, x_t ’s achieved *accuracy*, $A_t(x_t)$, and *latency*, $L_t(x_t)$ on a short segment (e.g., $\tau=0.2$ seconds) of the camera’s video stream is evaluated. The results up to iteration t are used to decide on the subsequent configuration x_{t+1} to be evaluated. The algorithm must be terminated upon identification of a near-optimal configuration. We refer to such algorithms as *iterative algorithms*.

Performance metrics. *Accuracy* $A_t(x_t)$ is calculated by comparing the obtained object detection results (i.e., bounding boxes and labels) using configuration x_t on a video segment against the ground truth. Following previous work [13], we use the F1-score metric. The detection F1-score is computed by verifying if a bounding box shares the same label and has sufficient spatial overlap with the associated ground truth. To

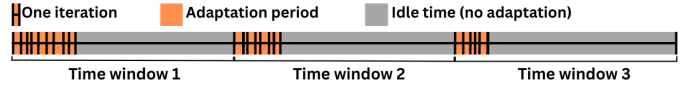


Figure 4: Three consecutive time windows.

account for the impact of frame rate on the accuracy, we use the location of objects from the previous sampled frame for a frame that is not sampled by the configuration. *Latency*, $L_t(x_t)$ is defined as the sum of the encoding latency by the camera, the network latency to transfer video segment from the camera to the server, the decoding latency on the edge server, and the inference latency to run object detection on the video segment with configuration x_t .

Proxy ground-truth. Since manually annotated ground truth is unavailable in real-time, we use the object detector’s output under the highest-resolution and highest-frame-rate *golden configuration* as a reference, following prior work [19], [31]. We refer to this reference as the *proxy ground truth*, or simply the *proxy*, and use it to estimate the accuracy of less resource-intensive configurations.

IV. ARCHITECTURE DESIGN

As discussed in Section I-A, each iteration of an iterative adaptation algorithm incurs substantial overhead. In particular, generating proxies (by definition) and sequentially switching between configurations and video streams incur network, computational, and temporal overhead. To execute such iterative algorithms efficiently, these overheads must be minimized. Therefore, an effective adaptation architecture should ensure that the system uses a near-optimal configuration majority of the time while incurring low overhead. Due to *unpredictable* variations in network conditions and video content, the optimal configuration shifts over time, requiring the adaptation process to be invoked *periodically*. The appropriate adaptation frequency depends on environment volatility. We refer to each adaptation cycle as a *time window*. A typical value for the time window is **1-2 minutes**. As illustrated in Fig. 4, at the beginning of each time window, adaptation resumes until a near-optimal configuration is found by the adaptation algorithm, which is then used for the remainder of the window. To keep adaptation overhead low, the adaptation period should remain small relative to the window length (e.g., below 10%). In this section, we illustrate how we designed ASTRA, starting from a baseline design.

A. Baseline Architectures

To perform online adaptation by running an *iterative algorithm* described in Section III, the baseline architecture for real-time configuration adaptation is shown in Fig. 5(a). In this architecture, during an adaptation period for each iteration t , the algorithm needs to observe both the accuracy and latency obtained under the **sampled configuration** x_t . This requires the system to momentarily process a short segment of video (for example, a fraction of a second such as 0.2s) using the channel associated with configuration x_t . Therefore, the system needs to switch rapidly between different channels

Table I: Channel and configuration switching latency. Values are mean (median, standard deviation).

Camera	Channel SW (ms)	Config SW (ms)	Reduction
Bosch	1217 (1207, 36)	57 (58, 11)	21.4×
HikVision	654 (597, 250)	25 (25, 7)	26.2×
Axis	411 (396, 34)	62 (63, 10)	6.6×

throughout the adaptation period until the algorithm converges to a near-optimal configuration.

To assess the feasibility of such rapid switching, we measured channel-switching latency for three COSMOS cameras: **HikVision** 8 MP Outdoor Dome, **Bosch** 4K UHD Outdoor Network PTRZ Dome, and **Axis** 8 MP Outdoor Dome. Switching latency is defined as the time from requesting a new channel until the first packet of its first frame arrives at the edge server. Since the COSMOS cameras are connected to the edge server over fiber with <10 ms network delay, the measured latency primarily reflects camera-internal switching. Each camera provides three channels: (3840×2160, 30 fps), (1920×1440, 15 fps), and (800×600, 10 fps).

Table I reports *channel switching* latency (Channel SW) from approximately **100** switches per camera among the three available channels. The latency is strongly camera-dependent, ranging from ~ 400 ms to over **1.2 s**; cameras such as Bosch, which use larger buffers and advanced streaming mechanisms, incur higher delays. These delays are significant when adaptation requires switching every 0.2 s.

To mitigate this latency and reduce the dependency on camera-specific buffering behavior, we improve upon the baseline architecture shown in Fig. 5(a) by introducing the modified design shown in Fig. 5(b). We replace the “*channel switcher*” component that repeatedly requests different channels from the cameras with a **down-sampler** component. This component only receives the highest-resolution, highest-frame-rate stream (Channel 1, indicated by green arrows in Fig. 5) and locally generates other configurations sampled by the iterative adaptation algorithm. Since Channel 1 with golden configuration, hereafter referred to as the **golden stream**, is already necessary to compute the proxy ground-truth, it can be mirrored internally at the edge server to perform the down-sampling, eliminating the need to request two separate streams from the camera.

Switching latency. Table I shows the switching results for the cameras, based on approximately **100** measurements per camera. Configuration switching via down-sampling reduces switching delay by about **20×** and reduces sensitivity to camera-specific buffering and jitter-control mechanisms.

Precise F1-score calculation. The improved architecture in Fig. 5(b) also enhances synchronization between the *ground-truth pipeline* (green box) and the *object detection pipeline* (dark purple box). Using a single stream from Channel 1 ensures that both pipelines operate on identically timestamped frames, avoiding frame alignment across separate channels whose timing can differ due to independent buffering and Network Time Protocol (NTP) drifts. Moreover, some cameras provide absent or incomplete Real-Time Control Protocol (RTCP) packets, making frame timestamps unreliable for

exact matching. The single-stream design therefore ensures accurate temporal alignment and reliable frame-level F1-score calculation.

Importance of temporal alignment. To quantify the impact of temporal misalignment on accuracy calculation during adaptation, we analyzed a **10-minute** recording from one COSMOS camera (2nd-floor view in Fig. 3). For each configuration, we computed the average F1-score over consecutive **0.2 s** adaptation iterations. The results are shown in Fig. 6.

Accurate F1-score measurement is critical for the *adaptation algorithm* to correctly model configuration accuracy. As shown in Fig. 6, even a one-frame misalignment between the *ground-truth pipeline* and the *object detection pipeline* can substantially alter the computed F1-score. In both resolution settings (Figs. 6(a) and (b)), although the highest frame rate configuration achieves the best true accuracy, a minor frame drift can make it appear as the lowest-performing configuration, potentially leading the iterative algorithm to select a suboptimal configuration.

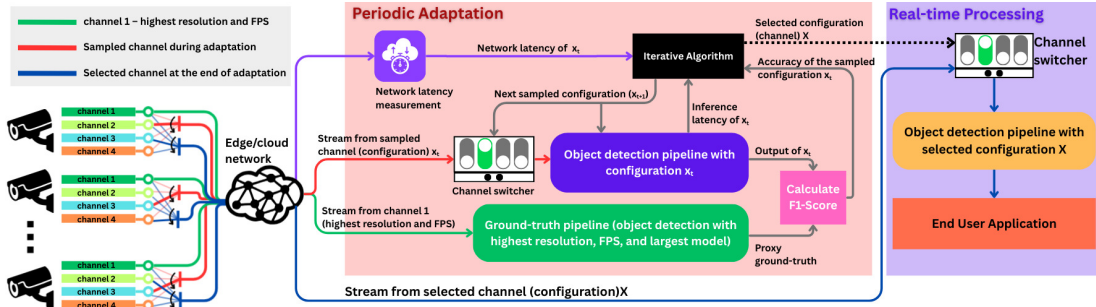
B. Our Architecture: ASTRA

The architecture should also manage differences in computational and time requirements among concurrent processes effectively to reduce their waiting (idle) time and ensure efficient resource utilization. For example, in Fig. 5(b), two major processes (outlined by white dashed contours) run in parallel during adaptation. *Process 1* (the down-sampling and object detection pipeline) operates on sampled configurations, while *Process 2* (the ground-truth pipeline) processes the golden-stream frames required to generate the proxy ground truth during a brief interval within the adaptation period.

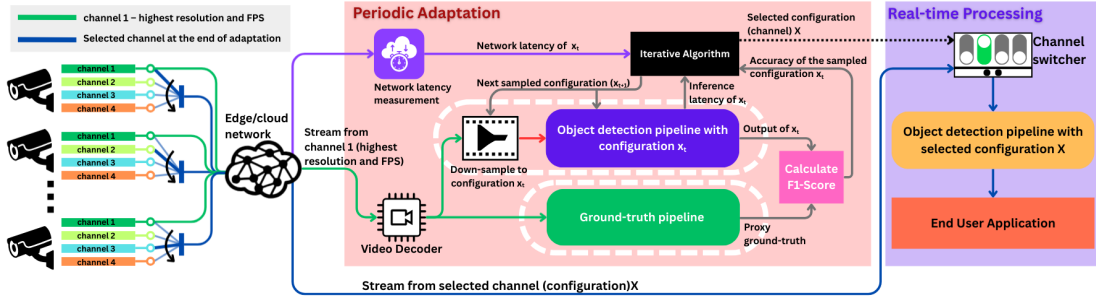
The runtimes of both processes vary across iterations: Process 1 depends on the selected configuration, while Process 2 depends on available GPU capacity. Thus, either process can become the bottleneck. To prevent one process from stalling the other and to maintain a high throughput, the architecture should execute these concurrent tasks asynchronously by introducing first-in-first-out (FIFO) queues at their inputs and outputs. This transforms the system into a classic *Fork-Join* model, where concurrent processes are decoupled through queued data flow. By structuring the adaptation processes as a Fork-Join system, the architecture decouples the concurrent tasks and eliminates global synchronization barriers, which significantly improves throughput and resource utilization. ASTRA is an asynchronous version of the architecture in Fig. 5(b) that incorporates this Fork-Join system.

Chunkification. To construct the video chunk queue, ASTRA includes a *chunkification* component that receives decoded frames and groups a small number of consecutive frames (e.g., 5) into a single video chunk. A natural choice for the chunk length is the **I-frame interval**, the number of frames between two successive I-frames in an H264/AVC bitstream.

Golden stream bandwidth consumption. In ASTRA (Fig. 7), the golden stream is transmitted only briefly during adaptation. In our evaluation setup (Section V), golden stream is active for only ~ 1 s per camera, while adaptation takes 4–7 s within a

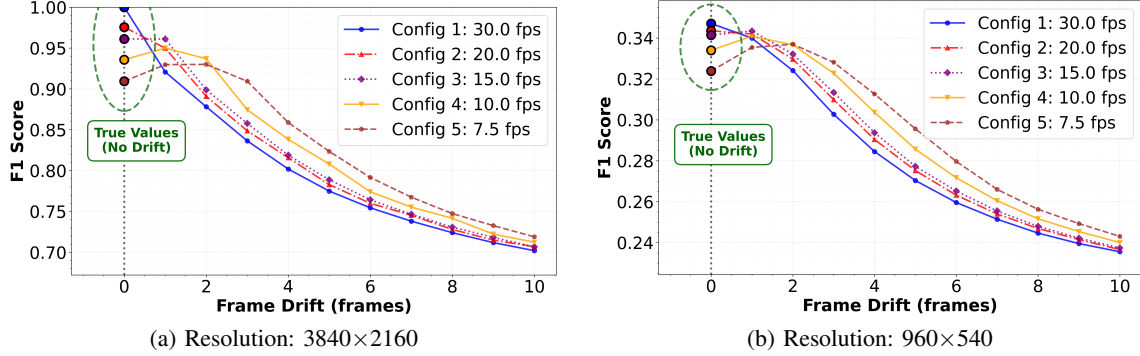


(a) Channel switching: Baseline architecture for online adaptation by executing an iterative algorithm.



(b) Configuration switching: Improved adaptation architecture that eliminates channel switching delay.

Figure 5: Comparison of (a) baseline and (b) improved architectures for online adaptation.



(a) Resolution: 3840×2160

(b) Resolution: 960×540

Figure 6: Impact of frame-drift between the *ground-truth pipeline* and the *object detection pipeline* on F1-score measurement.

60 s time window, and consumes approximately 10 Mbps while active. Moreover, ASTRA’s modular design allows *Process 1* (down-sampling and object detection) and *Process 2* (ground-truth generation) to be deployed near the cameras with high-bandwidth connectivity, while the continuously running analytics components (the “Real-time Processing” box in Fig. 7) can operate over bandwidth-constrained links.

V. EVALUATIONS

In this section, we evaluate ASTRA’s architecture in terms of accuracy, latency, and GPU utilization under diverse network conditions and video contents. We plug two iterative algorithms into the architecture (the “content” of the black box in Fig. 7): (i) GP-UCB-C derived from [32] and (ii) Chameleon++ derived from [13]. These algorithms represent two common classes of adaptation methods: (1) theoretically guaranteed algorithms (GP-UCB-C) and (2) empirically estimated algorithms (Chameleon++). We compare their performance against two baselines: (a) the offline optimal (upper

bound on achievable performance) and (b) a ground-truth-free adaptation approach that estimates accuracy using confidence scores rather than ground truth.

As described in Section III, the goal of the *iterative algorithm* is to select a configuration $x \in \mathcal{X}$ that maximizes a weighted accuracy–latency utility,

$$\max_{x \in \mathcal{X}} c_a A(x) - c_l L(x) \quad \text{s.t.} \quad A(x) \geq \alpha, \quad L(x) \leq \beta, \quad (\text{P1})$$

where $A(x)$ and $L(x)$ denote average accuracy and latency, $c_a, c_l \geq 0$ are application-specific weights, and α, β are application-specific accuracy and latency constraints.

GP-UCB-C: This is a constrained Bayesian optimization algorithm derived from [32]. It models the unknown accuracy and latency of each configuration using Gaussian Processes (GPs) [33]. At each iteration, it uses the GP confidence bounds to select a promising, feasible configuration, then updates the models using the observed performance. The algorithm stops once it identifies a near-optimal feasible configuration.

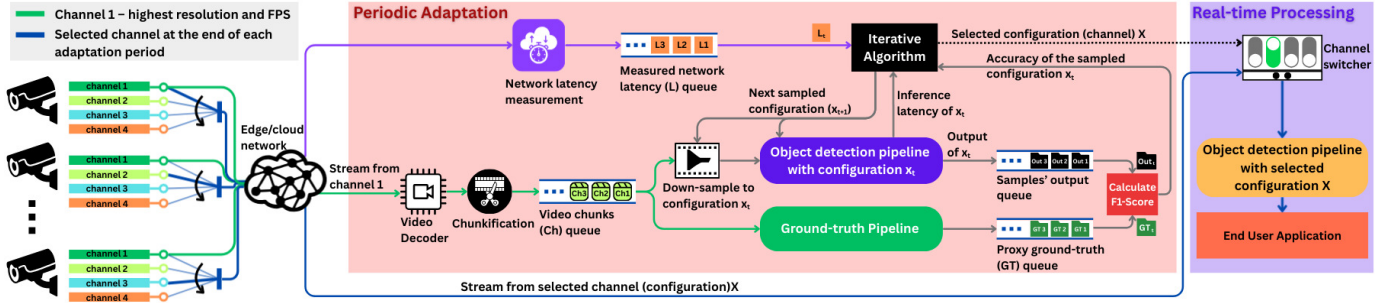


Figure 7: Overview of ASTRA: the design in Fig. 5(b) is converted into an asynchronous Fork-Join-style architecture.

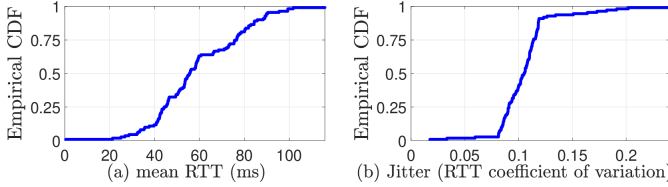


Figure 8: Network latency dataset distribution [34].

Chameleon++: It extends Chameleon [13], a profiling-based configuration search algorithm, by incorporating latency-awareness. It profiles a subset of configurations and estimates the accuracy and latency of the remaining configurations using parameter-independence assumption, allowing joint optimization of both metrics without exhaustive profiling.

AutoML++ (ground-truth-free adaptation): In this benchmark, the iterative algorithm is the GP-UCB-C algorithm that uses confidence scores generated by the object detection model as the accuracy metric instead of F1-scores that need proxy ground truth. This approach is inspired by the AutoML adaptation platform [15].

Offline optimal: This benchmark selects, for each time window, the best feasible configuration using full offline knowledge of $A(x)$ and $L(x)$. For multi-camera experiments, these metrics are averaged across cameras. Optimality is defined with respect to (P1), including its accuracy and latency constraints. Since this benchmark requires no adaptation and incurs no adaptation overhead, it serves as an upper bound on achievable performance.

Object detection models. In our experiments, the adaptation is performed for the object detection task, specifically pedestrians and vehicles detection, using YOLOv8 [35] (Section V-A) and YOLOv4 [36] (Sections V-B) models. Importantly, ASTRA is model-agnostic: any DNN-based object detection model, including newer versions of YOLO, can be integrated into its architecture.

Optimization setup and parameters. The evaluations are performed for two sets of parameters in (P1): (i) **accuracy constrained**, i.e., $c_a = 0$, $c_l = 1$, $\beta = +\infty$, and $\alpha \gg 0$ (Sections V-A and V-B) and (ii) **latency constrained**, i.e., $c_a = 1$, $c_l = 0$, $\alpha = 0$, and $\beta \ll +\infty$ (presented in [7]). The configuration set $\mathcal{X}$ used consists of 12 elements, each of which is a tuple (r_i, f_i) where r_i is one of the values $\{832, 608, 416\}$ and f_i is one of the values $\{30, 15, 10, 5\}$. A video stream with configuration (r, f) has resolution $r \times r$

and frame rate f fps. These three resolution levels and four frame rate levels provide sufficiently distinct operating points for the adaptation algorithm. Configurations with only small differences in resolution or frame rate typically exhibit similar accuracy and latency.

Accuracy and latency calculation. During camera i 's adaptation period in time window n , the system uses the configuration selected in the previous window, $x^{(i,n-1)}$. Once adaptation completes, it uses the newly selected configuration, $x^{(i,n)}$, for the remainder of the window. The achieved accuracy and latency in a time window are computed over the entire window: the performance under the previous configuration during adaptation and under the newly selected configuration after adaptation are weighted by their respective durations, and the resulting values are then averaged across the K cameras.

We evaluate ASTRA in two settings: (i) real-time adaptation of two live cameras in the COSMOS testbed (Section V-A) and (ii) real-time adaptation of up to eight emulated cameras on Google Cloud Platform (Section V-B).

Network latency dataset. To evaluate the performance of ASTRA under realistic network conditions between the cameras and the edge servers, we used a dataset of extensive real network latency measurements provided in [34] to induce extra latency. The dataset contains the RTT between end users and the nearest Alibaba Cloud site under 4G.

The empirical Cumulative Distribution Function (CDF) of the mean RTT and jitter (RTT coefficient of variation measured as the standard deviation divided by the empirical mean) are presented in Fig. 8. The detailed dataset was graciously provided by the authors of [34].

A. Deployment in COSMOS Testbed

We deployed ASTRA on the COSMOS testbed [37] and evaluated it using two *live* cameras (Fig. 3). In this setup, ASTRA runs on two Ubuntu 20.04 COSMOS servers, each with an Intel Xeon CPU@2.60GHz and an NVIDIA V100 GPU: one hosts the *real-time processing* and the other the *periodic adaptation* components shown in Fig. 7.

Selection of α . We varied α from 0.4 to 1 and observed that values below 0.85 compromised pedestrian detection accuracy, while values above 0.85 incurred high latency and GPU utilization. We therefore set $\alpha = 0.85$ for the **accuracy-constrained** scenario to balance accuracy and overhead.

We executed ASTRA with the GP-UCB-C algorithm during two 30-minute periods, representing peak (4 PM) and off-peak

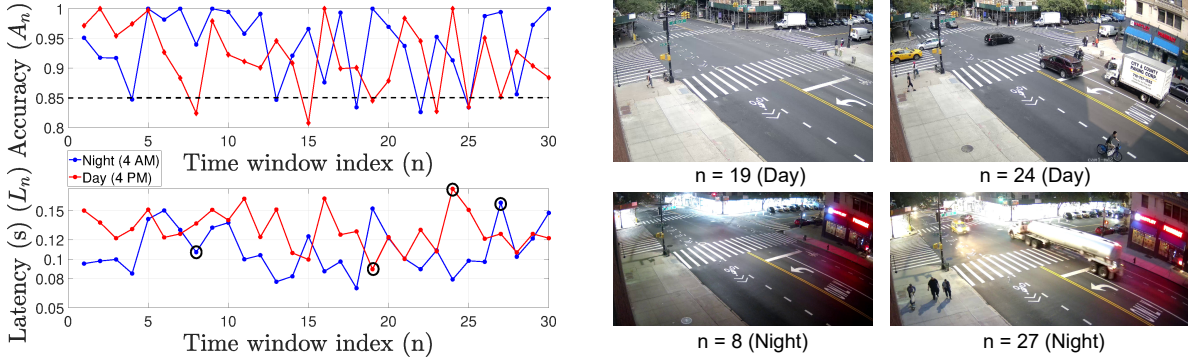


Figure 9: ASTRA's achieved average accuracy and latency during peak/day and off-peak/night.

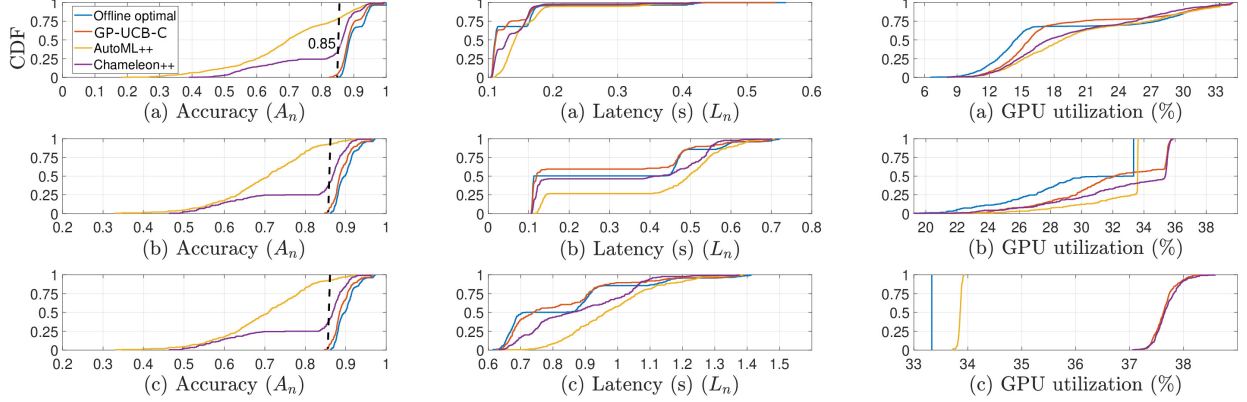


Figure 10: Empirical CDF of average achieved accuracy (with lower bound of 0.85), average achieved latency, and average GPU utilization per time window for the setting with (a) two, (b) four, and (c) eight cameras.

(4 AM) hours. Fig. 9 shows the average achieved accuracy, A_n , and latency, L_n , for each time window n . The results confirm that ASTRA consistently maintains accuracy above or close to the required threshold of **0.85** in both periods. Notably, latency is lower during off-peak hours, as reduced pedestrian and vehicle density allows the system to select lower resolutions and frame rates. To illustrate ASTRA's adaptability, we provide snapshots of the intersection at specific timestamps. At $n = 19$ (day), low traffic density allowed ASTRA to reduce the resolution/frame rate and latency while meeting the accuracy target; at $n = 24$, higher density and speed required a higher configuration to preserve accuracy. A similar pattern is observed at night ($n = 8$ and $n = 27$).

B. Controlled and Reproducible Evaluation

For a fair and reproducible comparison under identical video and network conditions, we deployed ASTRA on Google Cloud VMs and emulated multiple cameras using our comprehensive video dataset. Experiments were conducted using three VM instances hosted in the `us-east1` region (Moncks Corner, South Carolina). Two of these instances were equipped with an NVIDIA T4 GPU and were used to emulate the *real-time processing* (denoted by the light purple container in Fig. 7) and the *periodic adaptation* (denoted by the pale red container in Fig. 7). The third instance was dedicated to streaming H264-encoded videos over RTP (RTSP over TCP) using GStreamer, thereby emulating live cameras.

Video dataset. The dataset consists of videos recorded simultaneously from the two COSMOS' cameras (used in Section V-A). It includes 52 pairs of ten-minute videos (104 in total). Each pair is recorded from the two cameras simultaneously. The video pairs are captured eight times daily from morning to evening, in June, July, and Sept. 2022. The videos were purposely selected to represent a range of traffic densities and weather conditions (sunny, rainy, etc.).

GPU utilization is reported as the average over both GPU-equipped edge servers across the entire time window.

Performance comparison. Fig. 10 shows the system-level impact of increasing the number of cameras, and consequently the aggregate adaptation workload, from 2 to 4 and 8 cameras. Despite this increase, ASTRA with GP-UCB-C algorithm achieves near-optimal performance (less than 10% deviation) with GPU utilization around 2%, 4%, and 9% for 2, 4, and 8 cameras, respectively, which means $\sim 2\%$ per camera. Moreover, it satisfies the accuracy threshold of 0.85 in more than 90% of the cases. Note that an online method can occasionally appear better than the offline optimal on one individual metric (in this case, lower latency), but such cases correspond to a violation of the associated constraint (in this case, minimum accuracy of 85%) and therefore do not represent a better feasible solution to P1.

GPU utilization of ASTRA with Chameleon++ is close to GP-UCB-C. However, Chameleon++ does not meet the

specified accuracy target, $\alpha = 0.85$, in more than 30% of the times across all the evaluated scenarios. AutoML++ (GPU-UCB-C with no ground-truth) has the lowest GPU utilization as it does not require proxy ground truth. Despite having low GPU overhead, AutoML++ fails to achieve the minimum accuracy required around 75%, 85%, and 90% of times for the 2, 4, and 8 camera scenarios, respectively. This confirms the importance of incorporating proxy ground-truth for adaptation.

VI. CONCLUSIONS

We presented ASTRA, a lightweight runtime architecture for executing a broad class of iterative adaptation algorithms for real-time video analytics. ASTRA enables fast configuration adaptation with minimal overhead. Our evaluations show that ASTRA, when paired with a principled algorithm, consistently tracks near-optimal configurations and achieves performance close to the offline optimal benchmark.

ACKNOWLEDGMENT

This work was supported in part by NSF Center for Smart Streetscapes (CS3) under NSF Cooperative Agreement EEC-2133516, NSF grant CNS-2038984 and corresponding support from the Federal Highway Administration (FHWA), NSF grant CNS-2148128 and by funds from federal agency and industry partners as specified in the Resilient & Intelligent NextG Systems (RINGS) program, NSF grant CNS-2450567, and ARO grants W911NF2210031 and W911NF1910379.

REFERENCES

- [1] M. Ghasemi, S. Kleisarchaki, T. Calmant, L. Grgen, J. Ghaderi, Z. Kostic, and G. Zussman, "Demo: Real-time camera analytics for enhancing traffic intersection safety," in *Proc. ACM MobiSys*, 2022.
- [2] M. Ghasemi, S. Kleisarchaki, T. Calmant, J. Lu, S. Ojha, Z. Kostic, L. Grgen, G. Zussman, and J. Ghaderi, "Real-time multi-camera analytics for traffic information extraction and visualization," in *Proc. IEEE PerCom*, 2023.
- [3] M. Ghasemi, Y. Fu, X. Ouyang, P. Wang, M. K. Turkcan, J. Tavori, S. Kleisarchaki, T. Calmant, L. Grgen, Z. Kostic *et al.*, "Real-time video analytics for urban safety: Deployment over edge and end devices," in *Proc. ACM/IEEE SEC*, 2025.
- [4] D. Raychaudhuri, I. Seskar, G. Zussman, T. Korakis, D. Kilper, T. Chen, J. Kolodziejski, M. Sherman, Z. Kostic, X. Gu *et al.*, "Challenge: COSMOS: A city-scale programmable testbed for experimentation with advanced wireless," in *Proc. ACM MobiCom*, 2020.
- [5] M. Khani, G. Ananthanarayanan, K. Hsieh, J. Jiang, R. Netravali, Y. Shu, M. Alizadeh, and V. Bahl, "RECL: Responsive resource-efficient continuous learning for video analytics," in *Proc. USENIX NSDI*, 2023.
- [6] M. Ghasemi, Z. Kostic, G. Zussman, and J. Ghaderi, "Systems and methods for adaptive streaming for real-time video analytics (ASTRA)," U.S. Patent Application Publication US 2026/0163998 A1, Jun. 2026.
- [7] M. Ghasemi, Z. Kostic, J. Ghaderi, and G. Zussman, "ASTRA: Low-overhead runtime architecture for stream adaptation in video analytics," *arXiv preprint arXiv:2609.07020*, 2026.
- [8] COSMOS Project, "Hardware: Cameras," 2023. [Online]. Available: <https://wiki.cosmos-lab.org/wiki/Hardware/CamerasCameras>.
- [9] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karanakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *Proc. USENIX NSDI*, 2022.
- [10] M. Zhao, S. Liu, F. Wu, and G. Chen, "Responsive DNN adaptation for video analytics against environment shift via hierarchical mobile-cloud collaborations," in *Proc. ACM SenSys*, 2025.
- [11] K. Du, Y. Liu, Y. Hao, Q. Zhang, H. Wang, Y. Huang, G. Ananthanarayanan, and J. Jiang, "Oneadapt: Fast adaptation for deep learning applications via backpropagation," in *ACM SoCC*, 2023.
- [12] V. Nigade, P. Bauszat, H. Bal, and L. Wang, "Jellyfish: timely inference serving for dynamic edge networks," in *Proc. IEEE RTSS*, 2022.
- [13] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proc. ACM SIGCOMM*, 2018.
- [14] W.-J. Kim and C.-H. Youn, "Lightweight online profiling-based configuration adaptation for video analytics system in edge computing," *IEEE Access*, vol. 8, pp. 116 881–116 899, 2020.
- [15] A. Galanopoulos, J. A. Ayala-Romero, D. J. Leith, and G. Iosifidis, "Automl for video analytics with edge computing," in *Proc. IEEE INFOCOM*, 2021.
- [16] J. Yi, G. Lee, M. Jeong, S. Shin, D. Kim, and Y. Lee, "Towards end-to-end latency guarantee in MEC live video analytics with App-RAN mutual awareness," in *Proc. ACM MobiSys*, 2025.
- [17] K. Du, Q. Zhang, A. Arapin, H. Wang, Z. Xia, and J. Jiang, "Accmpeg: Optimizing video encoding for accurate video analytics," in *Proc. MLSys*, 2022.
- [18] K. Yang, M. Jeong, J. Yi, J. Lee, K. Park, and Y. Lee, "Logan: Loss-tolerant live video analytics system," in *Proc. ACM MobiCom*, 2024.
- [19] M. Zhang, J. Li, H. Zhao, L. Shen, and J. Liu, "Starstream: Live video analytics over space networking," in *Proc. ACM MM*, 2024.
- [20] M. Wong, M. Ramanujam, G. Balakrishnan, and R. Netravali, "MadEye: Boosting live video analytics accuracy with adaptive camera configurations," in *USENIX NSDI*, 2024.
- [21] Z. Li, Y. Zhu, S. Mumtaz, L. Kong, and B. Li, "RIVA: Communication-efficient streaming control for real-time industrial video analytics," *IEEE J. Sel. Areas Commun.*, vol. 43, no. 10, pp. 3548–3563, 2025.
- [22] Z. Li, M. Zhang, and Y. Zhu, "OAVS: Efficient online learning of streaming policies for drone-sourced live video analytics," in *Proc. IEEE/ACM IWQoS*, 2024.
- [23] M. Bala, A. Chanana, X. Chen, Q. Dong, T. Eiszler, J. Xu, P. Pillai, and M. Satyanarayanan, "The OODA loop of cloudlet-based autonomous drones," in *Proc. IEEE/ACM SEC*, 2024.
- [24] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proc. ACM SIGCOMM*, 2020.
- [25] S. Liu, T. Wang, J. Li, D. Sun, M. Srivastava, and T. Abdelzaher, "Adamask: Enabling machine-centric video streaming with adaptive frame masking for DNN inference offloading," in *Proc. ACM MM*, 2022.
- [26] I. Gokarn, Y. Hu, T. Abdelzaher, and A. Misra, "JIGSAW: Edge-based streaming perception over spatially overlapped multi-camera deployments," in *Proc. IEEE ICME*, 2024.
- [27] X. Hou, Y. Guan, and T. Han, "ViEdge: Video analytics on distributed edge," *ACM Trans. IOT*, 2025.
- [28] Y. Faye, F. Faticanti, S. Jain, and F. Bronzino, "VideoJam: Self-balancing architecture for live video analytics," in *Proc. IEEE/ACM SEC*, 2024.
- [29] Z. Fang, S. Hu, L. Yang, Y. Deng, X. Chen, and Y. Fang, "PIB: Prioritized information bottleneck framework for collaborative edge video analytics," in *Proc. IEEE GLOBECOM*, 2024.
- [30] H. Sun, X. Zhang, B. Zhang, K. Sha, and W. Shi, "Optimal task offloading and trajectory planning algorithms for collaborative video analytics with UAV-assisted edge in disaster rescue," *IEEE Trans. Veh. Technol.*, vol. 73, no. 5, pp. 6811–6828, 2024.
- [31] M. Zhang, F. Wang, and J. Liu, "Casva: Configuration-adaptive streaming for live video analytics," in *Proc. IEEE INFOCOM*, 2022.
- [32] W. Xu, Y. Jiang, B. Svetozarevic, and C. Jones, "Constrained efficient global optimization of expensive black-box functions," in *ICML*, 2023.
- [33] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995*, 2009.
- [34] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, and X. Liu, "From cloud to edge: a first look at public edge platforms," in *Proc. ACM IMC*, 2021.
- [35] J. Terven and D. Cordova-Esparza, "A comprehensive review of YOLO: From YOLOv1 to YOLOv8 and beyond," *arXiv preprint arXiv:2304.00501*, 2023.
- [36] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.
- [37] S. Yang, E. Bailey, Z. Yang, J. Ostrometzky, G. Zussman, I. Seskar, and Z. Kostic, "COSMOS smart intersection: Edge compute and communications for bird's eye object tracking," in *Proc. SmartEdge*, 2020.