Towards End-to-End Latency Guarantee in MEC Live Video Analytics with App-RAN Mutual Awareness

Juheon Yi
Seoul National University
johnyi0606@snu.ac.kr

Goodsol Lee
Seoul National University
gslee2@netlab.snu.ac.kr

Minkyung Jeong
Seoul National University
minkyung.jeong@hcs.snu.ac.kr

Seokgyeong Shin
Seoul National University
seokgyeong.shin@hcs.snu.ac.kr

Daehyeok Kim
The University of Texas at Austin
daehyeok@utexas.edu

Youngki Lee
Seoul National University
youngkilee@snu.ac.kr

Abstract

While mobile live video analytics apps require end-to-end latency guarantee for responsiveness and immersiveness, achieving consistent low latency is challenging due to complex fluctuations of wireless channel and scene complexity; for example, latency SLO satisfaction rate drops to as low as 26% in commercial 5G MEC platforms. Prior works mostly focus on either app-only (bitrate, DNN adaptation, or GPU allocation) or RAN-only (radio resource allocation) scheduling, with mutual ignorance of the other side resulting in mismatched scheduling decisions and frequent SLO violations. Coordinating the two schedulers is also challenging, as they are run separately by network and cloud operators with disjoint control. We present ARMA, an end-to-end live video analytics system with app-RAN mutual-awareness for high end-to-end latency SLO satisfaction in MEC. We design a mutually-aware decoupled scheduling mechanism on top of RAN Intelligent Controller (RIC) in Open-RAN architecture that fosters cooperative interaction between the two operators' schedulers while preserving operational proprietaries. We prototype an Open RAN-enabled 5G MEC testbed and evaluate ARMA, showing that ARMA achieves 97% SLO satisfaction rate.

CCS Concepts

• **Networks** → **Mobile networks**; **Cross-layer protocols**; • **Computer systems organization** → **Real-time systems**.

Keywords

Live Video Analytics, 5G, MEC, App-RAN Mutual Awareness

ACM Reference Format:
Juheon Yi, Goodsol Lee, Minkyung Jeong, Seokgyeong Shin, Daehyeok Kim, and Youngki Lee. 2025. Towards End-to-End Latency Guarantee in MEC Live Video Analytics with App-RAN Mutual Awareness. In *The 23rd Annual International Conference on Mobile Systems, Applications and Services (MobiSys '25), June 23–27, 2025, Anaheim, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3711875.3729139

1 Introduction

Live video analytics applications, such as AR/MR [52, 92, 93] or autonomous driving [55, 101], require real-time processing with strict latency constraints to ensure responsiveness. For instance, an autonomous vehicle must detect obstacles within 150 ms to avoid accidents (§2.1). Recently, Multi-access Edge Computing (MEC) architectures have emerged to enable low-latency services even for low-cost cameras without powerful on-board processors [59, 77].

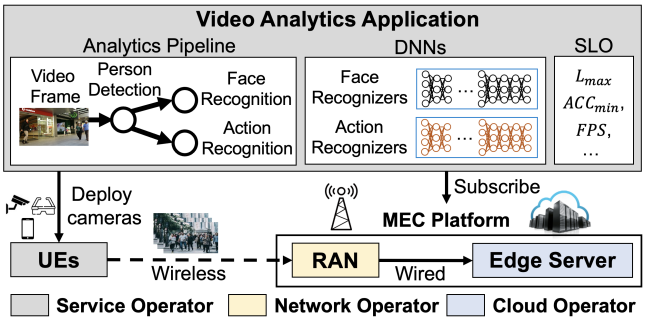


Figure 1: An example of MEC-based live video analytics.

Cellular network and cloud service providers offer such MEC platforms, including Wavelength from AWS and Verizon [80] and Azure Edge Zone from Microsoft and AT&T [8]. Figure 1 illustrates an example deployment scenario (§2.2): A police agency deploys cameras and equips officers with AR glasses for real-time criminal tracking and jaywalking detection [98]. User Equipment (UE) connects via a Radio Access Network (RAN) to an edge server located near the base station (BS), where captured videos are streamed and processed [20, 21].

However, satisfying end-to-end latency requirements—covering the entire pipeline from frame capture to processing output—is challenging. Latency fluctuations can occur both at the network stage (video streaming) and at the compute stage (Deep Neural Network (DNN) inference), due to dynamically changing wireless channel conditions and scene complexities. Our empirical measurements on commercial 5G MEC platforms show that the satisfaction rate for a 150 ms Service Level Objective (SLO) can drop as low as 26% (§2.3).

Existing approaches have designed adaptive live video analytics systems to address resource fluctuations but typically focus on either app-only (*e.g.*, bitrate adaptation, DNN model selection, or GPU scheduling [37, 40, 100]) or RAN-only solutions (*e.g.*, SLO-aware radio Resource Block (RB) allocation [44, 85]). This limitation arises because app and RAN schedulers are independently controlled by cloud and network operators, respectively, and their scheduling algorithms remain proprietary and undisclosed. Consequently, single-sided scheduling leads to sub-optimal performance (§2.4). Specifically, app-only scheduling (*e.g.*, bitrate and DNN model adjustments) frequently experiences high tail latency due to mismatches between scheduled bitrates and allocated RBs. Conversely, RAN-only scheduling, focusing on radio resource allocation, suffers from reduced throughput and fairness due to conflicting goals

between SLO satisfaction and network fairness. Moreover, simply combining app-only and RAN-only schedulers without mutual awareness also proves ineffective, as each scheduler cannot accurately estimate the other's resource availability and demands.

To overcome these challenges, we present ARMA (App-RAN Mutual Awareness), an end-to-end scheduling system designed to achieve high e2e latency SLO satisfaction in MEC live video analytics. ARMA's core is a *mutually-aware, decoupled scheduling* framework built upon the real-time RAN Intelligent Controller (RIC) within the Open-RAN architecture [63]. This framework fosters mutual awareness between app and RAN schedulers, enabling accurate tracking of each other's resource availability and demand. Consequently, this leads to more consistent scheduling decisions, satisfying stringent latency SLO requirements.

Specifically, ARMA achieves high latency SLO satisfaction through three mechanisms (§3.2): (i) *Bitrate-DNN interplay.* Leveraging recent findings that heavier DNNs can compensate for accuracy losses caused by lower bitrate and vice versa [37, 40, 78], ARMAjointly controls bitrate and DNN complexity to optimize latency SLO satisfaction under dynamically varying resource conditions. For instance, during high workloads, ARMA reduces DNN complexity to lower latency and increases bitrate to preserve accuracy. (ii) *Mutually-aware resource monitoring.* ARMA accurately tracks network and compute resource availability by enabling mutual awareness between app and RAN schedulers. Specifically, ARMA leverages physical-layer resource allocation information from RAN to improve bandwidth estimation accuracy at the application layer, facilitating better bitrate selection. (iii) *Decoupled scheduling with SLO splitting.* To ensure e2e latency SLO satisfaction despite separate schedulers, ARMA employs decoupled scheduling where app and RAN schedulers independently split the overall e2e SLO into feasible network and compute-stage deadlines. The two schedulers exchange only minimal information necessary for resource monitoring, without disclosing their scheduling algorithms.

Implementing ARMA, however, presents two significant challenges (§4.1): (i) *Coarse granularity of {bitrate,DNN} scheduling.* Adjustments to bitrate and DNN configurations occur at coarse time intervals (*e.g.,* in units of 1-2 seconds per Group of Pictures (GoP)) due to video compression efficiency constraints, making per-frame adjustments challenging amid frame-level fluctuations. (ii) *Handling sudden resource fluctuation.* Unexpected fluctuations (*e.g.,* sudden increases in workload due to user movements) inevitably introduce resource estimation errors. Decoupled schedulers following fixed SLO splits struggle to address these sudden changes effectively.

We address challenges with the following key ideas (§4.2).

- **Video-aware stochastic per-GoP scheduler** (§5). We build online stochastic models for network, compute latencies using video-specific insights (*e.g.,* scene content continuity, encoding algorithm) to account for per-frame latency fluctuation in coarse-grained, GoP-wise {bitrate,DNN} scheduling. Our insight is that the network, compute latencies can be modeled as Gaussian distributions and tracked using Kalman filters [81]. As scheduling {bitrate,DNN} configs to maximize expected latency SLO satisfaction is an NP-hard problem, we also devise an *iterative probability gradient algorithm* that computes an approximate solution with $O(MN)$ complexity for N users with M configs.
- **Mutually-compensating per-frame schedulers** (§6). Given the selected {bitrate,DNN} configs, we design RB and GPU schedulers capable of mutually compensating each other to robustly satisfy e2e SLO under resource fluctuations. Specifically, we split the e2e latency SLO into viable network, compute SLOs for the two schedulers based on expected latencies (*e.g.,* set network SLO more stringent when workload fluctuation is high). We also incorporate object-level preemption mechanism in the GPU scheduler to satisfy SLO even under unexpected latency increases due to abrupt bandwidth drop or workload increase.
- **Mutual awareness-enabling resource monitors** (§7). We design a real-time RIC for the RAN and server to exchange necessary scheduling information from their resource monitors. We design the interfaces to foster information exchange without exposing each other's proprietary scheduling algorithms. For example, ARMA's app scheduler hides e2e SLO, workload, and DNN inference latencies from RAN by obscuring them into a single network latency SLO value.

We implement ARMA on srsRAN-5G [72] with a multi-GPU edge server. Our evaluation across diverse video datasets, covering varied camera types, scene complexities, and mobility scenarios (static CCTV, vehicle dashcams, handheld cameras), demonstrates that ARMA achieves a 97% latency SLO satisfaction rate, representing up to 48% improvement over state-of-the-art baselines.

2 Motivation

2.1 Why Meeting Latency Criteria Matters?

Live video analytics apps require consistent real-time end-to-end latency, as illustrated in the following example scenarios.

- **Autonomous driving.** A vehicle moving at 20 m/s speed is running object detection to detect surrounding objects. To safely stop before crashing into a sudden obstacle appearing at 23 meters away, the end-to-end latency should be kept below 150 ms (assuming 10 m/s^2 deceleration).
- **AR person finding.** A police officer's AR glasses (60° FoV) run face detection and identification, and display results for the officer to confirm visually. When monitoring people at a 10m distance a horizontal span of 11m is covered. At this distance, an average man with a 0.4m shoulder width of appears as a 32-pixel-wide figure on a 1280×720 video frame (minimum detectable box size of RetinaNet [51]). To track a person moving at 3 m/s within a 1m area (≈10% of the FoV), latency must be <150 ms.
- **VR rendering.** A VR user's HMD captures his surroundings and streams the video to the server to detect objects and reconstruct them into 3D for rendering inside his VR space so as to grasp the existence of nearby physical objects and interact with them [69]. Consistent low latency and jitter is required for immersiveness; for example, 36 ms base motion-to-photon latency with 6 ms jitter (<42 ms 99-th tail latency) to avoid motion sickness [75].

2.2 Deployment Model

Figure 1 illustrates our deployment model, reflecting real-world deployments such as self-driving delivery robots [20] and smart crosswalks [21]. The model involves three operators: the video analytics service operator, the RAN operator, and the cloud operator.

Table 1: RTT (ms) and uplink throughput (Mbps) measurements in commercial 5G MEC platform.

	Outdoor (static)	Subway station (walking)	Train (fast moving)	Indoor (walking)
RTT ($min-max$)	24.4–39.7	27.6–86.7	27.2–42.1	25.8–53.7
tput ($mean \pm std$)	116.2±14.5	100.0±40.1	39.3±21.1	21.4±19.4

Specifically, the service operator deploys camera-equipped UEs at the target operation site and subscribes to an MEC platform (*e.g.,* AWS Wavelength [80], Azure Private MEC [8]) comprising two main components: (i) a *premium app latency guarantee plan* from the RAN operator, wherein the RAN scheduler prioritizes subscribed video analytics (VA) UEs over background non-VA UEs during radio resource (Resource Block, RB) allocation to meet the service operator-specified frame latency deadlines, and (ii) an *edge inference engine instance* from the cloud operator, offering exclusive access to a GPU-equipped edge server.

2.3 Latency in Commercial 5G MEC

We analyze end-to-end latency in commercial 5G MEC platforms, specifically focusing on uplink latency. We currently exclude downlink latency (*i.e.,* returning analysis results to UEs), as it is relatively negligible with minimal fluctuations. This is because the size of the analysis results (*e.g.,* bounding boxes) is substantially smaller than that of video frames. Moreover, typical 5G deployments allocate more bandwidth to downlink compared to uplink (*e.g.,* approximately a 3:1 ratio for the 5G DDDSU format [31]).

2.3.1 Experimental Setup

UE, 5G RAN and edge server. For the server, we use AWS EC2 g4dn.2xlarge instance equipped with an NVIDIA T4 GPU in an AWS Wavelength Zone connected to a major 5G operator's network in South Korea. The UE (Samsung Galaxy S23) streams video to the server over the operator's sub-6 GHz 5G (3.5 GHz frequency, 100 MHz bandwidth). Clocks are synchronized by exchanging timestamps using the Network Time Protocol (NTP) [62].

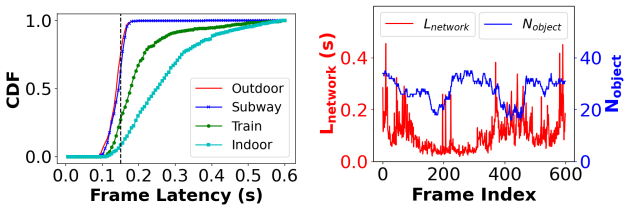
Video analytics pipeline. We assume a scenario where a police officer equipped with AR glasses is pursuing a criminal suspect through various urban environments. The analysis workload consists of a YOLOv8n [95] person detector and two ResNet-18 [34]-based analyzers for face and action recognition. The total inference latency ($L_{compute}$) scales linearly with the number of people detected (N_{object}) as follows:

$$L_{compute} = l_{detect} + N_{object} \cdot l_{analysis}, \quad (1)$$

where l_{detect} and $l_{analysis}$ denote the detection and per-object analysis latencies (6.1 ms and 1.66 ms respectively in our setup). We evaluate using the MOT17-02 [48] video dataset encoded at 1080p resolution at 30 fps and 20 Mbps bitrate.

2.3.2 Measurement Results

RTT and uplink throughput. Table 1 summarizes the Round Trip Time (RTT) and uplink throughput measured using Ping and Iperf. Results indicate strong potential for low-latency analysis, with RTT values as low as 24.4 ms[1] and throughput reaching 116.2 Mbps under favorable conditions. However, throughput significantly decreases and fluctuates in fast-moving trains and indoor environments.[2]



(a) End-to-end latency CDF. (b) Latency breakdown (*Indoor*).

Figure 2: End-to-end latency measurement results.

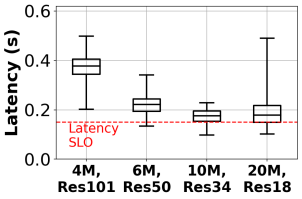


Figure 3: End-to-end latency of VideoEdge [37] (box: 25-75%, error bar: 0-95% range).

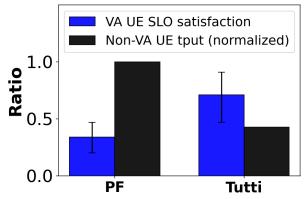


Figure 4: Throughput and network latency SLO satisfaction of Tutti [85].

End-to-end latency. Figure 2(a) shows that latency is generally below 150 ms under favorable channel conditions (*outdoor*, *subway station*), but significantly increases under bandwidth fluctuation. For instance, the SLO satisfaction rates for *train* and *indoor* environments drop to 26% and 8%, respectively, with 95th-percentile latencies exceeding 600 ms. Latency fluctuation occurs in both network and compute stages, as illustrated in Figure 2(b). (i) *Frame size and bandwidth.* The network latency $L_{network}$ is computed as:

$$L_{network} = F/BW \quad (2)$$

where F is the video frame size, and BW is the network bandwidth. The frame size varies depending on scene complexity and encoding frame type (*i.e.,* keyframe or interframe). Bandwidth also fluctuates based on the UE's location and mobility [41, 58, 94]. (ii) *DNN inference workload.* Compute latency fluctuates proportionally to the number of objects in the scene (Equation (1)), especially in dynamic environments (*e.g.,* crowded streets with fast-moving pedestrians).[3]

2.4 Limitations of Existing Approaches

Although mitigating latency fluctuations at both network and compute stages requires end-to-end scheduling across the application (UE and server) and RAN layers, prior work mostly focuses on either application- or RAN-only scheduling. They operate in a *mutually-agnostic* manner with schedulers at the other side, resulting in the following issues.

App-only scheduling suffers from long-tail latency. Recent systems adapt video encoding bitrate [22, 50, 52, 100], DNN model complexity [6, 17, 33, 47, 88], or both [37, 40], based on video content and available resources. Similarly, ML inference serving works

[1] Other measurements report even shorter RTTs of 10–20 ms [4, 85].

[2] A similar study reports throughput around 12 Mbps at speeds of 50 km/h [7].

[3] This also applies to other workloads (*e.g.,* vision-language models) as discussed in §9.

optimize GPU scheduling to ensure latency targets [18, 32]. However, they frequently exhibit poor SLO satisfaction due to mismatches between scheduled bitrates and the bandwidth allocated by the RAN scheduler, which remains unaware of application-layer frame sizes and latency SLOs. Figure 3 illustrates e2e latencies of various bitrate (4–20 Mbps) and DNN complexity configurations (ResNet18–101 [34], higher number indicates greater complexity) of VideoEdge [37] for MOT17-02 [48] video in the *train* scenario. Even the best-performing configuration ({10 Mbps, ResNet-34}) violates latency targets 77% of the time (95th-percentile latency is 0.25 s).

RAN-only scheduling suffers from throughput drop. A few recent works [44, 85] propose RAN schedulers that adjust UEs' priorities based on app-layer information (*e.g.,* flow size or latency deadlines [44, 85]), assuming either no or fixed compute-stage latency. For example, Tutti [85] increases a UE's priority based on its frame size and exponentially raises this priority as latency deadlines approach. However, RAN-only approaches suffer from reduced throughput efficiency and fairness, as UEs with urgent deadlines may not always have optimal channel conditions.

We analyze Tutti's performance [85] in a scenario with five UEs (three VA UEs and two non-VA file-transfer UEs) on our srsRAN-5G [72]-based MEC testbed (§8.1).[4] Each VA UE streams 20 Mbps video with a 100 ms network latency SLO but experiences different channel conditions due to varying mobility and UE-BS distances. Figure 4 shows the latency SLO satisfaction rates of VA UEs (bars indicate mean, error bars show min and max) and normalized throughput for non-VA UEs. Tutti achieves only a 71% latency SLO satisfaction rate for VA UEs, which, despite being higher than the 34% achieved by Proportional Fair (PF) scheduling, the de facto standard in commercial base stations [44], still remains low. Furthermore, Tutti reduces the throughput for non-VA UEs by 57%, significantly harming fairness.

Why does a simple combination of two schedulers fail? One might consider separately running the app-only and RAN-only schedulers (*e.g.,* VideoEdge [37] and Tutti [85]) to address the above challenges. However, such naive and mutually-agnostic combinations fail for two main reasons. First, the app-side scheduler incorrectly selects the video bitrate due to inaccurate estimation of the available network bandwidth, particularly the surplus capacity of the RAN (determined by the idle RBs and users' spectral efficiencies). Second, the RAN-side scheduler inaccurately sets latency SLO targets for RB scheduling, as it lacks knowledge of the compute-stage latencies at the UE and server. We elaborate on these issues in §3.2, and evaluate the performance of such naive, decoupled scheduling solutions in §8.

3 Our Approach

3.1 Goals

High latency SLO satisfaction. As motivated in §2.4, we aim at e2e scheduling of app and RAN to maximize e2e latency SLO satisfaction of VA UEs, while also maintaining high inference accuracy.

[4]Tutti [85] employs a frame-size predictor assuming fixed-bitrate, per-frame JPEG encoding, which does not apply directly to video encoding scenarios. For this experiment, we assume perfect frame-size knowledge to isolate the impact of conflicting scheduling goals.

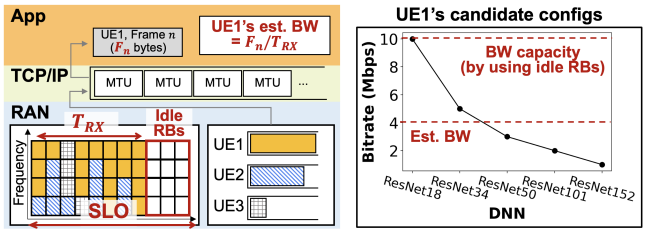


Figure 5: Conventional RAN-agnostic bandwidth estimation does not know the existence of idle RBs that can be additionally utilized within the SLO (left). Thus, it leads to bandwidth capacity underestimation (right).

App-RAN cooperation without disclosing scheduling algorithms. While coordinating the app and RAN schedulers for e2e latency SLO satisfaction, we aim to avoid the two schedulers from disclosing their proprietary scheduling algorithms to each other (*e.g.,* channel estimation algorithm of RAN operator, DNN models and GPU scheduler algorithm for service operator), so as to make our system practically deployable in split operator settings.

Compatibility. For wide applicability to commercial servers with various video codecs, DNN inference engines, and hardware, we control only the video encoding bitrate, DNN model complexity,[5] and GPU access timing for resource scheduling. We also aim to minimize RAN deployment overhead by designing standard-compliant system without modifications of proprietary UE cellular firmware.

3.2 Mutually-aware Decoupled Scheduling

Our approach is to foster mutual awareness between the app and RAN schedulers, so as to enable each scheduler to accurately track the resource availability and demand of the other, and make consistent scheduling decisions to satisfy the e2e latency requirement. We realize this approach using the real-time RAN Intelligent Controller (RIC) in O-RAN architecture [63], which are actively being designed to support <1 ms monitor/control delay [19, 27, 45]. Specifically, we achieve low e2e latency with the following mechanisms:

- **Bitrate-DNN interplay**. We jointly control the video encoding bitrate and DNN complexity under dynamic resources to maximize latency SLO satisfaction. For example, when compute workload is high, we use lighter DNN to reduce compute latency and increase bitrate to maintain the accuracy. Bitrate-DNN interplay is possible due to two reasons. First, heavier DNN with more layers and channels can compensate the accuracy loss from low bitrate (and vice versa), as empirically observed in several prior studies [37, 40, 78] (*e.g.,* EfficientNet [78]'s principles of scaling input resolution and DNN's layers and filters in tandem). Second, network and compute latency fluctuates with low correlation due to independence of scene complexity and wireless channel. For example, Pearson correlation of the two latency timelines in Figure 2(b) is -0.09.
- **Mutually-aware app, RAN resource monitoring**. The key to bitrate-DNN interplay is accurately estimating the resource demands and availabilities of both network and compute stages, so as to accurately track the selectable {bitrate, DNN} configs

[5]DNNs for various tasks commonly support multiple backbones with wide latency-accuracy tradeoffs (*e.g.,* ResNet 18-152, YOLOv5 n-x, EfficientNet B0- B7) [14, 79, 95].

given current resources. For this, we enable mutual awareness in app, RAN resource monitoring; especially, we leverage RAN's physical layer RB allocation information for network bandwidth estimation. Figure 5 illustrates the need for RAN-awareness: conventional packet arrival timestamp-based bandwidth estimation suffers from bandwidth underestimation and ignorance of the available bandwidth capacity from leveraging additional idle RBs (refer to §7.2 for details).

- **Decoupled scheduling with SLO splitting**. To enable e2e latency SLO satisfaction over split schedulers with operational proprietaries, we take a decoupled scheduling approach, where the app and RAN schedulers split the e2e SLO into viable network, compute SLOs that each can satisfy. The two schedulers only exchange the SLO and minimal information for resource estimation without disclosing their scheduling algorithms.

4 ARMA Design Overview

4.1 Challenges

Coarse granularity of {bitrate,DNN} scheduling. Video bitrate (and {bitrate,DNN} config accordingly) can be scheduled in coarse-grained time period (*e.g.,* in units of 1-2s Group of Pictures, or GoP) and makes it difficult to accurately adapt to frame-level fluctuations of channel status, frame size and scene complexity. This is due to two reasons. First, it is difficult to precisely control the size of each frame, especially in dynamic scenes, as the bitrate is typically configured as an average over a time window (GoP). Second, an intra-coded keyframe, which is much larger than inter-frames, must be inserted whenever the bitrate changes. As a result, frequent bitrate changes lead to excessive network resource utilization. For example, changing the bitrate every 500 ms in H.264 encoding using FFmpeg results in a 59% error between the target and actual encoded bitrate, as well as a 49% higher overall average bitrate. Thus, prior works [40, 100] mostly aim at optimizing average throughput and predict latency as a single average value, thus failing to consider such per-frame fluctuations.

Handling sudden resource fluctuation. Despite mutually-aware resource estimation, error from sudden, unpredictable fluctuation is inevitable (*e.g.,* abrupt workload increase due to user's head rotation to another view). With naive decoupled schedulers which follow only given split SLO, it is hard to handle such sudden fluctuations.

4.2 System Architecture

Figure 6 shows the system architecture of ARMA. It is composed of *data flow* (UEs streaming the encoded frames to the server for DNN inference) and *control flow* (monitoring network and compute resources, scheduling {bitrate,DNN} configs, and allocating RB and GPU resources). ARMA consists of the following components, which operate in the order depicted in Figure 7.

Video-aware stochastic per-GoP scheduler (§5). To account for per-frame latency fluctuations in coarse-grained {bitrate, DNN} scheduling, we build online stochastic models (§5.1) for network, compute latencies using video-specific insights (*e.g.,* scene content continuity, video encoding algorithm). Leveraging the predicted latency distributions, *RAN-aware {bitrate,DNN} scheduler* (§5.2) selects {bitrate,DNN} configs across UEs in units of GoP to maximize

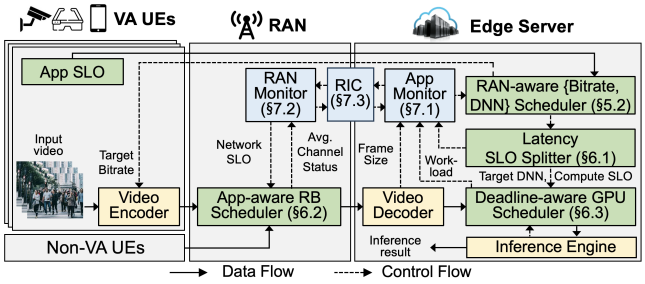


Figure 6: ARMA **system architecture (Yellow: existing video analytics pipeline, Green:** ARMA**'s resource schedulers, Blue:** ARMA**'s resource monitors).**

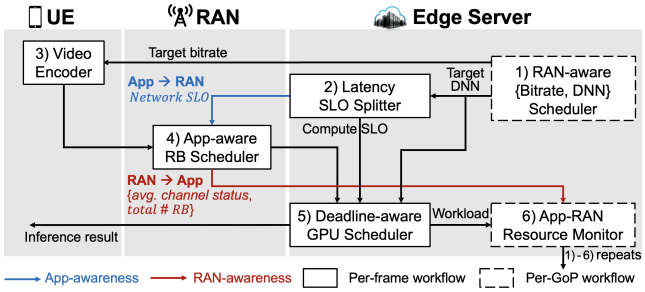


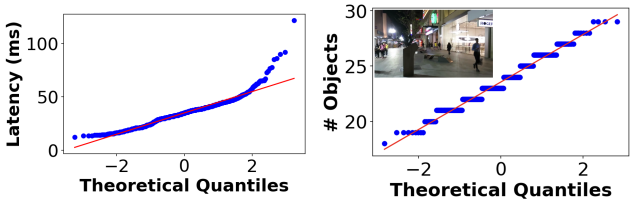
Figure 7: Mutually-aware decoupled scheduling workflow.

the overall expected latency SLO satisfaction. For example, when two candidate {bitrate,DNN} configs yield the same average latency, choose the one with smaller variance.

Mutually-compensating per-frame schedulers (§6). Given the selected {bitrate,DNN} configs, RB and GPU schedulers operate in a mutually compensating manner to satisfy e2e SLO even under sudden resource fluctuations. First, *latency SLO splitter* (§6.1) splits e2e SLO across the RB and GPU schedulers for each UE considering its predicted network, compute latencies; for example, it sets a stringent network SLO for a UE with high inference workload. Once the UEs encode their video frames at the target bitrate, *App-aware RB scheduler* (§6.2) allocates RBs in units of Transmission Time Interval (TTI)[6] to stream the frames within the splitted network SLO while also balancing the total RAN throughout and fairness with background non-VA UEs. Upon receiving the frames, *deadline-aware GPU scheduler* (§6.3) schedules GPU to finish DNN inference within the e2e SLO. To robustly satisfy the SLO even under sudden latency increase (*e.g.,* due to network bandwidth drop or abrupt scene change), it incorporates a preemption mechanism to accelerate the inference execution of a UE that is behind schedule by temporarily preempting other UEs who need fewer GPUs than expected.

Mutual awareness-enabling resource monitors (§7). *App, RAN resource monitors* (§7.1, §7.2) profile the resource demands and availability of network and compute stages, and deliver necessary scheduling information to the per-GoP and per-frame schedulers. *Mutual awareness-enabling RIC* (§7.3) enables the app and RAN schedulers to exchange resource information without disclosing their proprietary scheduling algorithms. For example, we hide app-side e2e SLO, workload, and DNN inference latencies from RAN by obscuring them into a single network SLO.

[6]Time duration of an RB in RAN (*e.g.,* 0.5 ms for 5G with 30 kHz sub-carrier spacing).



(a) Network latency in *subway* environment in §2.3.

(b) Compute workload (MOT17 [48]-10 video).

Figure 8: Q-Q plot of network latency and workload against normal distribution. Points aligning along the red line ($y = x$) indicates that the two distributions are similar.

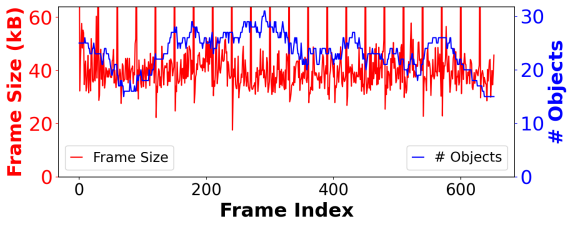


Figure 9: Frame size and workload timeline example (MOT17-10). Pearson correlation coefficient: 0.02.

5 Per-GoP Scheduling

In this section, we detail ARMA's per-GoP scheduling pipeline. We first build online stochastic latency models to predict latency distributions of network and compute stages over the next GoP (§5.1). We leverage the models and RAN-informed network resource availability to schedule {bitrate, DNN} configs to maximize overall expected latency SLO satisfaction (§5.2). Lastly, we interleave large keyframes across UEs to avoid network bottleneck (§5.3).

5.1 Online Stochastic Latency Modeling

Modeling unit. For each UE i, we model network transmission latency at 1 Mbps encoding bitrate ($L_{i,1\text{Mbps}}$) and workload (n_i) as independent Gaussian distributions as follows,

$$L_{i,1\text{Mbps}} \sim \mathcal{N}(\mu_{i,1}, \sigma_{i,1}^2), \quad n_i \sim \mathcal{N}(\mu_{i,2}, \sigma_{i,2}^2). \tag{3}$$

The rationales behind our modeling are as follows.

- $L_{\mathbf{1Mbps}}$**.** Network latency is determined as the frame size divided by bandwidth. Video-encoded frame is composed of motion vectors and residual data (*i.e.,* difference between the predicted and actual pixel values), where the residual typically dominates the frame size (*e.g.,* empirically >70% for H.264). As residual data are by nature random signals, it can be modeled as Gaussian distribution; for example, Shapiro-Wilk test [70] (used for validating whether a distribution is Gaussian) yields >0.96 score for various videos (§8.1). Bandwidth remains stable within a short GoP window (*e.g.,* 1s) as channel fluctuations are mitigated by the quantization in MCS mapping table (*e.g.,* in 2 dB steps [9]). Thus, network latency can be modeled as Gaussian. For example, Q-Q plot (scatter plot of sorted values of the two distributions) in Figure 8(a) shows that our measured latency distribution in §2.3 are overall well-aligned with normal distribution (modeling error handling for $> 2\sigma$ tail is detailed shortly).
- $\boldsymbol{n_i}$**.** Workload can also be modeled as Gaussian (Figure 8(b)), characterized by mean (average scene complexity) and variance (how fast objects newly appear and leave).
- **Independence.** We observe $L_{1\text{Mbps}}$ and $\boldsymbol{n_i}$ mostly have low correlation, as (i) scene content channel status are independent, and (ii) albeit objects leaving and disappearing affects the size of residual data, its impact is negligible than other scene changes (*e.g.,* lighting condition, object shape changes). For example, Figure 9 shows that frame size and workload have a Pearson correlation coefficient of only 0.02.

E2E latency modeling. Let $L_{i,k}$ denote the e2e latency of the i-th UE selecting using the k-th {bitrate, DNN} config $\{b_k, l_k\}$. Using the modeling units $L_{i,1\text{Mbps}}$ and n_i, we model $L_{i,k}$ as the sum of network and compute latencies $L_{i,net}$ and $L_{i,comp}$ as follows,

$$\begin{aligned} L_{i,k} &= L_{i,net} + L_{i,comp} = b_k \cdot L_{i,1\text{Mbps}} + l_0 + l_k \cdot n_i \\ &\sim \mathcal{N}\left(b_k \cdot \mu_{i,1} + l_0 + l_k \cdot \mu_{i,2}, b_k^2 \cdot \sigma_{i,1}^2 + l_k^2 \sigma_{i,2}^2\right), \end{aligned} \tag{4}$$

where l_0 is the detector inference latency. We model $L_{b\text{Mbps}} = b \cdot L_{1\text{Mbps}}$ from two observations. First, frame size is linearly proportional to encoding bitrate, as pixel quantization is a linear operation. Second, transmission latency typically dominates frame latency; initial uplink grant delay is typically negligible, especially considering recent proactive grant (pre-allocating UL resource before receiving Buffer Status Report (BSR)) for mobile devices [91].[7]

Model tracking at runtime. Latency models are updated per each frame by monitoring the encoded frame size, RAN-monitored bandwidth (§7.3), and the detected number of objects. We update use Kalman filter [81] for parameter update, which is lightweight and good at tracking time series data with Gaussian distributions.

Robustness to modeling errors. Modeling errors occur under sudden bandwidth/scene changes (*e.g.,* when a user abruptly rotates their head to view a new scene). However, they often have only a short impact (*e.g.,* each inter-frame is typically encoded by referencing only to the previous frame). Latency fluctuations from these errors are mitigated by preemptive GPU scheduling (§6.3).

5.2 RAN-aware {Bitrate,DNN} Scheduler

Using the latency models in §5.1, our goal is to select {bitrate, DNN} configs across UEs to maximize the expected overall latency SLO satisfaction (*i.e.,* $\prod_i P(L_{i,k} \le L_{\max})$) while satisfying the network and compute resource budgets. This is formulated as follows. Assume UE i $(= 1, ..., N)$ streams f fps video with workload model n_i and spectral efficiency R_i (Mbps per RB), which is shared from RAN (§7.3). UE i has K_i pareto-optimal accuracy-satisfying {bitrate, DNN} configs $C_{i,k} = \{b_{i,k}, l_{i,k}\}$ from the accuracy profiler (§7.1.4), where $j \in [1, K_i]$ and $b_{i,k}$ and $l_{i,k}$ are the encoding bitrates and inference latencies of the DNNs, respectively. We find an allocation $K^* = \{k_1^*, ..., k_N^*\}$ for the next GoP which minimizes the overall SLO satisfaction of $L_{i,k}$, by solving an one-step MPC problem[8]

$$\begin{aligned} \max \quad & \textstyle\prod_i P(L_{i,k} \le L_{\max}) \\ \text{s.t.} \quad & \textstyle\sum_i \left(b_{i,k_i}/R_i\right) \le N_{RB} \\ & \textstyle\sum_i \text{mean}(n_i) \cdot f \cdot l_{i,k_i} \le N_{GPU} \cdot T_{util} \quad \forall i \end{aligned} \tag{5}$$

[7] ARMA can also leverage the periodicity of video frames to reduce initial resource grant delay, which we leave as future work.

[8] Note that this yields the same solution as in conventional MPC with receding horizon control (optimizing a sequence of actions over a window), as our video analytics workload is GoP-wise independent.

where N_{RB} and N_{GPU} are the number of RBs in frequency axis per each TTI and GPUs, and T_{util} is the available GPU utilization time per each second (1 for dedicated edge server). Constraints specify that the average resource usage does not exceed the budgets.

Iterative probability gradient algorithm. Above scheduling is a mixed integer programming problem (NP-Hard). We design an *iterative probability gradient algorithm* that operates as follows. First, it chooses $(b_{i,k}, l_{i,k})$ per each UE that maximizes $P(L_{i,k} \leq L_{\max})$. Then, it iteratively adjusts allocations if bottleneck occurs. For example, if the sum of allocated bitrates exceeds the RAN's RB budget, we iteratively select the UE with the *minimum probability gradient* and adjust its config by a step (in the direction of reducing the bitrate and increasing the DNN) until the bottleneck resolves. The probability gradient of UE i (denoted as PG_i) with the currently selected config index k is defined as the

$$PG_i = |P(L_{i,k} \leq L_{\max}) - P(L_{i,k+1} \leq L_{\max})|, \tag{6}$$

assuming $\{(b_{i,j}, l_{i,j})\}$ is sorted in the ascending order of l. This lightweight heuristic efficiently finds an approximate solution in $O(MN)$ overhead for N UEs with M candidate configs; for example, it takes only <50 μs for $(M, N) = (5, 100)$ in our testbed (§8.1).

5.3 Inter-UE Keyframe Interleaving

Keyframe (which appears once at the beginning of each GoP) is encoded independently of others, and thus has higher size than inter-frames with temporal encoding (*e.g.,* 3-5×). We interleave keyframe offsets across UEs to avoid network bottleneck caused by simultaneous keyframe transmissions. We synchronize UEs' video capture timing and configure i-th UE's keyframe offset $offset_i$ as

$$offset_i = i \cdot \lfloor fps \cdot GoP / N_{UEs} \rfloor, \tag{7}$$

where fps is frame rate (uniform across UEs) and GoP is the GoP size (s), and N_{UEs} is the number of UEs.

6 Per-Frame Scheduling

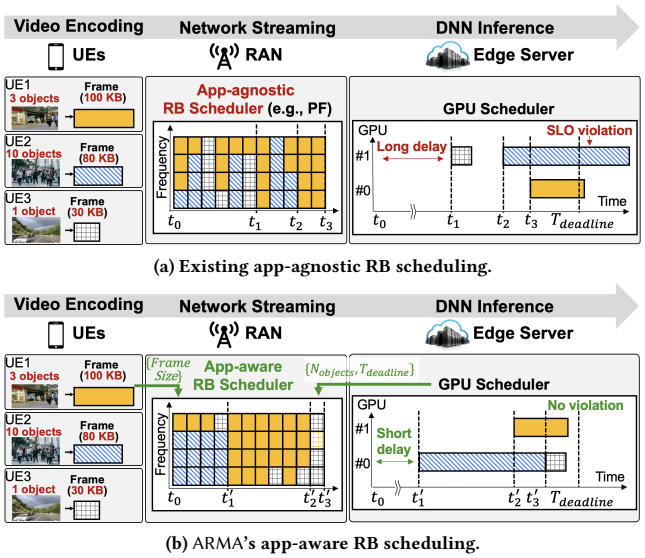
In this section, we detail ARMA's per-frame scheduling pipeline to satisfy the e2e latency SLO given the selected {bitrate, DNN} configs from per-GoP scheduler. *App-aware SLO splitter* first divides the e2e latency SLO across network and compute stages (§6.1) for each UE considering its expected latency distributions. App-aware RB scheduler (§6.2) and deadline-aware GPU scheduler (§6.3) allocate RBs and GPUs to satisfy the splitted SLOs.

6.1 Latency SLO Splitter

Upon each UE's latency model update, we split the UE's e2e latency SLO L_{max} into network and compute stage SLOs considering the expected latency distributions. Assume that the i-th UE has latency models $L_{i,net} \sim \mathcal{N}(\mu_{i,net}, \sigma^2_{i,net})$ and $L_{i,comp} \sim \mathcal{N}(\mu_{i,comp}, \sigma^2_{i,comp})$ as a result of the {bitrate, DNN} scheduling decision. We split network and compute latency SLOs $L^{max}_{i,net}$ and $L^{max}_{i,comp}$ as follows,

$$\begin{aligned} L^{max}_{i,net} &= L_{max} - \mu_{i,comp} - k \cdot \sigma_{i,comp}, \\ L^{max}_{i,comp} &= L_{max} - L^{max}_{i,net}. \end{aligned} \tag{8}$$

Higher k sets a more stringent network latency SLO to leave scheduling margin for unexpected workload increase, but may potentially hurt the RAN's throughput as RBs may be allocated to the UE even under bad channel. We empirically set k as 1.64, which corresponds to the 95% cumulative probability in the Gaussian distribution.



(a) Existing app-agnostic RB scheduling.

(b) ARMA's app-aware RB scheduling.

Figure 10: App-agnostic vs. -aware RB scheduling for 3 UE scenario. Leveraging frame size and workload information (highlighted as green), app-aware RB scheduling can prioritize UE2 to improve latency SLO satisfaction.

6.2 App-aware RB Scheduler

App-aware RB scheduler allocates RBs across UEs to stream their encoded frames to the server within the splitted network SLO. Figure 10 shows a three-UE scenario illustrating the benefit of app-awareness. Consider three UEs with different frame sizes (*e.g.,* intra- or inter-predicted) and compute workloads. App-agnostic scheduling (*e.g.,* TTI-level proportional fairness) results in a long makespan and SLO violation due to the ignorance on the frame size and workload (Figure 10(a)). Conversely, app-aware RB scheduling can prioritize latency-stringent UE 2 first, and reduce makespan by allocating RBs based on the granularity of app-layer frames.

Despite the opportunities, app-aware RB scheduling incurs non-trivial challenges. (i) *Conflicting goals.* Simply prioritizing UE with the most urgent latency deadline can hurt RAN resource efficiency, as it may not always be experiencing the best channel (Figure 4). (ii) *Fairness.* Prioritizing latency-critical Video Analytics (VA) UEs may hurt fairness with non-VA UEs. Few recent studies designed app-aware RB schedulers [9, 16, 44, 85, 97], but they mostly guarantee average throughput or minimize flow completion time and do not consider latency SLO satisfaction (§10).

We design a hierarchical two-stage scheduler to tackle the challenges. First, the outer scheduler allocates RBs across the VA and non-VA UE groups same as the PF scheduler The inner scheduler opportunistically reassigns RBs within the VA group to optimize latency SLO satisfaction without significantly impacting the outer scheduler's decision. Algorithm 1 shows the scheduling algorithm. The scheduler is triggered per each TTI. For each RB, we first calculate the PF coefficients of all UEs, and selects the UE with the maximum value (lines 2-4). If the selected UE u^* belongs to non-VA UE group, the RB is allocated directly allocated to it (lines 5-6). If otherwise, the inner scheduler identifies a sub-group of UEs who have similar PF coefficient (within 1-ϵ range) with u^* (line 8), and

Algorithm 1 Hierarchical RB scheduler

```
1:  procedure RB Scheduling():
2:      for each RB b do
3:          calculate PF coefficient p_{u,b} per each UE u
4:          select UE u* with the highest PF coefficient p*_{u,b}
5:          if u* ∈ non-VA UEs then
6:              allocate b to u*
7:          else if u* ∈ VA UEs then
8:              find VA UEs U' s.t. p_{u,b} ≥ (1 − ϵ) · p*_{u,b}
9:              select UE u** ∈ U' with maximum latency utility V_u,
10:             allocate b to u**
```

allocating the RB to a UE with the maximum latency utility V_u (lines 9-10) which is defined as,

$$V_u = R_{u,b} \cdot (F_u/B_u)^{\alpha} / (\max(D_u, \delta))^{\beta}, \tag{9}$$

where $R_{u,b}$ is UE u's spectral efficiency for RB b, F_u and B_u are frame size and remaining data size in buffer, D_u is the time remaining until the latency deadline (δ is a small constant to avoid negative values when latency exceeds deadline). This prioritizes UE with (i) favorable channel status, (ii) frame transmission almost finished ($B_u \approx 0$), and (iii) close deadline.

Scheduling overhead. Our hierarchical RB scheduling requires one additional iteration across VA UEs to track the latency utility. This keeps the scheduling complexity at $O(N)$ for N UEs which is the same as the original PF scheduler. It also incurs negligible CPU and memory overhead (§8.3.3).

Coexistence with non-VA UEs. Higher ϵ increases the chance that the inner scheduler selects another VA UE i' with higher $V_{i'}$ instead of VA UE i with the highest PF coefficient. As UE i' is likely to have the highest coefficient in the next TTI (assuming the channel status remains similar), it may temporarily affect fairness with non-VA UEs; however, average throughput is less affected, as non-VA UEs take more proportion after VA UEs finish their frames. We empirically set ϵ as 0.2 based on our evaluation, which imposes a negligible drop in fairness of non-VA UEs (§8.3.2).

6.3 Deadline-aware GPU Scheduler

Finally, GPU scheduler grants GPUs to UEs to complete the inference requests within the e2e latency SLO. To guarantee latency under resource estimation errors (*e.g.,* due to sudden drop in bandwidth or increase in workload), it incorporates a video analytics-aware, object-level preemption mechanism. It allows the scheduler to prioritize a UE's frame that needs more GPU resources than predicted by temporarily pausing the processing of other UEs' frames that need fewer resources than expected.

Basic operation. There are two schedulers: *inter-GPU* and *intra-GPU*. When a new frame arrives, inter-GPU scheduler forwards it to a GPU with the least load in its queue. For each GPU, intra-GPU scheduler chooses a frame with the least slack time (*i.e.,* minimum margin until the deadline after DNN inference) for processing.

Object-level preemption mechanism. When the intra-GPU scheduler detects a frame request whose deadline cannot be met, it searches for opportunities to offload the excess object recognition inference to other GPUs. Specifically, we determine that a GPU g can be preempted by a margin δ if all frame requests in its queue Q_g can be delayed by δ without missing their deadlines. It iterates over all GPUs to calculate the sum of the margins and preempts the GPUs if a set of GPUs capable of handling the excess inference workload is found. Otherwise, we drop the frame and interpolate its inference results with the most recent one from the corresponding UE. For this, each intra-GPU scheduler manages a preemptive inference queue, which are processed with higher priority than normal frame requests from the inter-GPU scheduler.

7 Mutually-aware Resource Monitoring

In this section, we detail components to gather scheduling information for ARMA's mutually-aware decoupled scheduling in §5 and §6. Specifically, app and RAN resource monitors (§7.1, §7.2) profile resource demands and availability for the video streaming and DNN inference stages, and exchange the information through mutual awareness-enabling RIC (§7.3).

7.1 App Resource Monitor

7.1.1 *Network Demand Monitor*

Network demand monitor notifies UEs' frame sizes to the RAN for app-aware RB scheduling. For this, each UE sends its encoded frame size in the app packet header prior to frame transmission, which is forwarded to the RAN. This approach does not require frame size prediction process as in Tutti [85] (which incurs estimation error and overhead), but incurs a notification delay: RAN does not know the frame size until the app header arrives (*e.g.,* $\approx RTT/2$=5-15 ms). We mitigate this by also notifying the RAN which frame type to expect next (§7.3), so that the RAN roughly estimates the incoming frame's size from its type before receiving the actual value.

Cross-layer traffic conversion. For each UE, the server converts its app layer frame size F_{App} to PHY layer size F_{PHY} as follows,

$$F_{PHY} = (F_{APP} + h) \cdot (1 + \epsilon), \tag{10}$$

where h is app layer packet header size, ϵ is the transport to physical layer protocol overhead (0.068 for 5G [84] assuming no retransmission). For each UE's frame with size F_{PHY}, RAN tracks the amount of data size remaining for transmission $F_{remaining}$ as

$$F_{remaining} = F_{PHY} - \sum_{t=t_{start}}^{t_{now}} TBS_t \cdot (1 - OH), \tag{11}$$

where t_{start} is the frame capture time, TBS_t is the allocated Transport Block Size (TBS) at time t (determined by the number of allocated RBs and MCS), and OH is the physical layer overhead (e.g., 0.08 for UL in the FR1 frequency range [3]).

7.1.2 *Compute Demand Monitor*

DNN inference latency. The inference latencies of all candidate DNNs are profiled offline, stored as a look-up table, and updated online upon each inference completion. To improve latency predictability, we make the following two system assumptions. (i) We do not use multithreading within each GPU to avoid tail latency increase (which could be as high as ≈100× [32]). (ii) We assume that all DNNs are loaded onto the GPU in advance, considering practical GPU memory sizes (*e.g.,* 48 GB for RTX A6000 vs. 4.9 GB for ResNet 18-152 weights and features with batch size 8). We discuss the generality and limitations of these assumptions in §9.2.

Workload. The number of objects per frame is also tracked to compute the latency modeling and prediction (§5.1).

```
struct frame_info{
    int UE_id
    int frame_id;
    int frame_size;
    int64_t deadline;
    uint8_t keyframe_interval;
    uint8_t keyframe_offset;
};
```

Figure 11: App→RAN RIC interface.

```
struct RB_sched_info{
    int N_total_RBs;
    vector<ue_stats> stat;
};
struct ue_stats{
    double TBS;
    int num_RBs;
};
```

Figure 12: RAN→App RIC interface.

7.1.3 *Compute Availability Monitor*

Compute availability monitoring is simplified in our single, dedicated edge server deployment model. However, GPU availability must be monitored in runtime for real deployment scenarios with competing background tasks (§9.2).

7.1.4 *{Bitrate, DNN} Accuracy Profiler*

Accuracy profiler provides the config scheduler (§5.2), which {bitrate, DNN} configs satisfy the accuracy requirements for each UE. As the accuracy-satisfying configs change over time due to dynamic scene content, online profiling is required. We leverage prior works for low-overhead profiling, composed of (i) *lightweight scene change detector* [38, 50, 52] to trigger profiling only upon significant content change, and (ii) *config accuracy interpolator* [40] which profiles each bitrate, DNN value axis with the remaining config value fixed, and interpolate the accuracies of the remaining configs.

7.2 RAN Resource Monitor

The RAN resource monitor's goal is to provide the app scheduler with an accurate estimate of UEs' bandwidth capacities (*i.e.,* how much the bitrates can be increased by using additional RBs). For this, two information are required: (i) how many RBs are available, and (ii) how much throughput each UE can achieve per each allocated RB. These cannot be achieved from conventional packet size and arrival timestamp-based bandwidth estimation in video streaming/analytics systems like WebRTC [13, 100], whose primary goal is conservatively reducing the bitrate to avoid congestion (as opposed to ARMA's bitrate-DNN interplay which increases bitrate by leveraging surplus RBs when workload is high). Figure 5 illustrates this: estimating a UE's bandwidth as frame size divided by reception time (F_n/T_{RX}) suffers from two problems: (i) it cannot estimate per-RB TBS, and (ii) it cannot know the *remaining* idle RBs, which could be additionally used within the latency SLO. Consequently, it results in a severe underestimation of each UE's maximum selectable bitrate.

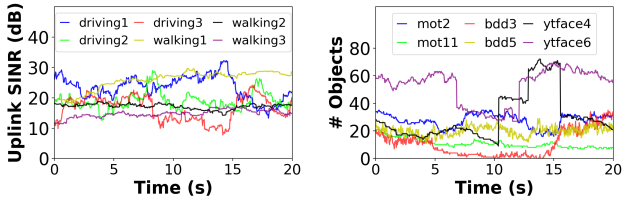
To overcome the limitation, we monitor (i) the total number of RBs, and (ii) number of allocated RBs per UE and achieved TBS to improve bandwidth estimation accuracy as shown in Figure 5(b). Using this, we calculate UE i's app layer spectral efficiency using cross-layer traffic conversion similar to Equations (10) and (11).

7.3 Mutual awareness-enabling RIC

RAN and server mutually exchange monitored resource information (§7.1, §7.2) and scheduling decisions (*e.g.,* latency deadline §6.1) over RIC. ARMA's RIC is implemented atop EdgeRIC [45], a real time co-located on the Distributed Unit (DU) of RAN for <1 ms monitoring/control delay. The app-RAN message exchange interfaces are implemented using O-RAN E2 [24]'s service models (SMs),

Table 2: Evaluation datasets.

	# Videos	Camera	Content	Workload ($\mu \pm \sigma$)	Mobility
MOT [48]	5	CCTV	Street	18.3±8.5	Static
BDD [96]	10	Dashcam	Driving	13.3±8.2	Driving
YTFace [82]	6	Mobile	Street	32.3±25.7	Walking



(a) Uplink SINR. **(b) Workload.**

Figure 13: Uplink SINR and workload trace samples.

which defines functionalities to monitor/control RAN. Specifically, we build our interfaces atop KPM (Key Performance Metric) [46] and RC (RAN Control) [68] SMs.

App→RAN. Each UE's frame size, index information (to estimate next frame size until the information arrives), and transmission deadline are shared from server to RAN using message format in Figure 11. It does not reveal any app information (*i.e.,* e2e latency SLO, DNN inference latency, and GPU scheduling algorithm) as they are all obscured as a single network latency deadline value.

RAN→App. Total number of RBs and UEs' spectral efficiencies (required for {bitrate, DNN} scheduling in §5.2) are shared to the server using message format in Figure 12. It discloses only the RB scheduling results (# RBs and achieved TBS), hiding channel state estimation, MCS selection, and RB scheduling algorithms.

8 Evaluation

8.1 Setup

Over-the-air testbed. We evaluate ARMA on a 5G RAN-enabled edge testbed (Figure 14). It consists of a commodity desktop equipped with Intel i9 CPU where we run srsRAN-5G [72] for BS, and a server (located in the same building) equipped with Intel Xeon Gold 5128 CPU and 8× RTX 2080 Ti GPUs to run video analytics apps. As an RF frontend, we use USRP X310 (TDD n78 band with 5DDDSU and single MIMO layer following [31, 44]) with Precision GPS Reference Clock [30] connected to the BS. For UEs, we use Google Pixel 6a smartphones with programmable SIM cards (sysmoISIM-SJA2) [76]. We modify EdgeRIC [45] to support ARMA's mutual awareness-enabling RIC (§7.3). We use H.264 in FFmpeg [25] 4.1.9 for video encoding and secure reliable transport (SRT) [74] for streaming. For DNN inference, we use PyTorch 1.10.1 [67] C++ API.

Trace-driven simulator. For repeatable experiments with more UEs, we use srsRAN-5G RF simulator [73], which is implemented using ZeroMQ [99] and GNURadio [29] to emulate wireless communication channel. We generate UL SINR trace using 3GPP channel model [2] for various UE-BS distance and mobility scenarios (static, walking at 1.4 m/s, driving at 35-90 km/h) as shown in Figure 13(a).

Video datasets and DNNs. We evaluate ARMA using multi-object tracking datasets with various scene complexity and change speed

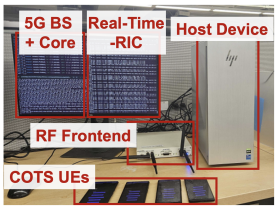


Figure 14: MEC testbed implementation.

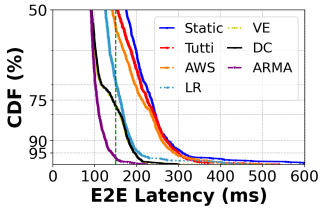


Figure 15: End-to-end latency comparison with baselines.

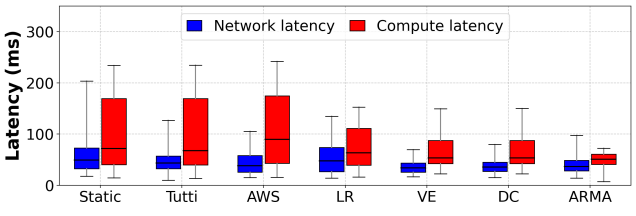


Figure 16: Latency breakdown (box: 25-75%, error bar: 0-95%).

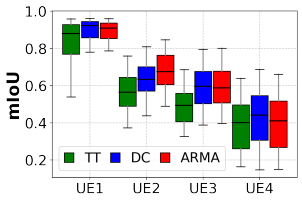


Figure 17: Streaming accuracy comparison.

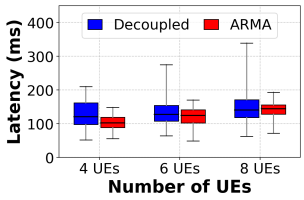
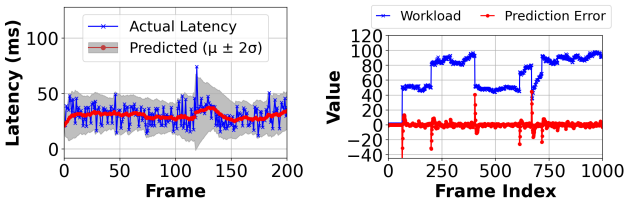


Figure 18: Latency for varying number of UEs.



(a) Network latency.

(b) Compute workload.

Figure 19: Stochastic latency modeling accuracy.

as shown in Table 2 and Figure 13(b). We sample different number of UEs (4–10) across datasets to create multi-UE scenarios. For DNNs, we use YOLOv8n detector and ResNet 18-152 [34] classifiers.

Baselines. ***Static*** uses static {bitrate, DNN} config and the PF scheduler in srsRAN-5G. ***LiteReconfig (LR)*** [87] is an app-only scheduling baseline which adapts DNN configs (with static bitrate) based on predicted scene content difficulty (*e.g.,* # objects, bounding box sizes and confidences). ***AWStream (AWS)*** [100] adapts bitrate configs (with static DNN) based on the available network bandwidth. ***VideoEdge (VE)*** [40] adapts {bitrate,DNN} config based on RAN-agnostic network bandwidth prediction and average compute workload, without latency modeling. ***Tutti*** [85] is a RAN-only scheduling baseline which adjusts UEs' priorities based on their predicted frame sizes and network latency deadlines (set as half of the e2e SLO). ***Decoupled (DC)*** runs VideoEdge [40] and Tutti [85] separately without mutual awareness (*i.e.,* no RAN-aware bandwidth estimation and latency SLO splitting). It also runs round robin GPU scheduler, as opposed to ARMA's deadline-aware scheduler.

8.2 System Macrobenchmarks

8.2.1 *End-to-end Latency and Accuracy*

End-to-end latency comparison. Figure 15 compares the e2e latency of ARMA and baselines for 4 UE scenario in the over-the-air testbed. Overall, ARMA achieves significantly higher latency SLO satisfaction rate (97.2%, which is up to 48.6% higher than ***Tutti***). Detailed analysis on the performance gain is as follows. First, ***Tutti*** suffers from low SLO satisfaction rate as analyzed in §2.4, mainly as it does not reduce UEs' bitrates under bad channel and suffers from low spectral efficiency. ***AWStream*** improves this by adapting the bitrate, but achieves minimal latency gain as it (i) cannot drastically reduce bitrate as it incurs inference accuracy drop, and (ii) cannot handle compute latency increase from workload fluctuation. Similarly, ***LiteReconfig*** suffers from high latency as it only adapts the DNN config. ***VideoEdge*** and ***Decoupled*** improve this by leveraging bitrate-DNN interplay. However, they still suffer from long tail latency due to two reasons. First, they cannot cope with per-frame fluctuations as they estimate resources by average, and do not have deadline-aware per-frame scheduling mechanisms on both network and compute stages. Second, RAN-agnostic bandwidth estimation leads to inaccurate estimation of the surplus bandwidth capacity (§7.2), thus missing the opportunity to increase bitrate and reduce DNN complexity under high workload. ARMA overcomes these limitations and achieves 97% SLO satisfaction. ARMA also achieves better tail latency and latency jitter (*i.e.,* difference between consecutive frames) performance compared to baselines: for example, it reduces the 95-th percentile latency from 193.80 ms (***VideoEdge***) to 138.81 ms, and average latency jitter from 30.23 (***Tutti***) to 17.20 ms.

Latency breakdown. Figure 16 shows that ARMA achieves latency improvements over baselines on both network and compute stages. As analyzed above, ***Tutti***, ***AWStream***, ***LiteReconfig*** achieves latency gains only on a single stage. ***VideoEdge*** and ***Decoupled*** improve latencies on both stages, but suffer from long tail latency due to mutual ignorance of the app, RAN schedulers.

Accuracy comparison. Figure 17 compares the streaming accuracy [49] (*i.e.,* accuracy of the analysis result compared to the latest frame available at its finish) from the obtained latencies, in terms of bounding box IoU (Intersection over Union). Box plot represents 5-100-th accuracies. It shows that ARMA's high latency SLO satisfaction leads to robust tail accuracy (*e.g.,* 0.56 vs. 0.63 vs. 0.68 for ***Tutti***, ***DC***, ARMA, respectively).

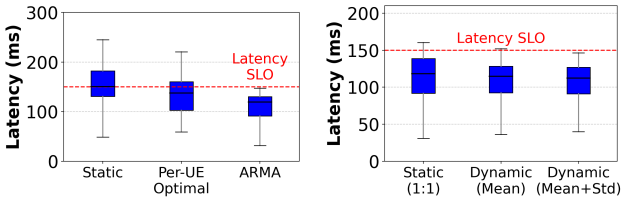
8.2.2 *Scalability*

Figure 18 shows that ARMA consistently achieves lower latency than ***Decoupled*** as the number of UEs increase. Especially, the tail latency of ***Decoupled*** increases significantly (*e.g.,* 95-th latency increasing from 210.7 to 339.7 ms for 4 UEs and 8 UEs), while ARMA keeps it low (*e.g.,* 339.7 vs. 193.5 ms for 8 UEs) due to RAN-aware {bitrate, DNN} selection and deadline-aware per-frame schedulers.

8.3 System Microbenchmarks

8.3.1 *Per-GoP Scheduler*

Stochastic latency modeling accuracy. Figure 19(a) shows the 20 Mbps streaming latency and predicted values with our latency



(a) {Bitrate,DNN} scheduler. (b) Latency SLO splitter.

Figure 20: ARMA **scheduler microbenchmark (box: 25-75%, error bar: 0-95% range).**

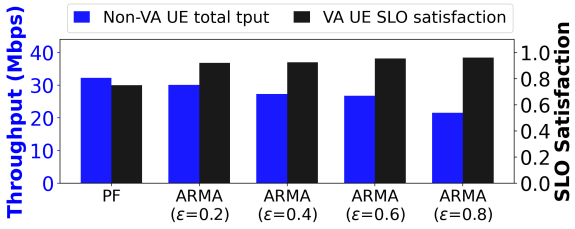


Figure 21: App-aware RB scheduler performance with non-VA file transfer UEs.

model for *subway* in Table 1. Our modeling achieves 6.37 ms Mean Absolute Error (MAE), with 93.56% of the latencies within 2σ range. Figure 19(b) also shows the modeled workload on YTFace video. MAE and 99-th error are 1.89 objects and 7.01 objects, respectively. Under sudden scene change, modeling error increases sharply; however, the model quickly adapts and reduces the prediction error.

{Bitrate,DNN} selection accuracy. Figure 20(a) compares ARMA's latency for 5 UE scenario using three policies: static, per-UE optimal, and iterative probability gradient. Static uses heavy DNN backbones, which incurs long inference latency and low latency SLO satisfaction rate (only 49%). Per-UE optimal reduces the average latency (*e.g.,* 75-th 182.4 vs. 160.2 ms) by reducing backbone and increasing bitrates, but does not effectively reduce the tail, as all UEs increasing the bitrates incurs network bottleneck. Iterative probability gradient algorithm adjusts the configs across UEs to reduce both the average and tail latencies (*e.g.,* 95-th 148.3 ms).

8.3.2 *Per-frame Scheduler*

App-aware latency SLO splitter. Figure 20(b) compares ARMA's latencies with various SLO splitting policies: static (1:1), dynamic (proportional to the average predicted latencies), and dynamic (Eq. (8), using both means and variances). As network, compute stage latencies fluctuate over time, latency modeling-based dynamic splitting improves latency SLO satisfaction from 90.9 to 95.2%.

App-aware RB scheduler. We evaluate the performance of ARMA's app-aware RB scheduler for various ϵ. We use two types of non-VA UEs: file transfer (emulated using Iperf) and web uploading (emulated by sending 20 KB buffer over TCP socket in random intervals sampled from Poisson distribution). (i) *File transfer UE throughput.* Figure 21 shows the throughput for 5 UE scenario: 2 VA UEs and 3 non-VA UEs. Higher ϵ increases the latency SLO satisfaction rate for VA UEs, at the cost of throughput drop for non-VA UEs (*e.g.,* for ϵ=0.2, 93% latency SLO satisfaction with 6.5% throughput drop). The effect of increasing ϵ on SLO satisfaction rate saturates, as aggressively allocating RBs to VA UEs despite bad channel status

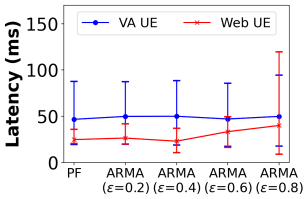


Figure 22: App-aware RB scheduler performance with non-VA web uploading UEs.

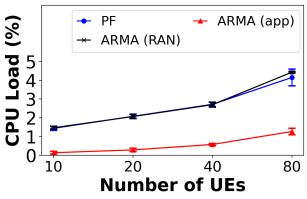


Figure 23: ARMA **schedulers' CPU overhead for varying number of UEs.**

Table 3: App-aware RB scheduler memory overhead.

	Component	Memory (per UE)
PF scheduler	*ue*	908 bytes
	ue_ctxt	96 bytes
	ue_history_db	104 bytes
ARMA **overhead**	*frame_info* (Figure 11)	22 bytes
	Latency utility (Eq. (9)-V_u, B_u)	8 bytes

results in lower resource efficiency. (i) *Web uploading UE latency.* Figure 22 compares web uploading UE latency for 5 UE scenario: 3 VA UEs and 2 non-VA UEs (error bars denote 0-95% range). Higher ϵ increases the latency of web UEs, especially the tail latency (*e.g.,* 41.73 ms for ϵ=0.2 vs. 119.76 ms for ϵ=0.8). VA UE latencies remain similar, as the web UE traffic size is negligible.

8.3.3 *System Overhead*

CPU overhead. We measured the CPU utilization of BS and server using Linux ps [66] in 4 UE scenario. Compared to ***Static***, ARMA incurs 0.4% higher CPU utilization for BS (*e.g.,* 5.2% vs. 5.6%) and 5% higher overhead for server, mainly due to scheduling logic and RIC. Figure 23 shows the scheduling CPU overhead for varying number of UEs; due to connection failure issue in srsRAN RF simulator for >10 UEs, we run the scheduling algorithms separately to measure the overhead. Overall, ARMA's app, RAN schedulers incur minimal overhead (*e.g.,* 1.27% and 4.43% for 80 UEs). As analyzed in §6.2, ARMA's RB scheduler has the same $O(N)$ complexity has PF, thus adding negligible overhead (*e.g.,* 4.15% vs. 4.43% for 80 UEs).

Memory overhead. Table 3 shows the memory overhead to implement ARMA's app-aware RB scheduler. For each UE, it incurs only 34 bytes overhead (3% of memory required in srsRAN's PF scheduler - *ul_sched()* function in *scheduler_time_pf.cpp*). We observed that latency overhead of one addition iteration over UEs required to calculate latency utility (Algorithm 1) is negligible.

RIC communication overhead. Mutual awareness-enabling RIC incurs minimal communication overhead and delay in both both directions. Specifically, app→RAN signaling (Figure 11) incurs 22 bytes overhead per each UE's frame, and RAN→app signaling (Figure 12) incurs 28+12·N bytes for N UEs.

9 Discussion and Future Work

9.1 Generality

Generality to VLM-based video analytics. While ARMA currently focuses on two-stage CNN inference pipeline, its app-RAN joint scheduling can be extended to handle workload fluctuation in

recent Vision Language Model (VLM) [57, 102]-based pipelines as well. Specifically, VLM encodes the input video frames and generates text response to the input query in an auto-regressive manner (one word per iteration, until the 'eos' symbol). The total inference latency of VLMs can be modeled as follows [56],

$$L_{compute} = l_{video\_encoder} + l_{TTFT} + N_{words} \cdot l_{TPOT}. \quad (12)$$

$l_{video\_encoder}$ is the video encoder latency, l_{TTFT} is the initial delay until the first output token (time to first token, TTFT), l_{TPOT} is the generation latency of each token (referred to as time per output token (TPOT) - 38 ms for Video-LLaMA-7B [102] on RTX A6000 GPU), and N_{words} is the response length. The inference latency fluctuates with the scene content as N_{words} increases proportionally to the scene complexity, albeit the relationship may not be linear as in CNNs (Equation (1)) and also varies depending on the query. We leave ARMA's latency model extension for VLMs as future work.

Generality to other apps. ARMA's mutually-aware app-RAN joint scheduling can be extended to various MEC apps involving real-time data streaming and server compute processing. For example, ARMA can be extended to response latency guarantee (*e.g.,* within 200 ms [71]) in voice assistant apps composed of user voice data streaming over RAN and speech recognition and question answering in server. ARMA can also be used for frame latency guarantee in multi-user VR [61] to jointly schedule edge server GPUs and RAN downlink RBs for frame rendering and streaming under dynamic scene content and wireless channel status.

Generality to edge-cloud cooperative inference. ARMA currently assumes fully-offloaded inference as it is challenging to deploy large DNNs on UE devices, and existing on-device inference systems [36, 38, 39, 90, 93] mostly lack adaptation capabilities for workload fluctuation. However, if local processing is available, edge-cloud cooperative inference can further improve ARMA's latency performance. For instance, we can dynamically adjust the portion of frame regions processed by UE and server considering the on-device inference latency, estimated bandwidth and workload (*e.g.,* increase on-device portion under high bandwidth fluctuation).

9.2 Limitations

Real-world deployment challenges. While ARMA shows promising results, following challenges remain for real-world deployment.

- **Handling cross-traffic scenario.** ARMA assumes that each UE only runs a dedicated single VA app (*e.g.,* low-cost CCTV, AR glasses). When a UE runs multiple apps concurrently, it can lead to inaccurate bitrate control and network delay estimation of the VA app. In such cases, we can differentiate and isolate the VA app traffic flow by using a separate 5G Data Radio Bearer (DRB) with its latency SLO specified by the 5G 5QI [1], which can be handled separately by ARMA's RB scheduler.
- **Handling server resource availability fluctuation.** ARMA assumes a dedicated edge server for VA UEs, with fixed inference latency and no model switching overhead (§7.1.2, §7.1.3). In case background tasks run concurrently, a more advanced multi-GPU scheduler is required to consider various factors that dynamically affect compute resource availability, including inference latency fluctuation from throttling and memory overhead from context switching [32], and fairness with background tasks.
- **Standard compliance.** We implement ARMA atop EdgeRIC [45], which assumes an additional Real-Time (RT) RIC deployed in the Distributed Unit (DU) alongside O-RAN's near-RT RIC in the Central Unit (CU). RIC architectures and interfaces for ms-level RAN monitoring/control is actively being investigated in the O-RAN community for standardization (*e.g.,* dApps [19], Microsoft's programmable RAN with dynamic SM [23, 28]). We plan to improve ARMA's RIC for standard-compliant operation, including handling UE handover across cells.

Co-design with video codec. Keyframe size peaks remains a major source of latency SLO violation in ARMA. We plan to mitigate this by extending our design to the video codec. For example, we can consider dividing the frame in units of slices supported in H.264 and H.265 [89] and interleaving the intra-encoded slice across frames.

10 Related Work

Live video analytics. A large body of works aimed at designing live video analytics for various apps (*e.g.,* monitoring [37, 40], AR/MR [52, 92, 93]). Several optimization techniques were studied to improve practicality, including adaptation [40, 83], continual learning [10, 60], model merging [64], and privacy [12]. ARMA is an e2e app-RAN joint scheduling system for improving latency predictability of video analytics apps.

Resource adaptation in live video analytics. Several works aimed at video content and network/compute resource-aware adaptation of video bitrate [11, 15, 22, 35, 42, 43, 50, 52, 65, 86, 100, 103], DNN [5, 6, 17, 33, 47, 87, 88], or both [37, 40]. However, such app schedulers operate in a *mutually-agnostic* manner with the RAN scheduler, resulting in inaccurate network resource availability estimation and latency fluctuation (§2.4).

App-aware RAN scheduling. OutRAN [44] aimed at RAN scheduling to minimize queueing delays of short flows (*e.g.,* webpage loading), but has not considered continuous video analytics workload. Tutti [85] also aims at app-aware RB scheduling, but only optimizes network stage latency, and considers fixed-bitrate, per-frame JPEG-encoding (small frame size variation). Other works aimed at UEs' demand and channel-aware RAN slicing to maximize resource utilization and throughput [16, 26, 53, 54, 97]. They can be complementarily used with ARMA to isolate live video analytics UEs for efficient resource scheduling.

11 Conclusion

We presented ARMA, a live video analytics scheduling system for high latency SLO satisfaction in MEC. To overcome the limitations of prior works in handling complex latency fluctuations over both the video streaming and DNN inference stages, we designed a mutually-aware decoupled scheduling mechanism to foster collaborative interaction between the two using real-time RIC. Evaluation shows that ARMA achieves 97% SLO satisfaction rate.

Acknowledgments

We sincerely thank our anonymous shepherd and reviewers for their valuable feedback. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. 2022R1A2C3008495 and No. RS-2024-00463802).

References

[1] 3GPP TR 23.501. NR; System architecture for the 5G System (5GS). ver. 19.1.0, Sept. 2024.

[2] LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) conformance testing (3GPP TS 36.141 version 14.3.0 Release 14) . https://www.etsi.org/deliver/etsi_ts/136100_136199/136141/14.03.00_60/ts_136141v140300p.pdf. Accessed: 9 Dec. 2024.

[3] User Equipment (UE) radio access capabilities (3GPP TS 38.306 version 17.0.0 Release 17). https://www.etsi.org/deliver/etsi_ts/138300_138399/138306/17.00.00_60/ts_138306v170000p.pdf. Accessed: 9 Dec. 2024.

[4] 5G MEC – Based Cloud Game Innovation Practice. https://www.gsma.com/futurenetworks/wiki/5g-mec-based-cloud-game-innovation-practice/. Accessed: 9 Dec. 2024.

[5] Ahmad, S., Guan, H., Friedman, B. D., Williams, T., Sitaraman, R. K., and Woo, T. Proteus: A high-throughput inference-serving system with accuracy scaling. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (2024), pp. 318–334.

[6] Apicharttrisorn, K., Ran, X., Chen, J., Krishnamurthy, S. V., and Roy-Chowdhury, A. K. Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems* (2019), pp. 96–109.

[7] Arunruangsirilert, K., and Katto, J. Practical commercial 5g standalone (sa) uplink throughput prediction. *arXiv preprint arXiv:2307.12417* (2023).

[8] Azure private multi-access edge compute (MEC). https://azure.microsoft.com/en-us/solutions/private-multi-access-edge-compute-mec. Accessed: 9 Dec. 2024.

[9] Balasingam, A., Kotaru, M., and Bahl, P. {Application-Level} service assurance with 5g {RAN} slicing. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)* (2024), pp. 841–857.

[10] Bhardwaj, R., Xia, Z., Ananthanarayanan, G., Jiang, J., Shu, Y., Karianakis, N., Hsieh, K., Bahl, P., and Stoica, I. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (2022), pp. 119–135.

[11] Canel, C., Kim, T., Zhou, G., Li, C., Lim, H., Andersen, D. G., Kaminsky, M., and Dulloor, S. Scaling video analytics on constrained edge nodes. *Proceedings of Machine Learning and Systems 1* (2019), 406–417.

[12] Cangialosi, F., Agarwal, N., Arun, V., Narayana, S., Sarwate, A., and Netravali, R. Privid: Practical,{Privacy-Preserving} video analytics queries. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (2022), pp. 209–228.

[13] Carlucci, G., De Cicco, L., Holmer, S., and Mascolo, S. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems* (2016), pp. 1–12.

[14] Chaurasia, A., and Culurciello, E. Linknet: Exploiting encoder representations for efficient semantic segmentation. In *2017 IEEE Visual Communications and Image Processing (VCIP)* (2017), IEEE, pp. 1–4.

[15] Chen, T. Y.-H., Ravindranath, L., Deng, S., Bahl, P., and Balakrishnan, H. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems* (2015), pp. 155–168.

[16] Chen, Y., Yao, R., Hassanieh, H., and Mittal, R. {Channel-Aware} 5g {RAN} slicing with customizable schedulers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)* (2023), pp. 1767–1782.

[17] Chin, T.-W., Ding, R., and Marculescu, D. Adascale: Towards real-time video object detection using adaptive scaling. *Proceedings of machine learning and systems 1* (2019), 431–441.

[18] Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., and Stoica, I. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (2017), pp. 613–627.

[19] O-RAN next Generation Research Group (nGRG), "dApps for Real-Time RAN Control: Use Cases and Requirements," ORAN-WG3.E2SM-KPM-v04.00 Technical Specification, October 2024. https://mediastorage.o-ran.org/ngrg-rr/nGRG-RR-2024-10-dApp%20use%20cases%20and%20requirements.pdf. Accessed: 9 Dec. 2024.

[20] Woowa Brothers Operate Self-Driving Delivery Robots with AWS Wavelength. https://aws.amazon.com/solutions/case-studies/woowa-brothers/?nc1=h_ls. Accessed: 9 Dec. 2024.

[21] Microsoft and AT&T demonstrate 5G-powered video analytics. https://azure.microsoft.com/en-us/blog/microsoft-and-att-demonstrate-5gpowered-video-analytics/. Accessed: 9 Dec. 2024.

[22] Du, K., Pervaiz, A., Yuan, X., Chowdhery, A., Zhang, Q., Hoffmann, H., and Jiang, J. Server-driven video streaming for deep learning inference. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (2020), pp. 557–570.

[23] Microsoft programmable RAN platform with dynamic service models. https://azure.microsoft.com/en-us/resources/research/microsoft-programmable-ran-platform-with-dynamic-service-models. Accessed: 9 Dec. 2024.

[24] Architecture & E2 General Aspects and Principles. Technical Specification O-RAN.WG3.E2GAP-v01.01. O-RAN Working Group 3.

[25] FFMPEG. https://ffmpeg.org//. Accessed: 15 Mar. 2024.

[26] Foukas, X., Marina, M. K., and Kontovasilis, K. Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture. In *Proceedings of the 23rd annual international conference on mobile computing and networking* (2017), pp. 127–140.

[27] Foukas, X., Radunovic, B., Balkwill, M., and Lai, Z. Taking 5g ran analytics and control to a new level. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking* (2023), pp. 1–16.

[28] Foukas, X., Radunovic, B., Balkwill, M., Lai, Z., and Settle, C. Programmable ran platform for flexible real-time control and telemetry. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking* (2023), pp. 1–3.

[29] GNU Radio. https://www.gnuradio.org/. Accessed: 9 Dec. 2024.

[30] Leo Bodnar Electronics. Precision GPS Reference Clock. https://www.leobodnar.com/shop/index.php?main_page=product_info&products_id=234. Accessed: 9 Dec. 2024.

[31] GSMA. 2020. 5G TDD Synchronisation Guidelines and Recommendations for the Coexistence of TDD Networks in the 3.5 GHz Range. https://www.gsma.com/spectrum/wp-content/uploads/2020/04/3.5-GHz-5G-TDD-Synchronisation.pdf. Accessed: 9 Dec. 2024.

[32] Gujarati, A., Karimi, R., Alzayat, S., Hao, W., Kaufmann, A., Vigfusson, Y., and Mace, J. Serving DNNs like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (2020), pp. 443–462.

[33] Han, S., Shen, H., Philipose, M., Agarwal, S., Wolman, A., and Krishnamurthy, A. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services* (2016), pp. 123–136.

[34] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

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

[36] Hu, Y., Gokarn, I., Liu, S., Misra, A., and Abdelzaher, T. Algorithms for canvas-based attention scheduling with resizing. In *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)* (2024), IEEE, pp. 348–359.

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

[38] Huynh, L. N., Lee, Y., and Balan, R. K. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services* (2017), pp. 82–95.

[39] Jeon, C., Kim, S., Yi, J., and Lee, Y. Mondrian: On-device high-performance video analytics with compressive packed inference. *arXiv preprint arXiv:2403.07598* (2024).

[40] Jiang, J., Ananthanarayanan, G., Bodik, P., Sen, S., and Stoica, I. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (2018), pp. 253–266.

[41] Jiang, J., Sekar, V., and Zhang, H. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies* (2012), pp. 97–108.

[42] Jiang, S., Lin, Z., Li, Y., Shu, Y., and Liu, Y. Flexible high-resolution object detection on edge devices with tunable latency. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking* (2021), pp. 559–572.

[43] Kang, D., Emmons, J., Abuzaid, F., Bailis, P., and Zaharia, M. Noscope: optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529* (2017).

[44] Kim, J., Lee, Y., Lim, H., Jung, Y., Kim, S. M., and Han, D. Outran: co-optimizing for flow completion time in radio access network. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies* (2022), pp. 369–385.

[45] Ko, W.-H., Ghosh, U., Dinesha, U., Wu, R., Shakkottai, S., and Bharadia, D. Edgeric: Empowering realtime intelligent optimization and control in nextg

networks. *arXiv preprint arXiv:2304.11199* (2023).

[46] O-RAN Working Group 3, "O-RAN Near-Real-time RAN Intelligent Controller E2 Service Model (E2SM) KPM 4.0," ORAN-WG3.E2SM-KPM-v04.00 Technical Specification, October 2023.

[47] Laskaridis, S., Venieris, S. I., Almeida, M., Leontiadis, I., and Lane, N. D. Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (2020), pp. 1–15.

[48] Leal-Taixé, L., Milan, A., Reid, I., Roth, S., and Schindler, K. Motchallenge 2015: Towards a benchmark for multi-target tracking. *arXiv preprint arXiv:1504.01942* (2015).

[49] Li, M., Wang, Y.-X., and Ramanan, D. Towards streaming perception. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16* (2020), Springer, pp. 473–488.

[50] Li, Y., Padmanabhan, A., Zhao, P., Wang, Y., Xu, G. H., and Netravali, R. 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* (2020), pp. 359–376.

[51] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 2980–2988.

[52] Liu, L., Li, H., and Gruteser, M. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking* (2019), pp. 1–16.

[53] Liu, Q., Choi, N., and Han, T. Onslicing: online end-to-end network slicing with reinforcement learning. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies* (2021), pp. 141–153.

[54] Liu, Q., Choi, N., and Han, T. Atlas: automate online service configuration in network slicing. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies* (2022), pp. 140–155.

[55] Liu, X., Ghosh, P., Ulutan, O., Manjunath, B., Chan, K., and Govindan, R. Caesar: cross-camera complex activity recognition. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems* (2019), pp. 232–244.

[56] Conduct your own LLM endpoint benchmarking. https://learn.microsoft.com/en-us/azure/databricks/machine-learning/foundation-model-apis/prov-throughput-run-benchmark. Accessed: 9 Dec. 2024.

[57] Maaz, M., Rasheed, H., Khan, S., and Khan, F. S. Video-chatgpt: Towards detailed video understanding via large vision and language models. *arXiv preprint arXiv:2306.05424* (2023).

[58] Mao, H., Netravali, R., and Alizadeh, M. Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication* (2017), pp. 197–210.

[59] ETSI. 2018. MEC in 5G Networks. https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf. Accessed: 9 Dec. 2024.

[60] Mehrdad, K., Ananthanarayanan, G., Hsieh, K., , J. J., , R. N., Shu, Y., Alizadeh, M., and Bahl, V. Recl: Responsive resource-efficient continuous learning for video analytics. In *USENIX NSDI* (April 2023).

[61] Meng, J., Paul, S., and Hu, Y. C. Coterie: Exploiting frame similarity to enable high-quality multiplayer vr on commodity mobile devices. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (2020), pp. 923–937.

[62] Mills, D. L. Network time protocol (version 3) specification, implementation and analysis. RFC RFC 1305, University of Delaware, 1992.

[63] Open-RAN Alliance. https://www.o-ran.org/. Accessed: 9 Dec. 2024.

[64] Padmanabhan, A., Agarwal, N., Iyer, A., Ananthanarayanan, G., Shu, Y., Karianakis, N., Xu, G. H., and Netravali, R. Gemel: Model merging for memory-efficient, real-time video analytics at the edge. In *USENIX NSDI* (April 2023).

[65] Pakha, C., Chowdhery, A., and Jiang, J. Reinventing video streaming for distributed vision analytics. In *10th USENIX workshop on hot topics in cloud computing (HotCloud 18)* (2018).

[66] Linux ps. https://man7.org/linux/man-pages/man1/ps.1.html. Accessed: 9 Dec. 2024.

[67] PyTorch. https://pytorch.org/. Accessed: 15 Mar. 2024.

[68] O-RAN Working Group 3, "O-RAN Near-Real-time RAN Intelligent Controller E2 Service Model (E2SM) RC 5.0," ORAN-WG3.E2SM-RC-v05.00 Technical Specification, February 2024.

[69] Seo, H., Yi, J., Balan, R., and Lee, Y. Gradualreality: Enhancing physical object interaction in virtual reality via interaction state-aware blending. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (2024), pp. 1–14.

[70] Shapiro, S. S., and Wilk, M. B. An analysis of variance test for normality (complete samples). *Biometrika 52*, 3-4 (1965), 591–611.

[71] Latency Requirements for Real-Time Speech Recognition in Edge AI Applications. https://massedcompute.com/faq-answers/?question=What+are+the+latency+requirements+for+real-time+speech+recognition+in+edge+AI+applications%3F. Accessed: 9 Dec. 2024.

[72] SRSRAN 5G. https://www.srslte.com/5g. Accessed: 9 Dec. 2024.

[73] SRSRAN 5G ZeroMQ-based multi-UE emulation. https://docs.srsran.com/projects/project/en/latest/tutorials/source/srsUE/source/index.html#multi-ue-emulation. Accessed: 9 Dec. 2024.

[74] Secure Reliable Transport (SRT) Protocol. https://github.com/Haivision/srt. Accessed: 9 Dec. 2024.

[75] Stauffert, J.-P., Niebling, F., and Latoschik, M. E. Effects of latency jitter on simulator sickness in a search task. In *2018 IEEE conference on virtual reality and 3D user interfaces (VR)* (2018), IEEE, pp. 121–127.

[76] Sysmocom. 2022. Programmable SIM cards from Sysmocom. https://shop.sysmocom.de/sysmoISIM-SJA2-SIM-USIM-ISIM-Card-10-pack-with-ADM-keys/sysmoISIM-SJA2-10p-adm. Accessed: 9 Dec. 2024.

[77] Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., and Sabella, D. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials 19*, 3 (2017), 1657–1681.

[78] Tan, M., and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (2019), PMLR, pp. 6105–6114.

[79] Tan, M., Pang, R., and Le, Q. V. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2020), pp. 10781–10790.

[80] AWS Wavelength. https://aws.amazon.com/wavelength/?nc1=h_ls. Accessed: 9 Dec. 2024.

[81] Welch, G. An introduction to the kalman filter.

[82] Wolf, L., Hassner, T., and Maoz, I. Face recognition in unconstrained videos with matched background similarity. In *CVPR 2011* (2011), IEEE, pp. 529–534.

[83] Wong, M., Ramanujam, M., Balakrishnan, G., and Netravali, R. {MadEye}: Boosting live video analytics accuracy with adaptive camera configurations. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)* (2024), pp. 549–568.

[84] Xie, Y., Yi, F., and Jamieson, K. Pbe-cc: Congestion control via endpoint-centric, physical-layer bandwidth measurements. 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* (2020), pp. 451–464.

[85] Xu, D., Zhou, A., Wang, G., Zhang, H., Li, X., Pei, J., and Ma, H. Tutti: coupling 5g ran and mobile edge computing for latency-critical video analytics. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking* (2022), pp. 729–742.

[86] Xu, M., Xu, T., Liu, Y., and Lin, F. X. Video analytics with zero-streaming cameras. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)* (2021), pp. 459–472.

[87] Xu, R., Lee, J., Wang, P., Bagchi, S., Li, Y., and Chaterji, S. Litereconfig: Cost and content aware reconfiguration of video object detection systems for mobile gpus. In *Proceedings of the Seventeenth European Conference on Computer Systems* (2022), pp. 334–351.

[88] Xu, R., Zhang, C.-l., Wang, P., Lee, J., Mitra, S., Chaterji, S., Li, Y., and Bagchi, S. Approxdet: content and contention-aware approximate object detection for mobiles. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems* (2020), pp. 449–462.

[89] Yang, K., Jeong, M., Yi, J., Lee, J., Park, K., and Lee, Y. Logan: Loss-tolerant live video analytics system. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking* (2024), pp. 1314–1329.

[90] Yang, K., Yi, J., Lee, K., and Lee, Y. Flexpatch: Fast and accurate object detection for on-device high-resolution live video analytics. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications* (2022), IEEE, pp. 1898–1907.

[91] Yi, F., Wan, H., Jamieson, K., Rexford, J., Xie, Y., and Michel, O. Athena: Seeing and mitigating wireless impact on video conferencing and beyond. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks* (2024), pp. 103–110.

[92] Yi, J., Choi, S., and Lee, Y. EagleEye: Wearable camera-based person identification in crowded urban spaces. In *Proceedings of the 16th Annual International Conference on Mobile Computing and Networking* (2020), ACM.

[93] Yi, J., and Lee, Y. Heimdall: mobile gpu coordination platform for augmented reality applications. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (2020), pp. 1–14.

[94] Yin, X., Jindal, A., Sekar, V., and Sinopoli, B. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), pp. 325–338.

[95] Ultralytics YOLO-v5. https://github.com/ultralytics/yolov5. Accessed: 9 Dec. 2024.

[96] Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., and Darrell, T. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687 2*, 5 (2018), 6.

[97] Zanzi, L., Sciancalepore, V., Garcia-Saavedra, A., Schotten, H. D., and

Costa-Pérez, X. Laco: A latency-driven network slicing orchestration in beyond-5g networks. *IEEE Transactions on Wireless Communications 20*, 1 (2020), 667–682.

[98] Zeng, X., Fang, B., Shen, H., and Zhang, M. Distream: scaling live video analytics with workload-adaptive distributed edge intelligence. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems* (2020), pp. 409–421.

[99] ZeroMQ. https://zeromq.org/. Accessed: 9 Dec. 2024.

[100] Zhang, B., Jin, X., Ratnasamy, S., Wawrzynek, J., and Lee, E. A. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (2018), pp. 236–252.

[101] Zhang, H., Ananthanarayanan, G., Bodik, P., Philipose, M., Bahl, P., and Freedman, M. J. Live video analytics at scale with approximation and {Delay-Tolerance}. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (2017), pp. 377–392.

[102] Zhang, H., Li, X., and Bing, L. Video-llama: An instruction-tuned audio-visual language model for video understanding. *arXiv preprint arXiv:2306.02858* (2023).

[103] Zhang, T., Chowdhery, A., Bahl, P., Jamieson, K., and Banerjee, S. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* (2015), pp. 426–438.