JOURNAL OF GEOSPACE SCIENCE

Vol. 6 No. 2 (2026)

E-ISSN : 2413-6093
https://jogss.org

Adaptive Edge-Cloud Data Scheduling forUrban Infrastructure Inspection Systems

Dai Teng
University of Illinois Urbana-Champaign Urbana, IL, USA
Email: *tengdai@ieee.org*

ABSTRACT

Vehicle- and drone-mounted inspection systems for urban bridges and roads generate image streams whose value depends jointly on analytic quality and timeliness, yet these streams must leave the field through cellular uplinks whose capacity fluctuates by an order of magnitude and drops to zero for tens of seconds. This paper presents ACES, an adaptive compression and edge-cloud scheduling method that decides, for every captured frame, which version to upload (resolution and JPEG quality), whether its defect analysis runs on the edge node or in the cloud, and in which order frames are transmitted and processed. ACES derives a per-frame decision rule from Lyapunov drift-plus-penalty optimization over three coupled backlogs (uplink, edge CPU, and cloud), so that backlogs act as prices for bytes and compute; it adds damped backpressure migration of uncommitted frames, an urgency-tiered preemptive order, and value-aware buffering with write-ahead-log recovery. We prove an [O(1/V),O(V)] utility-backlog trade-off for the core rule. The evaluation replays real data: rate-quality-compute profiles measured on 118 pixel-annotated urban road frames at 24 encoding versions, and five real cellular uplink traces (0.8–19.3 Mb/s, outages up to 78 s), in 1,480 trace-driven runs. ACES attains a normalized inspection utility of 0.980 versus 0.948 for the strongest baseline (paired gain +0.031, 95% CI [0.023,0.040], 24/25 wins), improves it by up to 5 points on congested low-capacity uplinks, delivers 96.8% of critical alerts within 4 s, stays ahead under edge crashes, bandwidth-estimation errors and a shared failing cloud, and spends 0.3 ms of computation per 0.2-s scheduling epoch.

KEYWORDS

edge–cloud computing
stream scheduling
infrastructure inspection
Lyapunov optimization
cellular uplinks
fault tolerance

1. Introduction

Urban bridges, viaducts and pavements are inspected increasingly by camera arrays mounted on survey vehicles and by drones, which turn periodic manual surveys into continuous image streams [1, 2]. A survey vehicle with four downward-looking cameras captures several frames per second; each frame must be analyzed for cracks and other surface defects, and the analysis results feed asset-management databases and, for severe defects, immediate maintenance alerts. The computation is split between an edge node carried by

the vehicle (or mounted beside a bridge) and a cloud back end, the canonical edge-cloud pattern [3]. The weak link of this pipeline is the wide-area uplink. Inspection fleets operate on public cellular networks whose uplink capacity varies rapidly with location and load [4]. In the five real driving traces we use (Table 1), the mean uplink capacity ranges from 0.84 to 19.3 Mb/s, the per-second coefficient of variation reaches 0.61, and individual outages last up to 78 s. At the same time, the data have heterogeneous urgency: a severe crack on a bridge deck must reach operators within seconds, a moderate crack within tens of seconds, and routine frames within minutes. Inspection data also have a characteristic rate-quality structure: in our measurements, uploading a frame at full resolution with JPEG quality 45 instead of 90 removes 64% of the bytes while losing only 1.7% of the crack-extraction F1, and analyzing the frame on the edge reduces the upload to a 3-KB result at the price of edge CPU time. These properties expose a bottleneck that existing mechanisms address only partially. Store-and-forward pipelines and industrial stream managers export data with persistence and static priorities [5, 6]; on the three low-capacity traces, our measurements show that this keeps critical alerts moving but lets most other frames expire (normalized utility 0.40–0.46). Bandwidth-adaptive streaming analytics degrades data quality to match the available bandwidth [7, 8, 9], but treats all frames alike and ignores edge computation. Computation offloading [10, 11] and minimum-completion-time dispatching [12] place tasks to minimize latency but are oblivious to the value of data and to the bandwidth-quality trade-off. What is needed is a joint, online decision of *what* to send (version), *where* to analyze (edge or cloud), and *when* (order), which stays stable under outages and failures.

This paper presents **ACES** (Adaptive Compression and Edge-cloud Scheduling) and makes the following contributions.

- **Formulation.** We model inspection streams as frames with urgency classes, deadline-decaying analytic utility and measured per-version size, quality and compute costs, served by three coupled queues (uplink, edge CPU, cloud) under trace-driven capacity, link outages and node failures (Sect. 3).
- **Method.** ACES selects each frame's version and analysis location by a closed-form drift-plus-penalty rule in which normalized backlogs act as prices for bytes and CPU time, re-optimizes uncommitted frames with damped backpressure migration, orders queues with an urgency-tiered preemptive policy, and manages a write-ahead-logged buffer with value-aware eviction. We prove an $[O(1/V), O(V)]$ utility-backlog trade-off for the core rule (Sect. 5).
- **Real-data evaluation.** We measure the rate-quality-compute profile of 118 pixel-annotated urban road frames [13] at 24 encoding versions and replay it over five real cellular uplink traces [4, 14] against six baselines in 1,480 runs, including ablation, sensitivity and robustness studies (Sects. 6–7). ACES improves the normalized inspection utility from 0.948 to 0.980 on average and by 4.0–5.0 points on congested low-capacity uplinks, with sub-millisecond decision cost.

2 Related Work

Inspection data systems. Image-based inspection and monitoring of concrete and asphalt infrastructure have been reviewed in [1, 2]; crack extraction from pavement images is commonly benchmarked on CrackForest [13], and multi-scale Hessian line filters [15] remain a standard parameter-light extractor. These works concentrate on detection accuracy; ACES treats the extractor as a measured, compressible task and focuses on moving inspection data through constrained networks.

Edge computing, offloading and migration. Edge computing places computation near data sources to cut latency and backhaul traffic [3]. Offloading decisions have been optimized dynamically with Lyapunov methods [10] and jointly with resource allocation in multi-server settings [11]; service migration follows user mobility [16]. Classical heuristics such as minimum completion time map independent tasks onto heterogeneous machines [12]. These approaches decide placement for given task sizes, whereas ACES jointly chooses the data version, which changes both the transfer size and the value of the result. Recent edge–cloud video analytics research further shows the value of making communication and computation decisions jointly. Wang et al. [17] use region-aware on-edge collaboration to raise video-processing throughput; Cao et al. [18] adapt edge-to-cloud transfers to the available bandwidth and to the difficulty of each input; Bai et al. [19] jointly configure video-analytics knobs and resources. These studies establish a

strong foundation for fine-grained, resource-aware video processing; ACES applies joint quality and placement control to inspection frames with heterogeneous alert deadlines and disruption-aware buffers.

Adaptive streaming analytics

AWStream learns accuracy-bandwidth profiles and degrades wide-area streams to match bandwidth [7]; Chameleon adapts video analytics configurations [8]; VideoStorm schedules analytics queries by quality and lag [20]; JCAB jointly adapts configurations and bandwidth with Lyapunov optimization [9]. ACES shares the profile-driven view but adds per-frame urgency, edge-versus-cloud placement, and failure handling, and uses uplink, edge and cloud backlogs as control state. Edge–cloud scheduling has also advanced through multi-agent controllers for distributed streaming analytics [21] and adaptive configuration, resolution and resource orchestration with Lyapunov optimization [22]. These results demonstrate the effectiveness of responsive cross-tier resource management in contemporary video workloads. ACES extends this research direction to time-sensitive inspection streams, with a profiled compression ladder and backlog-based scheduling of uplink, edge and cloud workloads.

Disconnection tolerance and reliability

Delay-tolerant networking stores and forwards data across intermittent links [6]. Industrial edge runtimes provide persistent, prioritized export streams for intermittent connectivity [5]. ACES builds on write-ahead logging [23] for crash recovery, uses hedged execution [24] for critical frames, and orders work with deadline-aware rules from real-time and job-shop scheduling [25, 26].

3 System Model and Problem Formulation

3.1 Architecture

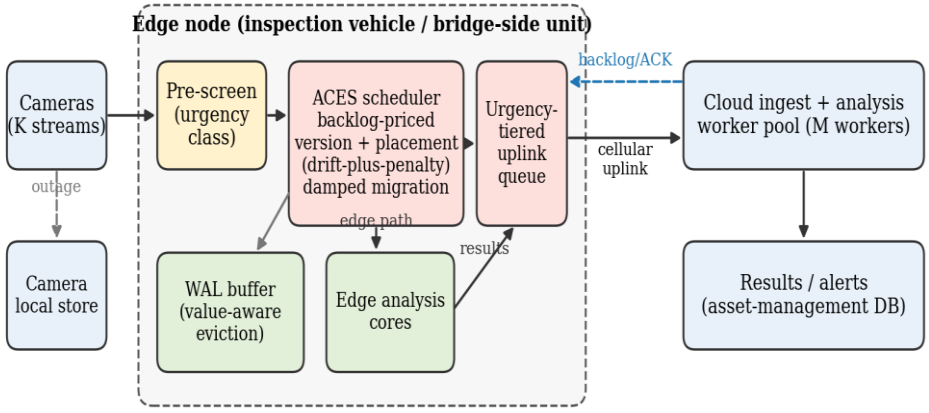


Figure 1shows the system. An edge node (on a survey vehicle or beside a bridge) receives frames from K cameras. A quarter-scale pre-screen assigns each frame i an urgency class c_i∈{critical,major,routine} with deadline D_c and weight w_c. The edge node has n_e

ACES architecture. Frames are pre-screened at the edge; the scheduler selects version and analysis location with backlog prices, migrates uncommitted frames, and orders an urgency-tiered uplink queue; a WAL buffer survives outages and crashes.

3.2 Frames, options and utility

Frame i arrives at a_i. Its option set is $\mathcal{O} = \{\varnothing\} \cup \{v_1, \ldots, v_L\} \cup \{e\}$: deferral $\varnothing$, uploading version v_ℓ (a scale-quality pair) for cloud analysis, or edge analysis e followed by uploading the compact result. Option o has transfer size $s_{i,o}$, edge work $p^E_{i,o}$, cloud work $p^C_{i,o}$ and analytic quality $f_{i,o}$, the F1 score of the crack map obtained from that option. All four quantities are measured per frame (Sect. 6). If the result of frame i reaches the cloud with latency L_i, it yields utility

$$U_i = w_{c_i} f_{i,o_i} g_{c_i}(L_i), \qquad (1)$$

where the timeliness factor $g_c(L)$ equals 1 for $L \le D_c$, decays linearly as $1 - (L - D_c)/G_c$ during a grace period G_c, and is 0 afterwards. We report the *normalized inspection utility* $\mathrm{NIU} = \sum_i U_i \,/\, \sum_i w_{c_i} f_i^{\mathrm{ref}}$, where

f_i^{ref} is the F1 of full-resolution edge analysis; NIU is 1 when every frame is analyzed at full quality within its deadline.

3.3 Queues, capacity and failures

Let $Q_N(t)$ be the uplink backlog in bytes, $Q_E(t)$ the edge CPU backlog in seconds and $Q_C(t)$ the cloud backlog in seconds. Each evolves as $Q(t+1) = \max\{Q(t) - b(t), 0\} + A(t)$, where $A(t)$ is the work admitted by the scheduling decisions and $b(t)$ the service: $C(t)\Delta$ bytes for the uplink (drawn from a real trace), $n_e\Delta$ for the edge and $M_{\text{up}}(t)\Delta$ for the cloud. Outages are periods with $C(t) = 0$ in the traces. Edge nodes crash with exponentially distributed inter-failure times and recover after a uniformly distributed downtime; cameras keep recording during downtime and hand their frames to the node after recovery, and the WAL restores queued frames and partially uploaded messages. Cloud workers fail independently; in-progress tasks are retried.

3.4 Problem

The scheduler chooses, online and without knowledge of future capacity, an option for every frame, the service order of the uplink and edge queues, and which frames to evict when the buffer is full, to

$$\max \quad \liminf_{T\to\infty} \frac{1}{T} \sum_{t<T} \mathbb{E}\left[\sum_{i\in\mathcal{A}(t)} U_i\right] \qquad (2)$$

$$\text{s.t.} \quad Q_N,\ Q_E,\ Q_C \text{mean-rate stable},$$

where $\mathcal{A}(t)$ is the set of frames decided in epoch t. Even the offline version with known capacities contains multiple-choice knapsack and weighted-tardiness scheduling as special cases, both NP-hard, which motivates an online, decomposable rule.

4 Measurement-Driven Observations

O1: Uplinks are volatile and intermittent. Table 1 summarizes the eight Mahimahi uplink traces recorded on U.S. operators while driving [4, 14]; Fig. 2 plots three of them. Capacities differ by a factor of 23 across operators and technologies, every trace has a heavy lower tail, and the three low-capacity traces (0.83–0.94 Mb/s) have 15–23 outages longer than one second. Our four cameras at 2 Hz offer 2.7 Mb/s at top quality, i.e., 0.14–0.44 of the capacity of the two high-capacity LTE traces but 2.9–3.3 times the capacity of the low-capacity traces.

Trace provenance

The traces are public recordings from earlier cellular-network studies [4, 14]; their measured variability and outages provide repeatable stress conditions under which all policies face identical network dynamics.

Table 1Real cellular uplink traces (1-s statistics). Outages: gaps >1 s between delivery opportunities.

Uplink trace	Dur. (s)	Mean (Mb/s)	CV	P10	Outages	Max out. (s)	Use
Verizon LTE (driving)	1365	6.22	0.61	0.67	18	14.4	eval
T-Mobile LTE (driving)	475	19.26	0.55	4.02	0	0.7	eval
AT&T LTE (driving)	1012	0.83	0.49	0.01	15	78.3	eval
T-Mobile UMTS (driving)	931	0.94	0.46	0.30	23	5.8	eval
Verizon EV-DO (driving)	1065	0.84	0.42	0.23	22	11.2	eval
Verizon LTE (short)	140	5.90	0.53	2.29	1	1.7	tune

T-Mobile LTE (short)	140	13.22	0.61	3.32	0	0.7	tune
AT&T LTE 2016	120	1.89	0.89	0.22	8	4.1	tune

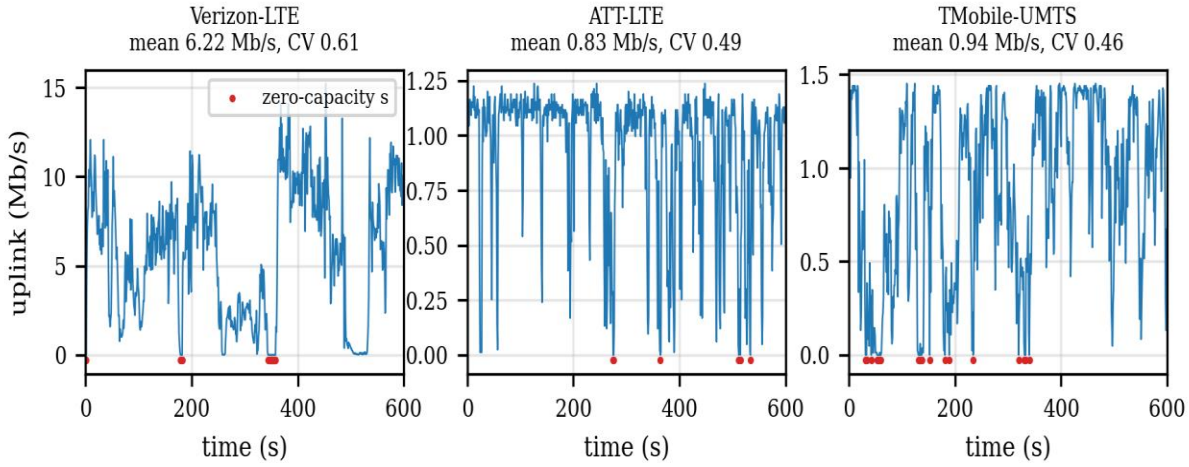


Figure 2Uplink capacity of three real traces over 600 s; red dots mark seconds with zero capacity.

O2: Analytic quality degrades slowly with compression, compute does not. Fig. 3 and Table 2 report the measured profile. Halving the resolution cuts the extraction time from 187 to 50 ms and the size by 70% at an F1 loss of 0.03; the steepest losses occur only below 3 KB per frame. Edge analysis yields the best F1 with a 3.0-KB result but costs 186κ ms of edge CPU, so the edge can absorb only a fraction of the stream.

O3: Volume control alone or priority alone is insufficient.

On low-capacity uplinks, store-and-forward with static priorities keeps critical alerts on time but lets major and routine frames expire (Table 4), while bandwidth-adaptive degradation keeps all classes flowing but uploads critical frames at the same reduced quality as routine ones and leaves edge cores idle. An effective scheduler must couple version, placement and order.

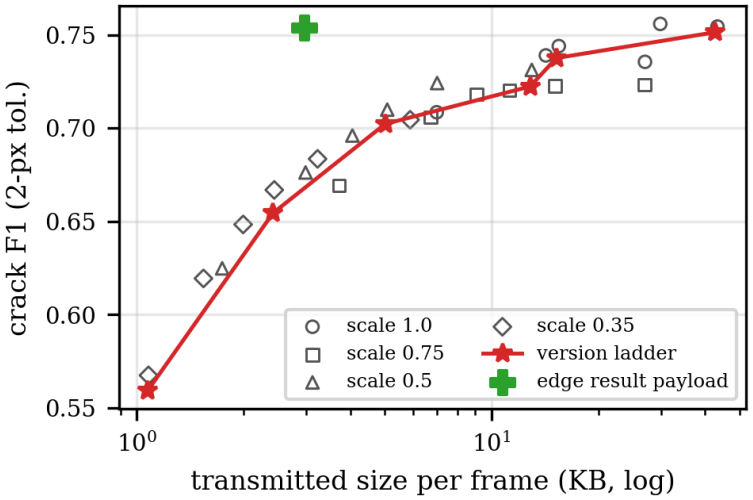


Figure 3Measured rate-quality profile of crack extraction (118 frames, 24 versions, F1 with 2-pixel tolerance). The red curve is the Pareto version ladder used by ACES; the green marker is the edge-analysis result payload.

Table 2Version ladder and edge option, measured on the evaluation frames. Proc.: extraction time on one cloud core.

Option	Scale	JPEG q	Size (KB)	F1	Proc. (ms)	$\hat{f}$
Cloud v_1	1.00	90	42.6	0.751	187	1.001
Cloud v_2	1.00	45	15.2	0.738	187	0.997
Cloud v_3	0.50	90	12.8	0.722	50	0.988
Cloud v_4	0.50	60	5.0	0.702	50	0.956
Cloud v_5	0.35	60	2.4	0.654	22	0.915
Cloud v_6	0.35	15	1.1	0.559	22	0.770
Edge e (result)	1.00	–	3.0	0.754	186× κ	1.000

5 ACES Design

ACES runs on the edge node and consists of (i) backlog-priced version and placement selection, (ii) damped backpressure migration, (iii) urgency-tiered preemptive ordering, and (iv) a reliability layer for estimation, buffering and recovery. Algorithm 1 summarizes one epoch.

5.1 Backlog-Priced Version and Placement Selection

We normalize the three backlogs to seconds of work, $Z_N = Q_N/\bar{C}$, $Z_E = Q_E/n_e$ and $Z_C = Q_C/M$, where $\bar{C}$ is a slowly varying estimate of the mean uplink capacity, and use the weighted quadratic Lyapunov function

$$\mathcal{L}(t) = 1/2\,(Z_N(t)^2 + n_e Z_E(t)^2 + Z_C(t)^2). \qquad (3)$$

Let $u(t) = \sum_i w_{c_i} \hat{f}_{o_i}$ be the utility admitted in epoch t, where $\hat{f}_o$ is the profiled relative quality of option o (Table 2). Standard manipulation [27] bounds the drift-plus-penalty expression by

$$\mathbb{E}[\mathcal{L}(t+1) - \mathcal{L}(t) - Vu(t)\,|\,\mathbf{Z}(t)] \le B + \sum_k \beta_k Z_k(t)\,\mathbb{E}[-b_k(t)]$$
$$-\mathbb{E}[\sum_i \left(V w_{c_i}\hat{f}_{o_i} - Z_N \frac{s_{i,o_i}}{\bar{C}} - Z_E\, p^E_{o_i} - Z_C \frac{p^C_{o_i}}{M}\right)|\mathbf{Z}(t)], \qquad (4)$$

with $\beta_N = \beta_C = 1$, $\beta_E = n_e$, and a constant B bounding the second moments of admitted and served work. Only the last sum depends on the decisions, and it separates across frames. Minimizing the bound therefore yields the per-frame rule

$$o_i^\star = \underset{o\in\mathcal{O}}{\operatorname{argmax}} \underbrace{V w_{c_i}\hat{f}_o \gamma_i(o)}_{\text{value}} - \underbrace{\left(Z_N \frac{s_{i,o}}{\bar{C}} + Z_E p^E_o + Z_C \frac{p^C_o}{M}\right)}_{\text{backlog prices}} \qquad (5)$$

where deferral has score 0. The product of a normalized backlog and the work an option adds is its price: when the uplink congests, Z_N rises and frames move to smaller versions or to the edge; when edge cores are busy, Z_E steers frames to the cloud; when cloud workers fail, the reported Z_C keeps frames on the edge. High-weight critical frames keep high-quality options at backlog levels at which routine frames are already compressed. The factor $\gamma_i(o) = g_{c_i}(\tau_i + \hat{L}_{i,o})$ scales the value by the timeliness predicted from the frame's age τ_i and its estimated completion time $\hat{L}_{i,o}$, computed from the backlog ahead of the frame's class, the capacity estimate $\hat{C}$, the profiled processing times and the reported cloud waiting time; with $\gamma \equiv 1$ the rule is exactly the minimizer of (4). The real sizes $s_{i,o}$ of the frame's own encodings are used, because encoding all six versions takes only 1–2 ms each.

Theorem 1 (Utility-backlog trade-off). Assume i.i.d. frame arrivals and service processes across epochs with finite second moments, and $\mathbb{E}[b_k] > 0$ for every queue. Let U^* be the maximum time-average utility of (2) and $u_{\max}$ a bound on the utility admissible in one epoch. Under rule (5) with $\gamma \equiv 1$,

$$\liminf_{T\to\infty} \frac{1}{T}\sum_{t<T} \mathbb{E}\,[u(t)] \quad \geq U^* - \frac{B}{V}, \qquad (6)$$

$$\limsup_{T\to\infty} \frac{1}{T}\sum_{t<T}\sum_{k} \mathbb{E}\,[Z_k(t)] \quad \leq \frac{B + V u_{\max}}{\epsilon}, \qquad (7)$$

where $\epsilon = \min_k \mathbb{E}[b_k] > 0$.

Proof sketch. Rule (5) minimizes the right-hand side of (4) over all decisions of the epoch, so the bound holds for any alternative policy. Comparing with the stationary randomized policy that attains U^* while keeping $\mathbb{E}[A_k] \leq \mathbb{E}[b_k]$, summing over T epochs and telescoping $\mathcal{L}$ gives the utility bound. Comparing with the all-deferral policy, which admits no work and thus has drift at most $-\epsilon \sum_k Z_k$, gives the backlog bound [27]. Deferral makes this comparison policy always feasible, so no further capacity condition is required. ▫

Migration (Sect. 5.2), delayed backlog reports and the reliability mechanisms of Sect. 5.4 build on this core rule; Sect. 7 quantifies their effect.

5.2 Damped Backpressure Migration

Prices change after a frame is admitted, for example when an outage begins. Every epoch, ACES re-scores all frames that are neither partly transmitted nor in edge processing, and moves a frame from its current option o to $o^\star$ if the score gain exceeds $\theta V w_{c_i}$, or defers it if all scores are non-positive. Evaluating all candidates against the same prices causes herding: when only two options exist, all candidates flip together and flip back in the next epoch. ACES therefore applies only the $\max(8, \lfloor 0.05\, n \rfloor)$ moves with the largest gains per epoch, where n is the number of candidates, which turns simultaneous best responses into a damped sequence. Migration covers both directions, from the cloud path to the edge path and back.

5.3 Urgency-Tiered Preemptive Ordering

The uplink transmits messages in chunks within each 20-ms slot, so a newly queued critical message preempts a bulk upload at the next slot. The queue is ordered in two tiers. Critical messages are served earliest-deadline-first [25]. Other messages are ordered by an apparent-tardiness-cost index [26],

$$I_m = \frac{w_c \hat{f}_o / G_c}{p_m} \exp\left(-\frac{\max(d_m - p_m - t, 0)}{k\,\bar{p}}\right), \qquad (8)$$

where p_m is the remaining transmission time at rate $\hat{C}$, d_m the deadline minus the expected downstream processing time, $\bar{p}$ the mean remaining time in the queue, and k the look-ahead. The weight $w_c \hat{f}_o / G_c$ is the rate at which utility (1) decays after the deadline, so frames far from their deadline yield to frames whose value is about to decline, and a stale routine frame cannot overtake a fresh critical alert. The edge CPU queue uses the same tiers with processing times instead of transmission times.

5.4 Reliability Layer

Bandwidth estimation and outages. The transport layer reports the delivery rate of each epoch. ACES uses $\hat{C} = \min(\mathrm{HM}_8, \mathrm{EWMA}_{0.5})$, the minimum of the harmonic mean of the last eight samples and a fast moving average, and an EWMA $\bar{C}$ with a time constant of 10 s for price normalization. A link that delivers nothing for 0.6 s is flagged as an outage; the short-term history is discarded when the link returns so that the estimate reflects the post-outage capacity.

WAL buffering and recovery. Every admitted frame, its option and the acknowledged byte offset of its upload are appended to a write-ahead log [23]. After a crash, the node restores the queues, resumes partial uploads from the logged offset, restarts interrupted edge jobs, and re-scores all restored and newly arrived frames with the current prices; the cloud de-duplicates results by frame identifier.

Value-aware eviction and expiry. When the buffer exceeds $B_{\max}$, ACES evicts the frames with the lowest value density $w_c\, g_c(\tau_i)/\mathrm{size}_i$ instead of the oldest frames, and every policy discards frames whose utility horizon $D_c + G_c$ has passed.

Hedged critical alerts. A critical frame still in flight after ηD_{crit} is duplicated on the other path (edge analysis for a cloud-path frame, a mid-ladder upload for an edge-path frame); the first result wins and the other copy is cancelled [24].

Algorithm 1: ACES scheduling epoch at an edge node. Input: pending frames, backlogs Q_N, Q_E, cloud report Q_C, delivery-rate sample.

1. Update $\hat{C}$, $\bar{C}$ and the outage flag; drop frames whose utility horizon expired.
2. For each newly pre-screened frame i: encode the ladder versions; choose o_i by rule (5); defer if the best score is ≤ 0; log (i, o_i) to the WAL and enqueue the upload or edge job.
3. Hedge critical frames older than ηD_{crit}.
4. Score uncommitted frames and apply the $\max(8, \lfloor 0.05n \rfloor)$ largest gains above $\theta V w_c$.
5. Order the uplink and edge queues: critical frames by EDF, other frames by the index I_m.
6. If the buffer exceeds $B_{\max}$, evict the frames with the lowest value density.

5.5 Complexity

Scoring a frame costs $O(|\mathcal{O}|) = O(L + 2)$. An epoch with n pending frames costs $O(nL)$ vectorized operations for re-scoring and $O(n \log n)$ for ordering; large queues are re-sorted once per second and new messages are inserted by binary search in between. Sect. 7.6 reports measured costs.

6 Experimental Setup

6.1 Real Data

Inspection frames. We use the 118 frames of the CrackForest data set [13]: 480×320 urban road images from Beijing with pixel-level crack annotations. A random 30% of the frames (35) form the *profiling split*, used only to build the ladder, calibrate thresholds and compute the profile means $\hat{f}_o$; the remaining 83 frames form the stream pool. The crack extractor flattens illumination, applies a multi-scale Sato line filter ($\sigma \in \{1,2,3\}$) [15] to dark ridges, applies hysteresis thresholding at the 98.5th percentile with ratio 0.6, and removes components below 100 pixels; its F1 against the annotation with 2-pixel tolerance is 0.754 at full resolution. For each frame we encode 24 versions (scales 1, 0.75, 0.5, 0.35; JPEG [28] qualities 90 to 15) and record encoded size, extraction F1 on the decoded version, and single-core extraction time (median of three runs per resolution). The version ladder is the six-level Pareto frontier of mean size and mean F1 on the profiling split (Table 2). Urgency classes follow the annotated crack area: the top 10% of frames are critical, the next 30% major and the rest routine. Sect. 7.5 replaces these labels by noisy labels and by a quarter-scale pre-screen computed on the edge in 14.3 ms.

Experimental scope. The 118 annotated CrackForest frames [13] give an auditable compression–quality–compute profile, and trace replay in the discrete-time emulator of Sect. 6.2 exposes all policies to identical, repeatable network and workload conditions.

Uplink traces. The five long driving traces of Table 1 form the evaluation set; the three short traces are used only to tune ACES's parameters. Each line of a trace is a 1500-byte delivery opportunity; traces repeat at their end as in Mahimahi [14], and each run starts at a random offset.

6.2 Trace-Driven Emulator

A discrete-time emulator with 20-ms slots replays, for every frame, its measured version sizes, extraction F1 and CPU times, and drains the uplink queue by the bytes the trace allows in each slot. It implements chunk-level preemption, edge cores, the cloud worker pool with priority queues, one-way latency, WAL persistence with resumable uploads, edge crashes and cloud worker failures. All policies share the pre-screen, the WAL, expiry, the transport delivery-rate samples and the cloud. Table 3 lists the defaults. Each configuration runs for a 600-s mission plus a 30-s drain period with five seeds (main comparison) or three seeds (other studies), which randomize trace offset, camera phases, frame sequence and failure times.

Table 3Default parameters

Parameter	Value
Cameras, capture rate	$K = 4$, 2 Hz each (8 frames/s)
Deadline / grace / weight (crit., major, routine)	(4, 30, 180) s / $G_c = D_c$ / (10, 3, 1)
Edge cores, slow-down	$n_e = 2, \kappa = 3$

Cloud workers, one-way delay	$M = 4$, 40 ms
Buffer, message header	$B_{\max} = 128$ MB, 300 B
Slot, epoch	20 ms, $\Delta = 0.2$ s
ACES	$V = 8, k = 2, \theta = 0.05, \eta = 0.5$
Failures (when enabled)	edge MTBF 300 s, down 30–90 s;
	cloud worker MTBF 300 s, down 60–180 s

6.3 Baselines

FIFO: store-and-forward of the top version, cloud analysis. **PRIO**: persistent export with strict class priority, the policy of industrial stream managers [5]. **AQ**: AWStream-style degradation [7], which selects the highest ladder level whose data rate fits 90% of $\hat{C}$ after reserving capacity to drain the backlog within 5 s, using the same profile as ACES. **EDGE-T**: edge-first offloading that analyzes a frame on the edge when the predicted edge wait is below 2 s and uploads the top version otherwise. **MCT**: per-frame minimum-completion-time placement [12] between edge and cloud with class priority. **MCT-AQ**: MCT placement with AQ versions and class priority, the strongest combination of existing mechanisms.

6.4 Tuning and Metrics

V and k were chosen on the three tuning traces at twice the default load: NIU varied by less than 0.003 for $V \in [2,32]$ and $k \in [1,4]$, and we fixed $V = 8$, $k = 2$. We report NIU, the deadline satisfaction ratio (DSR, fraction of frames of a class whose result arrives within D_c, counting lost frames as misses), the 95th-percentile latency of critical results, the mean F1 of delivered results, uplink volume and the fraction of frames analyzed on the edge.

7 Evaluation

7.1 Overall Performance

Table 4 and Fig. 4 summarize 175 runs. ACES achieves the highest NIU, 0.980, with the smallest spread across traces (± 0.020). MCT-AQ and AQ follow at 0.948 and 0.945; EDGE-T and MCT reach 0.85 because they upload the top version whenever they use the cloud and saturate low-capacity links; PRIO and FIFO collapse to 0.65 and 0.45. The per-trace view in Fig. 4 shows where the gains arise: on the congested ATT-LTE, T-Mobile UMTS and Verizon EV-DO traces, ACES exceeds the best baseline by 5.0, 4.0 and 4.9 points; on Verizon LTE it is within 0.004 of the best (MCT); on T-Mobile LTE, where capacity is abundant, all policies reach 0.995–0.998.

Table 4Main results over five real uplink traces and five seeds (mean; NIU also with standard deviation across the 25 runs). DSR: deadline satisfaction ratio; MB: uplink volume per 600-s mission; Edge: frames analyzed on the edge.

Policy	NIU	DSR_{crit}	DSR_{maj}	DSR_{rout}	$P95_{crit}$ (s)	F1	MB	Edge
FIFO	0.447±0.410	0.341	0.405	0.549	4.61	0.749	125	0%
PRIO	0.654±0.274	0.934	0.453	0.400	2.72	0.754	126	0%
AQ	0.945±0.036	0.960	0.994	1.000	1.38	0.714	96	0%
EDGE-T	0.852±0.115	0.953	0.959	0.439	2.28	0.756	91	45%

MCT	0.848±0.120	0.968	0.977	0.418	1.47	0.757	122	22%
MCT-AQ	0.948±0.034	0.975	0.995	1.000	0.76	0.713	96	0%
ACES	0.980±0.020	0.968	0.995	1.000	1.44	0.741	114	24%

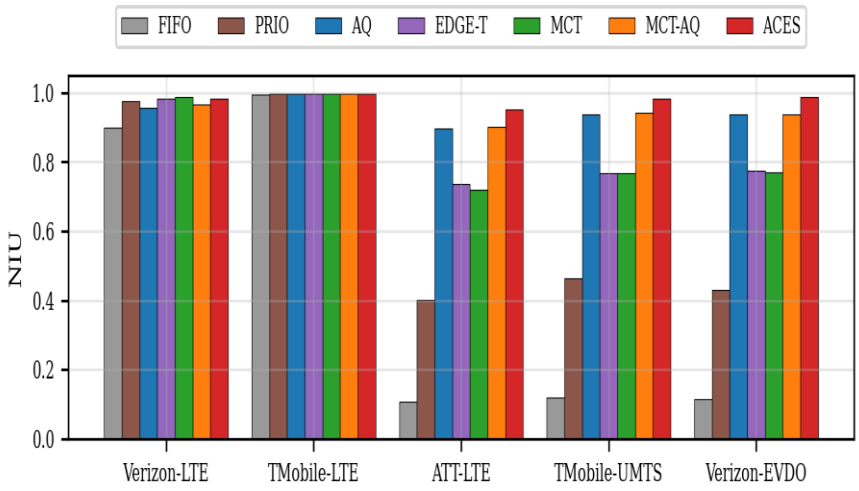


Figure 4Normalized inspection utility per uplink trace (mean of five seeds).

Table 5 compares ACES with each baseline on the same trace, offset, frame sequence and seed. ACES improves NIU over MCT-AQ by 0.031 (95% CI [0.023,0.040]) and wins 24 of 25 paired runs; the gain over AQ is 0.035 (24/25). Against MCT and EDGE-T the mean gain is larger (0.13) and the losses are confined to the LTE traces, where uploading every frame at top quality is affordable.

Table 5Paired NIU gain of ACES over each baseline (25 paired runs).

Baseline	**ΔNIU**	**95% CI**	**Wins**	**Wilcoxon *p***
FIFO	+0.533	[0.364, 0.702]	25/25	6.0e-08
PRIO	+0.326	[0.215, 0.438]	22/25	2.6e-06
AQ	+0.035	[0.027, 0.044]	24/25	1.2e-07
EDGE-T	+0.128	[0.084, 0.173]	19/25	1.6e-04
MCT	+0.132	[0.086, 0.178]	16/25	2.0e-03
MCT-AQ	+0.031	[0.023, 0.040]	24/25	1.2e-07

ACES reaches this utility by shaping quality per frame rather than per stream. It uploads 114 MB per mission against 96 MB for AQ and MCT-AQ, but its delivered results have a mean F1 of 0.741 versus 0.713, because the backlog prices compress routine frames first and keep critical and major frames at high quality; it analyzes 24% of the frames on the edge, mostly when the uplink is congested. All three classes meet their deadlines at least as often as under AQ: 96.8%, 99.5% and 100% for critical, major and routine frames.

7.2 Critical Alerts

Fig. 5 shows the latency distribution of critical results on a volatile LTE trace and on the ATT-LTE trace with a 78-s outage. FIFO queues critical alerts behind bulk frames and delivers only 34.1% of them on time; priority-based policies deliver most critical results within 1 s. ACES delivers 96.8% of critical alerts within 4 s, on par with MCT (96.8%) and within 0.7 points of MCT-AQ, while the p95 latency of 1.44 s leaves a margin of 2.6 s to the deadline. The lowest critical DSR occurs on the ATT-LTE trace (91.9%), whose 78-s outage stalls every policy.

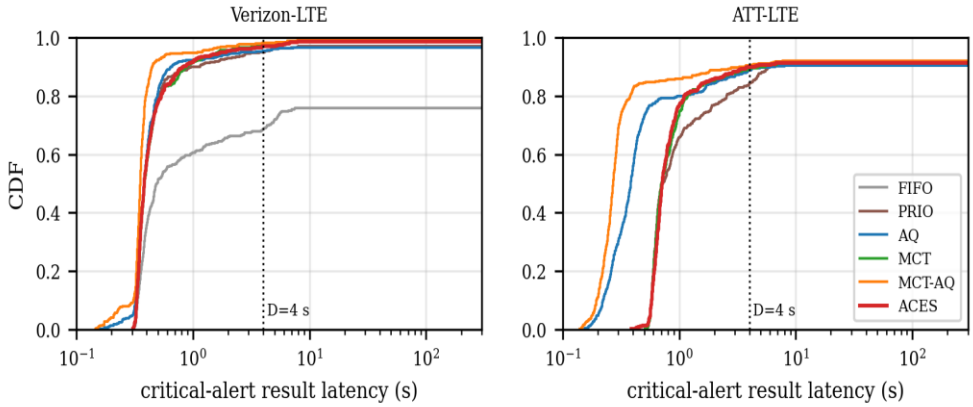


Figure 5CDF of critical-alert result latency (seed 0); undelivered alerts are placed at infinity. Dotted line: 4-s deadline.

7.3 Ablation Study

Table 6 removes one mechanism at a time (5 traces × 3 seeds), both nominally and with edge and cloud failures enabled. The version ladder is the dominant mechanism: restricting ACES to the top version or the edge reduces NIU by 0.052 (nominal) and 0.060 (failures). Edge placement adds 0.003–0.005 and damped migration 0.004–0.005; their contribution rises with failures, when prices change abruptly. Hedging, deferral, the outage-aware estimator, the timeliness term and the ordering refinement change NIU by at most 0.0005 under these conditions: the price feedback of rule (5) already moves new frames away from a stalled path, so these mechanisms act as low-cost safeguards. Table 7 probes them where they are designed to act. Value-aware eviction raises NIU by 0.016 with a 2-MB buffer, because oldest-first eviction discards frames regardless of their class, including critical frames held through an outage; with very small buffers, plain class order performs slightly better than the tiered ATC order (0.899 versus 0.890), so both orders are suitable. Under 5-Hz overload, a shared failing cloud and frequent edge crashes, all variants stay within 0.0015 of each other.

Table 6Ablation (NIU, 5 traces × 3 seeds). Failures: edge and cloud-worker MTBF 300

	Nominal		**With failures**	
Variant	NIU	**Δ**	NIU	**Δ**
ACES (full)	0.9762	+0.0000	0.9595	+0.0000
w/o version ladder	0.9246	-0.0516	0.8999	-0.0596
w/o edge placement	0.9728	-0.0034	0.9543	-0.0052
w/o migration	0.9722	-0.0040	0.9547	-0.0049
class order instead of tiered ATC	0.9765	+0.0003	0.9598	+0.0003
w/o deferral	0.9763	+0.0000	0.9595	-0.0001
w/o hedging	0.9761	-0.0002	0.9593	-0.0002
w/o outage-aware estimator	0.9767	+0.0005	0.9598	+0.0003
w/o timeliness term	0.9767	+0.0005	0.9592	-0.0003

w/o value-aware eviction	0.9762	+0.0000	0.9595	+0.0000

Table 7Safeguards under targeted conditions (NIU; ATT-LTE and T-Mobile UMTS, 2 seeds).

Variant	2-MB buffer	5 Hz load	shared failing cloud	edge MTBF 150 s
ACES (full)	0.8904	0.9396	0.9349	0.9051
w/o value-aware eviction	0.8741	0.9396	0.9349	0.9051
w/o deferral	0.8903	0.9402	0.9346	0.9049
w/o hedging	0.8909	0.9402	0.9348	0.9051
w/o outage-aware estimator	0.8920	0.9406	0.9352	0.9063
class order instead of tiered ATC	0.8986	0.9401	0.9351	0.9055

7.4 Sensitivity Analysis

Fig. 6 varies one parameter at a time on the Verizon-LTE and T-Mobile-UMTS traces (3 seeds). *Load* (a): as the capture rate grows from 1 to 5 Hz per camera, ACES declines from 0.988 to 0.968, while MCT-AQ declines from 0.973 to 0.918, so the advantage grows from 1.5 to 5.0 points; MCT and PRIO fall below 0.7. *Edge compute* (b): with slower edge cores (κ from 1.5 to 12), MCT drops from 0.990 to 0.719 because a slower edge pushes more frames onto the uplink at top quality, whereas ACES compensates with smaller versions and stays at 0.975 or above. *Buffer* (c): ACES is the best policy for every buffer size from 2 to 128 MB at 3 Hz load (0.937 at 2 MB versus 0.925 for MCT-AQ). *Deadlines* (d): halving or doubling all deadlines changes ACES's NIU by less than 0.01 and preserves its lead. *Trade-off parameter* (e): increasing V from 0.5 to 128 raises NIU from 0.9765 to 0.9845 while the uplink volume grows from 98 to 124 MB and the p95 critical latency from 1.57 to 1.81 s, the behavior predicted by Theorem 1; NIU is flat within 0.0003 over $k \in [0.5,8]$.

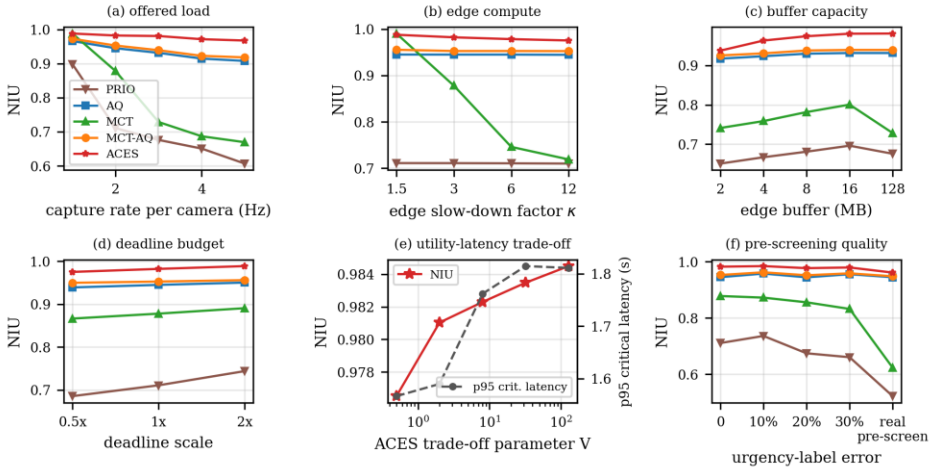


Figure 6Sensitivity (Verizon-LTE and T-Mobile-UMTS, 3 seeds): (a) capture rate, (b) edge slow-down factor, (c) edge buffer at 3 Hz, (d) deadline scaling, (e) ACES trade-off parameter V, (f) urgency-label quality.

7.5 Robustness

Edge crashes. With edge MTBF of 300 s and 150 s (downtime 30–90 s), ACES retains 0.966 and 0.948 NIU against 0.935 and 0.914 for MCT-AQ (Fig. 7a); at MTBF 600 s no crash occurred within the sampled missions. After recovery, the WAL restores the queues, and the frames recorded during downtime are priced together with the restored frames.

Bandwidth-estimation errors. Perturbing every capacity estimate by up to $\pm 75\%$ leaves ACES unchanged at 0.982 (Fig. 7b), whereas AQ and MCT-AQ, whose version choice is a direct function of $\hat{C}$, lose 0.7–0.8

points. ACES's prices depend on observed backlogs, which integrate the true service rate.

Urgency labels. Randomly corrupting 10–30% of the labels changes ACES's NIU by at most 0.005 (Table 8, Fig. 6f). With the quarter-scale pre-screen, whose severity estimate has Spearman correlation 0.32 with the annotated crack area, ACES still attains the highest NIU (0.961); the critical-alert DSR then follows the pre-screen accuracy (0.87), which motivates pairing ACES with stronger pre-screens on critical assets.

Table 8Effect of urgency-label quality (NIU / critical DSR).

Label source	PRIO	AQ	MCT	MCT-AQ	ACES
annotation (0% error)	0.711/0.91	0.945/0.95	0.878/0.97	0.953/0.98	0.982/0.97
10% random error	0.736/0.90	0.958/0.97	0.872/0.96	0.962/0.99	0.984/0.97
20% random error	0.674/0.81	0.944/0.95	0.856/0.91	0.952/0.98	0.977/0.95
30% random error	0.660/0.79	0.955/0.97	0.832/0.88	0.959/0.98	0.979/0.96
quarter-scale pre-screen	0.522/0.46	0.945/0.95	0.623/0.50	0.948/0.96	0.961/0.87

Shared, failing cloud. Fig. 7c lets 2, 4 or 8 edge nodes, each on a different trace, share a cloud with one worker per node whose workers fail with MTBF 200 s. Cloud-oblivious policies (PRIO, AQ) build cloud queues and fall to 0.53–0.81. ACES keeps 0.974–0.980, against 0.922–0.956 for MCT-AQ, with equal critical DSR (0.96–0.99), because the cloud backlog Z_C in rule (5) steers frames to edge cores while workers are down.

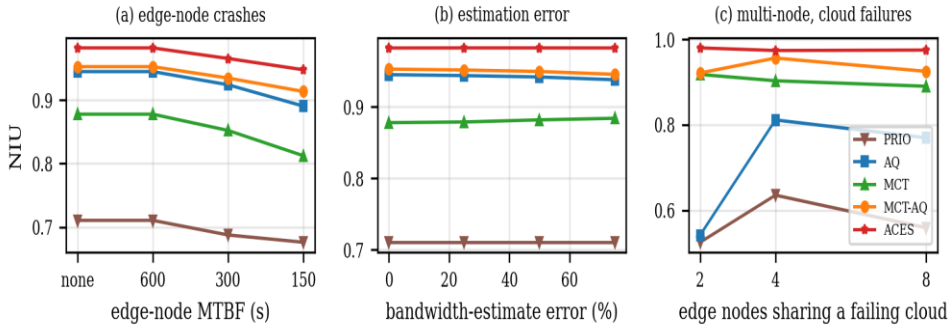


Figure 7Robustness: (a) edge-node crashes, (b) bandwidth-estimation error, (c) multiple edge nodes sharing a cloud whose workers fail.

Outage case study. Fig. 8 aligns the 78-s outage of the ATT-LTE trace with $t = 60$ s. Both ACES and MCT-AQ accumulate frames during the outage and flush the backlog within seconds after reconnection. ACES holds a larger backlog because it keeps critical and major frames at higher quality, and it achieves an NIU of 0.784 over this 240-s window versus 0.736 for MCT-AQ.

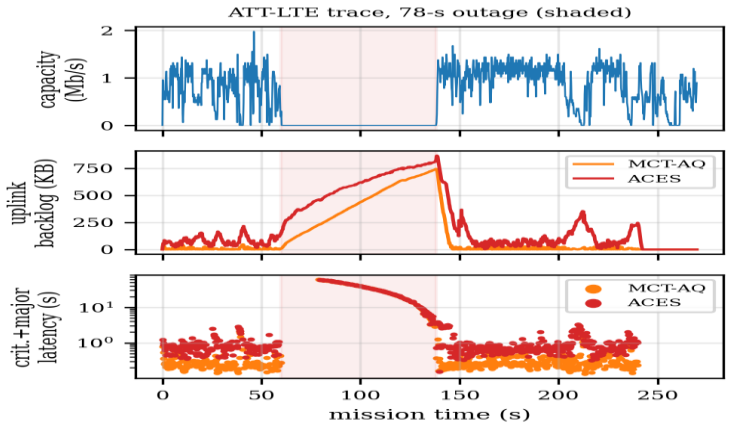


Figure 878-s outage on ATT-LTE (shaded): capacity, uplink backlog, and result latency of critical and major frames

7.6 Overhead

We timed the scheduling epoch of the Python implementation on one CPU core at 4 Hz per camera, twice the default load. An epoch took 0.19 ms on average on Verizon LTE and 0.31 ms on ATT-LTE (99th percentiles 0.91 and 0.83 ms; maximum 5.9 ms) with up to 388 pending frames, i.e., less than 0.2% of the 0.2-s epoch. Encoding the ladder adds 1–2 ms per frame, and the pre-screen 14.3 ms.

8 Discussion

Relation to adaptive execution-variant selection. Neuro-Elastic [29] selects among pre-compiled execution variants according to the current device state, balancing output quality against resource demand at the edge. ACES applies the same principle—profile the available options, then choose among them online—to inspection data with its own option set and decision rule: the 24 resolution/JPEG versions are profiled offline, the six Pareto-efficient versions form the ladder, and for each frame rule (5) chooses among these versions and edge analysis according to the uplink, edge and cloud backlogs. The version-ladder ablation (Table 6) and the edge-compute sensitivity study (Fig. 6b) quantify the benefit of this option-based selection for inspection workloads under changing communication and computation constraints.

Deployment

ACES needs only quantities an edge runtime already observes: queue lengths, a delivery-rate estimate from the transport, and acknowledgements carrying the cloud backlog. It therefore fits stream-manager style edge runtimes [5] and camera-ingestion edge appliances that analyze several IP-camera streams on premises [30] as a scheduling layer, without changes to the analysis code. The profile of Table 2 is the only application-specific input; re-profiling a few dozen frames when cameras, surfaces or extraction parameters change keeps the ladder current, as in profile-based analytics systems [7, 20].

Generality

The formulation applies to any inspection stream with versions of different size, quality and compute cost, such as bridge-deck imagery, structural vibration windows with different sampling rates, or point clouds with different densities; only the option set and the profile change.

Outlook

The paired comparison of Sect. 7 covers the six baselines of Sect. 6.3. Recent edge–cloud video analytics controllers [17–19, 21, 22] target general video workloads; evaluating them alongside ACES on inspection streams, together with field trials on 5G uplinks, additional inspection surfaces and embedded edge nodes, is the natural next step.

9 Conclusion

This paper addressed the scheduling of urban infrastructure inspection streams over volatile cellular uplinks with heterogeneous urgency, constrained edge computation and failures. ACES turns uplink, edge and cloud backlogs into prices that drive a per-frame choice of version and analysis location, re-optimizes uncommitted frames with damped migration, orders work by urgency tiers and survives outages and crashes through WAL-based buffering. A real-data evaluation with measured rate-quality-compute profiles and real cellular

traces shows a normalized inspection utility of 0.980 against 0.948 for the strongest baseline, gains up to 5 points on congested uplinks, stable behavior under failures and estimation errors, and sub-millisecond decision cost. Future work will extend the option set to multi-modal inspection data and coordinate several vehicles that share cellular cells.

References

Spencer Jr., B.F., Hoskere, V., Narazaki, Y.: Advances in computer vision-based civil infrastructure inspection and monitoring. Engineering 5(2), 199–222 (2019)

Koch, C., Georgieva, K., Kasireddy, V., Akinci, B., Fieguth, P.: A review on computer vision based defect detection and condition assessment of concrete and asphalt civil infrastructure. Advanced Engineering Informatics 29(2), 196–210 (2015)

Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Internet of Things Journal 3(5), 637–646 (2016)

Winstein, K., Sivaraman, A., Balakrishnan, H.: Stochastic forecasts achieve high throughput and low delay over cellular networks. In: Proc. USENIX NSDI, pp. 459–471 (2013)

Amazon Web Services: Manage data streams on Greengrass core devices (stream manager). AWS IoT Greengrass V2 Developer Guide, https://docs.aws.amazon.com/greengrass/v2/developerguide/manage-data-streams.html, last accessed October 2026

Fall, K.: A delay-tolerant network architecture for challenged internets. In: Proc. ACM SIGCOMM, pp. 27–34 (2003)

Zhang, B., Jin, X., Ratnasamy, S., Wawrzynek, J., Lee, E.A.: AWStream: adaptive wide-area streaming analytics. In: Proc. ACM SIGCOMM, pp. 236–252 (2018)

Jiang, J., Ananthanarayanan, G., Bodik, P., Sen, S., Stoica, I.: Chameleon: scalable adaptation of video analytics. In: Proc. ACM SIGCOMM (2018)

Zhang, S., Wang, C., Jin, Y., Wu, J., Qian, Z., Xiao, M., Lu, S.: Adaptive configuration selection and bandwidth allocation for edge-based video analytics. IEEE/ACM Transactions on Networking 30(1), 285–298 (2022)

Mao, Y., Zhang, J., Letaief, K.B.: Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE Journal on Selected Areas in Communications 34(12), 3590–3605 (2016)

Tran, T.X., Pompili, D.: Joint task offloading and resource allocation for multi-server mobile-edge computing networks. IEEE Transactions on Vehicular Technology 68(1), 856–868 (2019)

Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., Freund, R.F.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing 61(6), 810–837 (2001)

Shi, Y., Cui, L., Qi, Z., Meng, F., Chen, Z.: Automatic road crack detection using random structured forests. IEEE Transactions on Intelligent Transportation Systems 17(12), 3434–3445 (2016)

Netravali, R., Sivaraman, A., Das, S., Goyal, A., Winstein, K., Mickens, J., Balakrishnan, H.: Mahimahi: accurate record-and-replay for HTTP. In: Proc. USENIX ATC, pp. 417–429 (2015)

Sato, Y., Nakajima, S., Shiraga, N., Atsumi, H., Yoshida, S., Koller, T., Gerig, G., Kikinis, R.: Three-dimensional multi-scale line filter for segmentation and visualization of curvilinear structures in medical images. Medical Image Analysis 2(2), 143–168 (1998)

Wang, S., Urgaonkar, R., Zafer, M., He, T., Chan, K., Leung, K.K.: Dynamic service migration in mobile edge computing based on Markov decision process. IEEE/ACM Transactions on Networking 27(3), 1272–1288 (2019)

Wang, X., Shen, M.C., Yang, K.: On-edge high-throughput collaborative inference for real-time video analytics. IEEE Internet of Things Journal 11(20), 33097–33109 (2024). https://doi.org/10.1109/JIOT.2024.3424235

Cao, Z., Cheng, Y., Zhou, Z., Chen, Y., Hu, Y., Lu, A., Liu, J., Li, Z.: Edge-cloud collaborated object detection via bandwidth adaptive difficult-case discriminator. IEEE Transactions on Mobile Computing 24(2), 1181–1196 (2025). https://doi.org/10.1109/TMC.2024.3474743

Bai, T., Hou, B., Wang, Z., Liu, D., Nallanathan, A.: Content-aware joint knob configuration and resource allocation for edge video analytics. IEEE Transactions on Mobile Computing 24(6), 5536–5550 (2025). https://doi.org/10.1109/TMC.2025.3533596

Zhang, H., Ananthanarayanan, G., Bodik, P., Philipose, M., Bahl, P., Freedman, M.J.: Live video analytics at scale with approximation and delay-tolerance. In: Proc. USENIX NSDI, pp. 377–392 (2017)

Qian, B., Xuan, Y., Wu, D., Wen, Z., Yang, R., He, S., Chen, J., Ranjan, R.: Edge-cloud collaborative streaming video analytics with multi-agent deep reinforcement learning. IEEE Network 39(1), 165–173 (2025). https://doi.org/10.1109/MNET.2024.3398724

Liang, Y., Gao, G.: ASDMR: adaptive scheduling of DNN models and resources for collaborative edge-cloud video analytics. In: Proc. IEEE 31st International Conference on Parallel and Distributed Systems (ICPADS), pp. 1–10 (2025). https://doi.org/10.1109/ICPADS67057.2025.11323168

Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P.: ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Transactions on Database Systems 17(1), 94–162 (1992)

Dean, J., Barroso, L.A.: The tail at scale. Communications of the ACM 56(2), 74–80 (2013)

Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM 20(1), 46–61 (1973)

Vepsalainen, A.P.J., Morton, T.E.: Priority rules for job shops with weighted tardiness costs. Management Science 33(8), 1035–1047 (1987)

Neely, M.J.: Stochastic Network Optimization with Application to Communication and Queueing Systems. Morgan & Claypool (2010)

Wallace, G.K.: The JPEG still picture compression standard. IEEE Transactions on Consumer Electronics 38(1), xviii–xxxiv (1992)

Shang, W., Teng, D., Chen, T., Zhang, F., Ding, J.: Neuro-Elastic: a unified framework for hardware-aware adaptive quantization and dynamic sparsity in real-time edge intent prediction. IEEE Access 14, 111168–111187 (2026). https://doi.org/10.1109/ACCESS.2026.3715276

Amazon Web Services: What is AWS Panorama? AWS Panorama Developer Guide, https://docs.aws.amazon.com/panorama/latest/dev/, last accessed October 2026