Pendulum: Network-Compute Joint Scheduling for Efficient and Accurate MEC Live Video Analytics

Juheon Yi*†, Minkyung Jeong*†, Seokgyeong Shin†, Goodsol Lee†, Daehyeok Kim‡, Youngki Lee†
†Seoul National University, ‡The University of Texas at Austin
{juheon.yi, minkyung.jeong, seokgyeong.shin, youngki.lee}@hcs.snu.ac.kr, gslee2@netlab.snu.ac.kr, daehyeok@utexas.edu

***Abstract*—We present Pendulum, a live video analytics system with a novel network-compute joint scheduling in mobile edge computing (MEC) architecture. In practical scenarios, resource bottleneck frequently alternates across network (video streaming) and compute (DNN inference) stages due to independent fluctuations of wireless channel and scene content. However, prior single-stage scheduling systems suffer from throughput/accuracy fluctuation and resource wastage due to over-provisioning. To overcome the limitations, we newly leverage the interplay between video bitrate and DNN complexity to design an end-to-end system composed of (i) a resource-efficient network-compute demand curve profiler and (ii) a joint resource scheduler. Evaluation with various videos and state-of-the-art DNNs show that Pendulum achieves up to 0.64 mIoU gain and 1.29× higher throughput than state-of-the-art baselines. Pendulum also achieves near-optimal multi-user resource scheduling performance with minimal search overhead, achieving 25% cost reduction compared to the network-compute decoupled scheduling.**

I. Introduction

Live video analytics enables various services including traffic monitoring [27], surveillance [51], and AR/MR [32], [52]. Especially, recent mobile edge computing (MEC) architectures, where the server is located in the network edge close to the end-users, are enabling practical service deployment even on low-cost cameras at high throughput and low latency [35].For instance, a police agency drops CCTVs and officers with AR glasses in a city for real-time monitoring (*e.g.,* criminal chasing). Fig. 1 shows such a deployment model, where multiple clients stream video over shared cellular radio access network (RAN) to a GPU-equipped cloudlet [39] (edge server) located near the base station [40].

Live video analytics pipeline is composed of two stages: (1) *Network*: video streaming over RAN and (2) *Compute*: real-time Deep Neural Network (DNN) inference on edge server. The key to achieving high accuracy and throughput is to flexibly adjust the configuration (config) of video bitrate and DNN based on dynamic workload and resource availability, which is influenced by scene content complexity [60], [27], [58] and network bandwidth [59], [12]. However, achieving both goals is challenging due to independent fluctuation of the two factors, which causes *alternating resource bottlenecks* across network and compute stages over time (§II-A).

(*) Equal contributions. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00463802), the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2022-NR070595).

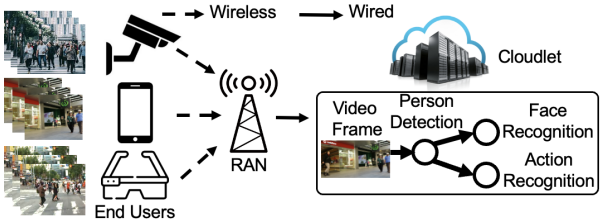


Fig. 1: Scenario: cloudlet-based person monitoring.

While previous studies have supported adaptation to dynamic contents and resources, their focus has been mostly limited to network stage-only scheduling (*i.e.,* adaptive encoding bitrate with fixed DNN) [59], [32], [12], [30].[1] These works are primarily designed for wide area network (WAN)-based cloud scenarios, where (i) the combination of wired and wireless links complicates accurate network bandwidth estimation, and (ii) the long network RTT hinders the server's ability to quickly apply bitrate adaptation decisions to the user-side video encoders, particularly under rapidly changing resource and workload conditions. Thus, they often aim at minimizing video bitrate based on conservatively estimated bandwidth, leading to throughput and accuracy drops due to limited adaptation space (§II-B1). Some research addresses compute-stage scheduling [60], [27], [41] in cloud scenarios where the bottleneck is DNN serving, and network bandwidth is sufficiently available. However, no prior studies have considered the joint scheduling of both stages in MEC scenarios. We observe that simply combining separate network and compute schedulers results in suboptimal throughput, accuracy, and resource costs (§II-B2).

We present Pendulum, an end-to-end system with a *network-compute joint scheduling* approach for efficient and accurate live video analytics in MEC. Pendulum leverages the new opportunities of MEC architectures (*i.e.,* simplified network architecture with shorter RTT) for precise network resource monitoring and agile adaptation of the end-to-end video analytics pipeline. Upon encountering a bottleneck in one stage, it conserves the resources in that stage and compensates for potential drop in inference accuracy by utilizing excess resources from the other stage. This joint scheduling approach is feasible due to the interplay between the bitrate and DNN complexity, offering two distinct benefits. First, it

[1]We refer to the single-stage scheduling as *controlling the network or compute resources independently*, albeit it indirectly affects the remaining stage (*e.g.,* adapting video resolution changes DNN inference latency).

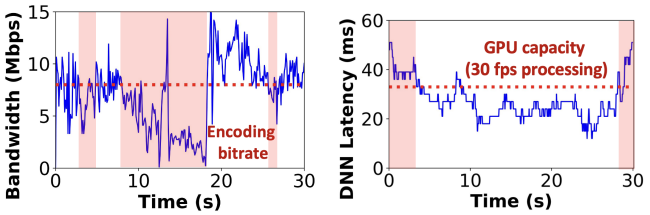
broadens the adaptation space. For example, by reducing the minimum bitrate that satisfies the app accuracy requirement (facilitated by using heavier DNNs for accuracy compensation), we can ensure a more reliable and higher throughput/accuracy than single-stage scheduling under bandwidth fluctuation. Second, it promotes a balanced allocation of network and compute resources, circumventing the need for resource over-provisioning.

While our approach sounds promising, realizing it entails the following challenges. (i) *Overhead of resource demand profiling*. {Bitrate, DNN} configs that meet the app accuracy requirement dynamically vary with the video content. Profiling them requires running multiple DNN inferences across different encoding bitrates, leading to substantial latency costs (*e.g.,* 3.4s per each profiling). (ii) *Complexity of joint resource scheduling*. It is challenging to allocate shared bandwidth and GPU resources to multiple users, each with its own set of resource demands and channel conditions. The scheduling falls into a multi-dimensional knapsack problem (NP-hard) with a large search space caused by 2D {bitrate, DNN} config space.

We address the challenges with following key ideas.

- **Dependency-aware network-compute demand curve profiling** (§IV). We design a runtime profiler that accurately estimates demand curve (*i.e.,* pareto-optimal {bitrate,DNN} configs that satisfy that accuracy requirement) with minimal overhead. It triggers profiling only upon significant content change. During each accuracy profiling event, it profiles the accuracy of a minimum number of $\{bitrate, DNN\}$ configs and interpolates the accuracies of the rest through *accuracy modeling*. Specifically, our key insight is that the accuracy along a single resource knob shows an activation–saturation shape, which can be modeled with a 3-parameter accuracy function. We also model the dependency of the network and compute stages (*i.e.,* accuracy gain from increasing DNN saturates as the bitrate becomes higher) for accurate multi-knob config space profiling.
- **Demand curve-aware joint scheduling** (§V). To make the NP-hard scheduling problem tractable, we devise an *iterative max-cost gradient algorithm* that computes an approximate solution with $O(MN)$ complexity for N users with M configs. It first finds the user-wise optimal configs, and iteratively adjusts the user with the *maximum cost gradient* (*i.e.,* maximum expected decrease in bottleneck resource usage by increasing a unit usage of the other resource) until bottleneck resolves.

We conduct extensive evaluation with various video datasets, state-of-the-art (SOTA) DNNs, and two testbeds: Linux tc [47]-based network emulator and OpenAirInterface 5G RAN software stack [38], [44]. Pendulum achieves up to 0.64 mIoU (mean Intersection-over-Union) accuracy gain (from 0.17 to 0.81) and 1.29× higher throughput compared to SOTA single-stage scheduling systems. Pendulum also achieves 25% lower resource cost than the network-compute decoupled scheduling baseline.



(a) Bitrate (red dashed line) vs. network bandwidth.

(b) GPU capacity (red dashed line) vs. DNN latency.

Fig. 2: Example bottleneck timelines (colored in red-3 network and 2 compute bottleneck events, respectively).

II. Motivation

A. Alternating Resource Bottlenecks

Network bottleneck occurs when the available bandwidth is less than the video encoding bitrate. Compute bottleneck occurs when the DNN inference workload cannot run in real-time on the available GPUs. We observe that the bottleneck events frequently alternate across two stages over time.

Study setup. We study the problem using a person analysis workload [58] in Fig. 1, composed of YOLOv8-m [55] person detection and two ResNet-50 [23]-based face and action classifiers. Total inference latency per frame is proportional to the number of people in the scene, denoted as

$$T_{total} = T_{detect} + N_{object} \cdot T_{analysis}, \quad (1)$$

where T_{detect}, $T_{analysis}$ are detection and analysis latency per each of N_{object} objects in a frame. We use MOT17-11 [29] video, with 720p@30fps, 8 Mbps encoding bitrate and LTE bandwidth trace [37]. We assume the user is allocated with 1 RTX A4500 GPU, which can process the 30fps video in real-time for up to 12 objects per frame.

Results. Figures 2(a) and (b) indicate the occurrences of the network and compute bottlenecks in red, respectively, for a 30s analysis window. Overall, there are 5 non-overlapping bottleneck events, composed of 3 and 2 network and compute bottlenecks, respectively, switching over time.

Why do bottlenecks alternate? Network and compute bottlenecks alternate as *wireless channel (network bandwidth)* and *video content (inference workload)* are highly dynamic. For example, maximum 5G uplink throughput is 107 Mbps [2] (sub-6G 100 MHz band, TDD with 5DDDSU format [20], numerology 1, single MIMO layer). With 10 contending users, each user experiences ≈10 Mbps bandwidth. Under channel fluctuation (*e.g.,* due to mobility [34], [54]), bandwidth drops below the encoding bitrate and causes network bottleneck. Video content also changes across time and location, incurring compute bottleneck in complex scenes (*i.e.,* high $N_{objects}$ in Eq. (1) that is beyond GPU processing capacity).

B. Limitations of Prior Works

1) Limitations of Single-Stage Scheduling

Throughput/accuracy drop. It is challenging to simultaneously achieve high throughput and accuracy by controlling only a single stage. Fig. 3 shows the changes in bitrate and accuracy over time when serving a MOT17 [29]-04 video

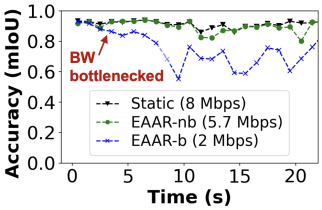


Fig. 3: Network-only scheduling performance (nb: no bottleneck, b: bottleneck).

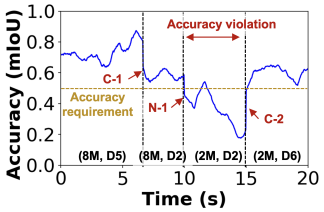


Fig. 4: Decoupled schedulers' accuracy timeline (C-1/2, N-1: compute, network scheduling events).

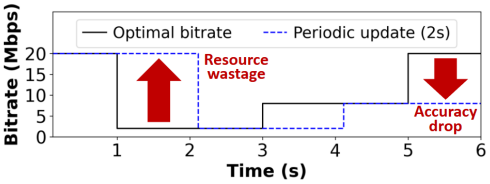


Fig. 5: Example scheduling under infrequent profiling

over an emulated RAN using EAAR [32], a network-only scheduling system that optimizes bitrate by adjusting the encoding quality of regions of a frame depending on whether or not objects were present in the previous frame. As an accuracy metric, we use the mean intersection over union (mIoU), which computes the overlap between the ground truth bounding box and the inference output. EAAR adjusts the bitrate to ≈5.7 Mbps with minimal accuracy drop compared to 8 Mbps encoding (EAAR-nb). However, when the network bandwidth is throttled to 2 Mbps at t=3s using `tc` [47], reducing the bitrate accordingly to avoid throughput bottleneck incurs a significant mIoU drop (EAAR-b).

Resource waste from over-provisioning. Allocating the right amount of resources to the non-targeted stage (*e.g.,* compute stage in the case of EAAR [32]) to avoid under/over-provisioning is also challenging. For example, in Fig. 2(b), allocating 1 GPU to the user results in a 14% latency violation rate, while allocating 2 GPUs incurs 62% resource waste. Elastic resource provisioning has higher operational costs [15] or may not be possible for edge server scenarios (*e.g.,* dedicated private cluster [26]).

2) Limitations of Existing Multi-Stage Scheduling

Limitations of simple combination of two single-stage schedulers. Simply running two network and compute-only schedulers in a decoupled manner (where each scheduler is unaware of the other) has two limitations. (i) *Throughput/accuracy drop from uncoordinated scheduling.* Fig. 4 shows an example serving accuracy timeline when serving a BDD [56] video using decoupled bitrate and DNN schedulers. The video is initially encoded at 8 Mbps, and we use EfficientDet [46]-D0–D6 detectors (D6 is the heaviest). Suppose the DNN scheduler first reduces the DNN from D5 to D2 (according to accuracy profiling results) to optimize resource usage without violating the accuracy requirement (C-1, t=6s). At t=10s, network bandwidth is throttled to 2 Mbps. Bitrate scheduler reduces the bitrate to below the bandwidth to avoid throughput drop (N-1), violating the accuracy requirement.

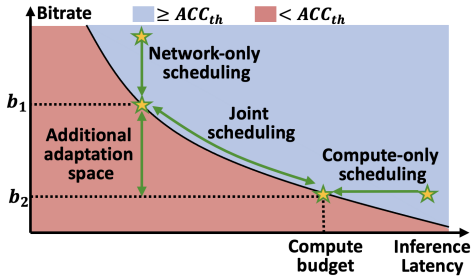


Fig. 6: Single-stage vs. joint scheduling comparison.

The problem persists until the next scheduling interval of the computer scheduler (C-2, t=15s), when it increases the DNN back to D6. (ii) *Sub-optimal resource allocation and waste.* As decoupled scheduler is unaware of different users' sensitivities to the network-compute demand curve that depends on video contents (*i.e.,* a certain user may require more resources on the remaining stage for compensation when reducing bottleneck stage resources).

Limitations of Recent multi-stage scheduling systems Simply incorporating both network and compute resources into configuration selection [27], [26], is insufficient in multi-user scenarios with shared resources. These systems optimize each user's configuration independently, without coordinating shared resources across users. This isolation can lead to inefficient system-wide resource usage—e.g., users may collectively exceed network bandwidth even when compute is underutilized, preventing all from meeting accuracy constraints.

Coordinating shared network and compute resources across users [53], is the next step in the right direction, but its effectiveness is limited without addressing the high cost of profiling. Profiling requires repeated inferences across a large configuration space (e.g., {bitrate, DNN}) and must be triggered frequently due to scene dynamics. As shown in Fig. 5, even with profiling every 2 seconds, the system suffers from both over- and under-provisioning, highlighting the difficulty of keeping up with dynamic demand at low overhead.

III. System Overview

A. Goals

High throughput and accuracy. We aim at end-to-end scheduling across end users and edge server to achieve both high throughput (*e.g.,* real-time 30 fps processing) and high DNN inference accuracy in alternating bottleneck scenarios.

Minimal operational costs. Network, compute resource costs vary depending on operators (*e.g.,* 5G 1 Mbps streaming: $0.36 [5]–$0.54 [48] per hour, cloud V100 GPU: $0.74 [19]–$0.91 [14] per hour). While achieving high throughput and accuracy, we aim to efficiently schedule resource usages to minimize operational costs.

B. Key Idea: Joint Scheduling

Our approach is to *jointly* schedule the network and compute resources. Fig. 6 shows a comparison between single-stage and joint scheduling. Network-only scheduling, which uses a fixed compute resource (DNN complexity), can only reduce

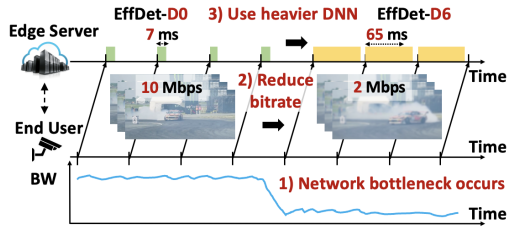


Fig. 7: Joint scheduling example for network bottleneck.

the video bitrate up to b_1 without violating the accuracy requirement (ACC_{th}). If network bandwidth drops below b_1, network bottleneck and throughput drop occurs. On the contrary, we utilize additional compute resources (*i.e.,* use heavier DNN up to compute budget) to further reduce the bitrate to as low as b_2. Fig. 7 shows an example. When network bandwidth is sufficient, we use high bitrate (10 Mbps) and lightweight EfficientDet-D0 [46] object detector (7 ms inference latency). When bandwidth drops, we reduce the bitrate accordingly to 2 Mbps for real-time streaming, and run a heavier EfficientDet-D6 (65 ms) to compensate the accuracy loss. A converse is applied upon compute bottleneck (*i.e.,* use lighter DNN and increase bitrate).

Joint scheduling is orthogonal to optimization techniques in existing single-stage scheduling systems (*e.g.,* RoI encoding [32] and frame filtering [30]) and can be generally integrated into many state-of-the-art systems. For example, EAAR [32] applies high encoding bitrate to regions where objects are predicted to exist, and low bitrate for background regions. In case of network bottleneck, we can reduce the two bitrates and use a heavier DNN to compensate for the accuracy drop. We evaluate the benefits of joints scheduling on DDS [12] and EAAR [32] in §VII-B.

C. Why is Joint Scheduling Possible?

Bitrate-DNN interplay. Joint scheduling draws from insights into the interplay between video bitrate and DNN complexity. That is, a heavier DNN with more number of layers and channels can compensate for the accuracy drop from low bitrate (and vice versa). This has been empirically observed in several prior studies (*e.g.,* EfficientNet [45]'s principles of scaling input resolution and model's layers and filters in tandem) [45], [26].

Independence of the bottleneck occurences. Network and compute bottlenecks occur from two independent factors (channel status and scene content changes), and thus mostly occur on a single stage at a time rather than simultaneously (*e.g.,* with 20% bottleneck probability each, bottlenecks occur simultaneously only 4% of the time, while bottlenecks alternate 36% of the time). Fig. 2 shows an example: Pearson correlation coefficient of the two bottleneck events is -0.04, indicating a weak correlation. Thus, surplus resource is mostly available on the non-bottleneck stage for joint scheduling (simultaneous bottleneck handling in §VI).

New opportunities in MEC. Compared to WAN-based cloud scenarios, MEC's simplified network architecture enables a more precise bandwidth monitoring. Also, shorter network RTT enables the edge server to agilely apply the bitrate adaptation decisions to the user-side video encoders, thus being able to agilely adapt the end-to-end pipeline to fast resource and workload changes.

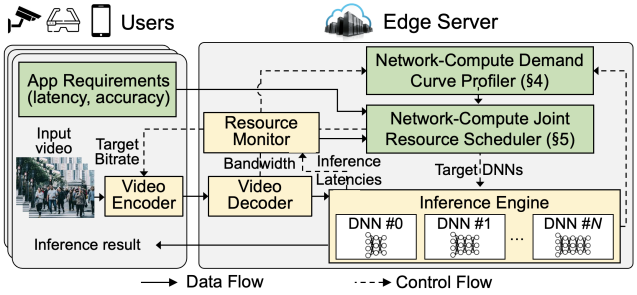


Fig. 8: Pendulum system architecture (Yellow: conventional video analytics pipeline, Green: Pendulum's components).

D. System Architecture

Figure 8 shows the system architecture of Pendulum.

Data flow. Users specify their app QoS requirements (latency, accuracy), and each user streams their live video encoded at the target bitrate. After receiving and decoding the frames, the server performs the DNN inference and aggregates the results.

Control flow. *Network-compute joint resource scheduler* finds resource allocations across users to minimize resource usage (§V). The *network-compute demand curve profiler* efficiently profiles the network-compute resource demands of users by (i) dynamically triggers profiling only during large scene changes and (ii) minimizes the number of config lookups (§IV).

IV. Network-Compute Demand Curve Profiler

In this section, we describe how Pendulum efficiently profile network-compute resource demands for joint scheduling. The profiler's goal is to analyze the Pareto-optimal demand configs of the user's live video stream. A {bitrate, DNN} config $\{b, t\}$ is considered Pareto-optimal if (i) it satisfies the accuracy constraint and (ii) no other $\{b', t\}$ and $\{b, t'\}$ exists s.t. $b' < b$, $t' < t$.

Pendulum aims to discover the full Pareto-optimal demand curve rather than a single satisfying configuration. This is particularly important in multi-user settings with shared network and compute resources, as multiple valid configurations enable flexible resource reallocation without compromising accuracy. These demand curves serve as the basis of our joint scheduling mechanism (§V).

Runtime profiling of demand curves poses two challenges: the large bitrate, DNN configuration space and the need for frequent profiling under dynamic scene changes. We address these challenges with a dependency-aware demand curve profiler (§IV-A) and a lightweight scene change detector (§IV-B).

A. Dependency-aware Demand Curve Profiler

Profiling is typically performed by scanning all candidate configs (*measurement-based profiling*). While precise, this incurs high overhead due to repeated inferences. We instead

Symbol	Description
C	Resource configuration bitrate, model
$Acc_t(C)$	Accuracy for config C at time t
sat_t	Maximum (saturation) accuracy at time t
$actv_t$	Activation threshold at time t
$curv_t$	Curve steepness at time t; controls how fast accuracy saturates

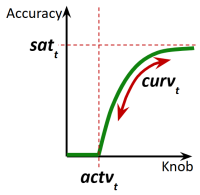


TABLE I: Notations for accuracy modeling.

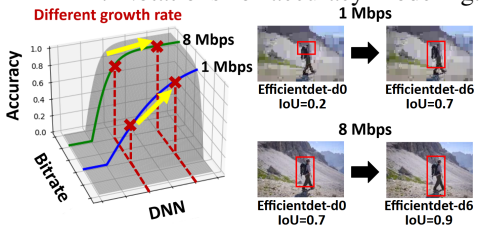


Fig. 9: Accuracy (IoU) gain from DNN increase (EfficientDet-D0 to D6) varies depending on bitrate.

adopt a lightweight *modeling-based profiling* approach. By fitting a simple model using a small number of configs (e.g., 3 for a single knob, 1 per additional knob), profiler can estimate accuracy across the full config space without exhaustive measurement. This approach reduces profiling overhead not only along individual knob axes, but also scales efficiently across multiple knobs with minimal additional cost.

1) Activation-Saturation Modeling

Accuracy along a single resource knob follows an activation–saturation pattern, where performance initially increases rapidly with improved resource quality (activation) and then plateaus as further gains diminish (saturation). This behavior can be explained from an information-theoretic perspective [17]. Following the infomax principle [31], a DNN can be viewed as an information channel whose performance reflects the information gain from input to output [3]. As input quality or model capacity increases, information gain grows and activates more layers, but eventually saturates due to limited input signal or model capacity. This pattern is also consistent with margin stability [7], where confident decision boundaries reduce the benefit of further resource increases.

We first model the accuracy along a single resource knob (e.g., bitrate or DNN) using a simple activation–saturation model (see Table I for notation).

$$Acc_t(C) = \begin{cases} 0 & if\ c < actv_t \\ sat_t \cdot tanh(curv_t \cdot (c - actv_t)). & otherwise \end{cases} \quad (2)$$

We adopt the hyperbolic tangent ($tanh$) for its smooth saturating shape and empirical effectiveness (see Fig. 22).

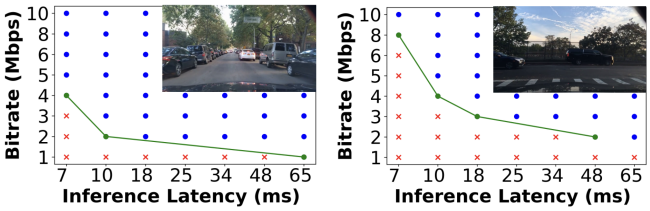
To extend this modeling approach to multiple knobs, we consider the interdependence between knobs. As shown in Fig. 9, increasing model capacity provides larger accuracy gains at low bitrates than at high bitrates, indicating that the impact of one knob depends on the setting of another. Assuming independence between knobs, as in Chameleon [27], therefore leads to estimation errors.

To reflect this, we model the multi-knob accuracy function as a joint function that captures the compound effect of all knobs, rather than treating them as independent. The accuracy

Algorithm 1 Subdivision-Based Curve Scanning Algorithm

Input: Configs C, activation threshold r_{thr}, division ratio N, minimum number of required sample min
Output: Scanned results $R = \{(c_1, r_1), ..., (c_m, r_m)\}$

$r_{golden} \leftarrow$ SCAN(c_{golden})
$actv \leftarrow$ previous $actv$ if exists else DIVIDE(c_{cheap}, c_{golden}, N)
$R \leftarrow$ SCAN($actv$)
if $R[actv] > r_{thr}$ **then** $left, right \leftarrow actv, c_{golden}$
else $left, right \leftarrow c_{cheap}, actv$
while $|R| < min$ **do**
  $next \leftarrow$ DIVIDE($left, right, N$)
  $R \leftarrow R \cup$ SCAN($next$)
  if $R[next] > r_{thr}$ **then** $left \leftarrow next$
  else $right \leftarrow next$
return R



(a) Static scene, good lighting (b) Dynamic scene, poor lighting

Fig. 10: Demand curves for different scenes. Blue/red points: configs above/below the accuracy requirement, green curve: Pareto-optimal configs.

at time t for a configuration $C = (c_1, c_2, \ldots, c_n)$ is defined as:

$$Acc_t(C) = \begin{cases} 0 & if \min\limits_{i=1,\ldots n}(c_i - actv_{t,i}) < 0 \\ sat_t \cdot tanh(curv_t \cdot \prod\limits_{i=1}^{n}(c_i - actv_{t,i})) \end{cases} \quad (3)$$

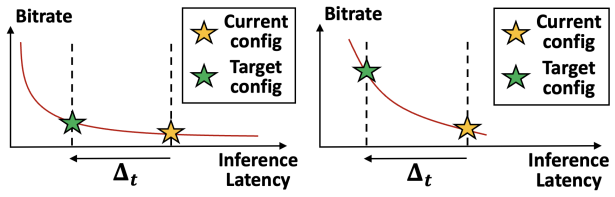
2) Model Fitting

Based on the above model, the profiler fits accuracy functions at runtime by scanning each knob independently while fixing the others to intermediate values, and then combining the results into a multi-knob model. To minimize profiling overhead, it scans only a small set of representative configurations. For a single knob, three samples suffice to fit the activation–saturation parameters, and each additional knob requires only one extra sample.

Samples are collected with a binary-style subdivision scan that repeatedly probes intermediate configurations until enough samples are obtained (Algorithm 1; subdivision factor 2). For multi-object scenes, overall accuracy is estimated by averaging models built per ground-truth bounding box.

B. Lightweight Scene Change Detector

Demand curve profiling should be triggered frequently, as the Pareto-optimal bitrate, DNN configuration dynamically changes depending on the scene content. Fig. 10 shows an example on a BDD [56] video for 8 bitrates and 7 EfficientDet [46] backbones. In a static scene (car not moving) with good lighting conditions (Fig. 10(a)), it is easy to detect objects from a low-bitrate video with lightweight backbones (small b, t values). In contrast, the values become larger for



(a) User A's tradeoff. (b) User B's tradeoff.

Fig. 11: Motivation of multi-user joint scheduling: additional bandwidth required to compensate Δ_t inference latency differs depending on the tradeoff.

dynamic scenes (moving cars) with poor lighting conditions (Fig. 10(b)).

To adapt to these changes while minimizing overhead, we trigger profiling only when a significant scene change is detected. This allows the profiler to avoid unnecessary profiling while still capturing significant shifts in demand curve. We empirically choose the following features which are highly correlated with the demand curve change: (i) motion vector sum (to detect camera motion), (ii) bounding box drift (to detect object motion), and (iii) color histogram difference (to detect lighting condition changes). The overhead of the scene change detector is negligible, as it uses low or zero-overhead features (*e.g.,* motion vectors are already available from the video codec).

V. Network-Compute Joint Resource Scheduler

In this section, we describe how Pendulum effectively solves the joint scheduling problem using the profiled network-compute tradeoff curve. Fig. 11 motivates the importance of good joint scheduling. Suppose two users (with different Pareto-optimal resource demand curves) using the configs marked as yellow stars. When a compute bottleneck occurs, the total inference latency should be reduced by Δ_t. Reducing user B's backbone requires a significantly larger bitrate increase for accuracy compensation than adjusting user A's, likely resulting in inefficient overall solutions. However, such scheduling is challenging due to the large search space; it involves $O(M^N)$ searches for N users (*e.g.,* 10s-100s), each with a large number of M configs (*e.g.,* 49 for 7×7 bitrate and backbones). This is a multi-dimensional knapsack problem (NP-hard).

A. Joint Scheduling Problem Formulation

Our goal is to dynamically allocate network and compute resources across users to minimize total resource cost, under time-varying resource availability and video contents. For each scheduling event, we assume the following inputs. Each user i has K_i accuracy-satisfying {bitrate, DNN} configurations, $C_{i,j} = (b_{i,j}, t_{i,j})$, where $j \in [1, K_i]$ and $b_{i,j}$ and $t_{i,j}$ are the candidate encoding bitrates and inference latencies of the candidate DNNs, respectively. User i experiences BW_i Mbps bandwidth, and the server has N_{GPU} GPUs, with maximum utilization time t_{th}. Given this input, we find the cost-minimizing allocation $J^* = \{j_1^*, ..., j_N^*\}$ by solving:

Algorithm 2 Iterative Max Cost Gradient Algorithm

Inputs: Accuracy-satisfying configs $C_i = \{(b_{i,j}, t_{i.j})\}$ for users $1, ..., N$, resource budgets $B_{network}, B_{compute}$.
Output: Cost-minimizing resource allocation $S = \{j_1, ..., j_N\}$
1: $S \leftarrow \{0, ..., 0\}$
2: **for** i in $1, ..., N$ **do**
3: $\quad S[i] \leftarrow GetCostOptimalConfig(C_i)$
4: **while** DetectBottleneck(S, $B_{network}$, $B_{compute}$) == $true$ **do**
5: $\quad j \leftarrow FindMaxCostGradientUser(C, S)$
6: $\quad S[j] \leftarrow AdjustConfigByStep(C[j], S[j])$
7: return S

$$\begin{aligned} \min_{J} \quad & Cost_{Network}\left(\sum\nolimits_i b_{i,j}\right) + Cost_{Compute}\left(\sum\nolimits_i t_{i,j}\right) \\ \text{s.t.} \quad & \sum\nolimits_i f_i \cdot n_i \cdot t_{i,j} \le t_{th} \cdot N_{GPU}, \quad b_{i,j} \le BW_i \\ & \forall i = 1, \ldots, N \end{aligned} \tag{4}$$

The first constraint enforces that the total inference latencies do not exceed the compute budget. The second constraint enforces that each user selects bitrate within his bandwidth constraint. The scheduling is dynamically triggered at runtime, upon users' network-compute demand curve changes or resource bottleneck occurences.

B. Iterative Max-Cost Gradient Algorithm

To make the problem tractable, we design an *iterative max-cost gradient algorithm* that (1) finds user-wise cost-optimal configs (from the demand curves obtained by the demand curve profiler in §IV) and (2) incrementally adjusts the allocation until bottleneck resolves. Algorithm 2 describes the algorithm. The scheduler takes the most recent profiled accuracy-satisfying demand curve $\{C_{i,j}\}$ for each user i along with the resource budgets $B_{network}$ and $B_{compute}$ as input, and outputs the cost-minimizing resource allocation S. First, it finds user-wise cost-optimal configs (lines 2–3). Then, it checks if the selected configs exceed the resource budgets. If a bottleneck is detected (line 4), it iteratively adjusts the config of the user with maximum cost gradient by a step until the bottleneck is resolved (lines 5–6). The cost gradient is defined as how much the resource cost of the bottleneck stage can be reduced by increasing the resource cost of the non-bottleneck stage. For example, if the network becomes a bottleneck, user i (with currently selected config j)'s cost gradient CG_i is

$$CG_i = \left| Cost_{Net}\left(b_{i,j} - b_{i,j+1}\right) / Cost_{Comp}\left(t_{i,j+1} - t_{i,j}\right) \right|, \tag{5}$$

assuming $\{(b_{i,j}, t_{i,j})\}$ is sorted ascending order of t.

Scheduling optimality. We empirically observe that the demand curves are convex as shown in Fig. 10, making Eq. (4) a convex optimization problem. Thus, our gradient-based heuristic effectively finds a near-optimal solution.

Scheduling overhead. Scheduling complexity is $O(N)$ for N users (§V) and lightweight; for example, it takes only <50 μs for N=100 users, each with 5 pareto-optimal configs each. Scheduling is triggered asynchronously with the serving path and does not affect throughput.

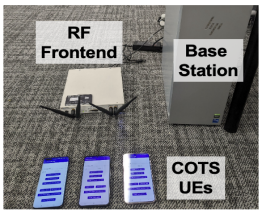
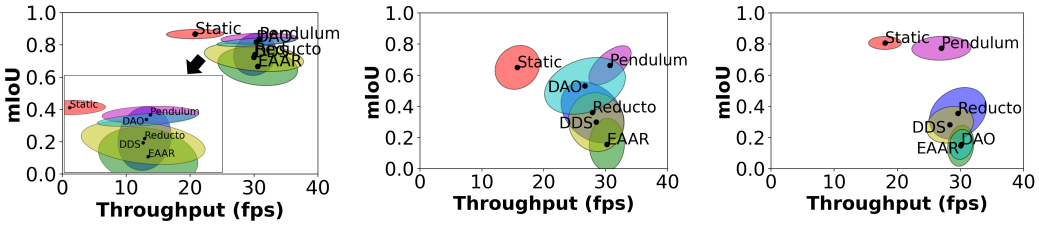


Fig. 12: Testbed implementation.



(a) MOT (moderate speed). (b) BDD (fast speed). (c) Game (very fast speed).

Fig. 13: Throughput-accuracy comparison in network bottleneck scenario.

VI. Practical Design Considerations

Handling network bandwidth fluctuations. All network-stage scheduling systems suffer from throughput/accuracy drops under unpredictable network bandwidth fluctuations. In contrast, Pendulum is more robust than the existing systems; Under bandwidth fluctuation, existing network-stage scheduling systems conservatively lower the video bitrate to avoid throughput drop [8], resulting in low inference accuracy. Pendulum follows a similar bandwidth estimation and bitrate selection policy: it estimates bandwidth as the minimum observed value within each scheduling window and selects the bitrate below it. However, Pendulum effectively compensates for accuracy by selecting a heavier backbone according to the profiled demand curve (evaluation in §VII-C).

Handling simultaneous bottlenecks. Pendulum's joint scheduler cannot find accuracy-satisfying configs when both stages are bottlenecked simultaneously (happens very rarely compared to alternating bottlenecks as analyzed in §II-A). In such a case, Pendulum falls back to prior systems by reducing both stages' resource usages below bottlenecks for real-time processing and best-effort accuracy.

VII. Evaluation

Implementation. We implement Pendulum on a server equipped with Intel Xeon Gold 5128 CPU and 8× RTX 2080 Ti GPUs. We use TensorFlow 2.6.2 C API + CppFlow [11] and PyTorch 1.10.1 C++ API for DNN inference. We use Secure Reliable Transport (SRT) [42] for live video streaming. We use FFMpeg 4.1.9 and H.264 for video encoding along with OpenCV 4.4.0 for image processing. For repeatable experiments, we use Linux `tc` [47] to shape network bandwidth following real-world LTE bandwidth traces [37]. We also validate the performance of Pendulum on over-the-air 5G MEC testbed (Fig. 12), composed of USRP X310 (TDD n78 band with 5DDDSU and single MIMO layer following [20]) and Google Pixel 6 with programmable SIM cards (sysmoISIM-SJA2) [44].

Datasets. We use three datasets with different scene contents and change speeds: five 30 fps CCTV videos (02, 04, 09, 10, 11) for MOT [29], 6 dashcam videos from BDD [56], and self-collected racing game videos (Games) from YouTube. All videos are scaled to 720p. We observe consistent results across datasets and report results on BDD unless specified.

Tasks and DNNs. We use two tasks: object detection and segmentation. For detection, we use YOLOv5 with 5 backbones (n/s/m/l/x), and EfficientDet with 7 backbones (D0-D6) [46]. For segmentation, we train FPN [28] with 7 EfficientNet backbones (B0-B6) [45] on BDD. Unless stated otherwise, we report performance using EfficientDet.

Single-stage baselines. Static uses a fixed {bitrate, backbone} config. **DDS [12]** uses two-path streaming (low-quality probe frame + high-quality feedback for regions with low-confidence inference results). **EAAR [32]** uses dynamic RoI encoding (high quality only for regions where objects existed in the previous frame) and motion vector-based frame filtering. **Reducto [30]** uses pixel/edge/area feature difference-based frame filtering (feature type is offline profiled per each task). **DAO [36]** adapts bitrate based on a lightweight predictor's accuracy estimation results (assuming a static DNN). **Backbone Adaptation (BA)** only adapts the DNN backbone (based on our profiler in §IV) and uses a fixed bitrate. This is equivalent to single-knob Chameleon [27].

Multi-stage baseline. Pendulum**-Decoupled** is a decoupled joint scheduler. When the network becomes a bottleneck, all users reduce their bitrates by the same amount until the bottleneck is resolved. Then, the server compensates for the accuracy drop by choosing the cost-optimal backbones that satisfy their accuracy requirements.

A. End-to-End Improvement

Throughput-accuracy. Fig. 13 compares the throughput-accuracy of Pendulum against baselines across three datasets. Ellipses show the 1-σ range of the results. Static, which uses a fixed (4 Mbps, EfficientDet-D1), suffers from throughput drops due to network bottlenecks. Overall, Pendulum consistently achieves ≈30 fps throughput and higher accuracy compared to the baselines: up to 0.64 mIoU gain (Game, Pendulum: 0.81 vs. EAAR: 0.17). Performance of baselines varies depending on the dataset. For MOT (moderate scene changes), all baselines effectively optimize the bitrate to below 3 Mbps (*e.g.,* EAAR: 2.98 Mbps), achieving comparable accuracy to Pendulum. However, for BDD and Game with faster scene changes, the accuracy drops significantly, especially for EAAR and DDS. For EAAR, the object region from the previous frame becomes highly stable. For DDS, fast-changing video encoded in a low-bitrate probe stream (*e.g.,* 1 Mbps) suffers from severe quality degradation, resulting in inaccurate DNN inference and feedback frame requests. Consequently, both

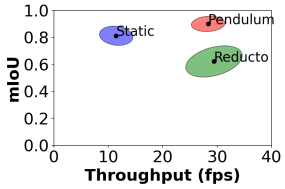


Fig. 14: Over-the-air performance.

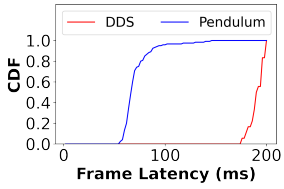
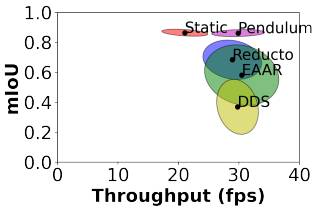
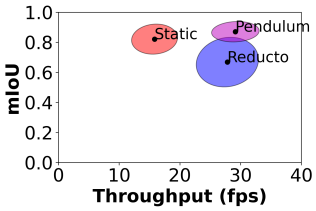


Fig. 15: Frame-wise latency comparison.



(a) YOLOv5 (detection), MOT.



(b) FPN (segmentation), BDD.

Fig. 16: Performance across various tasks & DNNs.

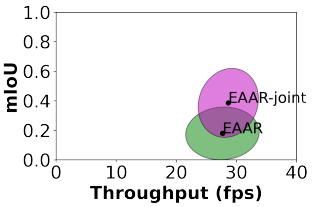


Fig. 17: Joint scheduling on EAAR [32].

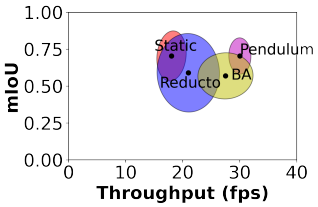


Fig. 18: Performance in compute bottleneck.

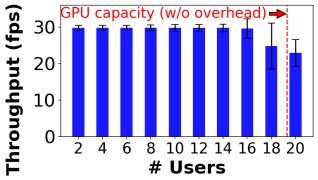


Fig. 19: Throughput for different number of users.

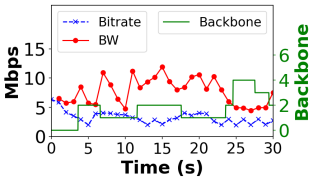


Fig. 20: Robustness under bandwidth fluctuation.

end up streaming the entire frame at low quality. Pendulum achieves higher mIoU even when DDS and EAAR use the heaviest D6 backbone (*e.g.,* 0.71 vs. 0.49, 0.48 in BDD).

Over-the-air performance. Fig. 14 shows the throughput-accuracy performance in the real physical channels using our USRP-based RAN implementation. We observe that Pendulum effectively handles network bottlenecks and achieves high throughput and accuracy compared to baselines.

Frame-wise latency. Pendulum also achieves low frame-wise latency. Fig. 15 shows that Pendulum achieves <100 ms per-frame latency, much less than DDS with two frame transfers and DNN inference per frame.

B. Performance on Various App Settings

Various DNNs. We repeat the same experiment as in Fig. 13(a), but with YOLOv5 detectors. We see a similar trend: Pendulum achieves 0.17, 0.32, and 0.52 higher mIoU than Reducto, EAAR, and DDS, respectively. Note that DDS's mIoU is lower than when using the EfficientDet backbones because the lightweight YOLOv5 backbone yields less accurate feedback regions on low-bitrate videos.

Various tasks. Fig. 16(b) also shows that Pendulum achieves similar performance for FPN segmentation models (*e.g.,* ≈30 fps throughput with 0.19 higher mIoU than Reducto).

Performance in compute bottleneck. Fig. 18 shows Pendulum's performance in compute bottleneck scenario on MOT Static (2 Mbps, EfficientDet-D6) suffers from throughput drop. Compared to Reducto and BA which only reduce compute resource usage either by reducing the number of frames (inferences) or backbone, Pendulum effectively increases the bitrate (from 1.98 to 5.42 Mbps) resulting in 0.12 higher mIoU and 1.29× higher throughput.

Joint scheduling on SOTA systems. We show the benefits of joint scheduling on existing system: EAAR [32]. Fig. 17(a) and (b) show the results on the BDD dataset. Joint scheduling achieves 0.17 and 0.20 higher mIoU than baseline DDS and EAAR, respectively, demonstrating its generalizability.

C. System Scalability and Robustness

Scalability. Figure 19 shows the system throughput with different numbers of users. To maximize the number of users that can be served, we use lightweight YOLO, and incur no network bottleneck. Compared to theoretical GPU capacity (number 30fps video streams that can be served in real-time assuming no profiling overhead), Pendulum achieves 29.56±2.66 fps with up to 16 users (throughput drops from 18 users due to profiling overhead), demonstrating that the system overhead is minimized and provide high scalability.

Robustness. To evaluate Pendulum's robustness under real-world bandwidth fluctuations, we shape the network bandwidth with tc using the real-world LTE trace in MahiMahi [37] ('Verizon-LTE-short', mean 7.25 Mbps, stdev 2.77 Mbps). Figure 20 shows the timeline. When bandwidth fluctuates in $t = 5s$ to $25s$ (red line), including a sudden drop at $t = 10s$, Pendulum refrains from increasing bitrate and selects conservative values so as not to incur network bottleneck. Instead, it selects a heavier backbone when necessary (due to scene content changes) to satisfy the accuracy requirement (§**??**).

D. Microbenchmarks

1) Network-Compute Demand Profiler

Profiling overhead. Figure 21 shows the comparison of profiling performance (overhead: ratio of profiling inference time to total inference time; accuracy: average mIoU across served frames). Compared to the fixed-interval full-search baseline (1 frames per 2s window, 2.79% overhead), AWStream [59] (full-search when current config's accuracy significantly drops) reduces overhead to 2.65% by skipping redundant profiling events. Chameleon [27], which uses dependency-agnostic interpolation, incurs 0.94% overhead with interpolation-based estimation. Our modeling-based profiler achieves the lowest overhead (0.58%), with only a marginal loss in mIoU.

Demand modeling functions. Figure 22 compares saturation functions for modeling demand on BDD. All achieve negligible fitting time compared to inference (e.g., ˜7.5ms), but

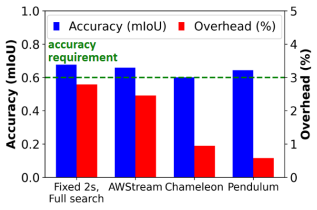


Fig. 21: Profiling overhead of various profilers.

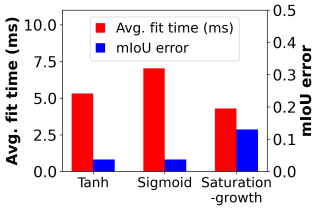


Fig. 22: Performance across demand modeling functions.

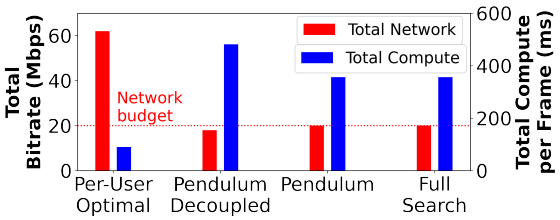


Fig. 23: Resource cost comparison of various schedulers.

Tanh results in the lowest mIoU error and is selected for our profiling design.

2) Iterative max-cost gradient scheduler

Figure 23 compares the resource cost of various schedulers given the same accuracy requirement (mIoU 0.5). We assume 10 users with randomly sampled demand curves from MOT and BDD (each with 4-6 Pareto-optimal configs, bitrate range in 1-10 Mbps and EfficientDet-D0-D6 backbones). We assume a network bottleneck scenario, where the total bitrate sum should not exceed 20 Mbps. We use linear cost models with unit network, compute costs set as \$0.36 and \$0.74 [18], [6]. *Per-User Optimal* chooses user-wise cost-optimal configs independently, resulting in severe network bottleneck. Our iterative max cost gradient algorithm finds the optimal solution (same with the full search due to empirical optimality guarantee in §V), with only 0.004% searches. It also reduces the total cost by 25% compared to Pendulum-**Decoupled**: this is because **Decoupled** is unaware of the network-compute demand curves and reduces the bitrate of users with low-cost gradients (*i.e.,* require large inference latency increase for accuracy compensation), whereas Pendulum efficiently chooses users with high-cost gradients.

VIII. Limitations and Future Work

Fluctuation in server compute resources. Pendulum currently assumes a dedicated edge server without background tasks. Thus the inference latency for each DNN is constant and can be profiled offline when solving the joint scheduling problem in Eq. (4). In practical scenarios, this assumption may not hold, as resource contention from background tasks may cause inference latency fluctuation (*e.g.,* due to context switching overhead [21]). We leave precise inference latency modeling and control in such scenarios to future work.

Generality to Video Language Models (VLMs). While Pendulum focuses on CNN-based video analytics pipelines, its core insight—the compensatory tradeoff between network and compute resources—also appears in VLM-based pipelines.

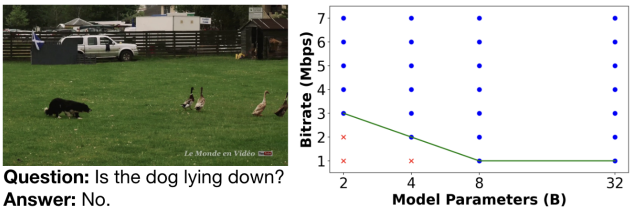


Fig. 24: Demand curve for a vision-language model (Qwen3-VL) on a vision question answering task.

We conducted a simple exploratory experiment using Qwen3-VL models [1] on a DAVIS video, varying the input video bitrate and model size for a fixed query. As shown in Fig. 24, lower bitrates require larger models to maintain correctness, suggesting that VLMs also expose a joint network–compute adaptation space. Extending Pendulum to such pipelines is left for future work.

IX. Related Work

Live video analytics. Live video analytics enables various useful apps including action/traffic monitoring [27], [33] and AR/MR [51], [32], [52]. Pendulum enables various apps in practical alternating resource bottlenecks.

Adaptive bitrate for live video analytics. A large body of works has designed adaptive bitrate techniques for live video analytics by controlling resolution [25], frame rate (filtering) [30], [9], [24], quantization [32], and a combination of all [59]. Other works have designed RoI filtering and streaming systems [61], [12] and super-resolution-enhanced streaming pipelines [57]. Quantization table optimization for DNNs [13], [62] has also been studied. However, they were limited to *network-only scheduling*.

Live video analytics serving on cloud. Several works aimed at high-throughput inference serving on cloud with content-aware adaptation [27], priority scheduling [60], [41], [16], [43], caching [22], [63], or multi-edge workload balancing [58], [50]. However, they are mostly *compute-only scheduling* (assume that videos arrive at the cloud without delay), lacking scalability in network bottlenecks.

Adaptive object detection on edge devices. Several works developed compute control knobs for adaptive object detection on mobile devices (*e.g.,* resolution [10], fps [49], [4]). Pendulum can also be leverage them as joint scheduling knobs.

X. Conclusion

We presented Pendulum, an end-to-end live video analytics system in MEC with network-compute joint scheduling. To overcome the limitations of single-stage scheduling systems in alternating resource bottleneck scenarios, we leverage the interplay between the video bitrate and DNN complexity. Based on this, we design Pendulum composed of (i) a joint scheduling mechanism (to estimate network, compute resource demands and availabilities as well as control resource usages), and (ii) joint resource scheduler. Pendulum achieves up to 0.64 mIoU gain and 1.29× higher throughput compared to state-of-the-art single-stage scheduling systems.

REFERENCES

[1] Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).

[2] 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: 1 Aug. 2025.

[3] ACHILLE ET AL., A. Where is the information in a deep neural network? *arXiv:1905.12213* (2019).

[4] APICHARTTRISORN ET AL., K. Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In *ACM SenSys* (2019).

[5] AT&T Cell Phone Plans. https://www.att.com/5g/consumer/. Accessed: 1 Aug. 2025.

[6] AWS Lambda Serverless Computing. https://aws.amazon.com/ko/lambda/pricing/. Accessed: 1 Aug. 2025.

[7] BARTLETT, P. L., FOSTER, D. J., AND TELGARSKY, M. J. Spectrally-normalized margin bounds for neural networks. *Advances in neural information processing systems 30* (2017).

[8] CARLUCCI ET AL., G. Analysis and design of the google congestion control for web real-time communication (webrtc). In *ACM MMSys* (2016).

[9] CHEN ET AL., T. Y.-H. Glimpse: Continuous, real-time object recognition on mobile devices. In *ACM SenSys* (2015).

[10] CHIN ET AL., T.-W. Adascale: Towards real-time video object detection using adaptive scaling. *PMLR* (2019).

[11] CppFlow. https://github.com/serizba/cppflow. Accessed: 1 Aug. 2025.

[12] DU ET AL., K. Server-driven video streaming for deep learning inference. In *ACM SIGCOMM* (2020).

[13] DU ET AL., K. Accmpeg: Optimizing video encoding for accurate video analytics. *MLSys* (2022).

[14] Amazon EC2 Pricing. https://aws.amazon.com/ko/ec2/pricing/on-demand/. Accessed: 1 Aug. 2025.

[15] EC2 on-demand vs. reserved instance pricing. https://aws.amazon.com/compare/the-difference-between-on-demand-instances-and-reserved-instances/?nc1=h_ls. Accessed: 1 Aug. 2025.

[16] FANG ET AL., B. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *ACM MobiCom* (2018).

[17] GOLDFELD ET AL., Z. The information bottleneck problem and its applications in machine learning. *IEEE Journal on Selected Areas in Information Theory* (2020).

[18] Google Cloud Serverless Computing. https://cloud.google.com/serverless?hl=en. Accessed: 1 Aug. 2025.

[19] Google Cloud Pricing. https://cloud.google.com/compute/gpus-pricing. Accessed: 1 Aug. 2025.

[20] 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: 1 Aug. 2025.

[21] GUJARATI ET AL., A. Serving DNNs like clockwork: Performance predictability from the bottom up. In *USENIX OSDI* (2020).

[22] GUO ET AL., H. Crossroi: cross-camera region of interest optimization for efficient real time video analytics at scale. In *ACM MMSys* (2021).

[23] 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.

[24] HSIEH ET AL., K. Focus: Querying large video datasets with low latency and low cost. In *USENIX OSDI* (2018).

[25] HU ET AL., J. Banner: An image sensor reconfiguration framework for seamless resolution-based tradeoffs. In *ACM MobiSys* (2019).

[26] HUNG ET AL., C.-C. Videoedge: Processing camera streams using hierarchical clusters. In *IEEE/ACM SEC* (2018).

[27] JIANG ET AL., J. Chameleon: scalable adaptation of video analytics. In *ACM SIGCOMM* (2018).

[28] KIRILLOV, A., HE, K., GIRSHICK, R., AND DOLLÁR, P. A unified architecture for instance and semantic segmentation, 2017.

[29] LEAL-TAIXÉ ET AL., L. Motchallenge 2015: Towards a benchmark for multi-target tracking. *arXiv preprint arXiv:1504.01942* (2015).

[30] LI ET AL., Y. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *ACM SIGCOMM* (2020).

[31] LINSKER, R. Self-organization in a perceptual network. *Computer* (1988).

[32] LIU, L., LI, H., AND GRUTESER, M. Edge assisted real-time object detection for mobile augmented reality. In *ACM MobiCom* (2019).

[33] LIU ET AL., X. Caesar: cross-camera complex activity recognition. In *ACM SenSys* (2019).

[34] MAO ET AL., H. Neural adaptive video streaming with pensieve. In *ACM SIGCOMM* (2017).

[35] ETSI. 2018. MEC in 5G Networks. https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf. Accessed: 1 Aug. 2025.

[36] MURAD ET AL., T. Dao: Dynamic adaptive offloading for video analytics. In *ACM MM* (2022).

[37] NETRAVALI ET AL., R. Mahimahi: Accurate Record-and-Replay for HTTP. In *USENIX ATC* (2015).

[38] NIKAEIN, N., MARINA, M. K., MANICKAM, S., DAWSON, A., KNOPP, R., AND BONNET, C. Openairinterface: A flexible platform for 5g research. *ACM SIGCOMM Computer Communication Review* (2014).

[39] SATYANARAYANAN ET AL., M. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing* (2009).

[40] Microsoft and AT&T demonstrate 5G-powered video analytics. https://azure.microsoft.com/en-us/blog/microsoft-and-att-demonstrate-5gpowered-video-analytics/. Accessed: 1 Aug. 2025.

[41] SHEN ET AL., H. Nexus: A gpu cluster engine for accelerating dnn-based video analysis. In *ACM SOSP* (2019).

[42] Secure Reliable Transport (SRT) Protocol. https://github.com/Haivision/srt. Accessed: 1 Aug. 2025.

[43] SUN ET AL., L. Biswift: Bandwidth orchestrator for multi-stream video analytics on edge. In *IEEE INFOCOM* (2024).

[44] 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: 1 Aug. 2025.

[45] TAN, M., AND LE, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *PMLR* (2019).

[46] TAN, M., PANG, R., AND LE, Q. V. Efficientdet: Scalable and efficient object detection. In *IEEE/CVF CVPR* (2020).

[47] Linux traffic control. https://man7.org/linux/man-pages/man8/tc.8.html. Accessed: 1 Aug. 2025.

[48] T-Mobile Cell Phone Plans. https://www.t-mobile.com/cell-phone-plans. Accessed: 1 Aug. 2025.

[49] XU ET AL., R. Approxdet: content and contention-aware approximate object detection for mobiles. In *ACM SenSys* (2020).

[50] YAN ET AL., Y. Visflow: Adaptive content-aware video analytics on collaborative cameras. In *IEEE INFOCOM* (2024).

[51] YI ET AL., J. EagleEye: Wearable camera-based person identification in crowded urban spaces. In *ACM MobiCom* (2020).

[52] YI ET AL., J. Heimdall: mobile gpu coordination platform for augmented reality applications. In *ACM MobiCom* (2020).

[53] YI ET AL., J. Towards end-to-end latency guarantee in mec live video analytics with app-ran mutual awareness. In *ACM MobiSys* (2025).

[54] YIN ET AL., X. A control-theoretic approach for dynamic adaptive video streaming over http. In *ACM SIGCOMM* (2015).

[55] Ultralytics YOLOv5. https://github.com/ultralytics/yolov5. Accessed: 1 Aug. 2025.

[56] YU ET AL., F. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv:1805.04687* (2018).

[57] YUAN ET AL., T. Accdecoder: Accelerated decoding for neural-enhanced video analytics. In *IEEE INFOCOM* (2023).

[58] ZENG ET AL., X. Distream: scaling live video analytics with workload-adaptive distributed edge intelligence. In *ACM SenSys* (2020).

[59] ZHANG ET AL., B. Awstream: Adaptive wide-area streaming analytics. In *ACM SIGCOMM* (2018).

[60] ZHANG ET AL., H. Live video analytics at scale with approximation and delay-tolerance. In *USENIX NSDI* (2017).

[61] ZHANG ET AL., W. Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading. In *ACM MobiCom* (2021).

[62] ZHU ET AL., A. Adastreamer: Machine-centric high-accuracy multi-video analytics with adaptive neural codecs. In *IEEE INFOCOM* (2024).

[63] ZHU ET AL., A. Crucio: End-to-end coordinated spatio-temporal redundancy elimination for fast video analytics. In *IEEE INFOCOM* (2024).