Better Never Than Late: Timely Edge Video Analytics Over the Air

Vinod Nigade*
VU Amsterdam

Ramon Winder*
VU Amsterdam

Henri Bal
VU Amsterdam

Lin Wang
VU Amsterdam & TU Darmstadt

ABSTRACT

Edge video analytics based on deep learning has become an important building block for many modern intelligent applications such as mobile augmented reality and autonomous driving. Various mechanisms have been developed to handle dynamic wireless networks, compute resource availability, and achieve high analytics accuracy via filtering, DNN compression, pruning, and adaptation. So far, limited attention has been paid to timeliness—providing strict service-level objectives (SLO) for edge video analytics pipelines, which is essential for the usability of user-interactive and mission-critical intelligent applications. In this paper, we analyze the challenges in achieving SLO for edge video analytics and present a system design for timely edge video analytics over the air leveraging a simple yet effective idea—feedback control. Our preliminary evaluation based on a system prototype and real-world network traces shows the potential of our design. We also discuss the limitations, calling for future work.

CCS CONCEPTS

• **Computer systems organization → Real-time system architecture**; • **Networks → Application layer protocols**; • **Human-centered computing → Ubiquitous and mobile computing**.

KEYWORDS

video analytics, edge computing, SLO guarantee

ACM Reference Format:
Vinod Nigade, Ramon Winder, Henri Bal, and Lin Wang. 2021. Better Never Than Late: Timely Edge Video Analytics Over the Air. In *The 3rd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT 21), November 15–17, 2021, Coimbra, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3485730.3493446

1 INTRODUCTION

The rapid development of Internet-of-Things (IoT) has led to the proliferation of modern intelligent applications driven by the recent advances of artificial intelligence (AI), such as augmented reality (AR), intelligent personal assistants, and autonomous driving [1, 2, 7, 21]. A remarkable number of these applications base their functionality on video analytics—analyzing continuous video streams generated by mobile and IoT devices to understand the environment [3]. Powered by advanced deep learning techniques [13, 26], video analytics typically relies on sophisticated deep neural networks (DNNs) that demand intensive computation. This poses critical challenges for the deployment of emerging intelligent applications that are based on video analytics, considering the generally limited capability of mobile and IoT devices.

Edge computing is deemed a promising computing paradigm for video analytics [28]. The main idea of edge computing is to deploy computing resources at the network edge (e.g., cellular base stations, wireless access points) to support computation-intensive intelligent applications. The edge platform helps reduce network latency and bandwidth consumption in the wide area, which are critical limitations of cloud computing. Meanwhile, edge computing provides more computational power for the deployment of intelligent applications with state-of-the-art DNNs, when compared with on-device computing.

Despite its high promise, edge computing also brings new critical challenges for video analytics. On the one hand, the edge-based setup introduces high system dynamics: (1) The video stream has to be transferred over a highly variable wireless network before being processed. (2) The edge platform may be shared with other concurrent processing tasks, which can bring resource contention, thus resulting in performance uncertainty. On the other hand, a variety of modern intelligent applications based on video analytics require timely processing. For example, user-interactive applications like virtual reality require that the motion-to-photon latency is bounded by 20ms [6, 21]. Other time-critical applications like autonomous driving also enforce such constraints to ensure safety [1]. Overall, these applications impose strict service-level objectives (SLO) on the end-to-end latency in order to be usable in production environments. Unfortunately, the SLO guarantee for edge video analytics has been overlooked in the literature so far.

In this paper, we make a case for *timely edge video analytics*, i.e., providing strict SLO defined on the end-to-end latency, including both the network transfer and the DNN inference serving stages, for edge video analytics pipelines. We identify the challenges in achieving SLO guarantees in edge video analytics and perform analysis on how the existing solutions fall short (Section 2). Based on the analysis, we propose a design based on the idea of feedback control, where the system adapts itself towards the goal of SLO guarantee by exploiting smoothed history information (Section 3). We build a system prototype and conduct experiments with real-world network traces. Our experimental results show that our design based on feedback control while being simple, is effective

*Both authors contributed equally to this research.

AIChallengeIoT 21, November 15–17, 2021, Coimbra, Portugal

ACM ISBN 978-1-4503-9097-2/21/11.
https://doi.org/10.1145/3485730.3493446

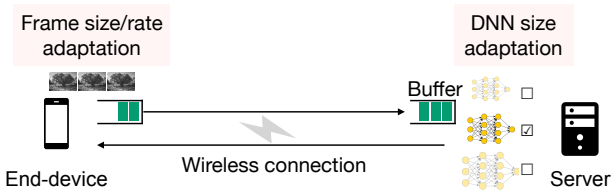

Figure 1: A typical edge video analytics pipeline.

in achieving strong SLO guarantees (Section 4). We also discuss the limitations of our design (Section 5) and present related work (Section 6), before concluding the paper (Section 7).

2 BACKGROUND AND MOTIVATION

In this section, we first present background information about edge video analytics. Then, we proceed to identify the challenges in the SLO guarantee for edge video analytics. Finally, we analyze the research gap in the literature and motivate our work.

2.1 Edge Video Analytics

The widespread adoption of high-resolution cameras and the rapid advancements of deep learning techniques have rendered deep learning based video analytics a key component in a variety of modern intelligent applications such as AR and autonomous driving [1, 21]. On the one hand, these deep learning techniques involve sophisticated DNNs to achieve high analytics accuracy, which typically requires intensive computation. On the other hand, the usability of these applications is subject to the service-level objective (SLO) on the latency [12, 17]. However, end-devices are typically equipped with limited computing resources due to portability and battery limitations. Thus, achieving high usability without dramatically sacrificing analytics accuracy for real-time video analytics is unlikely with on-device computing.

Edge video analytics leverages computing resources deployed at the network edge, e.g., at the cellular base stations and wireless access points, to perform video data processing. The end-device is connected to the edge platform with a one-hop wireless connection. An overview of a typical edge-based video stream analytics system is depicted in Figure 1. Due to the variability of the wireless network performance, achieving the highest analytics accuracy constantly in edge video analytics is hardly possible. To handle system dynamics and optimize analytics accuracy, *adaptive* approaches allows the system to adapt its video frame size/rate together with the employed DNN (e.g., from a pool of DNNs optimized at different levels with techniques including pruning and quantization), which conducts a tradeoff between the resource consumption and the analytics accuracy at runtime.

2.2 Being on Time Is Hard

Providing strict SLO on the end-to-end latency in edge video analytics is hard, and we identify the following challenges:

C1 Network variability of the wireless connection. The condition of the network connection (e.g., bandwidth and round-trip time, RTT) between the end-device and the edge platform suffers high variability due to the nature of the wireless connection which is susceptible to end-device mobility and environmental interference [14, 31]. This is confirmed by our collected WiFi traces (see Figure 5(top)) measured with a smartphone while moving around. The peak-to-valley bandwidth ratio can be as large as 5×. As a result, the time for sending a video frame over the network is also highly variable and hard to predict.

C2 Performance volatility of the edge platform. The edge platform consists of powerful but still limited computing resources which are likely to be shared among multiple processing tasks from different users/applications due to proximity considerations, i.e., users receive edge resources from the location closest to the end-device. Without sophisticated performance isolation mechanisms which are unlikely to be implemented at the resource-constrained edge, co-located processing tasks can incur significant performance interference with each other. This problem has been highlighted in Figure 5 in [30] where resource contention turns out to be a major source of system performance volatility. Thus, the time it takes to process an inference request (e.g., a video frame or a window of frames) can show a high level of uncertainty.

C3 Cascading effect of system adaptation decisions. The edge video analytics pipeline consists of two major steps, namely data transfer and DNN inference, with buffers holding pending inference requests before each of these steps, as shown in Figure 1. Consequently, the adaptation decision made for the current video frame may negatively affect those for future video frames in a complex manner due to the well-known "bufferbloat" issue. Such a cascading effect imposes extremely high complexity in system adaptation decision-making for SLO guarantee.

C4 DAG-based DNN inference serving. As discussed, edge video analytics involves both the data transfer and the DNN inference steps. For the DNN inference step, many existing works assume that only one DNN is involved in the analytics. Alternatively, the application can involve a more complex inference serving logic where multiple DNNs are orchestrated into a directed acyclic graph (DAG) for serving inference requests. Achieving strict SLO thus requires appropriate coordination among all the DNNs involved in the analytics pipeline, which further adds complexity to system adaptation.

2.3 Where Are We Standing?

Existing solutions to video analytics can be generally categorized into two directions: adaptive video analytics and DNN inference serving. The former concerns system adaptation to conduct tradeoffs between analytics accuracy and resource consumption to handle system dynamics. The latter mainly focuses on resource management on server clusters to achieve resource efficiency while meeting performance goals. A summary of the representative works in both research directions is shown in Table 1. Through this comparison, we make the following observations: (1) While some work (e.g., JetStream and AWStream) has considered latency in their system design, achieving strict SLOs for edge (or wide-area) video analytics has been rarely explored. (2) Very little attention has been paid on jointly considering the variability of both the network and compute steps in the design of adaptive edge video analytics pipelines. (3) DAG-based DNN inference has hardly been considered in adaptive

Solution	Network	Compute	DAG	SLO
Adaptive (wide-area or edge) video analytics				
JetStream [25]	✓	✗	✗	-
LAVEA [32]	✓	✓	✓	-
VideoStorm [35]	✗	✓	✓	-
AWStream [34]	✓	✗	✗	-
Chameleon [16]	✗	✓	✗	✗
VideoEdge [15]	✓	✓	✓	✗
Clownfish [24]	✓	✗	✗	✓
DDS [10]	✓	✗	✗	-
SPINN [19]	✓	✗	✗	✓
DNN inference serving				
Clipper [9]	✗	✓	✗	✓
DeepQuery [11]	✗	✓	✓	-
Nexus [29]	✗	✓	✗	✓
GrandSLAm [17]	✗	✓	✓	✓
InferLine [8]	✗	✓	✓	✓
ALERT [30]	✗	✓	✗	✓
Clockwork [12]	✗	✓	✗	✓
Llama [27]	✗	✓	✓	✓

Table 1: Summary of existing video analytics systems (✓: full support, -: partial support (latency considered but not guaranteed), ✗: no support).

edge video analytics systems. Overall, we see a clear gap between the state-of-the-art and our goal of providing strict SLOs for the end-to-end edge video analytics pipeline.

3 TIMELY EDGE VIDEO ANALYTICS VIA FEEDBACK CONTROL

In this section, we show how we can achieve timely video analytics over the air by leveraging a simple idea called feedback control. We first explain why feedback control is a suitable choice and then present our system design.

3.1 Why Use Feedback Control?

Generally speaking, achieving strict SLOs in edge video analytics can be done with two lines of approaches: *proactive* and *reactive*. Proactive approaches make system adaptation decisions based on real-time system status. Hence, they require precise information about the network status (typically done with bandwidth probing [34]), the availability of the compute resources, and the expected performance interference (e.g., based on a complex performance model [30]), all in real-time. However, the edge environment is highly dynamic, and obtaining such information precisely on time is not always realistic. On the contrary, reactive approaches rely on the history information based on the outcome of past system status and adaptation decisions. Overall, the benefits of reactive approaches include:

- No need for bandwidth probing and thus incurring no network overhead,
- No need for building explicit performance (interference) models for DNNs on the edge, and
- High scalability since the adaptation logic is relatively simple and can be placed on every end-device.

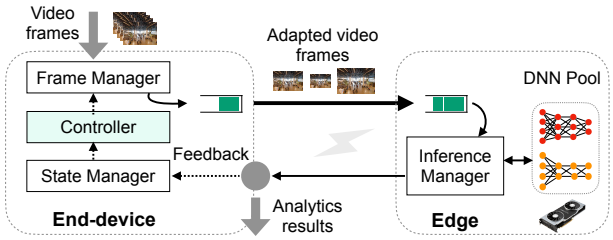

Figure 2: An overview of the system architecture and the main workflow.

Considering these benefits, we adopt a reactive approach for the SLO guarantee in edge video analytics. More specifically, we employ the feedback control technique, where we tune the system towards our goal based on the observed system outcome in the history. Our design is a proof-of-concept for reactive approaches. More sophisticated mechanisms such as PID control, model predictive control (MPC), and deep reinforcement learning (DRL) can also be explored later as they have shown big successes in other networked systems [22, 23, 33].

3.2 System Design

We now present our system design—timely edge-based video analytics over the air based on feedback control. Figure 2 depicts an overview of the system architecture and the general workflow. Overall, the system contains several modules that span across the end-device and the edge. The end-device represents a less powerful user device (e.g., a smartphone or an AR device), which uses a camera to capture the environment and generates the video stream for analytics, while the edge represents a relatively more powerful server equipped with high-end accelerators like GPUs at the network access point which the end-device is connected to via a wireless connection (e.g., WiFi, LTE, or 5G).

3.2.1 System workflow. The system works as follows: On the end-device, the Frame Manager module receives video frames at a fixed frame rate (e.g., 30 frames per second). The Frame Manager then resizes the video frames based on the control decision produced by the Controller module, encodes them, and sends them to the sending queue. The frames will be sent out to the edge over the wireless network. Upon receiving a frame from the end-device, the Inference Manager module on the edge picks a DNN with size matching the frame size from the pool of DNNs and uses the selected DNN to perform inference on the frame. Here, we choose an inference task involving only one DNN for simplicity, but our system supports DAG-based DNN inference tasks. When the inference is complete, the Inference Manager sends the inference result (typically containing text labels and bounding box coordinates which are small in size) to the end-device. On the end-device, a module called State Manager collects system state information, including the time it takes for every frame to be sent and processed by the edge. The Controller then relies on the information provided by the State Manager module and employs a control algorithm (detailed later) to make system adaptation decisions, i.e., deciding the frame size for the upcoming frames.

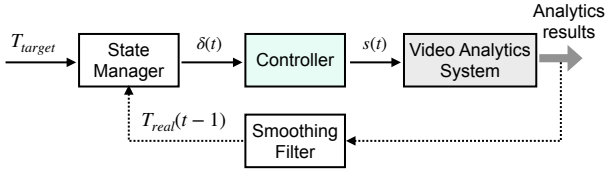

Figure 3: Feedback control internals.

3.2.2 State manager. This module manages the system state. Assume the SLO is given by T_{target}, which is the target per-frame end-to-end latency that the system aims to achieve. Specifically, the State Manager maintains the measured end-to-end latency of past frames and removes noise from the measurement results using a smoothing filter. Particularly, we employ an *error-based* adaptive filter [18], which has a good balance between sensitivity to large changes and stability to small fluctuations. Assume time is split into slots (e.g., one second). We denote the smoothed latency at time slot t by $T_{real}(t)$. We then compute the term deviation σ_t as an absolute difference between the target latency and the smoothed latency, i.e., $\sigma(t) = |T_{real}(t) - T_{target}|$. The error $e(t)$ is calculated by scaling the deviation $\sigma(t)$ where we use the power function as the scaling method:

$$e(t) = \begin{cases} -(\sigma(t))^a, & \text{if } T_{real}(t) \leq T_{target}, \\ +(\sigma(t))^b, & \text{otherwise} \end{cases} \quad (1)$$

Here, a and b are parameters and we set $b > a$ to assign a higher penalty to SLO misses.

The State Manager also maintains an observation window of up to W past time slots where the average of the errors in the observation window is used as an output of the State Manager, i.e., $\delta(t) = \sum_{i=t-W}^{t-1} e(i)/W$. We reset the window on every control decision (i.e., on frame size adaptation). Therefore, the past W error values accumulated in the window always represent errors under the same frame size.

3.2.3 Controller. We employ a threshold-based approach to make the adaptation decision, denoted by $s(t)$, as shown in the following equation:

$$s(t) = \begin{cases} s(t-1) + 1, & \text{if } \delta(t) \leq \theta^{low}, \\ \lfloor s(t-1)/2 \rfloor, & \text{if } \delta(t) \geq \theta^{high}, \\ s(t-1), & \text{otherwise.} \end{cases} \quad (2)$$

Intuitively, we gradually increment the (enumerated) frame size to the next level when the state output $\delta(t)$, i.e., the average error in the observation window, is lower than some threshold. Inspired by existing congestion control algorithms used for network transport, we decrease the enumerated frame size multiplicatively to respond fast to dramatic network bandwidth changes. The behavior of the adaptation can be further fine-tuned so that the change in the frame size matches the state output $\delta(t)$.

In this preliminary work, we chose values for the thresholds and other control parameters empirically. However, choosing threshold values adaptively based on the variance in the measurements and the amount of increase in the latency values for different frame sizes can be used to further improve the performance.

3.2.4 Incorporating Analytics Accuracy. Typically, the accuracy of DNNs increases with the increase in the input (frame) size, but not always. One option is to let the Controller return the largest frame size that can satisfy the SLO, but this may not provide the best analytics accuracy. Therefore, we choose to select the frame size that provides the highest accuracy among all feasible frame sizes—the frame sizes that are smaller or equal to the one returned by the Controller. Estimating a function that maps the frame size to the analytics accuracy during runtime is a challenging and open problem due to well-known issues such as video context drifts [4]. We leave it for future exploration.

4 PRELIMINARY EVALUATION

In this section, we introduce the experimental setup and discuss the evaluation results in detail.

4.1 Experimental Setup

4.1.1 System setup. We built a system prototype with an NVIDIA Jetson Xavier board as the mobile end-device and a server equipped with an Intel Xeon E5-2630 CPU, 64GB DRAM, and an NVIDIA RTX 2080 Ti GPU as the edge server. The two devices are connected via a 1Gbps Ethernet network, on which we use the Linux `tc` utility to replay wireless network traces (under mobility) for experiments. The system software is implemented in C++, where we use OpenCV for frame-based operations and Darknet for DNN inference on GPUs [5]. We evaluated our system on the object detection task performed by YOLOv4 DNNs. Our DNN pool consists of 19 YOLOv4 DNNs [5] with input sizes ranging from 64×64 to 640×640 with a step size of 32. We load as many DNNs to the GPU memory as possible and use the least recently used (LRU) policy to evict loaded DNNs to make room for new DNNs. For the experiments, we use videos from the PKU-MMD dataset [20], which contain multiple objects including persons, desks, and chairs. We set the SLO as 33ms, the minimal latency for achieving 30 FPS with a single GPU, which is also a much tighter number compared with what is used in other systems where the SLO is set to hundreds of milliseconds or even seconds [12, 17].

We implemented the client and server as a single process, multi-threaded module. The client module has three main threads for reading (or capturing) frames, sending frames, and receiving object detection results. The reading thread calls the controller in the same thread context and then preprocesses (resize plus encode) the frame according to the frame size returned by the controller. Upon receiving detection results, the receiving thread makes a call to the controller to handle feedbacks. Like the client module, the server module also has the receiving and sending threads and their associated queues. The server runs a main serving thread that calls the Darknet API (`detect`) to execute the desired DNN model from the model pool in the same thread context and uses a separate loading thread for provisioning a desired but passive model on the GPU. Note that one can use more advanced model serving frameworks like Clipper, INFaaS, or Clockwork as a server module, as long as the frameworks do not introduce highly variable delay such as waiting time or overhead in model switching (or scheduling).

4.1.2 Parameter values. We empirically choose the history window W size to be 20, i.e., observing the last 20 frames as feedback. The controller at time t does not wait for the direct previous frame feedback (i.e., a frame sent at $(t-1)$) as it introduces a delay that may create a cascading effect in case of short glitches in the system. Therefore, we make decisions on stale feedbacks but with a limit on the staleness (e.g., no more than one second elapsed or at least the feedback for $(t-30)$th frame must be available) under the assumption that the system's state remains relatively stable for a short period of time. We choose the value of threshold θ^{low} as a negative value of the product $(W \cdot 7.5)$ and θ^{high} as zero. Here, the multiplier 7.5ms roughly represents the increase in end-to-end latency plus the variance if the model size increases by one.

4.1.3 Evaluation metric. We use *end-to-end (E2E) latency* as a measure of timeliness, and it includes the processing (resizing plus encoding) time on the client, network transmission time to send frames and receive results, processing time (decode and maybe resize) on the server, queuing delay and the DNN inference time.

4.1.4 Controller overhead. One of the main goals in our controller design is to be scalable and lightweight in terms of compute, memory, and network usage. The control algorithm for every decision-making requires only a few simple arithmetic and logical operations that should be extremely fast on many end-devices. In terms of memory, the controller has to store only an array of past errors in the control window of length W and requires no extra network usage, as we do not have to monitor (or profile) network bandwidth actively.

4.2 Results

We perform experiments on our testbed with both synthetic bandwidth shaping values similar to those used in [34], and real-world WiFi network traces collected by ourselves when walking with a mobile device connected to a WiFi access point.

Figure 4 shows the results with the synthetic bandwidth shaping values. Here, we assume the end-device sends 30 frames per second to the edge server for analytics. We periodically choose the bandwidth limit values from the ordered list {25, 7.5, 5} Mbps and each value for 20 seconds. The top plot shows the network bandwidth variation over time. The middle plot shows the selected model compared with the desired model (the optimal one if the current network bandwidth condition is known beforehand). In our setup, the models that are neighbors in size of the currently active model are often pre-loaded according to the LRU model caching and replacement policy with enough GPU capacity. As a result, the model switching is almost always done instantaneously. The bottom plot depicts the measured end-to-end latency of every frame, where we can see that the SLO violation rate is as low as 1.03% despite the extremely tight SLO.

Figure 5 shows the results with real-world WiFi network trace collected by ourselves. Here, the network bandwidth varies at the granularity of seconds. As we can see, the variability is significant, confirming the need for system adaptivity for the SLO guarantee. Even under the highly variable network conditions, our proposed method meets the SLO for 98.3% of all the frames, confirming the

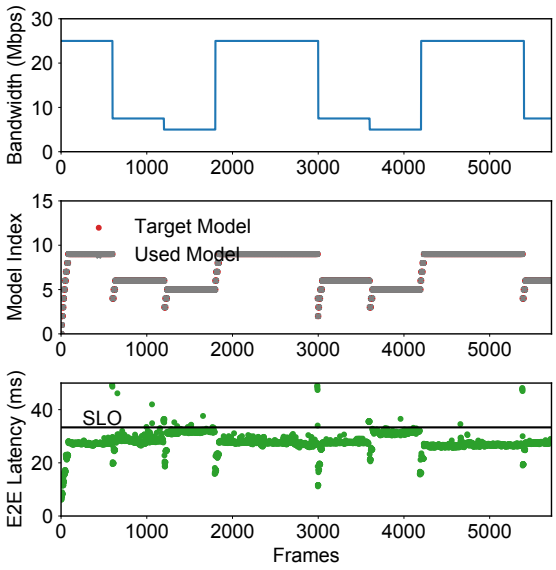

Figure 4: SLO guarantee performance under a synthetic network bandwidth trace.

effectiveness of our proposed reactive approach for the SLO guarantee.

Although we use the additive increase multiplicative decrease adaptation strategy, it can be seen from the figures (especially Figure 4) that our control algorithm quickly ramps up the DNN model as soon as the bandwidth improves significantly. This is because the end-to-end latency would be much lower than the target SLO, leading to the high error values defined in Equation 1. This ultimately leads to the selection of DNN models with larger sizes to quickly lower the gap between the target SLO and the measured per-frame end-to-end latency. The DNN model size is decreased dramatically when there is a significant drop in bandwidth, preventing violations due to slow reactions from the algorithm.

Figure 6 depicts the CDF of the end-to-end latency under both the synthetic and real-world network bandwidth traces. We can clearly observe that most of the frames (e.g., 99% with the synthetic trace and 98.3% with the real-world trace) are completed within the SLO. Note that, in both cases, the median latency of the frames finished before the deadline is around 27.53ms and 28.87ms, respectively, indicating that our method is not very conservative and does not significantly waste the latency budget to meet the SLO.

5 DISCUSSION

In this section, we discuss some of the limitations of our proposed approach and envision future work.

5.1 Responsiveness

While achieving promising results, we also notice some inevitable limitations in terms of responsiveness: (1) Feedback control, in its nature, is a reactive approach where the adaptation is done based on a smoothed history and is not sensitive to rapid changes in network bandwidth. (2) In our adaptation logic, we always choose

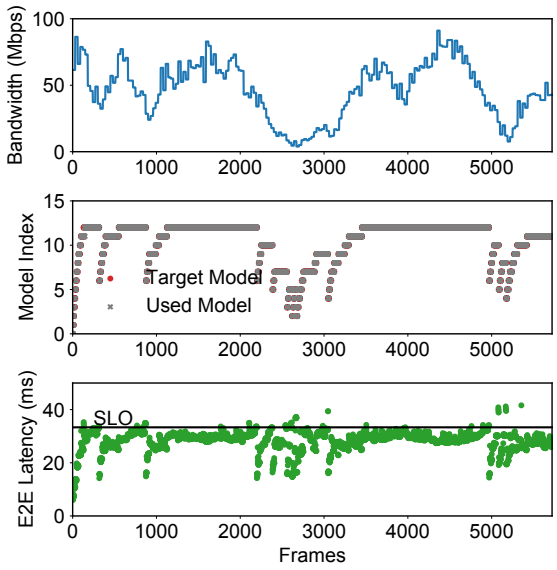

Figure 5: SLO guarantee performance under a real-world WiFi bandwidth trace.

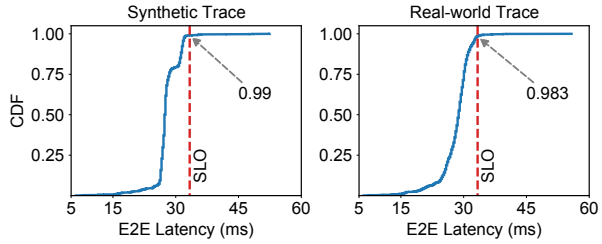

Figure 6: CDF of end-to-end latency under different bandwidth conditions.

a neighboring model to switch to, disallowing jumping decisions. While this helps with the model caching since future adaptation decisions are among the neighbors of the current DNN model, it also brings a delay in reacting to bandwidth changes. On the other hand, if we employ an adaptation algorithm that can freely change the DNN model size, i.e., more responsive to network bandwidth changes, we need to find a solution for model caching since loading a DNN to the GPU memory takes a few seconds. We will explore such algorithms in future work.

5.2 Model Selection and Batching

In this work, we assume the batch size during DNN inference to be one, which is suboptimal when (1) clients have higher latency SLOs, (2) many concurrent clients need to be served. Typically, DNN models offer higher throughput at large batch sizes but at the expense of higher inference latency. For clients with larger SLO (i.e., with a higher latency budget on the server), we can select larger models to satisfy the desired throughput (and accuracy as a consequence) by choosing larger batch sizes optimally. However, estimating the latency budget and determining the DNN model size together with the inference batch size in real-time under dynamic network conditions is non-trivial.

Furthermore, choosing models and batch sizes is complicated in the case of multiple clients issuing concurrent inference requests to the edge. In such a scenario, executing inferences at batch size one may lead to underutilization of the edge server and thus sub-optimal performance. The latest cloud inference serving systems promise to meet the latency SLOs but only on the server side and fail to consider the *end-to-end latency* which includes also the dynamic network part. As future work, we aim to design and implement a modern DNN serving system that can jointly consider the client's network dynamics and requirements (e.g., throughput), DNN management, and scheduling on the edge server with limited resources.

6 RELATED WORK

With the goal of achieving high analytics accuracy, adaptive edge video analytics systems such as AWStream [34] and Chameleon [16] conduct dynamic tradeoffs between resource consumption and accuracy via adapting system configurations, including frame size/rate and DNN model.

However, very few of these systems target the goal of achieving strict SLO guarantee. SLO guarantee for video analytics has been studied in the literature, but the focus is mainly on DNN inference serving on the server so far. Existing systems like Clipper [9], Nexus [29], and Clockwork [12] target efficient video analytics with strict SLO guarantees through informed resource management and request scheduling in server clusters. There are also systems such as GrandSLAm [17], InferLine [8], and Llama [27] that consider SLO guarantee in complex video analytics graphs on servers.

None of these works include the data transfer over the wireless network, which plays a critical role in edge video analytics pipelines. Unlike all these works, we propose to meet SLOs on the per-frame end-to-end latency for edge video analytics over the air.

7 CONCLUSIONS

In this paper, we made a case for timely video analytics at the edge by enforcing strict SLOs. We identified the challenges in achieving strict SLO guarantee on edge video analytics pipelines and analyzed the gap in the existing systems. We also proposed an adaptation mechanism based on feedback control which can make system adaptation decisions towards the specified SLO for edge video analytics. Preliminary results based on a prototype implementation showed that the proposed mechanism is promising, but imposes several limitations. In our future work, we will study more sophisticated adaptation mechanisms.

ACKNOWLEDGMENTS

This work is part of the Efficient Deep Learning (EDL) programme (grant number P16-25), financed by the Dutch Research Council (NWO). Lin Wang was partially supported by the German Research Foundation (DFG) Collaborative Research Center 1053–MAKI.

REFERENCES

[1] Fawad Ahmad, Hang Qiu, Ray Eells, Fan Bai, and Ramesh Govindan. 2020. CarMap: Fast 3D Feature Map Updates for Automobiles. In *USENIX NSDI*. 1063–1081.

[2] Ali J. Ben Ali, Zakieh Sadat Hashemifar, and Karthik Dantu. 2020. Edge-SLAM: edge-assisted visual simultaneous localization and mapping. In *ACM MobiSys*. 325–337.

[3] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-Time Video Analytics: The Killer App for Edge Computing. *Computer* 50, 10 (2017), 58–67.

[4] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Nikolaos Karianakis, Yuanchao Shu, Kevin Hsieh, Victor Bahl, and Ion Stoica. 2020. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. *CoRR* abs/2012.10557 (2020). arXiv:2012.10557 https://arxiv.org/abs/2012.10557

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *CoRR* abs/2004.10934 (2020). arXiv:2004.10934 https://arxiv.org/abs/2004.10934

[6] Kevin Boos, David Chu, and Eduardo Cuervo. 2016. FlashBack: Immersive Virtual Reality on Mobile Devices via Rendering Memoization. In *ACM MobiSys*. 291–304. https://doi.org/10.1145/2906388.2906418

[7] Michael Braun, Anja Mainz, Ronee Chadowitz, Bastian Pfleging, and Florian Alt. 2019. At Your Service: Designing Voice Assistant Personalities to Improve Automotive User Interfaces. In *ACM CHI*. 40.

[8] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. 2020. InferLine: Latency-Aware Provisioning and Scaling for Prediction Serving Pipelines. In *ACM SoCC*. 477–491. https://doi.org/10.1145/3419111.3421285

[9] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *USENIX NSDI*. 613–627.

[10] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. 2020. Server-Driven Video Streaming for Deep Learning Inference. In *ACM SIGCOMM*. 557–570. https://doi.org/10.1145/3387514.3405887

[11] Zhou Fang, Dezhi Hong, and Rajesh K. Gupta. 2019. Serving deep neural networks at the cloud edge for vision applications on mobile platforms. In *ACM MMSys*. ACM, 36–47. https://doi.org/10.1145/3304109.3306221

[12] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up. In *USENIX OSDI*. 443–462.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE CVPR*. IEEE Computer Society, 770–778.

[14] Junxian Huang, Feng Qian, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2012. A close examination of performance and power characteristics of 4G LTE networks. In *ACM MobiSys*. 225–238.

[15] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodík, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. VideoEdge: Processing Camera Streams using Hierarchical Clusters. In *IEEE/ACM SEC*. 115–131.

[16] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodík, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *ACM SIGCOMM*. 253–266.

[17] Ram Srivatsa Kannan, Lavanya Subramanian, Ashwin Raju, Jeongseob Ahn, Jason Mars, and Lingjia Tang. 2019. GrandSLAm: Guaranteeing SLAs for Jobs in Microservices Execution Frameworks. In *ACM EuroSys*. 1–16. https://doi.org/10.1145/3302424.3303958

[18] Minkyong Kim and Brian D. Noble. 2001. Mobile network estimation. In *ACM MOBICOM*. 298–309. https://doi.org/10.1145/381677.381705

[19] Stefanos Laskaridis, Stylianos I. Venieris, Mário Almeida, Ilias Leontiadis, and Nicholas D. Lane. 2020. SPINN: Synergistic Progressive Inference of Neural Networks over Device and Cloud. In *ACM MobiCom*. 37:1–37:15. https://doi.org/10.1145/3372224.3419194

[20] Chunhui Liu, Yueyu Hu, Yanghao Li, Sijie Song, and Jiaying Liu. 2017. PKU-MMD: A Large Scale Benchmark for Skeleton-Based Human Action Understanding. In *ACM VSCC*, Xiaobai Liu, Yadong Mu, Yu-Gang Jiang, and Jiebo Luo (Eds.). ACM, 1–8. https://doi.org/10.1145/3132734.3132739

[21] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge Assisted Real-time Object Detection for Mobile Augmented Reality. In *ACM MobiCom*. 25:1–25:16.

[22] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *ACM SIGCOMM*. ACM, 197–210. https://doi.org/10.1145/3098822.3098843

[23] Konstantin Miller, Dilip Bethanabhotla, Giuseppe Caire, and Adam Wolisz. 2015. A Control-Theoretic Approach to Adaptive Video Streaming in Dense Wireless Networks. *IEEE Trans. Multim.* 17, 8 (2015), 1309–1322. https://doi.org/10.1109/TMM.2015.2441002

[24] Vinod Nigade, Lin Wang, and Henri E. Bal. 2020. Clownfish: Edge and Cloud Symbiosis for Video Stream Analytics. In *IEEE/ACM SEC*. IEEE, 55–69. https://doi.org/10.1109/SEC50012.2020.00012

[25] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S. Pai, and Michael J. Freedman. 2014. Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area. In *USENIX NSDI*. 275–288.

[26] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *arXiv* (2018).

[27] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Llama: A Heterogeneous & Serverless Framework for Auto-Tuning Video Analytics Pipelines. *arXiv* (2021). arXiv:2102.01887

[28] Mahadev Satyanarayanan. 2017. The Emergence of Edge Computing. *Computer* 50, 1 (2017), 30–39.

[29] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: a GPU cluster engine for accelerating DNN-based video analysis. In *ACM SOSP*. 322–337.

[30] Chengcheng Wan, Muhammad Husni Santriaji, Eri Rogers, Henry Hoffmann, Michael Maire, and Shan Lu. 2020. ALERT: Accurate Learning for Energy and Timeliness. In *USENIX ATC*. 353–369.

[31] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. 2020. Understanding Operational 5G: A First Measurement Study on Its Coverage, Performance and Energy Consumption. In *ACM SIGCOMM*. 479–494.

[32] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. 2017. LAVEA: latency-aware video analytics on edge computing platform. In *ACM/IEEE SEC*. 15:1–15:13.

[33] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *ACM SIGCOMM*. 325–338. https://doi.org/10.1145/2785956.2787486

[34] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A. Lee. 2018. AWStream: adaptive wide-area streaming analytics. In *ACM SIGCOMM*. 236–252.

[35] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodík, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *USENIX NSDI*. 377–392.