Lightweight Optimization Strategies for Modern Video Applications Over Wireless Networks

A THESIS

submitted by

Shubham Chaudhary

Roll Number: PhD20021

under the guidance of

Arani Bhattacharya

for the award of the degree

of

Doctor of Philosophy

Department of Computer Science and Engineering

INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

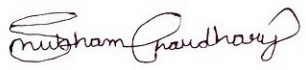
July 2026

Dedicated to my beloved parents.

Declaration

This is to certify that the thesis titled *Lightweight Optimization Strategies for Modern Video Applications Over Wireless Networks* has been authored by me. It presents the research conducted by me under the supervision of *Prof. Arani Bhattacharya*.

To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and to collaborations, with appropriate citations and acknowledgments, in line with the established norms and practices.

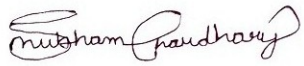


Shubham Chaudhary (PhD20021)
Department of Computer Science and Engineering
Indraprastha Institute of Information Technology Delhi
New Delhi, India, 110020

Declaration Regarding Use of AI

I acknowledge that I am fully responsible for the entire content of this thesis, including any sections assisted by online tools, including Artificial Intelligence-based tools.

I accept full accountability for any violations of ethical standards in publications arising from the use of such tools.



Shubham Chaudhary (PhD20021)
Department of Computer Science and Engineering
Indraprastha Institute of Information Technology Delhi
New Delhi, India, 110020

Certificate

This is to certify that the thesis titled **Lightweight Optimization Strategies for Modern Video Applications Over Wireless Networks**, being submitted by **Shubham Chaudhary (PhD20021)**, to the Indraprastha Institute of Information Technology, Delhi, for the award of the degree of **Doctor of Philosophy**, is an original research work carried out by him under my supervision.

The contents of this thesis, in whole or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Arani Bhattacharya
Assistant Professor
Department of Computer Science and Engineering
IIIT-Delhi, New Delhi, India, 110020

Acknowledgements

I am thankful to a large number of people who contributed directly or indirectly towards the successful completion of the works in this thesis. I am especially grateful and indebted to my supervisor, Prof. Arani Bhattacharya. I have been fortunate to have him as my advisor. His guidance and support extend beyond professional work and research. His encouragements always helped during setbacks and moments when things went south.

I am also grateful for the opportunity to collaborate and work closely with Prof. Mukulika Maity and Prof. Aruna Balasubramanian. I learned a lot from them. Their suggestions and comments reinforced my understanding and pushed me to think more. It helped me grow as an independent researcher. I enjoyed working with Prof. Mukulika on two of my works in this thesis. She helped throughout the ideation, analysis, and submission, and her contributions were indispensable. I am extremely thankful to Prof. Aruna for inviting me to Stony Brook University as a visiting research scholar. It was an amazing experience working there. It enabled me to meet with fellow labmates and some wonderful people. I got to spend six months at Stony Brook and worked in person with Prof. Aruna. I found an inspiring personality within her. I started my work on autonomous driving with her, and she always actively participated in discussions and meetings and gave constructive feedback.

I am also thankful to my labmates and peers at IIIT Delhi. They have always been generous, kind, helpful, and supportive throughout my PhD. Their constructive criticism during lab seminars and mock presentations always helped me identify my weaknesses and improve. I am indebted to my projectmates Navneet Mishra, Aruba Sood, Keshav Gambhir, Tanmay Rajore, Aryan Taneja, Anjali Singh, and Purbasha Roy for their contributions to the completion of the work in this thesis. Furthermore, I am extremely grateful to my beloved college, IIIT Delhi. It provided me with a nourishing environment and funds to attend conferences, and even monetarily supported my research visit to Stony Brook.

Finally, I would like to thank my loving parents. They have always encouraged and supported me throughout my PhD. They have always been the invisible force driving me to work hard and consistently. Lastly, I am grateful to God, who has always guided me and given me the courage and strength to keep pushing myself, even in moments of disappointment. Thank you so much to everyone who always stayed with me and guided me towards the successful completion of this thesis.

Abstract

The various video applications running over wireless networks face a common set of challenges, such as inconsistent bandwidth, high network variability, and sudden latency spikes. Considering such constraints, this thesis explores the possible strategies and design choices for developing data-intensive video applications such as real-time traffic surveillance, live video streaming, and cloud-assisted autonomous driving. Our focus is primarily on two techniques across three distinct video-streaming applications. The first technique is the intelligent use of tiled encoding available in modern video codecs, where the encoded video has independent rectangular spatial regions that can be manipulated in real time without re-encoding. The second technique is to develop data filtering strategies to minimize ingestion costs by pruning extraneous information.

In traffic surveillance, cameras stream video to servers for computer vision algorithms, consuming significant bandwidth. To reduce bandwidth usage, we use tile sampling to select spatial regions of frames that contain only moving objects, since the rest of the frame mostly has static backgrounds, such as the sky and buildings. To select such tiles, we propose an adaptive tile selection algorithm that samples only tiles with moving objects by leveraging their correlation with tile bitrates. Our evaluations across different lighting, weather, and traffic conditions, both using benchmark videos and a live deployment, show improved accuracy, reduced bandwidth usage, and lower overhead than existing systems.

For live streaming, we propose using the network interface of a nearby helper device to get an aggregated bandwidth. To stream videos, we design a tile-level aggregation strategy that partitions tiles into two subsets and schedules them independently across available network interfaces, based on their importance. We demonstrate, through extensive experiments, that such an aggregation strategy provides better QoE than conventional multipath strategies.

Lastly, we address the high data ingestion cost in self-driving cars. Autonomous vehicles rely on compressed models whose accuracy degrades over time as the distribution of real-world test data changes, requiring frequent retraining on a server. This necessitates limiting the number of selected training frames to minimize the end-to-end delay in obtaining the retrained model without impairing post-training accuracy. We sample only the most useful frames and adaptively encode them into a video based on the network bandwidth. Our lightweight sampling strategy seamlessly integrates with the existing workflow, achieving superior accuracy while minimizing update delay compared to baseline strategies across different network conditions.

List of Publications

Patent

Shubham Chaudhary, Aryan Taneja, Anjali Singh, Arani Bhattacharya, Mukulika Maity. "Method And System Facilitating Data Optimization In A Data Transmission Process". Indian Patent Application Number: 202211028440. (Granted)

Papers

1. **Shubham Chaudhary**, Aruba Sood, Navneet Mishra, Keshav Gambhir, Arani Bhattacharya, Mukulika Maity. "Tile-level Multipath Content-aware Live Video Streaming," In the ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM'26)

2. **Shubham Chaudhary**, Navneet Mishra, Keshav Gambhir, Tanmay Rajore, Arani Bhattacharya, Mukulika Maity. "COMPACT: Content-aware Multipath Live Video Streaming for Online Classes using Video Tiles," In the Proceedings of the 16th ACM Multimedia Systems Conference (MMSys'25) held in Stellenbosch, South Africa.

3. **Shubham Chaudhary**, Aryan Taneja, Anjali Singh, Purbasha Roy, Sohum Sikdar, Mukulika Maity, Arani Bhattacharya. "TileClipper: Lightweight Selection of Regions of Interest from Video for Traffic Surveillance," In the Proceedings of the 2024 USENIX Annual Technical Conference (ATC'24) held in Santa Clara, CA, USA.

4. **Shubham Chaudhary**, Arani Bhattacharya, Saket Anand, Aruna Balasubramanian. "Scalable and Sustainable Video Analytics on Edge using Sensor Clustering," In Proceedings of the 15th ACM Wireless of the Students, by the Students, and for the Students Workshop (S3) in conjunction with the 30th Annual International Conference on Mobile Computing and Networking (MobiCom'24) held in Washington, D.C., USA

5. Keshav Gambhir, Tanmay Rajore, **Shubham Chaudhary**, Taral Jain, Avishi Gupta, Mukulika Maity, Arani Bhattacharya. "NATIVE: Network Aggregation based Tiled Live Video Streaming," Demo Paper in Proceedings of International Conference on COMmunication Systems & NETworkS (COMSNETS) held in 2023, Bangalore, India.

Under Review

1. **Shubham Chaudhary**, Aruna Balasubramanian, Arani Bhattacharya. "Jaeger: Resource-Aware On-device Model Adaptation for Autonomous Vehicles," In the Proceedings of the 2026 ACM SIGOPS Annual Technical Conference (ATC) to be held in Shatin, Hong Kong.

2. **Shubham Chaudhary**, Saket Anand, Aruna Balasubramanian, Arani Bhattacharya. "Pipette: Adaptive Selection of Relevant Samples for Continual Learning in Autonomous Vehicles" (Yet to Resubmit)

Contents

List of Figures

List of Tables

Abbreviations

DNN	**D**eep **N**eural **N**etwork
BR	**B**it**r**ate
FG	**F**ore**g**round
BG	**B**ack**g**round
RTT	**R**ound **T**rip **T**ime
BW	**B**and**w**idth
QP	**Q**uantization **P**arameter
SCTP	**S**tream **C**ontrol **T**ransmission **P**rotocol
NTP	**N**etwork **T**ime **P**rotocol
CL	**C**ontinual **L**earning
AV	**A**utonomous **V**ehicle
QoE	**Q**uality **o**f **E**xperience
QoS	**Q**uality **o**f **S**ervice

Chapter 1

Introduction

As more of the world gets connected, the need and demand for video applications such as live video conferencing, online gaming, virtual reality, and video analytics/traffic surveillance have grown tremendously. All these video applications are bandwidth-hungry in nature and require a real-time performance guarantee. However, the current wireless infrastructure falls short of meeting such requirements. For developing countries like India, the situation is even worse. According to the recent OpenSignal survey [2], although 5G provides at least 6× higher download speeds than 4G, users' connectivity frequently falls back from 5G to 4G, reaching 38.6% on some network providers. The problem becomes even more pertinent, as reported by the Telecom Regulatory Authority of India (TRAI), 45.64% of the user base in India resides in rural areas [3]. Therefore, it is essential for developers to account for inconsistent bandwidth, latency spikes, and network disconnections when designing high-traffic video applications for current cellular networks.

Since network characteristics are inherently variable, video applications should employ intelligent strategies to address this variability. The most straightforward way is to reduce the amount of data being sent over the network. Albeit promising, this is not trivial because reducing the streamed data adversely affects the performance of the video applications. For example, in live video streaming, we can reduce the quality of the streamed frames, which significantly hurts the user's QoE. Therefore, the pruning strategy should be intelligent enough to adapt to network variations while maintaining a high QoE and QoS. Moreover, since these video applications run on devices with constrained compute and memory resources, the filtering algorithm should be lightweight to avoid significant overhead in the existing pipeline. This thesis proposes filtering algorithms and techniques tailored to various applications.

1.1 Background: Bandwidth Conservation for Video Streaming

Across all these applications, videos are typically first encoded using a standard. Such standards, also known as codecs, such as H.264, H.265, and AV1, use lossy compression techniques to compress content. However, there are additional parameters specified in the standards, such as resolution, quantization parameter (QP) and frame rate, which can be tuned depending on the application. Given the proliferation of video streaming applications, it has become essential to intelligently utilize these parameters to conserve bandwidth.

A number of prior works have utilized such parameters to counter the problem of high bandwidth consumption by video streaming applications. [4, 5, 6, 7, 8, 9, 10]. These works use one of the three techniques: frame pruning, quality pruning, and spatial pruning. Frame pruning refers to the selective filtering out of frames that do not significantly contribute to the improvement of the quality of the application. For example, many video conferencing software reduce the number of frames per second when the content shown is relatively static. Quality pruning refers to the utilization of QP or resolution to increase the level of quantization or the number of pixels per unit length of display, respectively. For example, YouTube adjusts both QP and resolution depending on the bandwidth. Spatial pruning refers to identifying the regions of interest and assigning fewer bits for the content outside them. These techniques have been extensively studied over the last 20 years.

A key challenge of these works is that they impose additional overhead on either the sender or receiver, making the schemes less practical. It hampers the real-time guarantees that modern applications require. For instance, in traffic surveillance, frame pruning [6] or spatial pruning [5] requires re-encoding the video on the camera. Similarly, quality pruning requires running an intensive algorithm to reverse the quality degradation it causes. Moreover, degrading the quality [4, 7, 11, 12] hurts the end application. In video analytics, it affects object detection accuracy, whereas in video streaming, it impairs the user experience. Further, since the cellular network is inconsistent [13], the amount of data to send should be adapted to reduce the streaming delays. Therefore, it becomes essential to consider these constraints when designing modern data-intensive video applications. This thesis addresses the shortcomings of prior solutions by proposing lightweight strategies that integrate easily with the existing pipelines.

1.2 Thesis Contributions

The primary argument of this thesis is that utilizing lightweight under-explored points of optimization in modern video applications has the potential to significantly improve the application while adding negligible computational overhead. We design and prototype three systems for three video streaming applications – (i) playing of educational videos using multipath networks, (ii) traffic surveillance, and (iii) use of cloud-assistance by autonomous vehicles. In the context

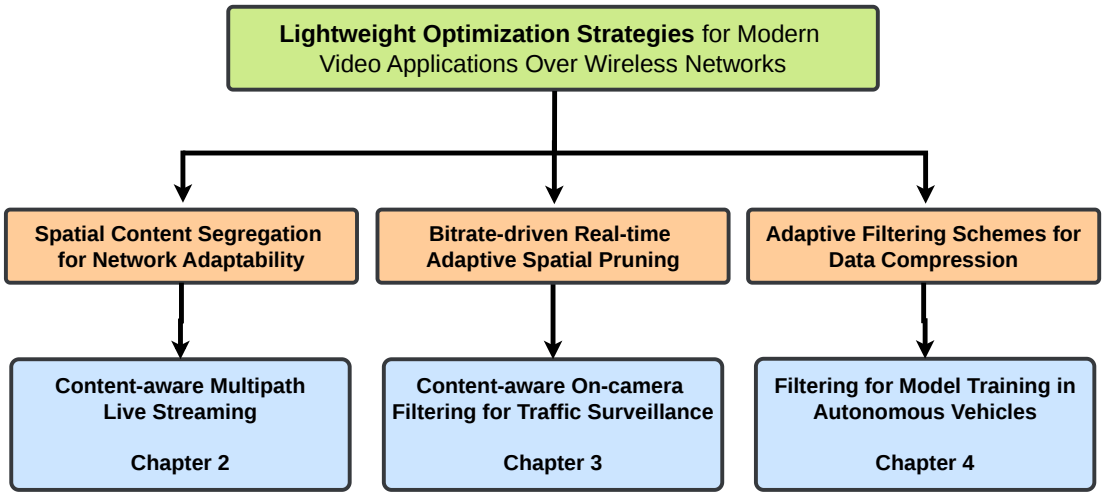


FIGURE 1.1: Organization and contribution of this thesis.

of educational videos, we identify the fact that the rendering of background and foreground content can be decoupled without significant degradation of the quality of experience. In traffic surveillance, we show that the level of compression of videos gives sufficient information about the presence of moving objects. In the use of cloud-assistance, we show that the confidence levels of lightweight object-detection models allow us to identify the frames that should be offloaded for retraining of the model.

Thus, this thesis focuses on devising methods and algorithms that leverage the spatial importance of content and an appropriate filtering scheme for three video applications: live video analytics/traffic surveillance, live video streaming, and cloud-assisted autonomous driving. These applications are widely adopted all over the world, including in developing countries like India. This thesis paves the way for the practical deployment by tackling the challenges and proposing appropriate solutions:

1.2.1 TileClipper for Real-time Traffic Surveillance

Real-time traffic surveillance has now become imperative for automatically enforcing traffic rules [14], controlling traffic lights [15], and detecting anomalous events such as accidents [16]. In surveillance, cameras stream video to a remote server that caters to the server-side application [7], such as object identification and vehicle tracking. However, a key challenge here is the associated extensive BW usage. We address this challenge by identifying regions of the frames that contain objects of interest (OoI) and streaming only those regions to the server. Generally, this requires running computer vision techniques, which are themselves computationally intensive. The camera usually lacks sufficient compute capability or power to run any compute-intensive algorithm. We propose TileClipper to solve this by observing the statistical patterns in the video BR across

regions containing OoI. This enables us to design a heuristic to identify such regions in real-time on the camera. TileClipper provides significant bandwidth savings without stressing the compute on the camera and the server-side.

1.2.2 Live Video Streaming Using COMPACT

The easy access to the internet on handheld smartphones has enabled live streaming, even in remote areas. Moreover, the number of smartphones is rising rapidly in developing countries. It is common for each family member to have a smartphone, or even for a single person to carry multiple smartphones [17]. Each of these smartphones supports both cellular and peer-to-peer Wi-Fi Direct connectivity. Therefore, we propose COMPACT to aggregate the network bandwidths of multiple devices at the user end to realize multipath streaming. To enable this, we reuse the idea of the region with the object of interest from our traffic surveillance work. Unlike conventional techniques, before streaming we detect the most important region of the frame for the user and separate it. Afterward, we design a scheduler to send the high-priority and the relatively low-priority regions over different networks. Using multiple paths increases the reliability and user QoE.

1.2.3 Pipette for Cloud-assisted Autonomous Driving

The rising adoption of vehicle automation ensures a safer commute. One such automation is to enable cars to drive completely autonomously. This requires the cars to run neural network models to observe the surroundings and perform necessary actions. One challenge associated with such automation is high inference rates. To achieve this, the vehicle typically runs specialized models that perform well only in limited scenes. Therefore, the model is occasionally retrained to adapt to changing environments. For this adaptation, the vehicle's frames are encoded into a video and streamed over the cellular network to a remote server, where they are used to retrain/finetune specialized models. Since the vehicles are mobile, the network characteristics are inconsistent. Ensuring a quick model update thus becomes challenging. We address this by designing Pipette that utilizes a frame filtering strategy based on the specialized models' predictions. Before streaming, we estimate the number of training epochs and the video encoding frame rate based on the current network characteristics. This provides an updated model with minimum model update delay.

1.3 Organization of This Thesis

This thesis is organized as follows:

- Chapter 2 gives details of our content-aware multipath approach, COMPACT, for improving user QoE. We discuss the results for driving and walking cellular network conditions and compare them with single- and multipath scenarios using different schedulers. A part of this work has been published in the proceedings of the ACM Multimedia Systems Conference (MMSys) held in 2025 [18], with an extended version accepted in the ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM). A practical demonstration of this work was done at the International Conference on COMmunication Systems & NETworkS (COMSNETS) held in 2023.

- Chapter 3 discusses our solution TileClipper's approach and methodology for traffic surveillance. It also provides all implementation details, extensive evaluation results conducted in challenging weather and traffic scenarios, and comparisons with state-of-the-art methods. A full paper based on this work has been published in the proceedings of the USENIX Annual Technical Conference (USENIX ATC) held in 2024 [19], and an Indian patent has been granted.

- Chapter 4, provides a detailed overview of our solution, Pipette, for cloud-assisted autonomous driving. We evaluate under both good and constrained BW conditions across different scenes. We compared our solution with state-of-the-art baselines. An extension of this work that performs on-vehicle training has been submitted to the 2026 ACM SIGOPS Annual Technical Conference (ATC).

- Finally, in Chapter 5, we conclude our current work and briefly discuss the possible future directions to explore.

Chapter 2

COMPACT: Content-aware Multipath Live Video Streaming for Online Classes using Video Tiles

In recent years, the popularity of online learning has increased [20, 21, 22, 23]. Students prefer using handheld devices for online learning [24, 25] primarily due to the availability of easier internet access on smartphones. Mobile learning enables students in even remote areas to listen to lectures from renowned universities and/or colleges. Live video streaming is the technology serving online learning [20, 26, 27, 28, 29, 30], supported by platforms such as YouTube and Twitch. Such services typically provide video content with a latency of the order of 5-10 seconds [31]. However, this growth in live video streaming for online learning comes with the challenge of providing students with a good quality of experience (QoE) to enhance their learning. Since most live-streamed online classes are interactive, where the students can ask questions either verbally or through chat or Q&A, all the modalities (text, audio, and video) should be in synchrony to maintain the live experience [26, 27]. The current wireless infrastructure falls short of catering to these bandwidth-hungry video applications. Although the newer generations of wireless technologies like 5G, 6G, and WiFi-7 promise to offer high bandwidth [32, 33, 34, 35], live video streaming still suffers from high latency and poor QoE [36, 37, 38, 39].

Several works [40, 41, 42, 43, 44, 45, 46, 47, 48] show the potential of using multiple connections to address the shortcomings of video streaming. Multipath streaming permits leveraging multiple available paths in conjunction. In addition, if one path goes down or has poor bandwidth (BW), the other can compensate, given it has relatively superior network characteristics [49]. Most students have access to multiple devices in their homes [50, 51, 52, 53, 54]. Live video streaming can benefit from aggregating all these devices' bandwidths. We show such a system in Fig. 2.1 where the user has two devices[1], a phone and a laptop connected to the Internet through two different Internet Service Providers (ISPs). Conventional techniques use only one device

[1]As a proof of concept, this work shows multipath with two devices only, which can be extended beyond. We leave it for future consideration.

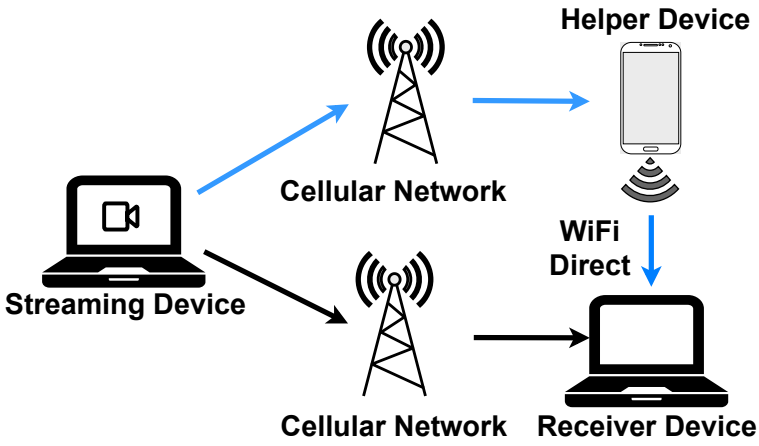


FIGURE 2.1: A schematic view of COMPACT, indicating in blue the lines for additional connectivity we intend to add.

(called primary) at a time to retrieve video packets (shown with black arrows). Having another supporting device (called a helper) in parallel facilitates multipath transmission (shown with blue arrows). The user's devices communicate within themselves using WiFi Direct/WiFi Peer-to-Peer connection. The use case for such a multi-device setup has been utilized and incentivized by MPBond [55], MicroCast [56], and OASIS [57], but they did not use it for live streaming.

Current multipath live streaming techniques utilize the extra bandwidth by distributing packets over the paths without considering the video content. Converge [40] used multipath WebRTC to meliorate video conferencing. Furthermore, they showed that if the current multipath protocols like MPTCP [58, 59, 60], MPQUIC [61, 62, 63], and MPRTP [64] are used as is for live video conferencing, they perform worse than single-path due to head-of-line (HoL) blocking caused by out-of-order packet arrival. TwinStar [41] sends frames in a round-robin fashion on the available paths, allocating bitrate jointly while encoding. AggDeliv [42] optimizes congestion control of the multiple paths to adapt them for mobile live video delivery. However, none of these techniques considers the relevance of spatial content. Therefore, these systems encode the whole content at a lower bitrate (BR) to accommodate it within the available BW.

The Case for Content-Aware Streaming: We first argue that the content of the video matters in terms of how packets should be scheduled over a multipath network. To illustrate this argument, we consider the case of a classroom where an instructor is teaching. The body, face, and gestures of the instructor are what the students focus on the most, indicated by the fact that they have the highest saliency [65, 66, 67]. Thus, these portions of the frames/video should have the highest quality. Therefore, streaming should be content-aware. Note that just cropping the face or blurring the background is not prudent, as it impairs the QoE. This is because online classes not only have scenes with the instructor talking, but also slides and whiteboards. Besides, occasionally, we even see instructors' videos hovering over the background (BG) with slides. These use cases make sending and rendering the BG with the foreground (FG)

indispensable. However, since the frequency of changes in the BG is lower and delays in rendering the changes in the background are not as significant as those in the instructor's body, face, and gestures, rendering an outdated BG suffices. Several works like [68, 69, 70, 71, 72] utilized content awareness in sports, lectures, gaming, and surgical videos. However, they focused only on playback speed control based on the content type and not on scheduling over multipath networks.

Our Solution: To resolve the problem of unreliable cellular networks, we propose COMPACT system that uses content-aware multipath live video streaming for live online classes, where we stream spatially segregated video content for transmission over different paths. For example, considering the case of the classroom, the portion of the video containing the teacher's face/body, the foreground (FG), is streamed at high quality through the path with the highest bandwidth and least latency (referred to as the primary path). The relatively static background (BG) content can be sent through the path with lower throughput, referred to as the secondary path. COMPACT reasons rendering the current FG with the previous BG in case the BG gets delayed, as the changes in BG are less frequent. Since modern smartphones have two interfaces suitable for video streaming – cellular and WiFi, it is sufficient to consider only a binary choice (FG and BG) to obtain an efficient solution.

To segregate a video into two different categories, FG and BG, we use the High-Efficiency Video Codec (HEVC/H.265) [73], as it supports creating *spatially independent rectangular blocks in a video, called tiles* (shown in Fig. 2.2(a)). Tiles can be removed, swapped, and/or added independently in real-time without affecting the rest of the video, provided the initial encoding uses tiles. Note that segregating only the speaker is computationally intensive, requiring image segmentation. Due to the changing BG, albeit with less frequency, BG subtraction gives poor accuracy. Using tiles gives us coarser yet faster FG detection using a face-detection model. We create two subsets of tiles; one has only the FG, typically containing the face and body of the speaker, while the other contains only the BG.

Content-aware Scheduler: Choosing the right path for the set of tiles is crucial because a wrong decision can lead to HoL blocking due to out-of-order delivery of tiles. For instance, if the BG tiles reach earlier, we would still have to wait for the FG tiles, causing unnecessary stalls. We found in our experiments that using conventional schedulers like MinRTT and Musher [74], used in MPQUIC and MPTCP, respectively, does not improve but leads to worse QoE (§2.1.3). Because the number of tiles to stream should depend on the asymmetric path characteristics. In cases where the FG set has fewer tiles than the BG set, a few tiles from the BG can be shifted to the FG if the path over which FG is being streamed has sufficient BW. Since conventional schedulers lack such knowledge, they fail to leverage the full potential of multiple available paths. Furthermore, the quality of the two tilesets can be varied independently based on the BW if streaming at a certain quality level is not possible.

COMPACT utilizes its own scheduler to fulfill the above requirements. The scheduler keeps track of the round-trip-time (RTT) and available bandwidth (BW) estimates of each path (we use two paths in this work). Depending on the RTT and the BW estimates, it either shifts tiles from BG to FG, varies the quality level, or does both. It finds an appropriate schedule by maximizing the utility function.

We implement COMPACT at the user level in Java using Stream Control Transmission Protocol (SCTP) [75, 76] under the hood. We use SCTP due to its in-order packet delivery with configurable retransmissions at the stream level to fix transport-level HoL blocking. Several prior works [43, 44, 45, 46] used it for video streaming applications. We evaluate COMPACT on the real network and in different scenarios using traces replayed using Mahimahi [77]. We find that COMPACT performs superior to the baselines, improving the median stall and E2E lag by 70.6% and 28.57%, and the tail stall and lag by 83.9% and $\approx$ 80% on a bus trace.

Contributions: Our main contributions are as follows:

1. We propose to leverage the different levels of importance of the spatial regions in streaming videos for online classes. This helps split the videos into foreground and background categories, which can be streamed through different network paths. To the best of our knowledge, we are the first to utilize such spatial segregation for multipath live video streaming.
2. We implement our own content and path-aware scheduler to address the shortcomings of the past multipath schedulers. COMPACT is open-sourced on GitHub[2] to spur further research.
3. We evaluate COMPACT on diverse network conditions, including a live test on the actual cellular network. It reduces the median stall by 3.4× and the $99th$ percentile tail stall by 6.2× compared to the single path. It further improves the median E2E lag by 28.57% and tail lag by $\approx$ 80%.

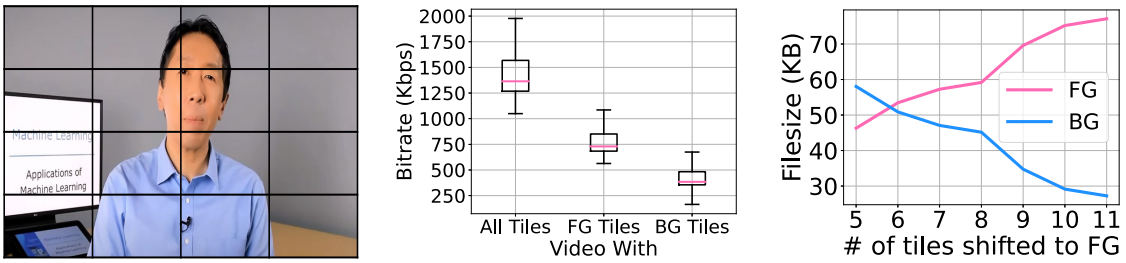
2.1 Background & Motivation

In this section, we first briefly discuss video tiles and their use cases in live video streaming. We then discuss the challenges of using tiles for multipath live streaming. Lastly, we study the use of conventional packet-level multipath schedulers at the tile level.

2.1.1 Video Tiles for Streaming

Modern video compression standards like HEVC [73], AV1 [78], and VVC [79] support encoding videos by splitting them spatially into a grid of rectangular blocks referred to as *tiles* (Fig.

[2]https://github.com/shubhamchdhary/COMPACT



(a) (b) (c)

FIGURE 2.2: (a) Video with 4×4 tiles. The foreground is highlighted in green and the background in blue. The video is publicly available under Creative Commons License [1]. (b) The bitrate of the full video is ≈1.7× and ≈4× the bitrate of foreground and background content, respectively. (c) shows that shifting tiles from the background (BG) to the foreground (FG) reduces the BG video size proportionally.

2.2(a)). These tiles are treated as independent entities, i.e., they can be manipulated without affecting the rest of the video. Due to their ease of manipulation and flexibility, tiles are widely used in 360° and VR videos [12, 80, 81, 82]. Tiles also have the advantage that removing them reduces video filesizes (refer to §). We show this in Fig. 2.2(c), where shifting a certain number of tiles from BG to FG increases FG filesize while decreasing BG filesize. This allows adjusting the number of tiles in each path to handle the differences in network bandwidths. Since tiles allow independent decoding, the FG can be rendered without waiting for the BG to arrive, reducing the overall latency. Tiles can also be encoded at different quantization levels, thus allowing another parameter to adjust according to the available network BW.

The separation of spatial content into FG and BG tiles reduces the overall video streaming bandwidth requirement. We show this in Fig. 2.2(b) where for a total of 400 video segments (a bunch of frames when encoded is called a video segment), the FG tiles bitrate (BR) is ≈ 1.7×, and the BG tiles bitrate (BR) is ≈ 4× lesser than the video with all tiles. This reduction in video bitrate demonstrates the utility of coupling tile-based streaming with multipath networking. This implies that if streaming a high-quality video over a single path is not viable, resorting to FG and BG tiles for multipath streaming makes it feasible without losing out on the quality of experience.

2.1.2 Tile-level versus Packet-level Streaming

Relying on tile-level streaming for live video delivery helps eliminate the packet-level HoL-blocking. We intuitively show this using Fig. 2.3. Consider the case of conventional multipath streaming at the packet level, where the encoded video segments are bifurcated into smaller packets, which are distributed across the multiple paths based on the network characteristics. At the other end (receiver-side), all packets should be received to decode and render the streamed segment. If any packet gets delayed, lost, or arrives out of order, the whole video reconstruction is hampered, leading to video stall/freeze. Moreover, if the paths are highly heterogeneous, the

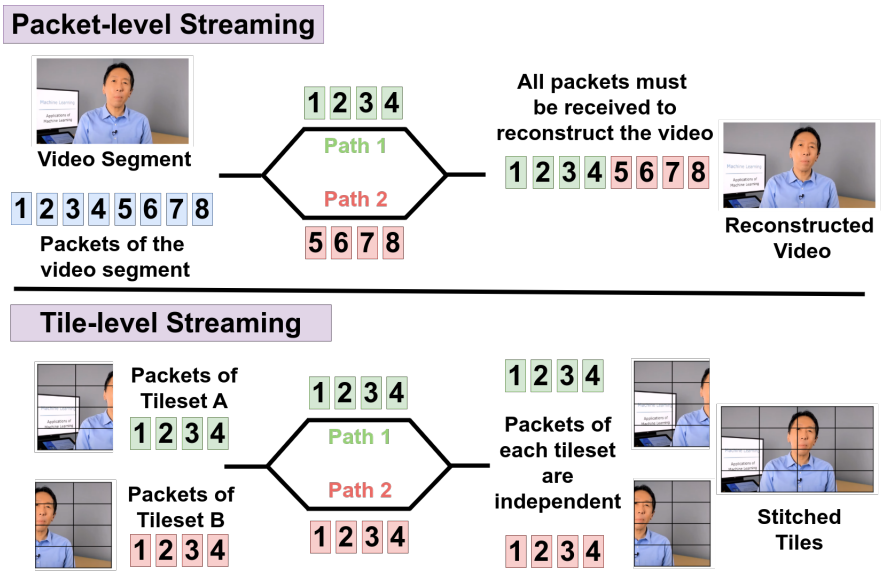


FIGURE 2.3: Difference between tile and packet-level multipath streaming. Streaming at the packet level distributes packets across the paths and mandates the reception of all the video packets for video reconstruction. While in tile-level streaming, all packets of a tile-set are streamed on a single path, making it independently receivable and decodable.

packets of the worst path will likely get delayed. Thus, even by reducing video quality and scheduling fewer packets on the worst path, there is no way to reduce video stalls.

Piggybacking on tile-level streaming helps resolve the packet-level HoL-blocking. This is true because, unlike distributing packets across multiple paths, in tile-level streaming, all the packets of a tile are streamed through one single path. This eliminates the possibility of HoL blocking caused by inter-path packet delay. We show this in Fig. 2.3 where the video is divided into two tilesets, A and B. *Tileset A* is streamed over *Path* 1 while *Tileset B* is streamed over *Path* 2. All the packets of a tileset are sent over a single path. Since tiles are independently decodable, even if the paths are highly heterogeneous, the tileset reaching first can be rendered without waiting for the other. Thus, it reduces the video stalls and allows FG and BG to be encoded at different quality levels.

2.1.3 Limitations of Conventional Schedulers

Fig. 2.4 demonstrates the stall duration and the application-level end-to-end (E2E) lag for different packet-level and tile-level schedulers (refer to §4.3 for details on the metrics and schedulers' implementation). We run the experiment on a 5-minute lecture video. We replayed Pensieve's [83] open-sourced bus trace files (bandwidth patterns shown in Fig. 2.10(b)) on both paths. We note that all the packet-level schedulers, PMin (MinRTT), PBF (Balanced Subflow Completion), and PMu (Musher), incur the highest median lag and introduce median stall similar to single-path streaming. This is evident because the path heterogeneity causes HoL blocking due

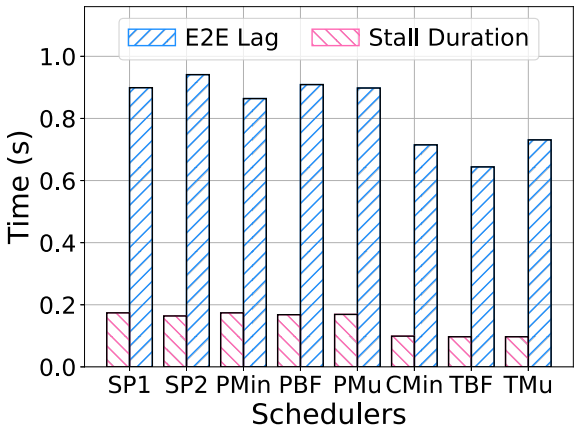


FIGURE 2.4: Median end-to-end (E2E) application-level lag and video stall duration of different packet-level (PMin, PBF, and PMu) and tile-level (CMin, TBF, and TMu) multipath schedulers compared to individual single paths (SP1 and SP2).

to out-of-order packet arrivals. Thus, conventional packet-level schedulers fail to fit for live video streaming.

If these schedulers are modified to work at the tile level, then all of them (CMin, TBF, TMu) perform better than packet-level schedulers (PMin, PBF, PMu) and single paths (SP1 and SP2) in terms of both E2E lag and stall. However, this improvement is marginal because of the tile-level HoL blocking. Although the tiles sent over the better path reach earlier, it has to wait for the rest to arrive, causing unnecessary lag. This problem persists even if the scheduler is content-aware, as illustrated by CMin (Content-aware tile-level MinRTT) being worse than TBF (tile-level balanced subflow). This example shows that being content-aware is not enough. This is due to the dependence of the FG on the BG. TBF achieves a superior tradeoff among all conventional schedulers. It manages to do so because it tries to equalize the completion times (time to send all the tiles of a set) of tiles across paths. TMu (tile-level Musher) behaves similar to CMin even though it distributes tiles in proportion to the available bandwidths. Therefore, content-aware schedulers need to distribute the tiles well to minimize the completion time.

2.1.4 Challenges of Using Tiles

While using tiles for multipath live streaming has advantages, three major challenges are associated with tile-level live streaming.

C1) Content-Obliviousness of Tiles: Tiles themselves do not have any knowledge of their content. Using an arbitrary splitting technique to get FG and BG sets leads to suboptimal performance (discussed in §2.4). Since there is no foreground and background, the receiver device is, therefore, obliged to wait for all the tiles to arrive before stitching. This causes unwanted HoL

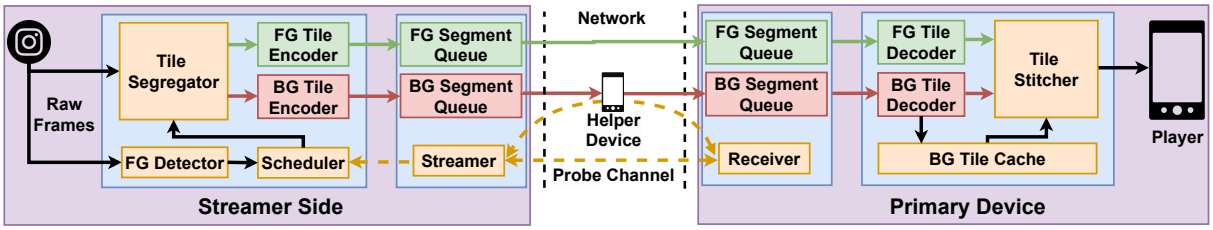


FIGURE 2.5: Architecture and workflow of COMPACT.

blocking, leading to high E2E lag and stall. Thus, it is imperative to divide tiles depending on the content so that if only the FG tiles reach, it is renderable with the recently cached BG tiles.

C2) Requirement of Fair Allocation of Tiles: After creating two sets of tiles, the BG can have more tiles than the FG. Since we stream BG over the low-bandwidth path, it may not fit within the bandwidth limit. In such cases, instead of encoding the BG tiles with a lower bitrate, some of its tiles can be allocated to FG because shifting reduces filesize (Fig. 2.2(c)). Such relocation of tiles gives the opportunity to stream BG tiles at relatively higher video quality as they are shifted to the FG tileset. It is, therefore, crucial for the system designer to consider appropriate tile allocation.

C3) Dependencies among FG and BG Tiles: Upon receiving all the tiles, the receiver stitches them as a single video and renders them on the screen. However, delayed reception of BG tiles can cause unwanted video stalls and HoL blocking. Although playing only the FG tiles is possible, it leads to worse QoE. It is important to break the strong dependency between FG and BG tiles.

2.2 Design of COMPACT

COMPACT's design is based on the following key observations: (1) at a particular time and location, all ISPs generally do not suffer equally from poor throughput [84], (2) a significant number of people often carry multiple smart devices that can be collaborated to leverage their aggregated bandwidth [50], (3) background in live online classes is mostly static, and changes are less frequent than the foreground, therefore, FG can be rendered with an outdated BG, (4) content-aware segregation of tiles allows streaming at a higher bitrate (better quality), and (5) BG can be encoded at a lower bitrate than FG because people pay more attention to FG [68, 69].

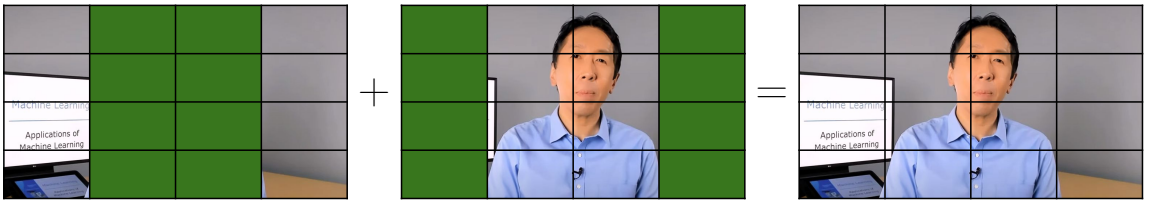
2.2.1 System Architecture

We show COMPACT's architecture and workflow in Fig. 4.7. At the streamer side, the camera sends raw frames to the system where *FG Detector* identifies the tiles containing any human face using a DNN (§4.3). Thereafter, the *Scheduler* uses the marked tiles to create FG and BG sets with tile indices. Based on the network statistics (RTT and BW), the scheduler decides what quality/bitrate to keep for the two categories (FG & BG), along with the number of tile shifts if needed (§2.2.2). Besides, it chooses the path with the minimum completion time $= \left(\frac{RTT}{2} + \frac{Filesize}{BW}\right)$, called the primary path, to stream the FG tiles. The other path is called the secondary. *BG tiles need not be streamed through the helper device only. It can be sent through the direct path too, depending on the network characteristics.*

Finally, *Scheduler* notifies the *Tile Segregator*, which passes the raw frames received from the camera to the FG and BG encoder to generate two tiled video segments, one for FG and another for BG. These encoded segments are stored in their corresponding sending queues (*FG Segment Queue* and *BG Segment Queue*). Although called *FG Tile Encoder*, the encoded segments can also have a few relocated tiles of BG. Subsequently, the *Streamer* module streams the video segments from the queues over the network through primary and secondary paths. In addition, it performs another important task of maintaining the network's RTT and BW estimates of each path using separate probe channels discussed in the next subsection. The *Helper Device* just relays the received tiles to the *Primary Device*.

As on the streamer side, the received tiles are first stored in queues on the primary device (where the user watches the video). The decoders for FG and BG fetch the segments sequentially from these queues and pass on the decoded frames to the *Tile Stitcher*. On receiving the FG and BG frames, the stitcher clubs these two subframes together to play as a single video (shown in Fig. 2.6). At times when the BG segments get delayed, the stitcher uses the last rendered BG stored in *BG Tile Cache* for rendering. The *Receiver* module communicates with the *Streamer* to echo back probe packets required for network RTT and BW estimation.

To get the right schedule, having a good enough estimate of network characteristics like RTT and BW is crucial. We create a total of four probe channels, two for RTT and two for BW. To estimate the RTT of each path, we send timestamps from the streamer machine to the primary, which is echoed back. The difference between the receiving time and the sent timestamp is the RTT. We attach timestamps in the custom headers of the sent video segment tiles to probe the network BW. The primary device dissects the header and replies with the demarcated timestamp. We use this timestamp to calculate the total sending time to infer network BW. The instantaneous RTT and BW are noisy in nature. Similar to prior works [85, 86], we use the harmonic mean (HM) of the past five values to smoothen the estimates.



(a) (b) (c)

FIGURE 2.6: Tile stitching: (a) background tiles and (b) foreground tiles stitched into a single video in subfigure (c).

Once the tiles are streamed from the sender based on an appropriate schedule, they might arrive at different moments in time at the primary device. If the BG tiles are delayed too much (more than the inter-frame delay), it can lead to stalls because we cannot render the FG tiles alone. Therefore, we need some way to handle the out-of-order arrival of the two sets of tiles. In online classes, the background (slides, whiteboard, or texts) is mostly static, and the changes are less frequent than those of the speaker (foreground). Therefore, the same BG can be rendered for multiple foregrounds. This opens the possibility of breaking the tight coupling between FG and BG tiles. In cases where the BG tiles are delayed, the previous BG can be stitched along with the current FG tiles to render.

2.2.2 Tile Scheduler

We now discuss our content-aware scheduler. Once we get the two sets of tiles for foreground (FG) and background (BG), respectively, timely delivery of both of them to the player side is crucial. Any delay caused increases the stall duration between two consecutive segments. For a video encoded at $25fps$, the inter-frame delay δ is $40ms$. Any extra time required beyond $40ms$ is, therefore, considered a stall/freeze. To have the best quality of experience, the quality (bitrate) of the played video should be the highest with the least stalls and quality switches within and across the segments. Borrowing from prior works on tiled streaming [87], we define a utility function to capture users' QoE as a weighted sum of user-perceived video quality, stall, inter and intra-segment quality switches.

Let there be total N tiles with m tiles in FG and n tiles in BG set. The variables Q_{FG} and Q_{BG} denote the quality defined in terms of the video quantization parameter (QP) of tiles in FG and BG, respectively. Let the size of each tile in FG and BG be $R_{FG}(Q_{FG})$ and $R_{BG}(Q_{BG})$ for a certain quality level Q_{FG} and Q_{BG} of foreground and background.

Filesize of Tiles: We note that our scheduler maps all tiles of FG and a limited number of tiles of BG to the primary path. Assuming a total of o BG tiles use the primary (i.e., the higher BW) path, the data F_P and F_S sent through primary and secondary paths are:

$$F_P = R_{FG}(Q_{FG}) \times m + R_{BG}(Q_{BG}) \times o; F_S = R_{BG}(Q_{BG}) \times (n - o) \tag{2.1}$$

User-perceived Video Quality: The video quality Q_t of a segment t, the user observes, depends on the quality of the FG and BG tiles rendered. Mathematically, if β_1 and β_2 denote the weights given to the quality of FG and BG, respectively, the video quality Q_t is:

$$Q_t = \beta_1 Q_{FG} + \beta_2 Q_{BG} = \beta_1 (m+o) q_t(l_{FG}) + \beta_2 (n-o)(\mathbb{1}(K) q_t(l_{BG}) - p(t)) \tag{2.2}$$

Here, the function $q_t(.)$ returns a normalized quality value for a given quantization level l_{FG} and l_{BG} for the FG and BG, respectively. In case of delayed delivery of BG tiles beyond a threshold, the player can choose to render received FG tiles by stitching them with the previous segment's BG tiles, caching the current ones. This depends on the completion time captured using the indicator function $\mathbb{1}(K)$ which is 1 if $\left(\left(\frac{RTT_S}{2} + \frac{F_S}{BW_S}\right) - \left(\frac{RTT_P}{2} + \frac{F_P}{BW_P}\right)\right) < \delta$ else 0. The path with the least completion time is referred to as the primary path P where FG tiles are scheduled. Here, BW_P and BW_S are the estimated bandwidths, and RTT_S and RTT_P are the estimated RTTs of primary and secondary paths, respectively. $p(t) = max(Q_t)$ penalizes if the BG tiles of past segments are rendered. We set β_1 and β_2 to 0.6 and 0.4, respectively to prioritize FG tiles.

Quality Switches: The quality levels of the tiles on the player side can be different across and within a segment. There should be a graceful degradation of quality for better QoE. We use I_1 and I_2 to penalize abrupt changes to control inter-segment and intra-segment quality switches, respectively. We define these as:

$$I_1 = |Q_t - Q_{t-1}| = |q_t(l_{FG+BG}) - q_{t-1}(l_{FG+BG})| \tag{2.3}$$

$$I_2 = |Q_{FG} - Q_{BG}| = |q_t(l_{FG}) - q_t(l_{BG})| \tag{2.4}$$

Stall Duration: The stall duration depends on the completion time of a video segment. A stall occurs when the tiles of the current segment reach after the play time of the previous segment P_{t-1}. Therefore, assuming primary path (P) is the fast path, for a segment t the stall L_t can defined as:

$$K_{max} = max\left(\frac{RTT_P}{2} + \frac{F_P}{BW_P}, \mathbb{1}(K)\left(\frac{RTT_S}{2} + \frac{F_S}{BW_S}\right)\right) \tag{2.5}$$

$$L_t = max((K_{max} - P_{t-1} - \delta), 0) \tag{2.6}$$

Here, K_{max} is the maximum of the sending times on the primary and secondary paths, the indicator function $\mathbb{1}(K)$ becomes zero when all BG tiles are shifted to FG, otherwise, it is one. The objective of the scheduler is to split the total tiles into primary and secondary paths in such a way as to maximize the utility function below.

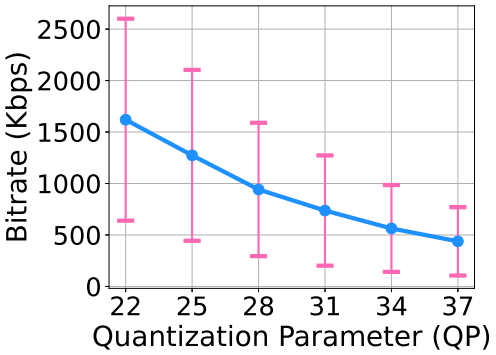


FIGURE 2.7: Video bitrate reduces with increasing quantization parameter (QP).

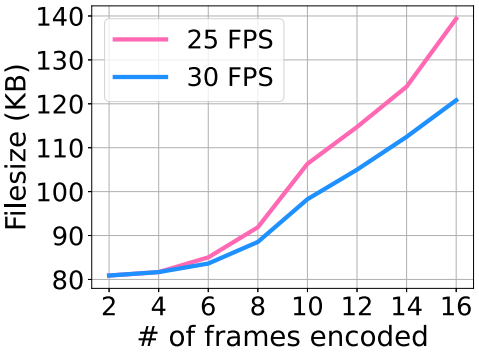


FIGURE 2.8: Increasing the # of frames in the encoded video segment increases filesize.

$$\textbf{Maximize } Utility = \alpha_1 Q_t - \alpha_2 I_1 - \alpha_3 I_2 - \alpha_4 L_t - \alpha_5 S_t \tag{2.7}$$

Here, S_t is the absolute difference in the number of tiles shifted from the background to the primary path between the last and the current segment. It refrains from abrupt tile relocations between two adjacent segments. We borrow the weights assignment strategy of [83, 88] keeping $\alpha_1 = max(Q_t)$, α_2 to α_4 equal to 1, and make $\alpha_5 = 0.1$. We use an exhaustive search over all possible tile shifts, FG, and BG QP changes to get the right schedule for every segment, which takes $<= 1ms$.

Moreover, we need to estimate the filesize of tiles for each QP value while scheduling because the segment is encoded after getting the right schedule. To estimate the filesize, we rely on Fig. 2.7, which depicts that for a QP reduction of 3, the filesize/bitrate increases by 1.2×. We use the filesize of the previously streamed segment and a multiplying factor of 1.2 to approximate the filesize for every QP during the exhaustive search.

2.3 Implementation

We implement the host/streamer and the joinee/viewer side of the COMPACT in Java. We use a Linux machine as the host device and two other Linux PCs as helper and primary devices. We now discuss each of the components in detail.

1. **Video Encoding:** Since COMPACT uses tiled videos for streaming, we use an open-source HEVC encoder, Kvazaar [89], to encode the raw frames into a grid of 4×4 tiles. Other configurations are also possible. However, we empirically found this configuration works well for us as it balances encoding overhead and stitching complexities similar to prior works utilizing tiles [9]. Our encoding pipeline begins by capturing raw frames from the screen (simulating a camera) using FFmpeg [90]. We then encode these frames into a tiled HEVC

video. We use GPAC [91] for tile manipulation and packing segments into an mp4 file. This process repeats for each video segment. We keep the framerate fixed at 25 (used by most live video streaming works [40, 92]). Each segment contains four frames, balancing encoding time and compression efficiency. We note from Fig. 2.8 that segments with 4 frames have approximately the same mean filesize as 2 frames, but it escalates faster from 6.

2. **Foreground Detection:** We argued in §4.2 that COMPACT is content aware. We use a pre-trained ResNet model available at [93] for face detection to detect the FG content. Once we have the coordinates of the bounding boxes of the detected faces, we identify the tiles containing these boxes as the FG tiles and the rest as the BG tiles. In the case of presentation slides, the model detects no human face, so we keep a horizontal split with eight tiles in both FG and BG.

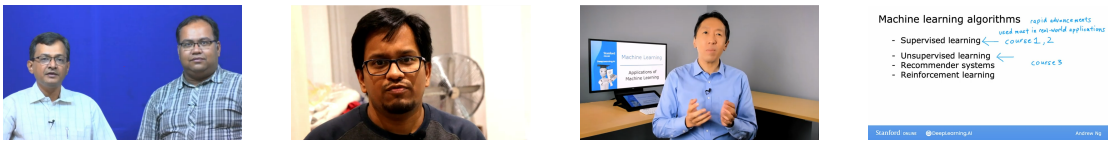
3. **Transport Layer:** We use Stream Control Transmission Protocol (SCTP) to stream the video segments because it provides in-order delivery, configurable retransmission, flow, and congestion control. We use UDP under the hood for all the control channels. Apart from SCTP, QUIC [94] is another transport protocol that precludes transport-layer HoL-blocking. QUIC is built on top of UDP, borrowing ideas from SCTP. Since SCTP is a relatively older protocol, multiple live-streaming techniques [43, 44, 45, 46, 95, 96, 97] have used it. QUIC, till date, is mostly explored in video-on-demand setups. While we have explored utilization of QUIC in our concurrent work, we do not discuss it here as the focus of our thesis is on the decision-making.

4. **Helper & Primary Device:** The helper's job is just to relay the received tiles to the primary device. To keep the overhead minimal, we use the Linux utility *socat (SOcket CAT)* [98], which acts as a socket-based bidirectional stream relay running on the helper. The primary device receives all the tiles, stitches, and plays them. Once the tiles of both paths are synchronized, we use OpenCV to stitch/join and render the FG and BG tiles.

The different tasks performed on the streamer and the receiver devices are asynchronous. We use multi-threading to parallelize the tasks. For instance, the video segment encoder can independently capture and encode videos, while the sender thread can parallelly stream the encoded videos. The encoder should not wait for the sender thread to complete sending. We use evicting queues[3] of size two to exchange information between threads.

Baselines: We compare COMPACT with the different conventional schedulers at both the packet level and tile level. All the tile-level schedulers, except CMin, are unaware of the video content.

[3]FIFO queues with auto-dequeuing on exceeding queue size are referred to as evicting queues.



(a) Video 1 (b) Video 2 (c) Video 3 (d) Video 4

FIGURE 2.9: Screenshots of the four videos used for evaluations.

1) Single Path: For the single path, we always stream on one of the paths without any scheduler. We denote as SP1 when using only Path 1 and SP2 when streamed on Path 2 only.

2) MinRTT: This is the default scheduler in MPQUIC and MPTCP. At the packet level (PMin), it sends all the segments' chunks (MTU-sized packets) through the path with minimum RTT. We even make it content-aware at the tile level (CMin) to show that, without tile shifting and out-of-sync FG and BG playback, it behaves similarly to oblivious tile-level schedulers. CMin sends FG tiles to the minimum RTT path and BG tiles to the other one.

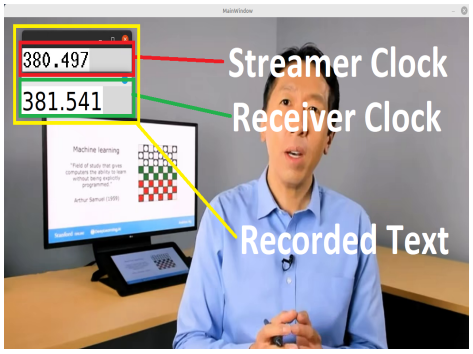
3) Musher: This is another MPTCP scheduler that allocates packets proportional to paths' bandwidth (PMu) [74]. At the tile level (TMu), we assign tiles based on the available bandwidth of the paths.

4) BFlow: We implement balanced subflow completion (DEMS [99]) at both packet (PBF) and tile level (TBF). The core objective of BFlow is to minimize the difference between the completion times of the sent data (packet for PBF and tiles for TBF) on both paths.

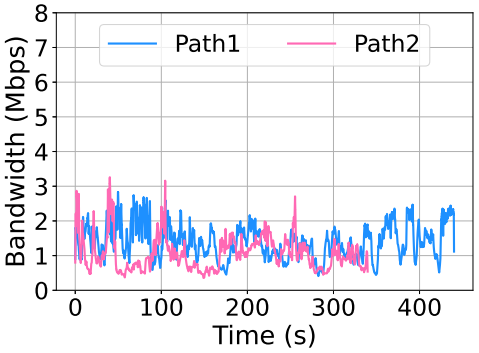
Dataset: We selected four lecture videos from YouTube [100, 101, 102, 103], each lasting 4-5 minutes. Fig. 2.9 shows screenshots of the videos used for the experiments. The videos have a mix of scenes, with Video 1 having two speakers delivering a lecture. Video 2 and Video 3 have one speaker in the FG with different BGs. Video 4 is similar to Video 3, with the only difference being that it has slides showing up frequently. We used these videos as they represent the typical online class setup and provide us with sufficient diversity to check the robustness of our technique. For each session, we replay these downloaded videos with a live digital clock to calculate E2E lag.

Performance Metrics: For extensive evaluations, we employ E2E lag and video stall to quantify the quality of experience. We do not report the quality of baselines individually in the evaluation because we kept it identical to COMPACT. This ensures a fair comparison of factors, such as end-to-end latency and stalls. Thus, we first ran COMPACT, logged its FG and BG quality for each segment, and then reused it in all other baselines while streaming.

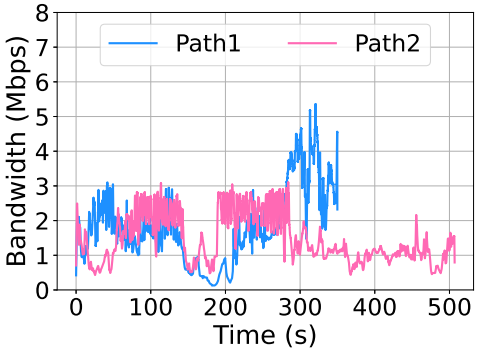
Our application-level E2E lag is defined as the difference between the time a frame appears on the streamer screen/camera and when it reflects on the viewer's screen. It depicts the actual lag that the user perceives. To measure E2E lag, we follow the strategy shown in [104], where we put



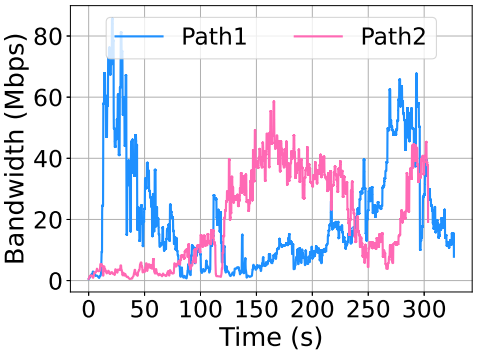
(a) E2E Lag Computation



(b) Bus trace



(c) Car trace



(d) Walk Trace

FIGURE 2.10: (a) Screenshot showing both the streamed digital clock and the local one. We record only the portion inside the yellow box for text detection to calculate lag. (b), (c) and (d) depict the bandwidth patterns of the replayed trace for the experiments.

a live digital clock on the video we stream. We also run the same clock on the receiver end and put it on the playing video. Afterward, we use a screen recorder to capture only the portion of the screen displaying the clocks (yellow box in Fig. 2.10(a)). Thereafter, we extract frames from the recorded video and run PyTessaract [105] OCR to recover the clocks' text. The difference in the detected texts is the E2E lag. While recording the clock texts, we reduced the capture rate to $10fps$ to reduce the number of OCR inferences, which is time-intensive. Note that we discard a few frames where the OCR fails to detect texts ($\approx 8\%$ on average). Our stall represents the inter-segment delay, which is the interval between the render time of the last frame and the first frame of two consecutive video segments.

2.4 Evaluation

We now discuss the performance of COMPACT under various network conditions and the overheads of its different components.

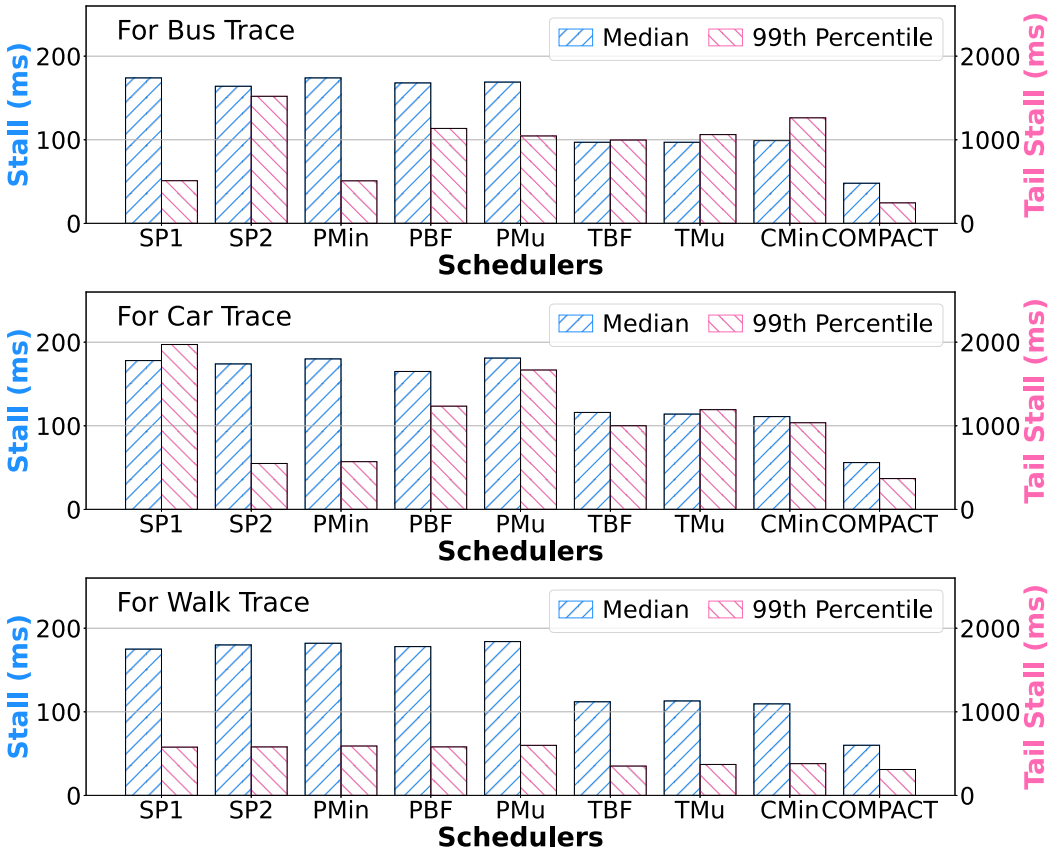


FIGURE 2.11: Stall with traces while traveling in a bus, traveling in a car, and walking

2.4.1 Trace-driven Results

To extensively evaluate COMPACT under various network conditions, we use trace files [106], open-sourced by Pensieve [83], collected while traveling in a bus and a car. Besides, we even collected two walking traces ourselves. We chose these traces to evaluate COMPACT under challenging network cases with high bandwidth fluctuations and RTTs. Since many countries in the world still witness poor bandwidth (< 3Mbps) [107, 108], we scale down the bus and car trace accordingly to the same range to evaluate for such scenarios. We report the bandwidth patterns in Fig. 2.10. In each case, we replay two traces of the same category on the two available paths using Mahimahi. If a trace ends early, it gets replayed automatically. We show the video stall and E2E lag in Fig. 2.12.

Stall: When streaming over a single path, we note that the median stall reaches $180ms$ on all traces. The median performance of packet-level schedulers (PMin, PBF, and PMu) is similar to or slightly worse than single paths. This is due to the HoL blocking caused by the out-of-order delivery of packets because of path heterogeneity. Therefore, packet-level schedulers fail to leverage the benefits of multiple paths. The network characteristics of bus and car traces are

distinct, therefore, the tail values ($99th$ percentile) of single paths differ by $\approx 3\times$ while on the walk trace, it is almost the same for packet-level schedulers and single paths. Except for the PMin, the rest of the packet-level baselines experience larger tail stalls. Since PMin picks the path with the minimum RTT to stream all packets through that path only, it manages to avoid out-of-order packets.

At the tile level, conventional schedulers reduce the median stall by at least 35% compared to the best single path (single path with the least stall) and observe a lower stall than packet-level baselines. Even the basic content-aware tile-level MinRTT (CMin) scheduler shows a trend similar to oblivious tile-level schedulers on all traces, validating that mere content awareness is insufficient. Tile-level schedulers are superior to packet-level schedulers because tiles intended for a path are sent only through that path, thus completely removing inter-path HoL blocking because only a smaller size BG is scheduled on poorer paths, unlike packet-level schedulers that do not consider this. However, the tail stall still dominates due to the dependency between FG and BG tiles.

Compared to all baselines COMPACT induces the least median and tail stall with $48ms$ and $245ms$, respectively, on the bus trace, $59ms$ and $379ms$ on the car, and $59ms$ and $307ms$ on the walking trace. On the bus trace, it experiences $3.4\times$ less median stall than the single path best, $3.5\times$ less than the best packet-level scheduler (PBF), and $2\times$ than the best tile-level scheduler (TBF). Similarly, the reduction is $2.9\times$ (single path best), $2.4\times$ (PBF), and $1.9\times$ (CMin) on the car trace and $\approx 3\times$ (single path best), $3\times$ (PMin), and $1.9\times$ (CMin) for the walking trace. In terms of tail stalls, on the bus and car traces, COMPACT manages to reduce it by at least $1.5\times$ compared to the single path best and packet-level best, while $2.6\times$ in contrast to tile-level best. On walking trace, its tail is $44ms$ smaller than TBF (best) and $84ms$ than PMin. The asynchronous playback of FG and BG tiles of COMPACT helps alleviate the video stalls dramatically.

E2E Lag: In terms of E2E lag, on all traces COMPACT experiences either the least lag or is comparable to tile-level schedulers. However, in terms of tail lag, it is always better than TBF and TMu on all the traces. The lag (median E2E lag) of single paths and packet-level baselines is almost similar in all the scenarios. Tile-level schedulers are marginally better than packet-level schedulers in terms of lag. Except for MinRTT (PMin and CMin), the rest of the schedulers witness large tail lags on the car trace. Since the bandwidth is superior in our collected walking trace, all the schedulers manage to keep that median lag below $1s$ and tail lag below $4s$.

COMPACT's median lag is 28% lower than single path best, 23% than the lowest of packet-level schedulers, and similar to the lowest of tile-level. Compared to TMu and TBF (both perform almost the same), the $99th$ percentile lag is $1.7\times$, $1.1\times$, and $2.5\times$ less on bus, car, and walking trace, respectively. We note that all the schedulers are consistently better on the walking trace due to the higher bandwidth (Fig. 2.10(d)). The evenness vanishes when the paths'

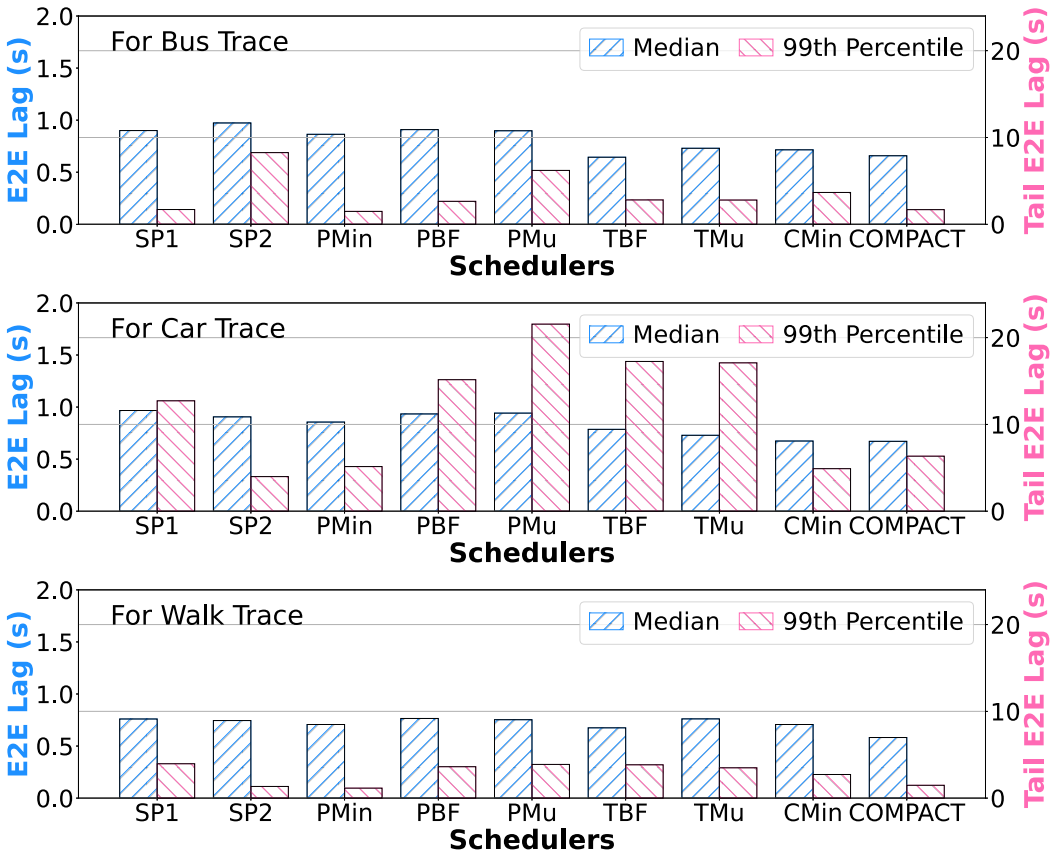


FIGURE 2.12: E2E lag with traces while traveling in a bus, traveling in a car, and walking

characteristics differ under the bus and car traces. Due to the same streaming video quality, all schedulers encounter a similar median E2E lag. All the above observations vouch for using tiles and multipath-based live-streamed classes.

Out-of-order FG and BG stitching: In the cases where BG gets delayed, COMPACT stitches the current FG with the cached most recent BG. This helps significantly reduce video stalls and lag. We quantify in Fig. 2.13 the number of times such out-of-order stitching occurs while running each of the traces for all four videos. We note that these out-of-sync playbacks happened 46.8%, 43.6%, and 40.3% on the bus, car, and walk trace, respectively. These events are fewer on the walking trace due to the more consistent bandwidth compared to the bus and car traces.

Perceived Video Quality: For all the packet and tile-level schedulers, the player renders the streamed video without any changes. Therefore, there is no scope for quality degradation compared to the encoded video streamed from the streamer. However, COMPACT stitches the previous BG in case it gets delayed. Now, we show that such asynchronous stitching does not hurt the user experience because the BG remains mostly static. VMAF is known to capture human-perceived video quality more accurately than PSNR and SSIM. Therefore, we report the

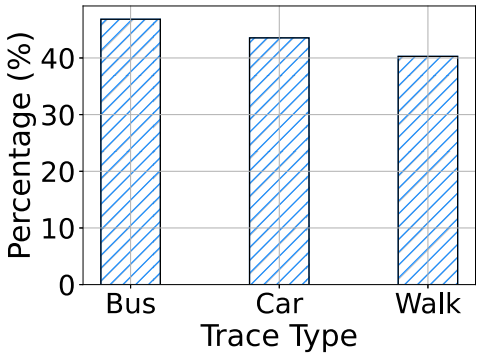


FIGURE 2.13: Out-of-sync Playback: shows the proportion of time FG and BG tiles played out-of-sync.

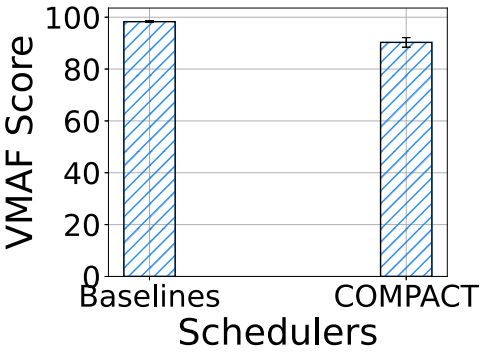


FIGURE 2.14: Perceived mean video quality (VMAF score) for all four videos on the bus trace.

VMAF score. We show the mean video quality of all four videos perceived by the viewer in terms of VMAF score [109] for the bus trace in Fig. 2.14. We utilize the open source [110] implementation of VMAF for calculations. We note that compared to the baselines, the VMAF score of COMPACT drops marginally by 8.15% even though it renders a previous BG 46.8% (Fig. 2.13) of the time on the bus trace. This implies that although out-of-order FG and BG stitching theoretically degrades the video quality, it is marginal in practice. Note that the quality for all the baselines remains the same because we use the same streaming quality for all of them.

2.4.2 Adaptability to Path Fluctuations

To validate COMPACT's adaptability to drastic and abrupt network fluctuations, we create two traces for each path. We fixed $Path1$'s bandwidth to $1.6Mbps$ and $Path2$'s to $1.2Mbps$ with a standard deviation of $10Kbps$. We make the $Path2$'s bandwidth suddenly dip to $100Kbps$. We replay these traces and run COMPACT for 4 min on a video with only slides. To confirm the desired trace patterns, we run Iperf to get the bandwidths of both paths in Fig. 2.15.

To capture COMPACT's response to the abrupt change in the BW, we show the number of tiles scheduled on each path in Fig. 2.15. We observe that before $135s$, our scheduler distributes the tiles proportionally to the paths' estimated bandwidths. However, soon after, when $Path2$ went down, COMPACT detected the change and adapted accordingly by gradually shifting most of the tiles to $Path1$ only, demonstrating robustness to sudden network variations.

2.4.3 Live Experiment

For the live experiment, we run COMPACT on the actual cellular network. As shown in Fig. 2.1, we keep the same setup and use three Linux desktops: one acting as the streamer to live

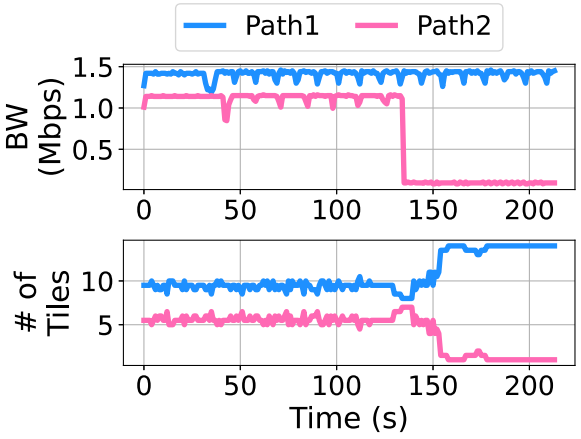
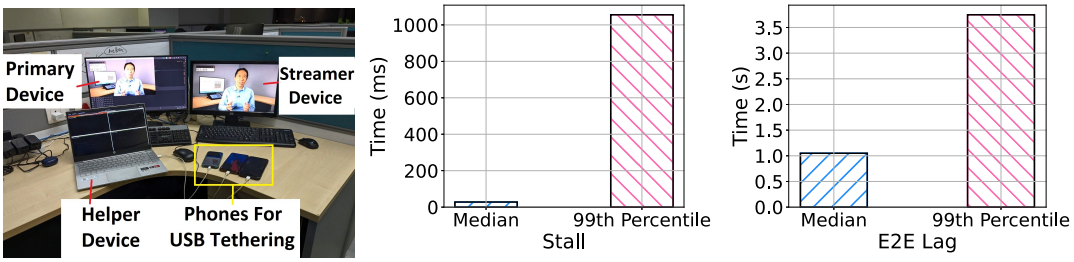


FIGURE 2.15: Number of tiles scheduled on each path when the network changes drastically.



(a) Our Setup (b) (c)

FIGURE 2.16: (a) shows our live experiment setup. COMPACT observed (b) stall and (c) E2E lag while streaming over the actual cellular network during the live experiment.

stream videos (with 15th Gen Intel i7 having 16GB RAM), another as the helper (with AMD Ryzen 5 with 16GB RAM), and the third one as the client playing the video (equipped with 12 Gen Intel i7 with 32GB RAM). All machines access the internet via the cellular network, using three smartphones via USB tethering. Furthermore, we connect the primary and helper desktops together using a separate WiFi connection to relay the tiles. In actual deployment, the application on the primary can be made to discover and connect to an available helper device. Like trace-driven experiments, we streamed one of the four videos that ran a live digital clock to measure E2E lag and stall (reported in Fig. 2.16). We utilized the publicly accessible IPv6 addresses from the helper and the client to connect to the streamer. We note from Fig. 2.16 that the experienced median stall is $28ms$ with a tail of $1055ms$. The median lag is close to $1s$ with a tail latency of $3.75s$. The mean QP for the FG and BG tiles during the streaming was 33. Here, the lag values are worse than trace-driven results due to a poor cellular network, with a mean BW and RTT in the order of $833.4Kbps$ and $425ms$, respectively.

Table 2.1: Overheads of different components of COMPACT

Encoding	FG Detection	Scheduling	Stitching
277 ms	56 ms	1 ms	1 ms

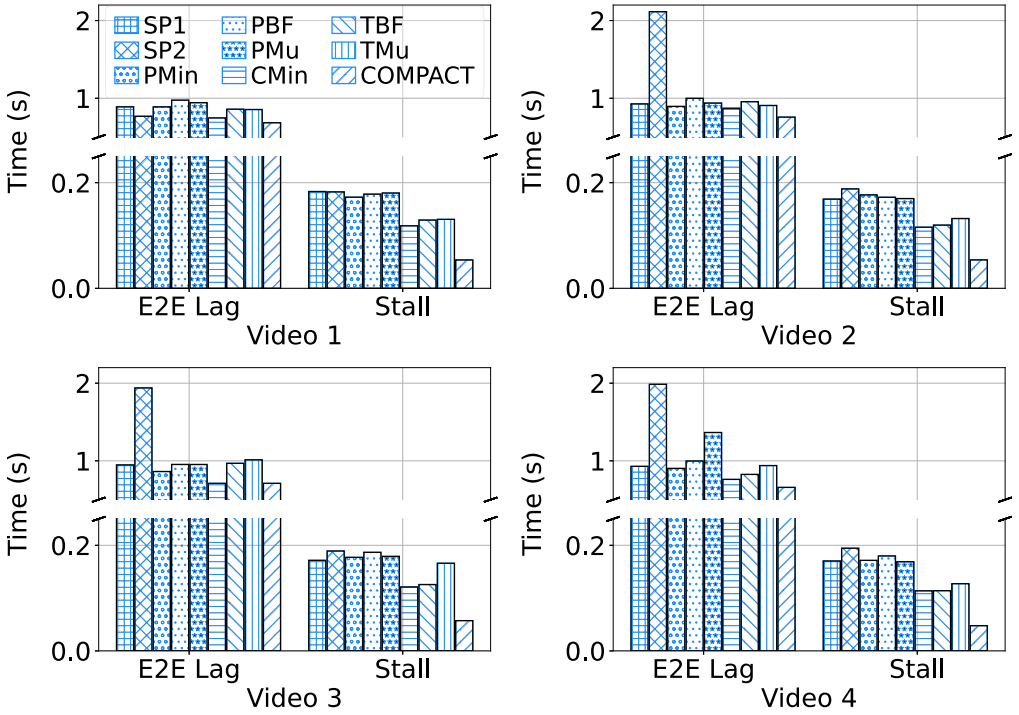


Figure 2.17: End-to-end (E2E) application-level lag and video stall duration of the different schedulers on individual videos used in the large-scale evaluation for the bus trace. COMPACT consistently performs better than the baseline schedulers

2.4.4 Microbenchmarks

Overheads: We log in Table 2.1 the mean overheads (in ms) of each individual component of COMPACT. We observe that scheduling and stitching take almost negligible time of $1ms$. Tiled video encoding incurs the greatest latency due to the software-based encoder Kvazaar. We rule out using a faster preset as it degrades the final video quality. Moreover, screen capture using FFmpeg also contributes to the encoding overhead.

Our encoding happens in two steps: raw frame capture and HEVC encoding, and FG and BG generation, where the former takes most of the time ($> 85\%$). On average, the model used for foreground detection takes $56ms$ to recognize human faces in the streamed videos. We parallelize detection with the frame capture thread to amortize this overhead.

Video-wise Results: We show COMPACT's median lag and stall on individual videos in Fig. 2.17 for the bus trace to show the effect of different video types. $Video1$ consists of two speakers with no slides, and the other three videos have only one speaker. Specifically, in $Video2$

and $Video3$, the instructor uses slides to convey whatever he is teaching. $Video4$ has animation playing in the video in between.

We note that COMPACT consistently achieves lower lag and stall than the baseline schedulers across all videos. All the packet-level schedulers incur stalls similar to single-path streaming, thus failing to benefit from multiple paths. Because inter-path out-of-order packet delivery is absent, tile-level schedulers are better than packet-level schedulers. Among the tile-level schedulers, CMin gives the fewest stalls. The out-of-order playback helps COMPACT to have the least stall. Since only $Video2$, $Video3$, and $Video4$ have frequently changing screens between slides and the speaker, they are encoded at a higher bitrate than $Video1$. Therefore, the E2E lag and stall are highest for them for SP2 (the worst path).

2.5 Related Works

Content-aware video streaming: Streaming video with consistent quality over a cellular network is difficult due to its variable bandwidth, packet loss, and latency. HotDASH [111] relies on temporal content awareness for adaptive video streaming. They decide the video's encoding rate based on the network characteristics. This can lead to poor streaming quality due to the single path's limited bandwidth. Works like [11, 80] prioritize content based on the user's viewport to adjust quality in 360° videos using tiles. ZGaming [72] predicts frames based on content type to improve cloud gaming. However, none of these works employ multipath.

Multipath Video Streaming: Several works [43, 44, 45, 46, 47, 112] utilized multipath to bolster the video streaming performance. XLINK [48] improved the quality of short video streaming using MPQUIC. MPDASH [113] employs MPTCP [58, 59, 60] to stream video adaptively across multiple interfaces. MPRTP [64] creates multiple subflows to deliver media packets over multiple paths. However, live streaming is more challenging than video-on-demand due to its tight deadline requirement. Converge [40] and TwinStar [41] used multipath WebRTC for live video conferencing and live cloud gaming, respectively. AggDeliv [42] optimize congestion control in MPQUIC for mobile live video delivery. Chorus [112] integrates adaptive bitrate streaming with multipath networks to optimize the quality of experience. However, none of these works utilize content-awareness to address the packet-level HoL blocking.

Multipath Schedulers: Several multipath schedulers, such as Musher [74], DEMS [99], ECF [114], and DAMS [115], have been proposed and work at the packet level. Although DEMS and DAMS consider the deadline requirements, they are unaware of the content of the packets. Thus, they fail to mitigate inter-path HoL blocking at the application level. Employing these schedulers in live video streaming worsens the situation (§2.1.3). Therefore, the scheduler tailored for live streaming should be content-aware to distribute packets well across multiple paths.

2.6 Discussion and Limitations

Having multiple peers: COMPACT currently supports peer-to-peer video streaming with only one participant. Extending to a multi-participant setup will require obtaining network statistics for each attendee for scheduling. This can incur overhead on the streamer side. We intend to explore possible mitigation strategies to amortize overhead costs.

Catering to diverse applications: Currently, we show results for live online classes. However, the setup can serve different applications. The major challenge is correct foreground detection, which depends on the video application. A possible direction is to use a saliency-based FG segragator. Moreover, COMPACT captures a few frames and encodes them into segments (similar to HTTP low-latency live streaming), which are streamed over different paths. Although the encoding is parallelized via pipelining, the overhead of the first segment dominates, resulting in high latency. One possible mitigation strategy is to stream on a frame basis, as done by WebRTC. Here, frames are captured, encoded, and streamed continuously. Such an architecture can make COMPACT suitable for video conferencing-like applications.

Accurate bandwidth estimation: Our scheduling decision relies on bandwidth estimation. Underestimation leads to poor quality, and overestimation can cause unwanted video stalls. COMPACT currently uses a relatively simple bandwidth estimation technique, leaving the use of a more sophisticated, accurate predictor for future work. An accurate predictor can improve link utilization, albeit at the cost of increased computation.

2.7 Conclusions

In this work, we propose a system COMPACT for live video streaming intended for online classes over cellular networks. It utilizes multiple devices at the user end to distribute video tiles. We observe that the conventional packet-level scheduler incurs inter-path HoL blocking. Therefore, COMPACT utilizes a video content-aware scheduler that splits a video into foreground and background using tiles and sends these tiles over different network paths. We evaluate COMPACT under various network conditions using simulated, controlled, and real setups. On a bus trace, COMPACT manages to limit the median stall by 3.4× and the tail stall by 6.2× compared to the single-path case. It also reduces the median E2E lag by 28.57% and tail lag by $\approx$ 80%. COMPACT, therefore, is likely to increase access to education in connectivity-challenged regions. Since tiles can be manipulated in real-time without re-encoding. We envision its use in traffic surveillance as a natural extension. This is possible because, as in an online class, the scenes captured by traffic cameras consist of vehicles and other irrelevant regions, such as trees and buildings. Therefore, tiles can be utilized to filter out these regions. We discuss this extension of our current strategy in the next chapter.

Chapter 3

TileClipper: Lightweight Selection of Regions of Interest from Videos for Traffic Surveillance

In recent years, real-time traffic surveillance has become important for automated enforcement of traffic rules [14], traffic light control [15], and the detection of anomalous events such as accidents [16]. Cities like Shanghai, New Delhi, and New York have installed hundreds of thousands of surveillance cameras[1].The video feeds from these cameras are either processed locally or sent to cloud/edge servers for application of computer vision algorithms. These algorithms run deep neural networks (DNNs), which are inherently compute-intensive. Processing it locally requires expensive hardware (such as GPUs and NPUs), thereby increasing traffic surveillance costs and limiting scalability. On the other hand, a major challenge for techniques that send video feeds to servers is the high data rate, up to 1 Mbps per camera [7], leading to high network bandwidth consumption. Thus, it is essential to find techniques to reduce bandwidth consumption without sacrificing the quality of traffic surveillance.

Current techniques of reducing bandwidth typically utilize one or more of two strategies. The first technique is to intelligently select the objects or frames of interest to be sent to the cloud server [6, 7, 116]. However, video is usually encoded so that pixels are defined as offsets from their neighboring pixel values, a compression strategy where the neighbors can be either spatial or temporal. Thus, only sending the objects or frames of interest would require re-encoding, which mandates either integrating the algorithm into the camera's firmware [117] or re-encoding on the device directly connected to the camera [6, 118]. Adding such a capability to the cameras or devices attached to them would require a substantial investment. Further, while this technique can save significant bandwidth during off-peak hours, it is difficult to do so when traffic is congested. The second technique is to perform additional computation on the cloud server by either running more powerful models [119] or sending a signal to the camera to send additional data only when needed [7]. This technique saves bandwidth at the cost of additional GPU usage,

[1] https://www.comparitech.com/vpn-privacy/the-worlds-most-surveilled-cities/

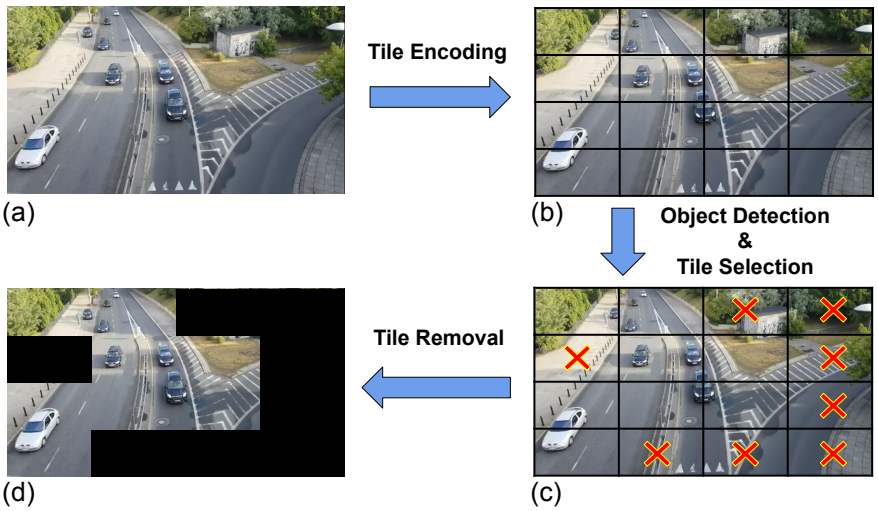


FIGURE 3.1: Illustration of TILECLIPPER. Steps: (a) encoding the video in the form of tiles supported by the HEVC format, (b) selecting only the relevant tiles by using the statistical pattern of bitrates, (c) clipping out the unnecessary tiles, and (d) sending this filtered video to the server for surveillance.

which is also expensive and energy-intensive. Thus, a solution that runs *without increasing computational cost and is easy to integrate with existing systems* is essential to reducing the cost of traffic surveillance.

In this work, we focus on traffic surveillance of *moving* objects, such as vehicles and pedestrians. To this end, we avoid the re-encoding problem proposed in prior works as follows. Video standards like HEVC or H.265 [73] allow the videos to be split into tiles (Figure 3.1b). This allows us to send tiles containing only the objects of interest (moving vehicles or pedestrians) while omitting the other tiles without re-encoding. Figure 3.1 shows the workflow of our system. The key advantage of removing tiles is that it runs in real time, even on embedded platforms like the Jetson Nano, unlike techniques that filter frames or objects of interest and re-encode videos. As surveillance cameras with native HEVC support become increasingly available [120], removing tiles is easy to integrate into deployed surveillance systems.

However, identifying tiles containing objects of interest generally requires running DNNs, which are themselves computationally intensive. The camera usually lacks sufficient compute capability or even power to run any such algorithm. Thus, the key challenge of selecting tiles with objects of interest is to *avoid any increase in the compute involved either on the camera side or on the server side*. We address the challenge of identifying tiles using a statistical approach grounded in a few observations. Our first observation is that since our cameras are stationary, only the objects of interest, such as vehicles and people, are in motion or can change. This implies that identifying just objects in motion is sufficient to identify objects of interest. Our second observation is that tile file sizes (or bitrates) increase significantly when there is any motion in them. These two observations enable us to design a heuristic to identify tiles with objects of interest. Note that the tile bitrates are also influenced by other factors, such as weather

conditions, local road conditions, the color of the objects, and the camera's position. Finally, we design a system TileClipper that applies a threshold to the past tile bitrates to identify tiles that contain objects of interest. TileClipper computes this threshold by performing a grid search on the statistics of tile bitrates during calibration. The grid search shows the correct percentile values for the tile bitrates, separately for tiles with and without moving objects. It then uses the midpoint between these identified percentile values as the threshold. This threshold automatically adapts based on weather and traffic conditions for each tile independently. We evaluate TileClipper across a variety of videos by running it on embedded platforms such as the Jetson Nano and Raspberry Pi 4B (RPi). We present results from object detection, as our system is agnostic to applications such as identification and tracking. Our dataset includes 55 videos from standard benchmarks under various weather, lighting, and traffic density conditions, as well as our own recorded videos.

Our evaluation shows that TileClipper achieves, on average, over 92% accuracy in identifying objects while running in real-time, even on RPi and Nano. This accuracy is higher than state-of-the-art frame filtering techniques like Reducto [6] and comparable to techniques like DDS [7], and CloudSeg [119] that use additional server computation. It also requires lower compute for calibration than the baselines, as it requires only one-time calibration instead of periodic re-calibrations. We summarize our contributions as follows:

- We identify that tiles with moving objects have higher bitrates and envision using them for traffic surveillance. To the best of our knowledge, this is the first work that uses tile bitrate to identify mobile objects.
- We utilize the correlation between tile bitrate and moving objects to design a thresholding strategy that identifies the tiles with moving/changing objects. This strategy is adaptive to different weather and traffic conditions.
- We evaluate TileClipper on RPi and Jetson Nano and obtain $> 92\%$ accuracy. It works in real time and reduces data sent by up to 40%. We compare it with DDS, Reducto, and CloudSeg to show that it is less compute-intensive, enabling scalable, cheaper deployment.
- We demonstrate a live deployment of TileClipper on our university campus, with a camera attached to an embedded board and a 4G connection. It saves over 55% bandwidth while providing an accuracy of over 87%.

3.1 Background & Motivation

In this section, we provide the background needed for our work and then present the motivation behind TileClipper.

3.1.1 Working of Video Encoders

We first discuss the video compression strategies used by video codecs. In general, video codecs operate at a coarser level than pixel-level dependencies, partitioning frames into spatial blocks (macroblocks in H.264 and coding tree units (CTUs) in H.265) [121]. Blocks with similar content (dependencies) can use the same bit allocation for representation, thus reducing the total number of bits needed. Matching blocks are estimated using motion vector predictor algorithms [73, 122]. Therefore, the objective of video encoders is to minimize the distortion (D) after compression and the number of bits (b) required to encode block dependencies (also referred to as the rate R). This is known as R-D minimization. For frames split into K blocks, the objective function is shown mathematically as [123, 124]

$$\text{Minimize} \quad \sum_{k=1}^{K} (D(s_k, s'_k) + \lambda b_k), \tag{3.1}$$

where s_k is the original kth uncompressed block, s'_k is its decoded version of the compressed representation, $D(s_k, s'_k)$ is a distortion function that returns the mean squared error (MSE) between s_k and s'_k, λ is Lagrange multiplier assigning relative importance to distortion and the extra bits, and b_k is the number of extra bits needed to encode spatial (within frames) and/or temporal (across frames) dependencies of the kth block in a frame from a reference frame. In codecs such as HEVC and AV1, blocks in highly dynamic, complex scenes are further subdivided into smaller sub-blocks for more precise motion prediction [73, 124]. This is shown in Figure 3.2, where blocks are subdivided into multiple levels to encode the area with a moving car. Thus, the number of bits b_k for a block k and the total number of bits B for a segment with S total blocks that are subdivided using a single-level quadtree are given by:

$$b_k = \sum_{i=1}^{I} b_k^i, \text{ and } B = \sum_{k=1}^{S} b_k, \tag{3.2}$$

where b_k^i is the number of bits required for motion prediction of the ith sub-block of the kth block and I is the number of sub-blocks[2]. Note that while decoding the exact content requires algorithms, it is possible to obtain the total number of bits used in a segment B simply by parsing the statistics of the segments present in its headers.

Note that tiles are different from macroblocks. A tile consists of multiple blocks. During encoding, tiles are enforced with the constraint that the spatial and temporal dependencies among blocks are within the tile boundaries [125]. Therefore, the dependencies are localized and based only on

[2]The number of sub-blocks depends on the codec used. For example, HEVC and AV1 use a quadtree structure with four recursive sub-blocks, whereas VVC uses a multi-tree structure.

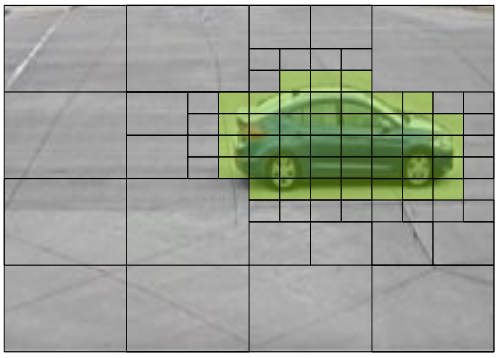


FIGURE 3.2: Video codecs use sub-blocks to encode complex scenes.

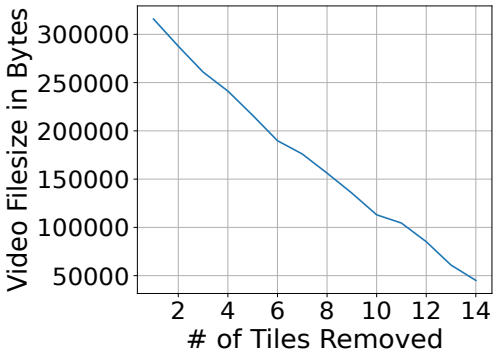


FIGURE 3.3: Removing tiles reduces the filesize of a video linearly.

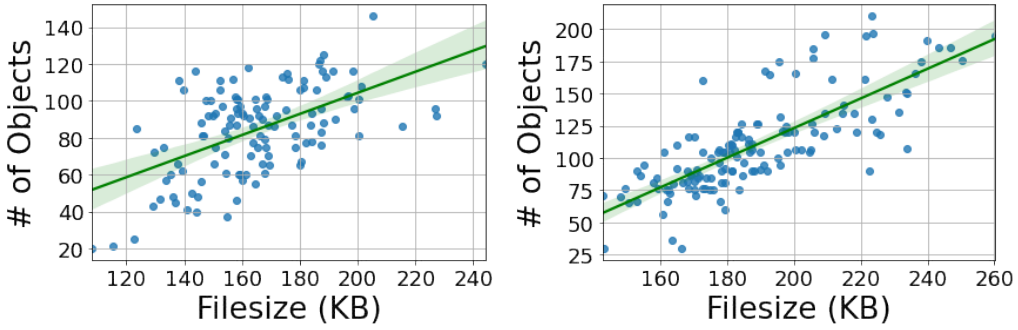


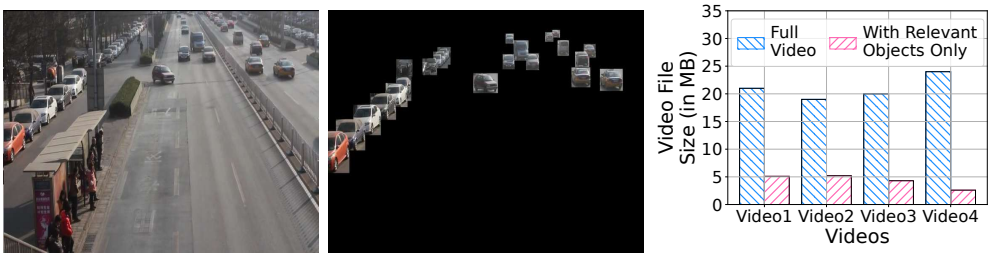
FIGURE 3.4: The plots show the correlation between video segment sizes and the number of moving objects for two different videos. The Spearman correlation values are 0.49 and 0.80, respectively.

the content within a tile. Since tiles are independently encoded, removing them does not impact the decoder. The decoder can easily play the video by placing empty patches in the places where tiles are missing (Figure 3.1d). An important advantage of using tiles is that removing them reduces the video file size, thus saving network bandwidth. We confirm this in Figure 3.3, where we prune tiles one by one in 45 segments of a tiled video and observe that the average filesize of the segments reduces linearly. This implies that removing tiles with irrelevant scenes, such as backgrounds with no objects, sky, trees, or buildings, can reduce unnecessary bandwidth usage.

3.1.2 Correlation Between Number of Objects and Segment Bitrate

In surveillance settings, the camera is fixed in place but must monitor for any movement/changes. Due to the compression strategy of video codecs (discussed in §3.1.1), any sudden appearance and/or movement of objects would lead to a corresponding increase in the amount of residual data. Intuitively, this would lead to an increase in the bitrate (i.e., filesize) of the video segments.

We first explain the intuitive reason behind the possible correlation between the number of moving objects and segment bitrates. Note from Eq. (3.2) and Figure 3.2 that regions of the video that have more motion and objects of interest are subdivided into smaller blocks and thus require a larger number of bits to encode. Therefore, videos with moving objects and dynamic



(a) (b) (c)

FIGURE 3.5: Amount of possible savings in four sample videos. In (a) and (b), we show that the removal of the frame background leads to no loss of information. In (c), we quantify the possible data savings. Each video is 1 minute long and has a resolution of 940×540. For "Full Video", we send the entire video unchanged. For "Relevant Objects", we remove the background.

content have higher bitrates [126, 127]. Note that motion prediction and the use of smaller blocks for dynamic scenes are used by almost all modern video standards, such as H.264 [128], HEVC [129], AV1 [130], and VVC [131]. Such higher bitrates have been observed by prior studies [126, 127, 132].

We confirm this hypothesis by studying the correlation between video file size and the number of objects, where videos are segmented into $0.5s$ segments. Figure 3.4 shows the scatter plots of two distinct videos in different settings (details of the benchmarks used are in §4.3). We note that the file size and the number of objects in the segments are moderately correlated, with correlation values ranging from 0.48 to 0.90. Such an observation motivates us to design a surveillance system that can use the bitrates to detect the presence of objects of interest, i.e., moving objects.

However, utilizing this observation to skip segments is difficult. This is because, as seen in Figure 3.4, no segment has zero objects, indicating that it is impossible to remove any segment without hurting the accuracy. Thus, an alternative strategy of pruning unnecessary regions is needed.

3.1.3 Shortcomings of Existing Systems

Recent work has proposed two major strategies to reduce the amount of data sent to servers. The first strategy, frame-level filtering, involves transmitting only a limited number of frames and then reconstructing the position of the moving objects from them [116]. The second strategy is to send only a low-resolution version of the video. The server-side can then compensate for the low resolution by either running a more computationally intensive DNN [119] or, if needed, sending queries to the camera to fetch frames/regions of interest at higher resolutions [7]. Figure 3.5 shows the amount of data that can be saved by sending only the relevant portions of the frame. We observe a reduction of $3.5 - 19\times$ in the file sizes that need to be sent if only the objects identified (known as regions of interest) are transmitted over the network.

However, sending only the regions of interest has a couple of challenges. First, traditional object-detection techniques require some form of DNN. Current techniques, such as DDS [7] and Cross-ROI [9], can only identify the spatial regions by running these computationally intensive DNNs. Although a few recent models, such as Yolo-Ret [133], can run in real time on embedded devices with a GPU, such as the Jetson Nano, their reported accuracy is only comparable to that of the weakest full models (i.e., the nano version of Yolo5).

While smart cameras with specialized compute capabilities for running DNNs are available on the market, they are at least 5× as costly as conventional cameras [134]. Thus, current techniques either depend on server-side calibration heuristics as proposed by DDS [7] or Reducto [6] or utilize specialized hardware such as FPGA-based boards [135]. Second, pruning unnecessary regions or frames of interest often requires integrating the logic into the camera's firmware to avoid the overhead of re-encoding the video [6]. Both server-side techniques and firmware-integrated logic increase the system's complexity and cost.

Next, we present TileClipper, which overcomes the limitations of prior works by leveraging the key idea that bitrates and the presence of moving objects are correlated.

3.2 TileClipper's Architecture & Design

3.2.1 Design Choices and Overview

To resolve the challenges discussed earlier, TileClipper selects video tiles for transmission to the server in real time. Unlike prior works, TileClipper side-steps the problem of integration with firmware by using the tiled encoding built into the video standard. The main advantage of tiles is that removing some of them is not computationally intensive and can be done in real time, even on embedded platforms. TileClipper's identification of relevant tiles has the following motivations:

1. **Minimal computation requirement on the camera side:** The amount of computation on the camera side should be low enough to allow cheap edge devices like RPi to run the tile selection technique in real-time. This allows easier integration of the technique into existing deployments.
2. **Substantial bandwidth savings:** The number of tiles sent to the cloud server should be small enough to provide significant bandwidth savings.
3. **Sufficient robustness:** The selection of tiles should be robust enough so that server-side object detection accuracy does not suffer by missing out on necessary tiles under different light and weather conditions. The calibration of the system should not introduce a lot of additional computation overhead on the cloud/edge server.

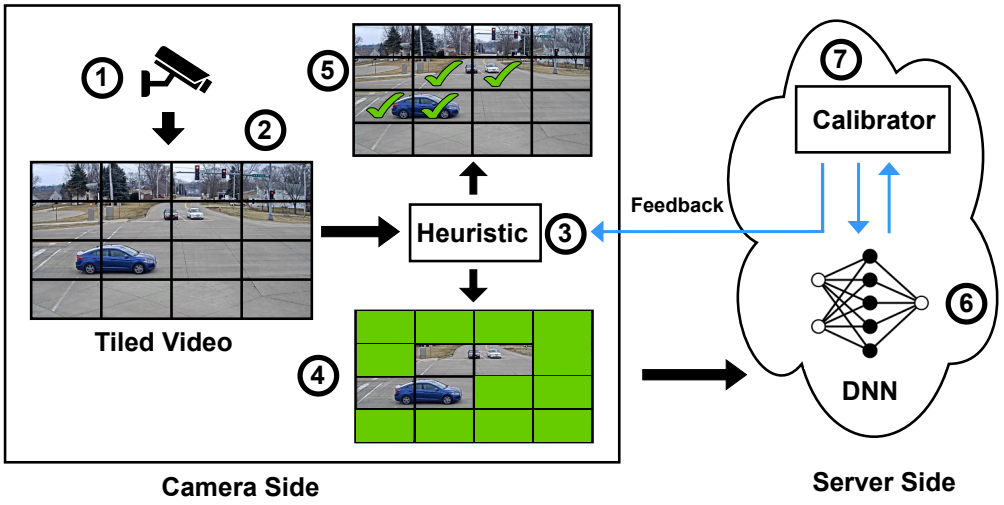


FIGURE 3.6: Workflow of TILECLIPPER. On the camera side, the camera generates tiled videos and runs the thresholding strategy to remove unwanted tiles. The server side runs the initial calibration and the DNN to recognize objects.

TILECLIPPER Overview: The key idea behind TILECLIPPER is that video encoders utilize a technique of encoding that generates higher bitrates for tiles with moving or complex scenes than with simpler static scenes. This enables TILECLIPPER to identify regions of interest without decoding the raw frames (i.e., the video is not decoded for TILECLIPPER's inference). This makes TILECLIPPER different from existing state-of-the-art techniques that often rely on optical flow [5] or low-level frame features [6]. This technique further has the advantage that it is easy to integrate TILECLIPPER with newer codecs, such as AV1 and VVC, that utilize increasingly complex encoding and decoding algorithms.

Figure 3.6 shows the workflow of TILECLIPPER. At the camera side, (1) camera hardware captures frames, (2) provides the HEVC encoded tiled video, (3) we apply TILECLIPPER's tile selection heuristic to select the relevant tiles from tiled video using a threshold. TILECLIPPER depends on a crucial observation to select tiles. We identify the tile bitrate (in kbps) as a key statistic that can indicate whether a tile contains a moving object. This is feasible because of the way video encoding schemes work in practice – where any static background material can easily be inferred via prediction from previous frames, but moving objects need to be encoded with additional bits [126, 127]. This increases the size of a group of tiled video frames with objects, as opposed to those with only a static background. (5) TILECLIPPER uses this insight to design a selection strategy. It calculates each tile's statistics separately and then individually identifies the appropriate threshold for each by calibration (discussed in §3.2.3). (4) Any tile that exceeds the threshold is sent to the server for detailed post-processing for traffic surveillance. (6) At the server side, we use YOLO-v5 [136] for object detection and calibration. (7) The calibrator uses the first 30s of videos and runs a DNN to send the right parameters as feedback to the camera. While we have assumed calibration to run on the server side, it is lightweight and rare enough to be performed even on edge devices with GPUs such as nVidia Jetson Nano. Furthermore

(as discussed in §3.4.2), calibration need not be triggered even by changes in weather/lighting conditions.

3.2.2 Utilization of Tile Bitrate Statistics

Correlation Between Tile Bitrate and Moving Objects: A major challenge of TileClipper is to identify the spatial regions of interest. TileClipper focuses on the smaller rectangular regions (tiles) as opposed to entire segments because there are few segments with no objects of interest. To utilize tiles, we first divide each segment into a 4×4 tile configuration, as justified by prior work such as CrossROI [9], since it provides a good balance between accuracy and encoding overhead. We obtain the bitrate statistics for each tile and use the YOLOv5 computer vision algorithm to count the number of objects in each tile (illustrated in Figure 3.1). A key point is that obtaining the bitrates for each tile does not require decoding. Thus, this technique can be utilized on encoded videos by parsing only metadata.

We then obtain the correlation between the tile bitrates (file sizes) and the number of objects (shown for two tiles in Figure 3.7) for each individual tile. We find a very strong correlation, with the Spearman correlation coefficient consistently exceeding 0.75. Figure 3.8 (for a subset of 32 tiles) shows that only the tiles containing the moving car have higher bitrates. Note that even though the car is partly present in tiles, all of them show higher bitrates because the car is moving through them. Therefore, objects split across tiles exhibit a similar signature in all tiles that contain them. Such a strong correlation can be intuitively explained by the fact that since each tile is independently encoded, their sizes are affected by even a small number of moving/changing objects (see Figure 3.2). These changes cannot be as efficiently compressed by the encoding algorithm, leading to larger tile bitrates (in §3.1.1).

This strong correlation no longer holds if the comparison is performed across tiles with distinct scenes, even of the same segment (shown in Figure 3.9(a)). This is because the background encoded in each tile affects the bitrate. We note that the tile bitrates have significantly different distributions, with median values differing by a factor of 2.5×. Thus, each tile has its own bitrate characteristic that cannot be directly inferred from its neighbors.

Lighting and Weather Conditions Affect Tile Bitrates: We also note in Figure 3.9(b) that both lighting and weather conditions affect tile bitrates. Again, this is intuitive since the background content can affect the tile bitrates. We find that in the rain, the droplets reduce the intensity of the colors, thus reducing the median tile size by over 30% compared to noon time. Since light and weather conditions can change periodically, these observations imply that the tile bitrate can be a good metric *only over the short term*. Thus, while the tile bitrate can be used as a metric, the actual threshold should depend on a moving window of the bitrate statistics. An alternative strategy would be to recalibrate periodically, which incurs additional overhead

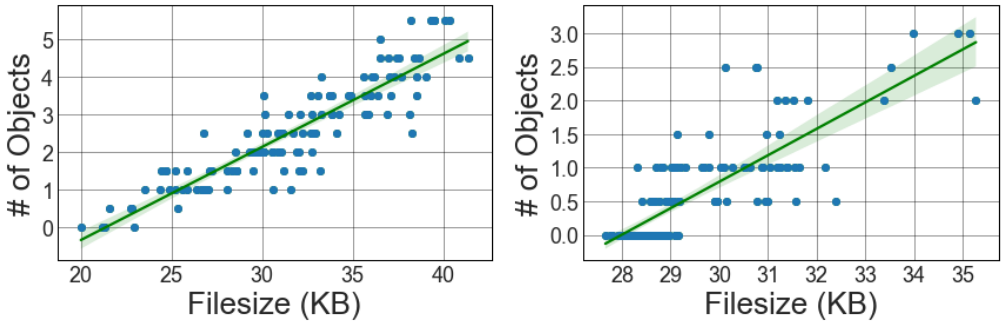


FIGURE 3.7: The plots show the correlation between video tile filesize and the number of objects for two different tiles. The Spearman correlation values are 0.78 and 0.90.

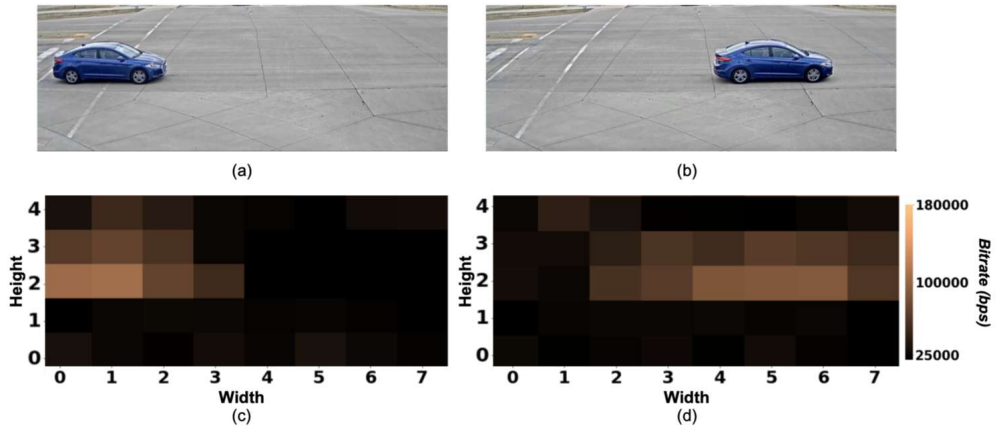
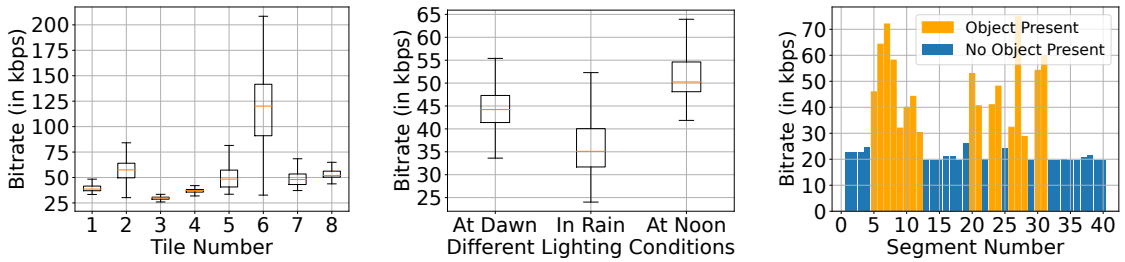


FIGURE 3.8: Relation between the presence of moving objects and the bitrate (in bps) of tiles. Note that only the tiles where the car is moving have a higher bitrate than the other tiles.

(as shown in Figure 3.17). However, we will show in §3.4.2 that using percentile statistics over a moving window performs as well as periodic recalibration.

Presence of Noise in Tile Bitrates: We next observe in Figure 3.9(c) that while the bitrates of a tile are noisy in nature, the amount of noise in general is small. The higher peaks usually indicate moving objects, while the smaller peaks tend to reflect minor changes, such as shadows or changes in lighting conditions. Therefore, just considering any tile bitrate higher than a certain arbitrary value does not necessarily imply the presence of an object. Thus, we discard smaller peaks by considering only the spikes as most probable to have objects with sizes greater than a threshold. Choosing this threshold via an intelligent strategy is therefore crucial, as it varies across tiles and directly affects TILECLIPPER's accuracy and savings. Moving objects of no interest, such as the movement of foliage, also gives a considerable spike in bitrate distribution. However, they should be discarded because they are irrelevant to traffic surveillance. Our calibration phases set the threshold high to reduce false detections in these cases.



(a) (b) (c)

FIGURE 3.9: (a) and (b) show bitrates of the same video across 600 segments of different tiles and the same tile under different lighting conditions, respectively. (c) shows the temporal (for 20s) distribution of video segments of a tile; spikes with high bitrates (in orange) have objects.

3.2.3 Our Thresholding Technique to Filter Tiles

Problem Formulation: The goal of TILECLIPPER is to identify as many tiles with moving objects as possible while discarding tiles without such objects. The ratio of tiles selected with moving objects to the total tiles with moving objects is called recall. The ratio of tiles with objects present to the total tiles sent is called precision[3]. A high recall ensures that the most relevant objects are selected, whereas a high precision ensures that substantial bandwidth is saved. Let the bitrates of tiles $T_j (j = 1, \ldots, t)$ of a segment s_i be denoted as $B_i(T_j)$. Let their corresponding precision and recall be denoted by P_{ij} and R_{ij}, respectively. TILECLIPPER needs to identify the right threshold r_{ij} on bitrate $B_i(T_j)$ of each $< s_i, T_j >$ pair to classify if objects are present. An accurate classification would imply balancing both recall and precision. This is typically done using F_β-score, which is the weighted harmonic mean (HM) of precision and recall [137][4] Mathematically,

$$\textbf{Maximize } F_\beta = \frac{1+\beta^2}{2} HM(P_{ij}, \frac{1}{\beta^2} R_{ij}), \forall s_i, T_j \tag{3.3}$$

To assign the value of β, we note that recall is usually given higher priority than precision in traffic surveillance, as it is often used in safety-critical applications [138]. In such cases, $\beta = 2$ is used as it gives 4× higher weight to recall than to precision. We, therefore, maximize the F2-score. However, to explore the trade-off between accuracy and savings for use cases requiring higher precision, the value of 2 can be replaced with other commonly used values, such as 1 or 0.5.

Calibrating the Thresholds and Selecting Tiles: During the calibration, we send all the tiles of the first S segments to the server to run the $CALIBRATE$ procedure. It starts by

[3]We use precision and recall instead of the absolute numbers of true positives and true negatives, as the data is often imbalanced [137].

[4]Harmonic mean gives higher weightage to the term that has the least value. Thus, for the harmonic mean, unlike in the arithmetic or geometric mean, a higher weightage to a term is given by dividing the term by the weight.

running the object classification (using YOLO-v5x) to identify tiles with moving objects inside by tracking bounding boxes using StrongSORT [139] object tracker in each frame of the segments (Line 2 of Algorithm 1). We use the intersection over union (IoU) of tracked bounding boxes to separate tiles into static and mobile object categories. We accordingly classify the tiles of the segments into two categories: *true* (with moving objects) and $false$ (without moving objects). For each category, we compute the $10^{\text{th}}, 20^{\text{th}}, \ldots, 80^{\text{th}}$ percentiles[5] of the bitrates for each tile separately for all S segments, denoted by P^{true}_{10}, $P^{true}_{20}, \ldots, P^{true}_{80}$ and $P^{false}_{10}, P^{false}_{20}, \ldots, P^{false}_{80}$ respectively. We then consider the threshold as the mean of P^{true}_{v1} and P^{false}_{v2} for all values of $v1, v2 = 10$ to 80. We identify the pair of percentile values $v1$ and $v2$ using an exhaustive search that maximizes the F2-score for thresholding at the camera side (Line 4-8 in Algorithm 1). For each video, we use the first S=60 segments ($30s$) for calibration, which worked well for us during sensitivity tests (in §3.4.4). After calibration, the server returns the best percentiles p_t and p_f and the clusters Q_t and Q_f, along with the bitrates of the segments in S belonging to the *true* and $false$ categories, respectively, for each tile.

With the thresholds now calibrated, we use the $SELECT$ procedure in Algorithm 2 at the camera to obtain the bitrate threshold. This is called for all the tiles of every segment. The procedure computes percentile values for the segment bitrates in each cluster (Q_t and Q_f) of each tile. To adapt the thresholds temporally, we maintain two different clusters of size 10 (sensitivity towards size shown in §3.4.4) for each category. Note that each tile has its own independent cluster with ten recent bitrate values. We compare the bitrate with the tile-specific midpoint of $< P^{true}_{p\_t}, P^{false}_{p\_f} >$ (Lines 8-10 in Algo 2), where the values p_t and p_f are chosen during the calibration phase. The tile-specific clusters are updated in each call (Lines 12 and 14). A segment classified with no object is added to cluster Q_f, automatically removing the older element (because we use evicting queues[6] as clusters). A segment with objects is pushed to cluster Q_t. This update to clusters (an instance of a sliding window) ensures they remain up to date with the changing bitrate distribution, making them adaptive to scene changes. Note that for tiles where we encounter no or very few segments with objects ($< 10\%$, i.e., < 6 segments) during calibration, this technique does not work because the clusters are not populated with enough values to represent the bitrate distribution. We fall back to an outlier detection approach for these cases (Lines 3-6).

Fallback to Outlier Detection: In a few tiles ($\approx 21\%$ of the total tiles in our dataset), the calibration phase finds no or too few segments with objects (no object ratio $O < 10\%$) within the S segments. These cases happen either because the tile focuses on an area outside the roads or because there is less traffic. We cannot decide to remove these tiles altogether because $\approx 26\%$ of the 21% (5.72% of the total) tiles start showing objects after calibration. Since these events

[5] We choose percentiles instead of weighted means due to their lower bias towards outliers, which are introduced by mistakes in the ground truth [140]. Furthermore, we use statistics rather than absolute values to account for content drift. We later show that this avoids the need for periodic re-calibration.

[6] Evicting queues are FIFO queues with auto-dequeuing when enqueue exceeds the queue size.

Algorithm 1 TileClipper's strategy for calibration

INPUT: Bitrate of each tile $B_i(T_j)$, list of percentile values V
OUTPUT: percentiles of true cluster and false clusters $< p\_t, p\_f >$, tiles in the cluster with objects Q_t, tiles in cluster with no objects Q_f

1: **procedure** CALIBRATE($B_i(T_j), V$)
2: Q_t, Q_f = Object recognition and tracking on S segments
3: $o \leftarrow [\,]$ // Empty list of objective values
4: **for** each value $v1$ in V **do**
5: **for** each value $v2$ in V **do**
6: $f2 \leftarrow$ ComputeF2Score($Q\_t, Q\_f, v1, v2$)
7: o.push($< v1, v2, f2 >$)
8: $< p\_t, p\_f > \leftarrow \arg\max_{<v1,v2>}(o)$
9: **return** $< p\_t, p\_f >, Q\_t, Q\_f$

are rare, we model them as the tail of a Gaussian distribution. To identify these, we compute the median P_{50} and the standard distribution σ of the bitrates of the tiles across past S segments, which is updated at each step (Line 6 of Algo 2). We then utilize $P_{50} + \gamma \times \sigma$ as the threshold (Line 5). To identify the optimal value of γ, we run TileClipper on these tiles to produce Figure 3.10, which shows how false positives and misses vary with increasing γ. We choose $\gamma = 1.75$ (corresponding to the 96^{th} percentile), which gives a mean miss rate of < 0.05 and a mean false positive rate of ≈ 0.075. Note that we choose a slightly lower value of γ over the one that minimizes the total errors in Figure 3.10 for providing higher weightage to recall than precision.

Time Complexity of Calibration and Thresholding: We note that computing the percentile statistics of the tiles takes $O(S \log S)$, where S is the number of segments used for calibration. If there are t tiles, then this computation is repeated for each tile. Thus, calibration has a total time complexity of $O(tS \log S)$. For the SELECT procedure, again, we compute the percentiles of the two queues, which have a time complexity of $O(|Q| \log |Q|)$, where $|Q|$ is the maximum size of the two clusters (10 in our case). We also use enqueue operations, which take constant time. Since the SELECT procedure is called for all tiles t, this gives a total time complexity of $O(t|Q| \log |Q|)$. Since both t and Q are relatively small in magnitude (< 20 each), the time complexity is small enough to run in real time on cheap devices.

3.3 Implementation and Dataset

TileClipper's Implementation: We implement both the camera-side version of TileClipper and server-side calibration in Python3. We obtain the bitrates of the tiled segments using the ffprobe tool from the FFmpeg tool suite [90]. We run the camera-side component on Raspberry Pi 4 and Jetson Nano. Our server uses Ubuntu 18.04 and has an nVidia GeForce RTX

Algorithm 2 TileClipper's strategy to estimate the threshold for tiles of segments.

INPUT: Current tile's Bitrate B, no object ratio O, # used for calibration S, two evicting queues Q_t and Q_f containing bitrates of the past segments and best percentiles p_t and p_f to use for true and false clusters respectively, value of γ to use.

OUTPUT: Threshold on a tile.

procedure SELECT($Q\_t, Q\_f, O, S, B, p\_t, p\_f, \gamma$)
    // Calculate threshold r
    if $(sizeOf(Q\_t)/S) < O$ **then**
        // Not enough objects; use fallback
        $r \leftarrow (median(Q\_t) + \gamma * std(Q\_t))$
        $Q\_t.enqueue(B)$ // Update cluster
    else
        $u \leftarrow percentile(Q\_t, p\_t)$
        $l \leftarrow percentile(Q\_f, p\_f)$
        $r \leftarrow 0.5 * (u + l)$ // Mean of both percentiles
    if $B > r$ and $(sizeOf(Q\_t)/S) >= O$ **then**
        $Q\_t.enqueue(B)$
    else if $B <= r$ and $(sizeOf(Q\_t)/S) >= O$ **then**
        $Q\_f.enqueue(B)$
    return r

2080 Ti GPU. For the experiments, we store the encoded videos on the storage of a Raspberry Pi or a Jetson Nano, and then run a script to start our workflow. We have made our source code and datasets available on GitHub at https://github.com/shubhamchdhary/TileClipper to spur further research.

Video Encoding: While a number of traffic surveillance public datasets are available, they are typically not encoded in a tiled format. Since TileClipper requires the input video in tiled form, we first need to encode the videos into tiled format using a system of 4×4 tiles. Other tiling configurations are possible, but we will discuss in §3.4.4 (as have prior works like CrossROI [9]) that this configuration performs well in practice. We first split the entire video into segments of $0.5s$ duration. We choose this duration because a shorter duration wastes more bandwidth due to poor compression, while a longer one reduces the scope of tile removal because fast-moving vehicles exit tiles quickly, leaving them mostly empty. We then encode it using the open-source HEVC encoder Kvazaar [141] at 30fps, and finally pack it into mp4 using the tool GPAC [142].

An encoding parameter that influences video quality is the Quantization Parameter (QP). A high QP value implies that both the amount of compression and the amount of loss are high. However, traffic surveillance videos do not require the same level of quality as videos for end consumers [4, 7]. Thus, we experimentally identify the QP parameter that yields no loss in object detection accuracy with YOLO-v5s (see Figure 3.11). We observe that a QP parameter of 30 does not lead to any such loss while also reducing the filesize by almost $4\times$. Therefore, we encoded all of our videos at 30 QP.

TABLE 3.1: Summary of the dataset used for evaluation.

Dataset	# of Videos	Resolution	Duration	Type
AICC21	14 14	1920×1080 1280×960	5 min	Benchmark
DETRAC	20	960×540	1-2 min	Benchmark
Others	4	1280×720	6-8 min	Chaotic
OurRec	3	1280×720	13-25 min	Flyover
Total	**55**	-	-	-

Datasets and Baselines: For large-scale evaluation, we use YOLO-v5s as the ground truth, as it is widely used in surveillance, and run it on each tiled video segment to identify objects. As in Reducto and DDS, we utilize publicly available benchmarks and a few of our own videos (Others and OurRec) in chaotic traffic and at flyovers (summarized in Table 3.1). The dataset has been carefully selected to include more video hours and a wider range of traffic conditions than the original studies that proposed the baseline strategies.

As a baseline, we reimplement DDS with a few parameter changes to make its performance comparable to TILECLIPPER. We recall that DDS first sends the videos at a higher QP of 36 with a lower $0.8\times$ resolution and identifies regions containing objects. The regions where objects are identified with low confidence are then queried from the camera and sent at a lower QP than the original video (30 in our case, with $0.8\times$ resolution). Our version of DDS uses HEVC video encoding and YOLO-v5s as the neural network on the server side (considering its state-of-the-art performance) [143], with a threshold value of $0.2 - 0.25$ to fetch an object for the second phase[7]. Although DDS always sends the video frames at a resolution $0.8\times$ the original, we still calculate the savings by comparing them with $1\times$ resolution to retain their setting and make the reported savings compatible with their implementation.

We use the open-source Reducto implementation as a second baseline, with fixes to address the low-QP video issue. Reducto's frame filtering uses a calibration system to first cluster low-level pixel feature values. We reproduce the technique and then let the calibration run for $120s$, as we observed that this calibration time minimizes the need for future recalibration.

Finally, we include two additional orthogonal baselines – CloudSeg and StaticTileRemoval (STR). We were able to run the open-sourced version of CloudSeg available at [144] without any changes. CloudSeg sends the videos at a lower resolution than the original and applies super-resolution on the server before object detection. We send the videos from the camera end at a resolution of $0.5\times$ the original and then use super-resolution at the server end for upscaling. In STR, we annotate the tiles that do not contain any road area and remove them. This allows us to test the

[7] A larger range of values leads to fewer misses and/or misclassifications, but at the cost of lower saving. We observed in experiments that increasing this range by as little as 0.05 leads to a large increase in the number of objects recalled in the second phase, allowing very low or even negative savings compared to sending the whole video. This also highlights the critical dependency of DDS on DNNs' confidence score.

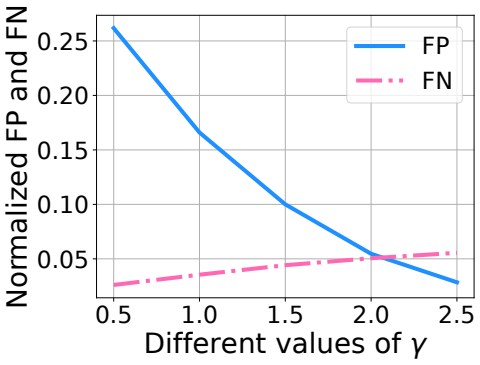


FIGURE 3.10: Increase in the value of γ decreases the chance of selecting tile falsely (FP) and increases the probability of misses (FN).

FIGURE 3.11: Varying video QP vs. filesize and YOLOv5 inference performance. YOLOv5 does not lose objects till 30 QP.

utility of TILECLIPPER's thresholding strategy. We do not compare against AccMPEG [4] as its implementation requires changes to the codec that are currently not compatible with HEVC.

Performance Metrics: We report the following metrics: **(1) Accuracy:** Accuracy is the ratio of objects that are detected after tile pruning to the total number of objects detected without filtering.

(2) Precision: Precision is the proportion of objects detected that are actually relevant (moving objects). High precision indicates substantial bandwidth savings.

(3) Recall: Recall is the proportion of objects correctly identified. Having a high recall is essential to ensure that our system does not miss any moving objects.

(4) Bandwidth saving: The bandwidth saving in percentage is the amount of data reduction achieved as compared to the original data if the pruned video is sent.

(5) Execution time: We measure the execution time in frames per second on the camera side. We measure server-side computations as GPU usage time in minutes.

3.4 Evaluation

We evaluate TILECLIPPER's performance, justify some of its design choices discussed in §3.2.1 across various scenarios, and compare it with DDS, Reducto, and CloudSeg.

3.4.1 Comparison with Baseline Techniques

Accuracy: Figure 3.12(a) compares the accuracy of TILECLIPPER and other baseline techniques across datasets. We list down our observations as follows: (1) TILECLIPPER is able to achieve a mean accuracy of over 0.85 on all the datasets. It exceeds 0.90 on AICC21, DETRAC, and

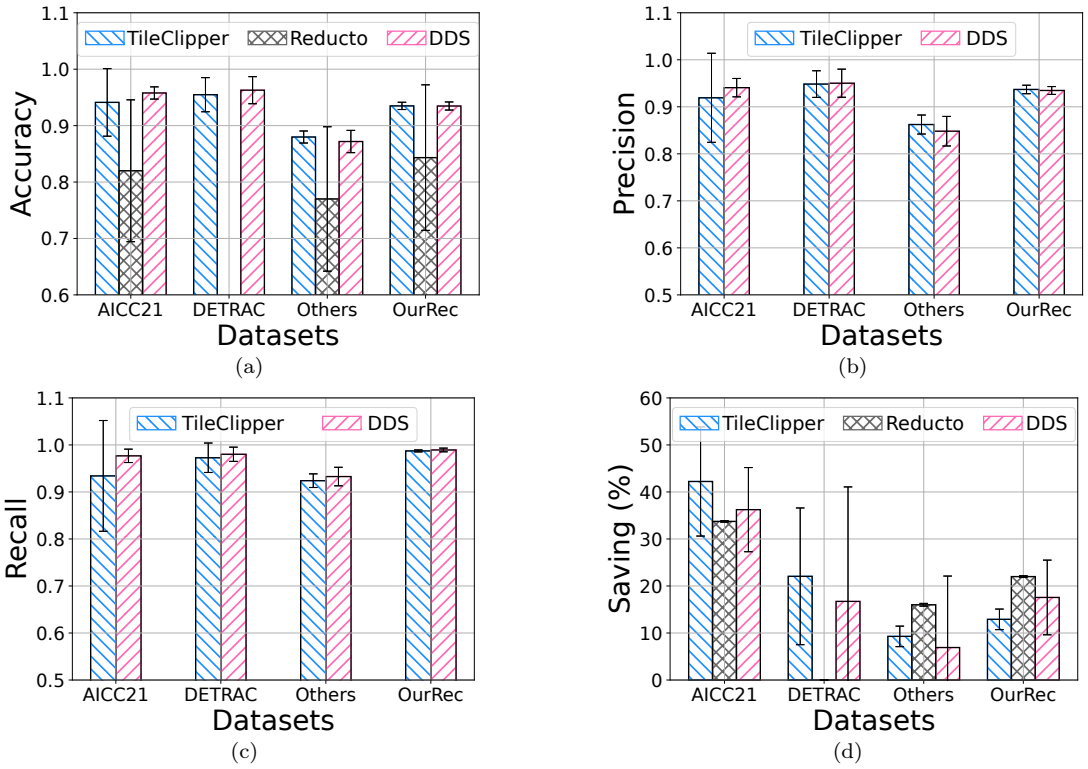


FIGURE 3.12: Performance of TILECLIPPER compared to Reducto and DDS in terms of (a) accuracy, (b) precision, (c) recall, and (d) bandwidth savings. TILECLIPPER provides accuracy, precision, and recall comparable to DDS, but with larger bandwidth savings. TILECLIPPER provides much higher accuracy than Reducto (12-15%), though TILECLIPPER's savings are lower on average by 10%. We omit Reducto's precision and recall results because it never produces false positives, and we also omit its results on the DETRAC dataset because it requires more calibration time than the videos are long.

OurRec. This indicates that very few moving objects are left undetected. (2) TILECLIPPER has comparable accuracy with DDS. Only on AICC21 videos, DDS performs marginally better than TILECLIPPER. This happens primarily because, in some cases, the vehicles stop for a while at a traffic signal or due to other vehicles; these stationary vehicles do not cause a significant bitrate spike, therefore TILECLIPPER discards the tiles containing these objects. Thus, TILECLIPPER misses these immobile vehicles, resulting in a marginally lower accuracy than DDS. However, such misses do not hurt the end traffic surveillance, as the vehicle was detected while still moving. Furthermore, if needed, the original video without tile pruning can be retrieved from the camera, which locally stores it for future use. (3) Reducto suffers in terms of accuracy at testing time, leading to the lowest accuracy on all the datasets. Note that Reducto prepares a hash table of thresholds, and the camera uses one of them for filtering. Due to high quantization, pixel-level differences are not captured by the existing clusters, resulting in lower accuracy. We observe that this does not work well in practice on lower-quality videos, as corroborated by [118]. Further, we cross-verified Reducto's performance by running on videos with lower quantization to reproduce the accuracy originally reported.

TABLE 3.2: Comparison with CloudSeg and StaticTileRemoval (STR).

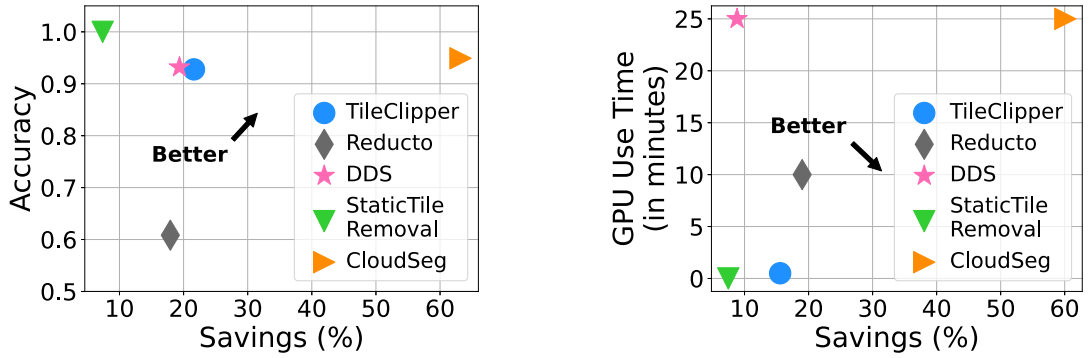
Dataset	Mean Accuracy (%)			Mean Saving (%)		
	Tile-Clipper	Cloud-Seg	STR	Tile-Clipper	Cloud-Seg	STR
AICC21	94.12	96.77	100	42.23	57.34	17.44
DETRAC	95.48	97.04	100	22.06	61.41	07.05
Others	87.98	90.80	100	09.28	69.89	00.00
OurRec	93.50	95.02	100	12.91	64.13	05.04

Precision and Recall: Figure 3.12(b) and 3.12(c) shows the precision and recall of TILECLIPPER and DDS. Since Reducto's frame filtration never produces false positives, we omit its precision calculation. We observe that TILECLIPPER provides > 0.85 precision and > 0.90 recall, which is high enough for most safety-critical applications [138]. Both DDS's recall and precision are comparable to TILECLIPPER's, demonstrating that TILECLIPPER's threshold effectively selects tiles. We further separately check the precision and recall values on the DETRAC dataset, as it contains such different traffic densities. We find that the fallback approach yields precision and recall of 92.65% and 96.48%, respectively, which is only 2% lower than when normal calibration is possible.

Bandwidth Savings: Figure 3.12(d) shows a comparison of the bandwidth saved in percentage for each technique. We observe that TILECLIPPER provides $> 40\%$ savings for AICC21 and $> 20\%$ for DETRAC videos. For OurRec and Others, the saving is close to 10%. This is due to the nature of the videos: in these datasets, the number of cars and traffic flow is very high, occupying most of the frame. This leaves less room for saving by removing unnecessary tiles. Except for OurRec, on all other datasets, TILECLIPPER gives higher bandwidth savings than DDS.

The savings for DDS are generally lower than TILECLIPPER, even though it uses a higher QP and a $0.8\times$ video resolution. DDS also shows higher variance than other techniques. This is attributed to the second phase, which requests a high-resolution video if the confidence is low. Reducto achieves higher savings than both DDS and TILECLIPPER across all datasets, except on AICC21, where its savings are slightly lower. This is because Reducto's heuristic filters out frames even with dense traffic. These gains in bandwidth, however, come at the cost of accuracy, as shown in Figure 3.12(a). Note that when TILECLIPPER's accuracy and saving degrade, the accuracy of DDS and Reducto also drops. This implies that the accuracy and saving drop are dataset-specific and are not a shortcoming of TILECLIPPER.

Comparison with Orthogonal Techniques: We also compare TILECLIPPER with the complementary works CloudSeg [119] and StaticTileRemoval (STR) in Table 3.2. STR removes the off-road tiles. Hence, none of the moving objects are missed, which leads to 100% accuracy at the cost of poor savings. Such savings even reach 0 on Others videos while TILECLIPPER



(a) (b)

FIGURE 3.13: (a) Accuracy and bandwidth saving tradeoffs of TILECLIPPER and baselines across the entire dataset. (b) GPU usage time and trade-offs across all systems for a video of ≈25 minutes.

provides 9.28% mean savings. Savings of TILECLIPPER is at least 2× that of STR. CloudSeg delivers the highest accuracy and savings, as expected, by sending videos at half the resolution at the cost of increased server-side GPU use. TILECLIPPER's accuracies on AICC21, DETRAC, and OuRec are comparable to CloudSeg. In the case of the OurRec and Others dataset, due to occlusions and types of vehicles not present in the training dataset (COCO) of YOLO-v5 (our server-side DNN is trained on), TILECLIPPER's calibration misses objects, leading to a drift from the optimal value. This makes TILECLIPPER filter tiles incorrectly. We leave, utilizing a better server-side object recognition for future work. The adaptive spatial tile pruning in TILECLIPPER achieves higher savings than StaticTileRemoval across the datasets.

Tradeoffs: Figure 4.9(b) shows the tradeoffs of all the techniques between accuracy, saving, and server-side GPU usage. We find that TILECLIPPER provides a better tradeoff between accuracy and bandwidth saving than the baselines (Figure 3.13(a)). CloudSeg offers the best trade-off between accuracy and bandwidth savings, at the cost of the highest server-side GPU usage. At the server, since the majority of the cost comes from energy and GPU usage, we compare server-side GPU usage to process a ≈ 25 minute video against the savings in Figure 3.13(b). Here, TILECLIPPER provides the best tradeoff between server-side GPU use and bandwidth savings. Although Reducto's savings are slightly better than TILECLIPPER, its GPU usage is 20× higher and gives the lowest accuracy. This implies that TILECLIPPER can maximize its accuracy and savings while using the least server-side resources.

Execution Time: We now test TILECLIPPER, Reducto, DDS, and CloudSeg benchmarks on two distinct edge devices – Jetson Nano and Raspberry Pi 4B (RPi). Figure 3.14(a) shows TILECLIPPER's processing speed along with the baselines on each device in terms of frames per second (fps) on the camera side. We note that TILECLIPPER runs at 22 fps on RPi, ≥ 16 fps [145], [146], which are needed for most real-time analytics, while on Jetson Nano it runs at 57 fps. The higher speed on Jetson Nano is due to its support for hardware-based HEVC encoding. CloudSeg performs similar to TILECLIPPER on RPi because it also requires only video encoding. DDS is slower than TILECLIPPER on each device, as it requires extraction of the spatial portions

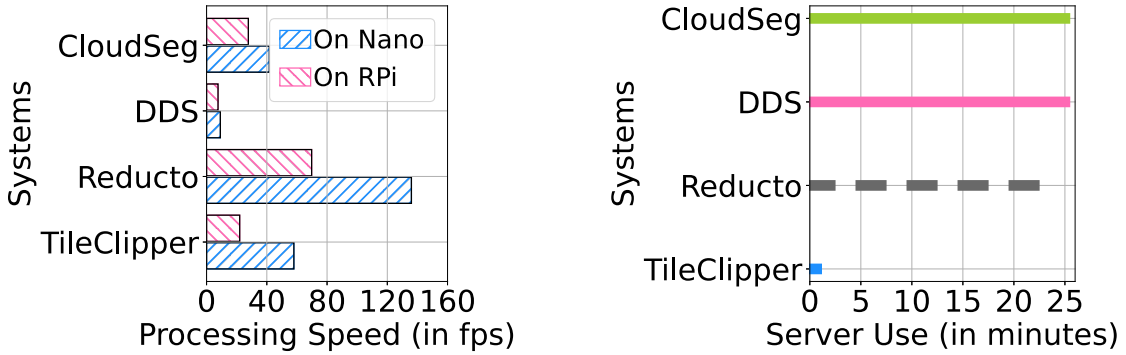


FIGURE 3.14: (a) shows mean video segment processing time (in fps) of TILECLIPPER, Reducto, DDS, and CloudSeg at the camera side on different hardware. (b) shows the server usage time for all systems for a $\approx 25min$ video.

from the frames (in its second phase), which is computationally expensive. Reducto runs fastest because it needs to encode fewer frames.

Cloud Server Cost: Figure 3.14(b) shows the server-side cost in terms of server GPU use time of each system to process a video of $\approx 25min$. We note that the most common object detectors available today are single-shot detectors, in which the computational cost is independent of the scene. We observe that TILECLIPPER uses less computation (GPU) on the server side than others. It requires computation initially only during calibration for 60 video segments ($30s$). On the other hand, CloudSeg continuously uses super-resolution to scale up low-resolution videos from the camera, whereas DDS requires processing all frames through a DNN to recall objects. Both are computationally intensive and use GPUs to run super-resolution and object detection algorithms. In our experiments, Reducto required frequent recalibration every 2 minutes using the server's GPU. However, note that Reducto's computation for inference may also reduce slightly due to fewer frames eventually sent by the server.. In summary, while TILECLIPPER runs in real time on edge devices and is computationally cheaper, it also uses less power (reducing costs) on the server side, making it an overall more efficient and scalable system.

3.4.2 Effect of Environment

Varying Traffic Density: Figure 3.15(a) and 3.15(b) show the accuracy and bandwidth savings achieved under different traffic densities. As expected, we observe in Figure 3.15(b) that videos with lower traffic density allow more tiles to be removed, resulting in greater bandwidth savings. TILECLIPPER saves 22% more than DDS in both conditions. Furthermore, Figure 3.15(a) shows that TILECLIPPER works equally well in low traffic conditions, indicating that its fallback approach also gives good accuracy in practice.

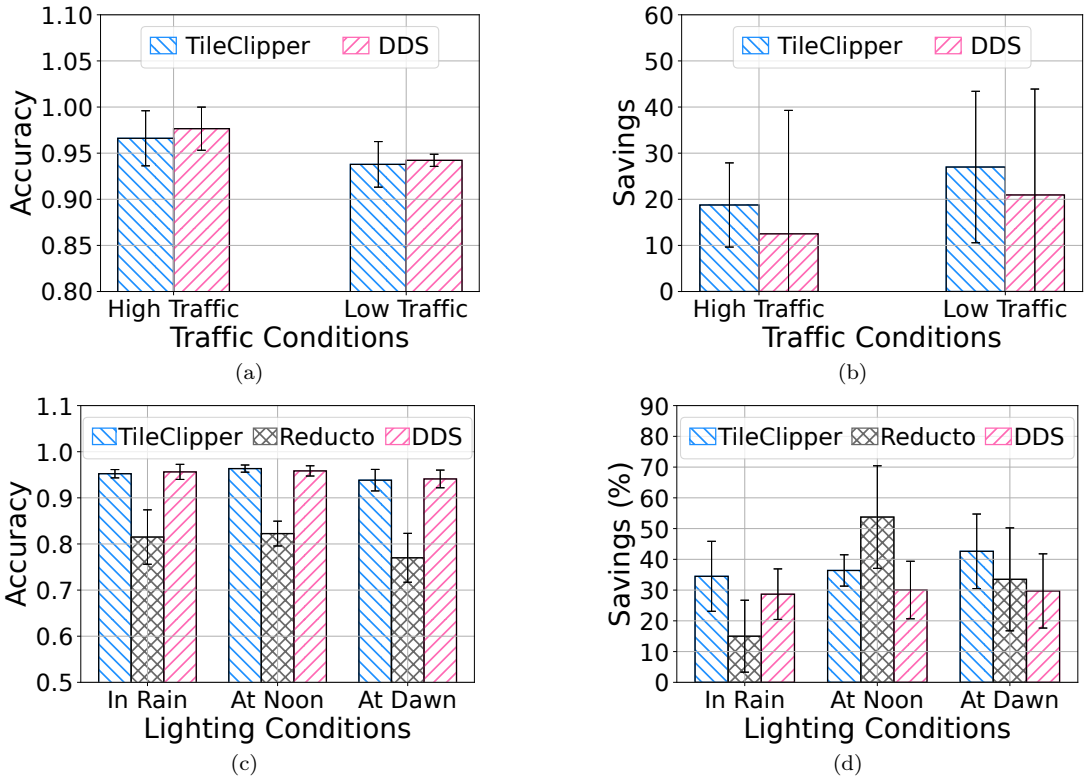
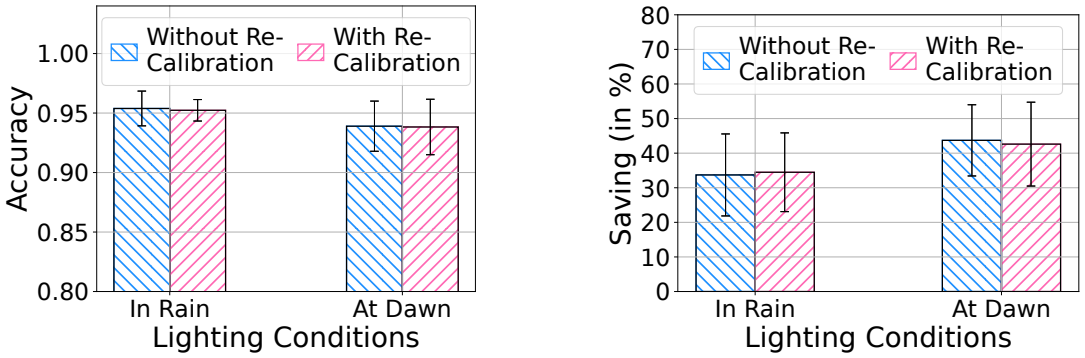


FIGURE 3.15: Performance in different traffic densities and lighting conditions. (a) and (b) shows the accuracy and bandwidth savings under high traffic ($>$ 10 moving vehicles within a segment) and low traffic density. (c) and (d) shows the accuracy and bandwidth savings in the rain, at noon, and at dawn.

Varying Lighting Condition: We also have a set of three videos from the AICC21 dataset from the same camera with different lighting conditions – noon, rain, and dawn. We find that the accuracy values (shown in Figure 3.15(c)) do not change significantly. They are all more than 0.92 across all settings and comparable with DDS. Reducto performs worst, achieving $<$ 0.85 accuracy across all environments. TILECLIPPER's bandwidth savings (Figure 3.15(d)) are consistently $>$ 30% as compared to DDS in all the conditions, showing its robustness in all weathers. Reducto achieves the highest savings only at noon, performing poorly in the rain because raindrops reduce the correlation between pixels in the frames to filter.

Impact and Need for Recalibration: We argued in §3.2.3 that any change in weather and light conditions can be handled by TILECLIPPER's cluster-based adaptive threshold without recalibration. We confirm this in Figure 3.16(a) and 3.16(b). We first calibrate on the noon videos, then run TILECLIPPER both with and without re-calibration on videos in other conditions, such as rain and dawn. We note that recalibration has a negligible impact on accuracy and savings. This shows that TILECLIPPER's thresholding strategy, which uses percentile statistics of the clusters, is robust enough to adapt to lighting and/or weather conditions, and that additional recalibration is unnecessary to maintain consistent performance.



(a) (b)

FIGURE 3.16: Performance with and without re-calibration (a) shows accuracy and (b) shows bandwidth savings.

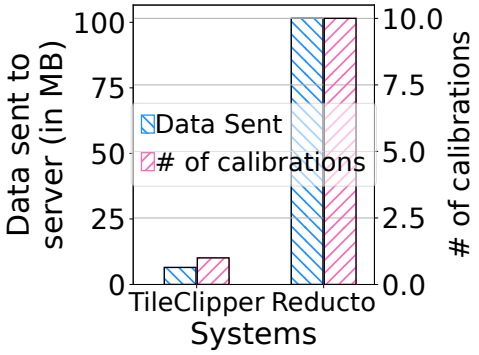


FIGURE 3.17: The amount of data sent to the server and the total number of recalibrations in TILECLIPPER and Reducto.

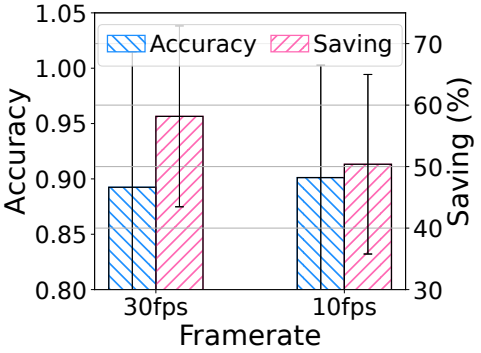


FIGURE 3.18: Accuracy and the bandwidth savings during the live experiment at different video encoding rates of 30fps and 10fps.

Recalibration Overheads: We now compare in Figure 3.17 the overheads involved in calibrating TILECLIPPER and Reducto in terms of data sent to the server during calibration and the total number of calibrations needed. We omit DDS and CloudSeg as they have no notion of calibration. We observe that for a $25min$ video, Reducto sends 15× more data to the server than TILECLIPPER for calibration. Further, the overall number of calibrations needed is also 10× more in the case of Reducto. This is because Reducto does not inherently adapt to scene changes and requires frequent recalibration triggers to get updated thresholds from the server. On the other hand, the adaptive algorithm (discussed in §3.2.3) of TILECLIPPER based on the clusters of the past 10 video segments can align itself with the temporal changes, making it more robust to changes in weather and traffic volume changes, imposing lower server overheads.

3.4.3 Live Deployment of TILECLIPPER

We also evaluate TILECLIPPER in a live deployment where the key challenge was that surveillance cameras available today do not utilize tiled encoding. Live encoding in software, as done by Kvazaar, requires support for vector instructions that are unavailable on ARM processors. Thus,

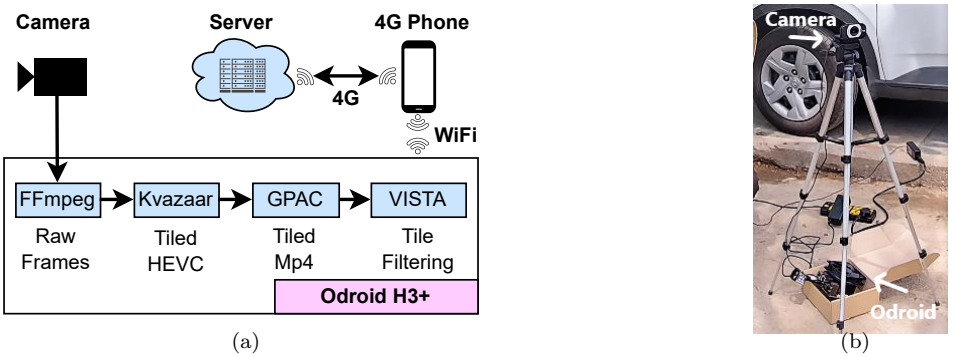
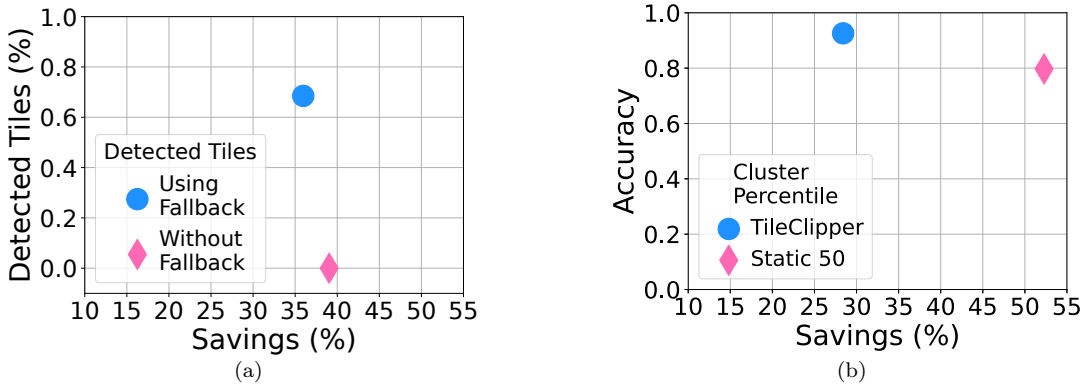


FIGURE 3.19: Live deployment, (a) Workflow on Odroid H3+, (b) On-site setup with a tripod-mounted camera connected to the Odroid.

we designed a setup on Odroid H3+ [147], which had an Intel Jasper Lake N6005 x86 processor clocked at 2.0 GHz frequency. This costs around $130, which is similar to the cost of Jetson Nano.

Setup: Figure 3.19(a) shows the process of capturing frames, tiling, encoding, and running TILECLIPPER on top of it. To get 0.5 s of tiled video segments (needed for TILECLIPPER) at an fps of N, we first capture $N/2$ raw video frames from the camera using FFmpeg, encode them into HEVC tiled videos using Kvazaar (with a superfast encoding preset), and finally use GPAC to package them into tiled mp4. All these tools run in cascade and transfer data via Linux pipes on the Odroid H3+ board. We run the TILECLIPPER's thresholding algorithm on the encoded tiled video segments to discard tiles with no objects. We used a smartphone with 4G connectivity as a hotspot. After obtaining regulatory and IRB approval, we installed the setup about 4-5 meters from a two-way road intersection near our campus, having a vehicle speed limit of 30kmph. The traffic consisted of a mix of vehicles and pedestrians. We use a Logitech C615 1080p webcam as our camera connected to the Odroid H3+ board (shown in Figure 3.19(b)). Setting a bitrate filtering threshold in the deployment was similar to the design in §3.2.1. We sent the first 60 segments to our hosted server for calibration over the 4G network. Once the server returns the feedback, we let the tile filtering run. TILECLIPPER did not require any recalibration during the entire live experiment.

Results: We run two experiments of 30min and 15min duration at frame rates of 30 and 10 fps, respectively. The accuracy and savings for both scenarios are reported in Figure 3.18. We observe that in this setup, TILECLIPPER obtains an accuracy $> 88\%$ and 90%, and saving $> 55\%$ and $\approx 50\%$ on 30 fps and 10 fps segments, respectively. However, even though the bandwidth saving in terms of percentage is similar, the amount of data sent is 53% higher when encoded at 30fps than at 10fps. We also observe that the streaming latency of filtered tiles is 50% and 30% lower after using TILECLIPPER compared to streaming the entire video at 30 fps and 10 fps,



(a) (b)

FIGURE 3.20: Ablation studies to (a) measure the effect of using fallback, and (b) show the effect of choosing a single fixed percentile for all tiles vs the grid-searched value of TILECLIPPER..

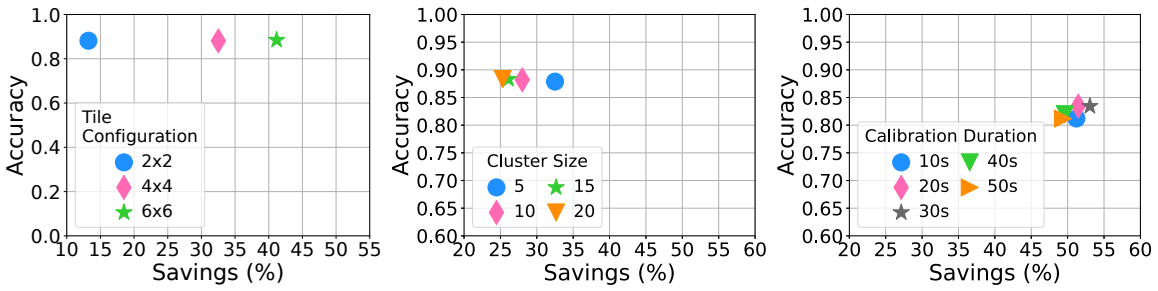
respectively. This shows that TILECLIPPER's bandwidth savings also reduce streaming latency, unlike DDS.

3.4.4 Sensitivity Tests and Ablation Studies

We now justify the decision choices made in TILECLIPPER, including the parameters used during calibration and in the algorithmic strategies, through sensitivity tests and ablation studies. The study utilizes the same dataset as in §3.4.4.

Effectiveness of Fallback: To validate the effectiveness of our fallback strategy for outlier detection (discussed in §3.2.3), we conducted a study with and without the fallback on 43 videos out of all 55 where the calibration phase identified no or too few segments with objects of interest (Figure 3.20(a)). Note that the no-fallback approach omits all tiles, assuming none of them contain objects. Thus, any strategy would show some fall in savings. TILECLIPPER's fallback strategy demonstrated a mere 8% reduction in savings compared to the scenario where it was not employed, as depicted in Figure 3.20(a). Here, we show results for the tiles where clusters could not be formed during calibration due to the absence of objects. We observe that more than 65% of the tiles with objects were detected using the fallback strategy, with only 7% false selections. This implies that using our fallback strategy is more suitable than removing tiles altogether.

Without Search of Tile Statistics: We first recall that TILECLIPPER uses a search of the percentile statistics to find out the optimal one. To verify that this search is essential, we perform our study with a threshold value fixed at 50^{th} percentile for both the clusters across 8 videos, selecting 2 videos from each of the 4 datasets randomly. This results in more savings ($\approx 2\times$) at the cost of lower accuracy when compared with the study run on the same dataset but with calibration as illustrated in Figure 3.20(b). This happens due to the static nature of the threshold value. The tiles with objects of interest having a lower bitrate than the fixed threshold are missed,



(a) (b) (c)

FIGURE 3.21: Sensitivity to different parameters of TILECLIPPER on a subset of the videos. (a) shows the effect of different tile configurations on accuracy and savings, (b) shows the effect of different cluster sizes on accuracy and savings, and (c) shows the effect of varying calibration duration on TILECLIPPER. We choose 30s as it gives the best tradeoff between accuracy and savings.

affecting accuracy. Our selection of 50^{th} percentile is deliberate, as opting for any other lower value would have led to a compromise in accuracy.

Using different tile configurations: We compare the accuracy with 4×4 tiles, among 2×2, 4×4 and 6×6 configurations in Figure 3.21(a). In each case, the encoder uses constant-bitrate encoding to ensure that the size of the video does not change with tile configurations. While the accuracy remains relatively consistent irrespective of the setting used, we obtain substantial savings (over 30%) only with 4×4 and 6×6 configurations. While selecting 6×6 yields significant savings, it also escalates calibration complexities and increases the extraction time of bitrates by $3.5\times$ as opposed to 2×2 tiles. Thus, we selected the tile configuration of 4×4 since it strikes the optimal balance between savings and operational intricacies. A similar configuration was also chosen by CrossROI, citing an identical reason [9].

Varying cluster sizes: In Figure 3.21(b), we show experimentation with different cluster sizes. Interestingly, we find that varied cluster sizes do not affect accuracy. However, we observe a decrease in savings by 3.34% when the cluster size changes from 10 to 15. It is important to note that we avoid a cluster size of 5 over the choice of 10 as it captures the underlying bitrate distribution with a mere 7.5% dip in savings. Furthermore, we also note that choosing any size above the cluster size of 10 leads to a progressive decline in savings.

Varying calibration duration: Figure 3.21(c) demonstrates the calibration phase across different time durations, we deliberately chose videos from the dataset where the Tileclipper exhibited its worst performance. The motivation behind our experiment is to observe the effect of varied calibration duration. We observe that changing the calibration duration has relatively small effects on the overall accuracy and savings. Since increasing the calibration time did not contribute to an improvement in accuracy or savings, instead, we chose to utilize an average calibration of 30s to reduce the incidents of fallback.

3.5 Related Works

Works on traffic surveillance fall into two categories: optimization of traffic surveillance using a variety of intelligent strategies and design of lighter DNNs.

Optimization of Traffic Surveillance: Traffic surveillance is increasingly used in practice, and thus is an important area of research [6, 8, 119, 119, 132, 148, 149]. Chameleon [8] and Spatula[150] use the traffic correlation from multiple cameras to identify segments to send to reduce bandwidth consumption. CASVA [132] uses a deep reinforcement learning mechanism to adapt the video parameters to the changing bandwidth configuration. AccMPEG [4] optimizes server-side DNN accuracy by tuning video codec-specific parameters to reduce bandwidth usage. This requires it to rely on the server side to get the right parameters. CloudSeg [119] sends the video at a low resolution but then uses super-resolution on the server side. Like CloudSeg, DDS [7] similarly also sends the video at a low resolution but then requests additional parts of frames separately when the DNN has low confidence. Clownfish [148] and AdaMask [5] extract the background content from video frames and send only the objects, reducing data volume. However, identifying the objects requires a lightweight deep neural network, thus requiring a more expensive device on the camera side. Reducto [6] and SmartFilter [118] use a set of pixel-level operations to filter out irrelevant frames. MRIM [151] sends mixed-resolution frames with higher resolution in areas with objects. Unlike TileClipper, these works all work at the frame level, thus requiring the raw frames. Furthermore, like TileClipper, CoVA [121] analyzes videos in the compressed domain to identify objects of interest. However, unlike TileClipper, it uses neural networks on the server side.

Design of Lightweight Neural Networks: A number of works reduce the computation needed for vision tasks. For example, [152], [133], and [153] all run object detection on smaller edge devices like Jetson Nano. However, they still require a GPU for computation, and their speeds come at the cost of accuracy. FastDeepIoT [154] and DeepAdaptor [155] identify and then prune the nodes in the neural network for pruning to make it light enough for execution on embedded systems. NoScope [156], RECL [157], and Ekya [158] all design lightweight DNNs based on the situations observed by the cameras. Such techniques are orthogonal to our work.

3.6 Discussion on Design Choices, Limitations, and Future Work

We discuss a few aspects and limitations of TileClipper:

Missing Tiles with Static and Small Objects: TileClipper relies on the presence of moving objects to select tiles and, thus, misses static objects. As static objects would not raise the bitrate, the thresholding strategy would treat small movements as noise. Similarly,

slow-moving objects could also be missed by TileClipper, as they might not create sufficient changes in the bitrate. A possible mitigation strategy for slow-moving objects would be to use techniques like optical flow to interpolate missing frames/regions within videos. Since the changes in the case of slow-moving or static objects are relatively small, such interpolation can be done via optical flow estimation followed by pixel synthesis [159].

We also note that there is a slight drop in accuracy on the TileClipper with the Others and OurRec datasets, as they contain a few such missed cases. Furthermore, small objects ($< 1/20th$ of the tile dimensions), such as pedestrians or vehicles far from the camera, do not increase the bitrate sufficiently to get detected by TileClipper. They are considered as noise. However, missing such objects does not often hurt server-side application accuracy because such smaller objects ($< 1\%$ of the entire frame) are also missed by DNNs at the server.

Choice of Video Codecs in TileClipper: We have evaluated TileClipper on videos encoded using HEVC at constant bitrate (CBR). We choose HEVC as it is used in today's cameras [120, 160, 161], and there is open-source software available to manipulate its tiles easily. As newer codecs like AV1 and VVC support tiles and utilize similar encoding strategies, we expect TileClipper to also work with them. Note that H.264 encoders do not support tiled encoding, so it is difficult to use TileClipper with H.264. Furthermore, we utilize CBR encoding as the tool used for video encoding, Kvaazar, does not support variable bitrate (VBR) encoding [162]. As VBR allows changes in bitrates of segments depending on the complexity of content, such changes in the individual tiles (utilized by TileClipper) are even more strongly visible. Recent prior works on traffic surveillance, such as AdaMask, also utilized another similar technique called constant rate factor (CRF). Note that we do not consider adaptive bitrate (ABR), since none of the traffic surveillance systems currently use it. However, we believe that TileClipper can still function by initially calibrating itself for each of the possible quality levels on the server.

Reduced Precision with Unstable Camera: The DETRAC dataset has a few videos with unstable cameras, causing movement across the frame, leading to lower precision and savings. A possible solution (we did not explore) is to use reinforcement learning-based calibration to handle such problems by studying the correlation across tiles of segments. As the motion of moving objects on traffic would follow a pattern that leads to a temporal correlation among tiles. Such reinforcement learning-based camera calibration has shown good results in CamTuner [163].

Applying TileClipper on Other Applications: Our evaluation discusses object recognition, because TileClipper is agnostic to applications. This is unlike the baselines, such as DDS and Reducto, whose parameters need to change depending on the query sent to the camera. Because TileClipper does not make any changes at the frame level or quality of videos, it has no direct effect on applications such as object classification, reidentification, or tracking. TileClipper could also potentially be used to infer the level of congestion on roads since any

change in the number and/or movement of objects in a tile would trigger TileClipper to send the content.

3.7 Conclusions

In this work, we present a system TileClipper for traffic surveillance that substantially reduces bandwidth consumption by selecting spatial regions (tiles) of interest for further analysis at the server. TileClipper's tile selection leverages the observation that tile bitrates strongly correlate with the number of moving objects within a tile. We then implement TileClipper on inexpensive edge devices and demonstrate that it runs in real-time. We further show that it outperforms prior systems in accuracy, bandwidth savings, and/or computational cost across a wide range of scenarios and in live deployment.

Note that TileClipper's working is only evaluated here to infer the presence of objects. These techniques are insufficient while deciding which frames to send for re-training. In the next chapter, we look at a different strategy required when we filter frames for training.

Chapter 4

PIPETTE: Adaptive Selection of Relevant Samples for Continual Learning in Autonomous Vehicles

The market size of autonomous vehicles (AV) is drastically increasing, with an estimated increase of 6× by 2030 [164, 165]. These AVs extensively use deep neural network (DNN) models to analyze the data from their cameras to make decisions about control and/or predictions. Typically, AVs come integrated with additional hardware such as GPUs for onboard processing.

Since integrating such onboard processing is expensive, the DNNs used in AVs need to be smaller in size [156, 157, 158, 166]. Such smaller models, typically achieved either by designing shallow architecture neural networks [156], pruning [167, 168], and/or quantization [169, 170], exhibit limited generalizability and thus suffer from data drift [10, 157, 158, 171]. In other words, as the real-life test data varies from the training dataset in terms of lighting conditions [10, 171], perspective [172], distribution of objects or type of scenes [157, 158], the accuracy of such smaller models degrades. This problem is exacerbated by the fact that AVs are mobile, and thus, the scenario observed often changes. Our experiments show that the mean average precision (mAP)[1] of the nano version of the Yolov8 [173] model, on average, varies by 35.72% with time.

Continual learning [174, 175] is one promising technique to address the problem of data drift. The primary idea behind it is to occasionally retrain or finetune the lightweight compressed model (also referred to as the student model) on the new samples. Fig. 4.1 shows a typical retraining pipeline on an AV. Here, the student model continuously performs inference on the vehicle and stores the frames for future training. When triggered, the frames are offloaded to a remote cloud/edge server. On the server, a deeper, more accurate model, called a teacher model, annotates the samples to generate the ground truth required for training. Once retrained, the updated model is sent back to the vehicle for inference. This strategy of training a student model

[1]We use mAP with an Intersection over Union (IoU) threshold of 0.5 throughout this Chapter to measure the object detection accuracy of the models. This is a widely adopted metric by the vision community.

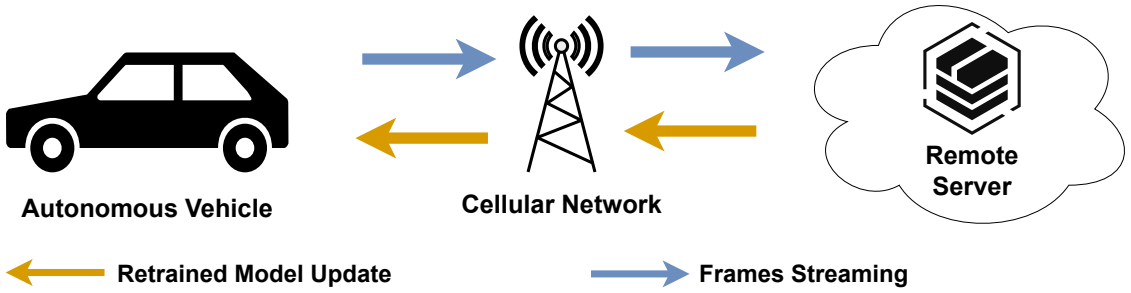


FIGURE 4.1: Overview of a typical model retraining pipeline.

using the labels of the teacher is known as knowledge distillation, which has been utilized by several prior works in the systems community [10, 157, 158, 166, 172, 176, 177, 178].

Challenges of Integrating Continual Learning into AV: Integrating continual learning into AVs, however, is non-trivial. The primary challenge is the substantial data transfer requirements for fine-tuning and the update delay associated with the model updates. At the typical sampling rate used in the context of AV continual learning [158], a vehicle can send a considerably large volume of data for retraining.

To exemplify, Ekya [158] uses frames of every $200s$ for retraining, assuming a framerate of $30fps$, it consists of 6000 frames. When encoded into $1080p$ video using H.264 [179] video compression standard (codec), it becomes $\approx 270MB$ of data. Study like [180] shows that the median 5G uplink speed even goes below 1 MBps. Uploading $\approx 270MB$ over such a channel will take $270s$, which exceeds the retraining window ($200s$) even without training. Moreover, since the retraining triggers are frequent (every $200s$ in Ekya), the streaming overheads become considerable. This hampers inference accuracy due to delayed model updates, which critically affects the control and safety of AVs [157]. This problem is likely to become more acute as the number of sensors per vehicle and AVs increases. A lower sampling rate, on the other hand, hurts the post-training accuracy.

Therefore, our goal in this work is to design a dynamic sampling algorithm that (a) reduces the total model update delay, (b) maintains high accuracy, and (c) adapts to changing bandwidth (BW) conditions experienced by the AVs.

Our Approach: We propose a system called PIPETTE that minimizes data ingestion cost and the model update delay by selecting only the samples relevant to the current training. We first run the student as usual for inference and pick the frames where the student's mean confidence score of all predictions within those frames remains below a certain threshold. Our experiments depict (in §4.1) that such a confidence score-based frame selection strategy is as good as using all frames (shown in Fig. 4.3) while reducing the number of training samples by 51.65%. Once selected, we encode these frames as videos before streaming to the server.

Furthermore, we design an algorithm that adaptively selects an appropriate encoding frame rate and training epochs on the AV before streaming, based on the current network characteristics, by minimizing an objective function (refer to Eq. (4.4)). Employing our adaptive frame sampling strategy serves three purposes: a) it reduces the network overhead by selecting only relevant training samples, b) it minimizes the model update delay, and c) it ensures safety by faster model update.

Since PIPETTE adapts the number of training epochs on the AV before streaming frames for training, it needs a profiler that can interpolate to obtain a more accurate estimate. Prior work on continual learning, such as [10, 158], obtains these estimates via one-time offline profiling. We find this sub-optimal for AVs, as it does not account for the complexity of the scene, which continuously changes in AVs. The accuracy improvement after training is a function of the scene complexity. Therefore, we propose an adaptive online profiler that updates itself after each model update. Along with the updated model, the server returns the updated profiler to the vehicle. This happens for each AV independently.

Evaluation & Contributions: We evaluate PIPETTE across 14 dashcam videos totaling 15.5 hours. We find that, compared to no retraining, PIPETTE leads to 9.37% improvement in test accuracy after retraining, while reducing the model update delay by 14% compared to a fixed-sampling strategy. To put the accuracy gains in perspective, the object detection mAP of the most accurate large variants of Yolo models has only improved by 6.3% from Yolov5 to Yolov8 in the last five years [181]. PIPETTE integrates seamlessly with the existing inference pipeline on the AVs, without adding any extra compute overhead. Furthermore, it does not require any server-side dependency for frame filtering.

Our contributions are summarized as follows:

1. We design PIPETTE with a confidence-score-based frame-sampling strategy to select hard samples relevant for training. It does not add any extra compute overhead to the existing inference pipeline on the autonomous vehicle.

2. We identify that a fixed encoding frame rate fails to provide a deadline guarantee due to varying network BW. We design an algorithm that adaptively selects the optimal encoding frame rate and training epoch by jointly optimizing them.

3. We design an adaptive online profiler that generates estimates for each AV independently and updates itself over time to adapt to the varying observed scenes of the AVs.

4.1 Background & Motivation

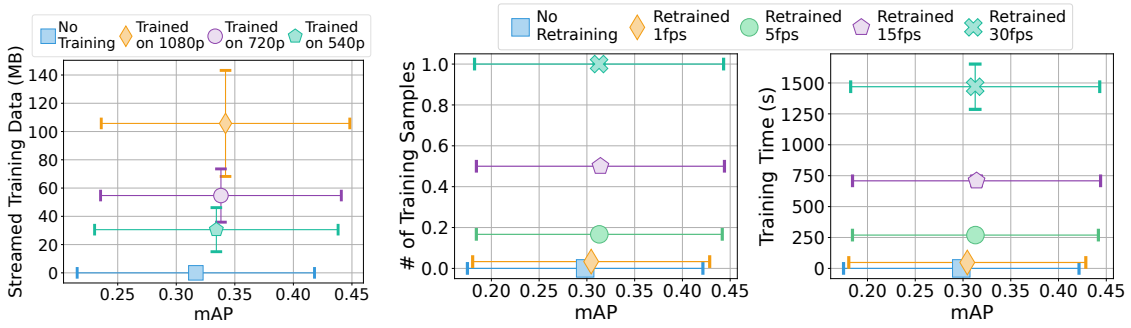
4.1.1 Effect of Video Parameters

Typically, the frames are streamed after encoding to improve compression efficiency and save bandwidth, rather than in the raw format. We first explore the possibility of selecting fixed video parameters for optimal compression. There are three possible knobs to play with to reduce the filesize of the encoded video: 1) resolution, 2) framerate, and 3) quantization parameter. Among these, the quantization parameter (QP) offers the most significant size benefits at the expense of quality. Distortions in the quality affect the model's inference accuracy as shown by [4, 7, 182]. During training, such degradations can lead to poor-quality teacher annotations that impair a student's learning. Therefore, to reduce the amount of streamed training data, we only vary the resolution and framerate.

To study the impact of these two parameters, we conduct experiments on a subset of eight videos (each $\approx 1hr$ long) from our total dataset. We use the state-of-the-art Yolov8 [173] model. The model details and videos are provided in §4.3. For each experiment, we use the first $570s$ of the video for training[2] and the rest for testing. We note from Fig. 4.2(a) that reducing the resolution of the training frames to half, i.e. $540p$, drops the mAP of the trained model only by 2.3% and 1.2% compared to using $1080p$ and $720p$ resolution, respectively. This drop becomes insignificant when we consider the reduction by 3.5× and 1.8× in the streamed training data. Therefore, we choose to use $540p$ frames for training.

Video Framerate vs. Model Accuracy: Videos consist of a sequence of highly correlated frames across the temporal dimension. Video codecs exploit this property for compression. Due to the high level of similarity, consecutive frames are almost identical. Ideally, training samples should be independent and identically distributed (i.i.d.) for robust and reliable training. This implies that when passed for training, the highly correlated frames of videos provide a marginal gain. We validate this by training on $1080p$ frames sampled at 1, 5, 15, and $30fps$. We report the mAP, number of samples, and training time in Fig. 4.2(b). We observe the improvement in mAP saturates after $5fps$ boosting, it only increases by 0.4% at $15fps$ in contrast to $5fps$ while using 3× more samples. The number of samples increases linearly with an increase in the sampling rate. Similarly, the training time rises linearly with an increase in frame sampling rate. Therefore, using a sampling rate of $5fps$ or less is more efficient and economical. Besides, training with $1fps$ sampled frames improves mAP too, though not as much as $5fps$. This is particularly helpful in cases where the network bandwidth (BW) and round-trip time (RTT) are insufficient to stream at higher frame rates (fps), as this would inadvertently increase the model update delay.

[2]Throughout this Chapter, by training we mean finetuning, i.e., reusing the weights of a previously trained model for retraining.



(a) mAP vs. streamed data

(b) mAP vs. normalized # of samples and training time

FIGURE 4.2: mAP versus streamed data, training time, and number of samples.

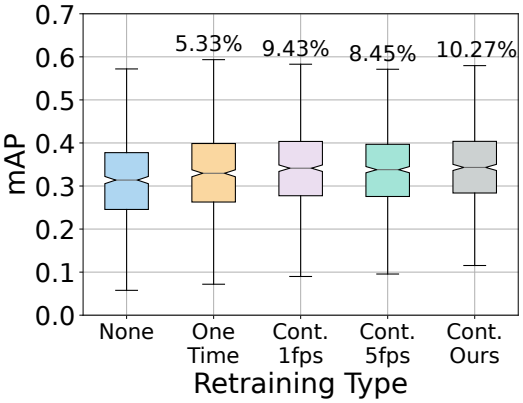
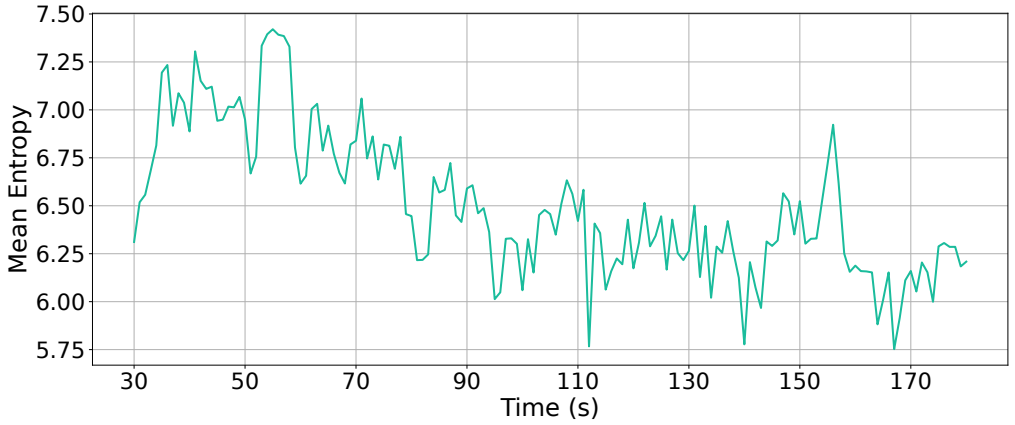


FIGURE 4.3: mAP improvement when using different training schemes for continuous retraining.

4.1.2 Benefits of Continuous Training

We report in Fig. 4.3 the mAP with one-time and with continuous retraining. Similar to the previous subsection, in the one-time scheme, we train the model only once on $570s$ of the video, while in the continuous retraining scheme, we periodically retrain the model in every window (i.e., every $30s$[3]). We find that compared to no training (None), retraining the model once provides 5.33% better mean object detection accuracy. Furthermore, continuously retraining in every window with a frame sampling rate of $1fps$ and $5fps$ (Cont. 1fps and Cont. 5fps) improves the mean mAP by 9.43% and 8.45%, respectively. This is approximately 2× improvement compared to one-time training. This highlights that periodically retraining the model helps it adapt to adverse, unseen scenarios, thus leading to improved accuracy. We show such a drastic scene change in Fig. 4.4. We put the snapshots of frames before and after such a change in Fig. 4.4(b) and Fig. 4.4(c). To show the gradual change in the scene, we use the mean entropy of frames. A higher entropy represents a larger number of distinct pixels within a frame (e.g., during daylight), while a low entropy implies a larger number of similar pixels (such as during night). The plot

[3]We use a fixed retraining window like RECL [157], Ekya [158], and AMS [166]. Dynamic window size selection is not in the scope of this work.



(a) The mean entropy of the frames gradually drops as the scene becomes dark at night, necessitating a quick model update.

(b) At the beginning of the video (evening scene).

(c) After 50*s* of the video (dark night scene).

FIGURE 4.4: (a) The entropy of the frames changes with the change in the video scene, implying a domain shift from scene in (b) to scene in (c).

in Fig. 4.4(a) depicts the gradual decrease in the entropy of the frames when the scene becomes dark (Fig. 4.4(c)). This change requires a faster model update to handle the change in the scene.

The last box in Fig. 4.3 shows the mAP when using our confidence-based frame sampling strategy for continuous training. Compared to None, it achieves 10.27% mean accuracy, which is 21.5% higher with 51.65% fewer samples in contrast to the Cont. 5fps scheme. These observations imply that periodic retaining is inevitable. Therefore, it becomes pertinent to leverage a rational frame selection strategy to minimize the training data sent to the server, where our confidence score-based frame selection strategy is particularly promising. Note that such optimization strategies are not possible if we directly borrow techniques used in streaming for applications like video-on-demand (VoD). This is because such applications are designed to enhance the human experience, whereas in AVs, the optimizations should focus on training neural networks [183].

4.1.3 Need for Adaptive Frame Sampling

The above discussions validate the idea of streaming at a reduced resolution and frame rate. We now ask the question, *is streaming with a carefully chosen static video parameter enough?* To answer this question, we select a fixed sampling rate of 5fps and encode the frames of 30s as a

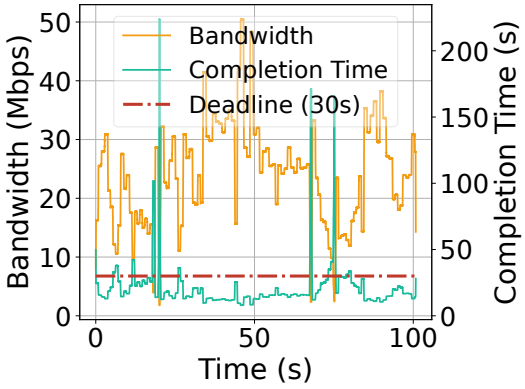


FIGURE 4.5: Completion time of sending $50MB$ of training data over a varying bandwidth network often exceeds the retraining window duration of $30s$.

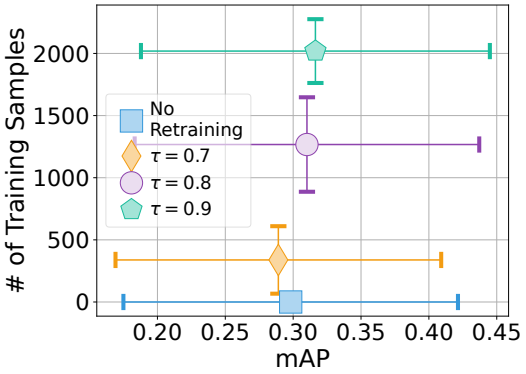


FIGURE 4.6: Effect of thresholds (τ) for the confidence score-based sampling on the object detection accuracy (mAP) and the number of training samples.

video, resulting in $50MB$ of streaming data. Assuming that we have to stream the same amount of data in all windows, we compute the completion time of sending this video over a network with varying BW as shown in Fig. 4.5. We note that the completion time often exceeds our retraining window of 30s even without training involved. This emphasizes the need for an adaptive frame sampling mechanism that can adjust the number of samples according to the available network bandwidth. Note that the total model update delay depends on both streaming and training time. Therefore, for minimizing total delay, the joint optimization of both these components is essential.

4.2 PIPETTE's Design and Architecture

The design of PIPETTE leverages the benefits of both the static and dynamic parameter settings. Static knobs include the video resolution and the filtering threshold of the confidence score. PIPETTE streams the video at half resolution ($540p$) to the server for training and uses a confidence threshold of $\tau = 0.8$. Fig. 4.6 shows the effect of the different thresholds on the accuracy (mAP) and the number of training samples for one-time training. We find that $\tau = 0.8$ gives the best tradeoff. It improves the mAP by a similar amount as 0.9, while reducing the number of samples by 36.17%. Note that we keep the minimum detection confidence score to 0.5. Therefore, the number of samples with confidence scores below 0.6 was zero most of the time. Hence, we only experimented with thresholds from 0.7 to 0.9. We now discuss PIPETTE's architecture, frame sampling strategy, and its dynamic encoding frame rate selection algorithm.

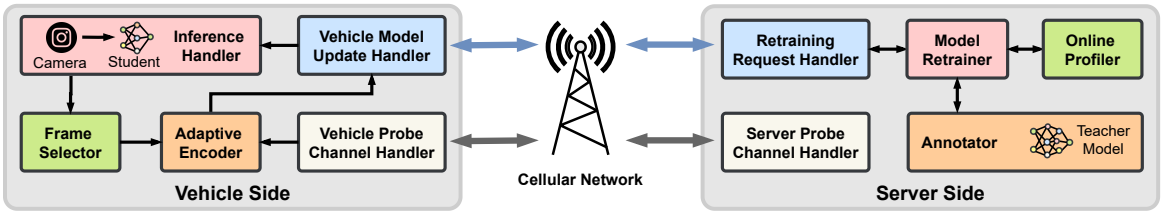


FIGURE 4.7: PIPETTE's Architecture and Workflow

4.2.1 Design Choices and Architecture

PIPETTE's frame sampling algorithm has the following motivation:

1. **Minimal overhead on the vehicle:** Since an AV has to continuously detect the objects in realtime, it is essential to use a lightweight frame selection algorithm.
2. **Adapt to the network to speed up model update:** The streamed frames and the training epochs should adapt to the current network conditions to update the model quickly.
3. **Accurate but fast retraining:** The selected samples should be representative enough to provide improvements after training, even with reduced training time.

PIPETTE's Workflow: Fig. 4.7 depicts the architecture and workflow of the different components of PIPETTE. Both the vehicle and the server-side consist of five independent components. On the vehicle side, initially, the *Vehicle Probe Channel Handler* establishes a connection with the corresponding handler (*Server Probe Channel Handler*) on the server side. This handler periodically (every $100s$) probes the network to reckon the round-trip-time (RTT) and bandwidth (BW). We use Iperf3 [184] to get the estimates. *Adaptive Encoder* uses the estimates from *Vehicle Probe Channel Handler* to dynamically select the encoding frame rate and the number of training epochs based on the current RTT and BW of the network.

The *Inference Handler* continuously fetches frames from the vehicle camera and performs inference on them for the vehicle to maneuver itself. After inference, it passes the frames and the prediction results to the *Frame Selector*, which decides whether to keep or discard the current frame using our confidence score sampling (§4.2.2). At every retraining window ($500s$), *Frame Selector* passes all the sampled frames to the *Adaptive Encoder*, which runs the algorithm in Algo. 4 to get the right frame rate (r) for encoding and the training epochs (e) depending on the current network RTT and BW. Afterwards, it encodes the selected frames as video using H.264 [179] video codec at a frame rate of r and sends it to *Vehicle Model Update Handler* for streaming. *Vehicle Model Update Handler* attaches the training epoch (e) in the header of the streamed video, which the server dissects to get back e. It waits until training finishes. It notifies

the *Inference Handler* to switch the current student model to the updated one once the server sends the retrained model.

In summary, PIPETTE has two sequential stages of frame selection before streaming them to the server: (1) selecting relevant frames for training using the confidence score, and (2) choosing the right frame rate for encoding, depending on the current BW. We refer to the rate of these two stages (frame sampling rate and encoding frame rate) combined as the adaptive frame sampling rate of PIPETTE.

On the server side, *Server Probe Channel Handler* manages the Iperf3 server for the vehicle's Iperf3 client. The *Retraining Request Handler* receives the streamed video from the AV and passes it to the *Model Retrainer* for training. *Model Retrainer* first initializes *Annotator* to generate the ground truth for training by running the teacher model. Once all the frames of the video are annotated, the retraining runs for e epochs on the *Model Retrainer*. Subsequently, *Model Retrainer* triggers *Online Profiler* to fit a new profiling curve for accuracy prediction on the AV. Ultimately, *Retraining Request Handler* sends back the retrained model and the new profiling curve to the AV. The AV receives the updated model to replace the older one and updates the profiler (§4.2.4).

4.2.2 Hard Sample Selector

Model training updates the weights using the gradient of the difference between the predictions and the ground truths. Thus, the training is beneficial only if the gradients are large, i.e., the model's predictions are significantly different from the ground truths. Using training samples where the model is already good enough does not improve accuracy, but rather wastes resources. Therefore, it is imperative to pick relevant training samples. The confidence score associated with each model prediction represents the model's confidence in its own predictions. A lower value represents that the model struggles in detection. We utilize this confidence score to select frames (SELECT procedure of Algo. 3) where the student model gives poor predictions. We call such frames hard samples. Training on these hard samples benefits the student model the most. We define a function to pick the *hard samples*. Let there be n objects detected in a frame I_t at time t. Let the corresponding confidence scores associated with each prediction i be $p_i(I_t)$. Then the confidence score of each image $C(I_t)$ is approximated using:

$$C(I_t) = mean(p_i(I_t)), \forall i \in \{1 \dots n\} \tag{4.1}$$

For every frame of a window (Line 3 to Line 8 in Algo. 3), the *Frame Selector* calculates the mean confidence score using Eq. (4.1). If $C(I_t)$ is greater than the threshold τ, it implies that the model is good enough on those frames, hence they are discarded. Only the frames with $C(I_t) < \tau$

Algorithm 3 PIPETTE's confidence score-based frame selection.

INPUT: List of frames I, confidence score threshold τ.
OUTPUT: List of selected frames in the current window.

```
1:  procedure SELECT(I, τ)
2:      O ← [ ]                          ▷ Empty list to store selected frames
3:      for each frame f in I do
4:          P ← [ ]                      ▷ Empty list to store predictions
5:          for each object d in f do
6:              p ← GetConfScore(d)
7:              P.append(p)
8:          C ← ComputeMeanConfScore(P)
9:          if C < τ then
10:             O.append(f)              ▷ Store current frame
11:     return O
```

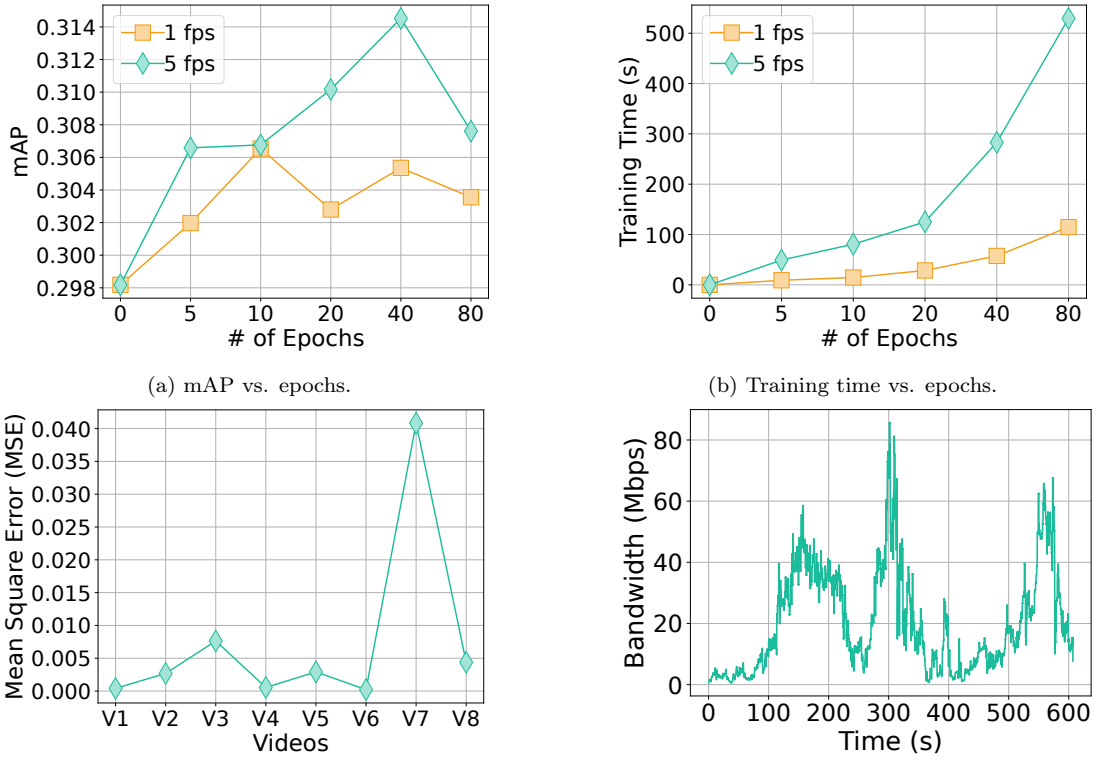
are selected for training (Line 10). A sensitivity study with different metrics to calculate $C(I_t)$ is discussed in §4.3.5.

At the end of the current window, *Frame Selector* passes all the selected frames to *Adaptive Encoder* for encoding. Since this sampling strategy does not entail any complex computation, it has negligible overhead. Moreover, running our confidence score-based sampling strategy on top of adaptive encoding helps reduce the number of samples, minimizing the model update delay. This sampling strategy is better than simple frame sampling, which always selects frames regardless of the model's current behaviour.

4.2.3 Adaptive Encoding Frame Rate Selection

We argue in §4.1 that the encoding frame rate of the video should be dynamic and depend on the available network BW. Intuitively, during periods with constrained BW, the encoding frame rate should be lower compared to relatively higher BW periods. This strategy ensures that the model update delay remains small. However, the total model update delay is the sum of the video streaming completion time, the streaming delay, and the training and annotation time. Therefore, to ensure minimal model update delay, we capture and identify the optimal values for all these components.

Problem Formulation: We design a utility function to capture the contribution of each and every component associated with the model update pipeline. Let N and e be the total number of selected training frames and epochs, respectively. Function $S(N, r)$ returns the filesize (s) and $B(N, r)$ returns the number of subsampled frames (f) when N frames are encoded at the frame



(a) mAP vs. epochs.

(b) Training time vs. epochs.

(c) MSE of fitted curve.

(d) BW pattern of the trace.

FIGURE 4.8: (a) and (b) show the variation of mAP and training time with the number of epochs. (c) shows the mean square error of accuracy prediction when using a single profiler across various videos. (d) shows the bandwidth (BW) pattern of the network trace used in Mahimahi for experiments.

rate of $rfps$. Then $F(s)$ calculates the streaming delay of the encoded video with $f = B(N, r)$ frames as:

$$F(s) = F(S(N, r)) = \frac{RTT}{2} + \frac{S(N, r)}{BW} \tag{4.2}$$

Assuming $G(f)$ denotes the ground truth generation time on $f = B(N, r)$ frames of the video encoded at $rfps$. It is a linear function and depends on the teacher model and the server compute. $T(f, e)$ and M denote the training time and the delay of streaming back the updated model. Then, mathematically, the total delay Δ is:

$$\Delta = F(s) + G(f) + T(f, e) + M = F(s) + G(B(N, r)) + T(B(N, r), e) + M; \forall r \in R, \forall e \in E \tag{4.3}$$

Here, R and E represent the sets containing the total number of possible encoding frame rates and training epochs. Our objective is now defined as:

$$\textbf{Minimize}_{\mathbf{r,e}}\ \Delta \tag{4.4}$$

$$s.t. \quad A(f,e) - \alpha F(s) >= \delta, r \in R, e \in E \tag{4.5}$$

Here, $A(f,e) = A(B(N,r),e)$ denotes the accuracy of the model when trained on $B(N,r)$ frames for e epochs. We pick r and e which minimizes Δ where $A(f,e) - \alpha F(s)$ is greater than or equal to δ. We define δ as the p^{th} percentile of $A(f,e) - \alpha F(s)$. We empirically set $\alpha = 0.2$ and use Fig. 4.16 to set 60^{th} percentile for δ.

Solution: To solve the above optimization problem, we use a brute force method. We first reduce the search space by limiting the range of E and R. From Fig. 4.8(a) and Fig. 4.8(b), we note that increasing the number of epochs beyond 20 provides diminishing returns in terms of mAP. Therefore, we restrict the range of epochs within $\{5, 10, 15, 20\}$. Similarly, we keep the encoding frame rate within $\{1 \dots 5\}$ as discussed in §4.1. Thus, the total combinations become 20, which is small enough for an exhaustive search (SEARCH procedure in Algo. 4) to solve without much computational overhead.

4.2.4 Design of Profiler to Estimate Performance

Pipette solves the optimization problem on the AV itself before streaming the encoded frames to the server. Therefore, it necessitates estimating all the components of Eq. (4.3) beforehand. This implies that Pipette should estimate the improvement in accuracy if it chooses a certain encoding frame rate and number of training epochs. For this, we design a profiler that can predict the accuracy, training time, and the filesize of the encoded frames given a certain encoding frame rate (r) and epoch (e) from R and E. We start by reusing the approach in [10, 158], where the authors perform an offline profiling over a subset of the dataset to obtain a polynomial fit. Then the fitted polynomial can be interpolated to get estimates. However, during our experiments, we observe that such a globally fitted curve fails to generalize across the videos. Fig. 4.8(c) shows, for a subset of videos, the mean square error (MSE) of the predictions of the globally fitted curve and a curve fitted individually on every video. We find the MSE to vary by $\approx 40\%$. Moreover, the fitted curve should be updated continuously in every window. Therefore, unlike the prior works, we design a profiler that keeps updating the fitted polynomial in every window to minimize the error in the predictions. The annotation time is a linear function of the number of frames and depends on the teacher model and the type of server GPU. We perform an offline profiling to get the linear function. We use the first training window of Pipette to fit a curve for the accuracy and training time profilers.

Filesize Predictor: Since the AV has only the frames, it is necessary to estimate the filesize of the encoded video at different encoding frame rates. Similar to the result in Fig. 2.7, we perform a small offline experiment to study the relation between the filesize and encoding frame

Algorithm 4 PIPETTE's algorithm to solve Eq. (4.4).

INPUT: List of possible encoding frame rates R and training epochs E, percentile threshold p^{th}, per frame annotation time λ, uplink bandwidth and RTT B_u and R_u, downlink bandwidth and RTT B_d and R_d, model size Ω, the coefficients of training time and accuracy prediction curve W_t and W_a.
OUTPUT: Encoding frame rate r, number of training epochs e.

procedure SEARCH($R, E, p^{th}, \lambda, \Omega, B_u, R_u, B_d, R_d, W_a, W_t$)
  $U, D, V \leftarrow [\,], [\,], [\,]$
  for each rate r in R **do**
    $f \leftarrow N/r$ ▷ Subsampled frames when rate is r, B(N, r)
    for each epoch e in E **do**
      $S \leftarrow GetFileSize(f)$ ▷ Encoded frames filesize
      $F \leftarrow GetFrameStreamingDelay(B_u, R_u, S)$
      $G \leftarrow \lambda.f$ ▷ Annotation time
      $T \leftarrow GetTrainingTime(f, e, W_t)$
      $M \leftarrow GetModelDownloadingDelay(B_d, R_d, \Omega)$
      $A \leftarrow GetAccuracy(f, e, W_a)$
      $\Delta \leftarrow F + G + T + M$ ▷ Total delay
      $u \leftarrow A - \alpha F$ ▷ Current utility
      U.append(u)
      D.append(Δ)
  for $v \leftarrow 1 \ldots |U|$ **do**
    if $U[v] >= percentile(U, p^{th})$ **then**
      V.append($D[v]$)
  $m \leftarrow argmin(V)$
  $ii \leftarrow indexWhere(D, V[m] == D)$ ▷ index of min(D).
  $r, e \leftarrow R[ii], E[ii]$
  return r, e

rate. We note that increasing the encoding frame rate from $1fps$ to $2fps$, $3fps$, $4fps$, and $5fps$ increases the filesize by $\approx 30\%$, $\approx 55\%$, $\approx 80\%$, and $\approx 100\%$ respectively. We use this relation to interpolate the filesize using the size of the video from the previous update window.

Accuracy Predictor: For the accuracy predictor, in the first window, we use an encoding frame rate of 5 and epochs of 20 for training. On the server, we fit a degree four polynomial using the output of the first training, which is returned to the AV. In the subsequent windows, we use this curve for accuracy prediction and keep updating the curve. This procedure happens for every video individually. During every update window, the coefficients of the polynomial are updated using the equation:

$$V_{new} = \mu V_{current} + (1 - \mu) V_{old} \tag{4.6}$$

Here, V_{new} is the updated value, $V_{current}$ is the actual value fitted in the current window, and V_{old} is the previous value. We empirically set $\mu = 0.1$. Note that this update happens on the server because the new relation between the epoch and the mAP becomes available on the server after training. Once updated, the values are returned to the AV along with the updated model.

Training Time Predictor: Our training time predictor uses the same approach as the accuracy predictor, with the only difference being the degree of the fitted polynomial, which is one. Besides, unlike the accuracy curve, we do not adapt the fitted curve for training time prediction.

4.3 Implementation & Evaluation

We implement both the vehicle and server-side components of PIPETTE separately in Python3. We utilize the multiprocessing library to create independent processes for the handlers (§4.2) running concurrently. The profiler fits the polynomial curves using NumPy.

Testbed: Our server runs on a PC with an AMD Ryzen 9 7950X 16-Core Processor CPU equipped with 32 GB of RAM and Nvidia GeForce RTX 4080 GPU. The AV side codebase runs on Nvidia AGX Xavier, which has an 8-core Nvidia Carmel ARMv8.2 CPU with 512 CUDA cores and 32 GB RAM. Both the server PC and the AGX are connected via Ethernet. To emulate an actual wireless network, we use Mahimahi [185], which uses a pre-recorded network trace and replays it to simulate realistic network characteristics. We run our own collected network trace (shown in Fig. 4.8(d)) inside Mahimahi.

Models: We conduct all the evaluations for the object detection application, which is the most crucial part of AV. For this, we use the state-of-the-art Yolov8 [173] DNN model pretrained on the MSCOCO dataset [186]. We then select its smallest model (Yolov8n) and create specialized student expert predictors by pruning them using Torch-Pruning [168]. Since the compressed expert models are known to achieve superior accuracy with smaller prediction categories [156, 187, 188], similar to [171], we restrict the number of classes to ten (the first ten classes of the MSCOCO dataset), which are relevant for AV. Note that new unseen classes can be added during retraining. On the server, we use Yolov8x (the largest variant of Yolov8) as the teacher model, which generates the ground truths required for training.

Dataset: To achieve meaningful data drift, the evaluation videos should be sufficiently long. Therefore, similar to prior works [10, 157, 171], we pick 14 dashcam/driving videos from YouTube [189]. The videos are captured in various cities across different countries, under diverse environments, and at different times of day. Thus, these represent a diverse set of environments and traffic conditions that an AV witnesses. Our included videos have a length of $\approx 30min$ to $1hr30min$. Thus, the total length of the video becomes $\approx 15.5hrs$. All videos have a resolution

of $1080p$ and a frame rate of $30fps$. We finetune the specialized student model on the first $570s$ of every video before running experiments to remove any learning discrepancy introduced by the pruning. This acts as our pretrained baseline and ensures that the model does not benefit from any specific video. We repeat all experiments on each and every video independently. While streaming the distilled frames for training in every window, we encoded them into $540p$ videos to harness more savings due to reduced resolution (§4.1). However, for inference on the AGX (acting as our AV), we use the original $1080p$ videos. We use a retraining window of $500s$ for all experiments (sensitivity study in Fig. 4.17).

Baselines: We compare PIPETTE with the following baselines.

1) No Retraining (NR): For this, we run the model on the videos without any training (except the finetuning on the first $570s$), which behaves similar to our pretrained model.

2) One Time Training (OTT): We finetune the pretrained model once again on the $570s$. The rest of the video is used for testing. This baseline helps illustrate the benefits of continuous retraining.

3) Continuous Retraining With Fixed Sampling (FS): Works like Ekya [158] and RECL [157] use a fixed frame sampling strategy and a fixed retraining window. Using it as a baseline justifies the requirement of having adaptive frame sampling. For training, we fix the sampling rate to $5fps$ and the encoding frame rate to $1fps$ to minimize model update delay and set the epochs to 20 to maximize accuracy. This can be seen as the best possible case.

4) Continuous Retraining With Content-based Adaptive Sampling (ASR): Prior works such as AMS [166] and Gecko [176] use content-based adaptive sampling. The idea is to adjust the frame sampling rate based on the scene's dynamism. When the AV moves slowly, they pick a lower sampling rate and increase it when the speed is higher. We borrowed the AMS implementation from their open-sourced codebase and integrated it into our code. We keep all parameters the same as in the original paper, including epochs=20, except for the sampling rate range. Originally, AMS used sampling rates between 0.1 and 1, but we keep it between 1 and 5, as PIPETTE does, for a fair comparison. The encoding frame rate varies with the sampling rate.

Note that for all the baselines, to have a fair comparison, we keep the retraining window to $500s$ similar to PIPETTE and encode at $540p$[4] when streaming to the server for training.

[4]Note that the original papers streamed at the original video resolution (higher than $540p$). This overburdens the network, significantly increasing the model update delay. However, for a fair comparison, we still use 540 for the baselines, which reduced their update delay.

4.3.1 Evaluation Metrics

We now evaluate PIPETTE to assess its performance and validate the design choices for object detection. However, PIPETTE can be generalized to other similar tasks like semantic segmentation. We use the following metrics for PIPETTE's evaluations:

(1) Model Update Delay: We define the model update delay as the time taken from the start of a retraining window to the time a retrained model is received at the AV. Having a smaller model update delay is optimal. It encompasses the delay (in seconds) introduced by all the components (encoding, frame streaming, annotation, training time, and model download time) in the training pipeline.

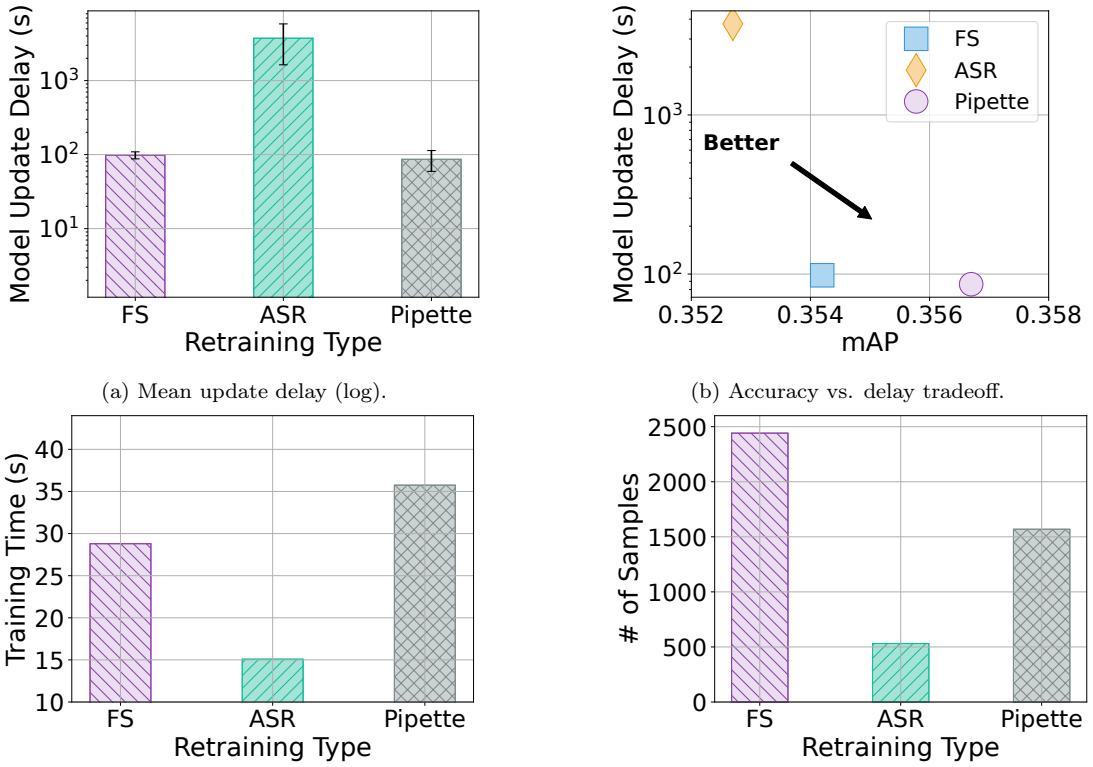
(2) Accuracy: We test PIPETTE on the object detection task and assess its accuracy using mean average precision (mAP) at an intersection over union (IoU) threshold of 0.5. This is a widely adopted metric for object detection within the computer vision community, as it is known to be more robust compared to metrics such as the F1 score. A higher mAP implies superior detection across all classes.

(3) Number of Selected Frames for Training: We showed in Fig. 4.3 that our confidence score-based frame selection strategy picks relevant frames (hard samples) for training, which is fewer than the fixed frame selection technique. We log the number of such samples that PIPETTE chooses.

(4) Training Time: We log the duration of the training session (in seconds) in each retraining window. The training time depends on the number of training epochs and the number of training samples.

4.3.2 Comparison with Baselines

Model Update Delay: Once we get the accuracy, we then quantify how long it takes to get the updated retrained model back from the server. This is crucial because obtaining the updated model as quickly as possible enables the AV to take correct actions for improved safety [157]. We report the model update delay associated with each of the baselines in Fig. 4.9(a) on a log scale. We observe that ASR results in orders of higher delay compared to the other two baselines. This happens because ASR intermittently (every $10s$) sends the sampled frames to the server to calculate the sampling rate to use for subsequent frames, depending on the content dynamicity. These frequent streamings introduce delays, which, when aggregated, escalate. PIPETTE provides the updated model with the least delay of $86.37s$, which is $\approx 14\%$ faster than FS. These observations imply that PIPETTE incurs the least overhead in the entire training pipeline. Since NR and OTT do not rely on the server for training, we skip their results.



(a) Mean update delay (log).

(b) Accuracy vs. delay tradeoff.

(c) Mean training time.

(d) Mean # of selected frames.

FIGURE 4.9: Comparison with the baselines across all performance metrics: accuracy (mAP), model update delay, training time, and number of selected frames in each retraining window. The key takeaway is that PIPETTE achieves the best mAP vs delay tradeoff by balancing the total number of training frames and training time.

Accuracy vs. Model Update Delay Tradeoff: To highlight the benefits of PIPETTE, we report the tradeoff between mAP and delay for all the baselines in Fig. 4.9(b). Having a higher accuracy and smaller update delay is ideal, as this implies superior accuracy using a quick reception of the updated model. We find that PIPETTE achieves the best tradeoff among all the baselines. That is, $\approx 14\%$ and $\approx 43\times$ faster model update than FS and ASR, respectively. While achieving mAP 0.7% better than FS and 1.2% better than ASR across all videos. The improvements of PIPETTE become much more considerable under extreme conditions: low bandwidth and severe scene change (Fig. 4.10 and Fig. 4.12). Thus, PIPETTE is able to balance the number of samples and the number of training epochs according to the current network characteristics, achieving an appropriate tradeoff between the two metrics.

Training Time: During our experiments, we note that, although adaptive, ASR always selected a frame sampling and encoding rate of $1fps$. We speculate that this happens because the original paper [166] was designed for a semantic segmentation task. Therefore, the designed formula may not generalize to the object detection tasks we evaluate for. Due to the smaller sampling and encoding frame rates, the total number of samples is less (shown in Fig. 4.9(d)). Having fewer

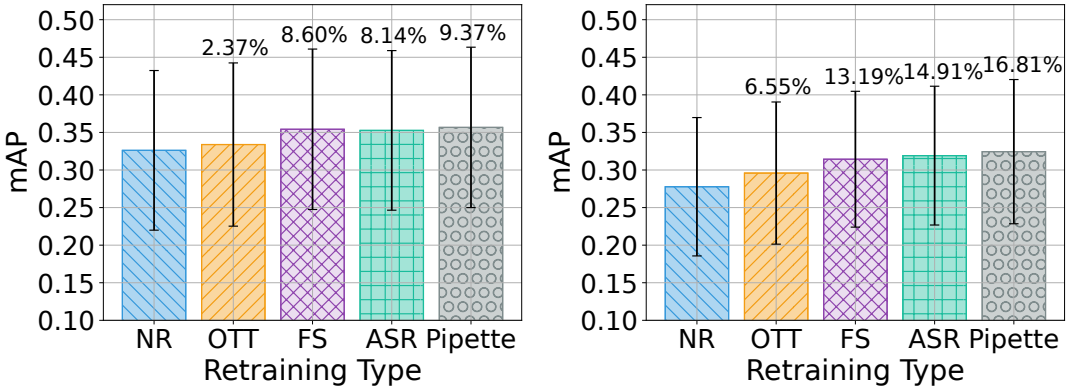


FIGURE 4.10: Mean mAP on all videos (left) and videos with drastic change (right), respectively.

training samples implies smaller training time, as shown in Fig. 4.9(c). PIPETTE shows the highest training time 31% more than FS. This happens because PIPETTE uses a higher encoding frame rate than FS and ASR. On average, it picks an encoding rate of $1.5fps$, which is higher than the $1fps$ rate of FS and ASR. However, the smaller number of selected samples in PIPETTE amortizes this cost, minimizing the total update delay.

Number of Selected Training Samples: We argued in §4.1 that our confidence score-based selection strategy picks only relevant samples for training, which is better than a fixed frame selection scheme. To validate this choice, we log the absolute number of selected training frames in each window. Fig. 4.9(d) shows the mean number of selected frames by each technique in each retraining window. We note that ASR samples the least number of frames while incurring the longest update delay. PIPETTE picks 35.7% fewer samples than FS while achieving a better mAP at the same time. Unlike FS and ASR, the samples selected by PIPETTE are the hard samples that are more relevant for training. By adaptively balancing the number of samples and training time, PIPETTE achieves superior accuracy and update delay.

Accuracy: Fig. 4.10 depicts the mean accuracy of all the baselines, aggregated across all videos. The percentage values above each bar represent the mean improvement compared to NR. We note the following observations: (1) training once (OTT) improves the accuracy only by 2.37%, stressing that the one-time adaptation is not sufficient. (2) All the continual retraining techniques, namely FS, ASR, and Pipette, achieve at least 6% better accuracy compared to OTT. (3) ASR achieves the worst accuracy improvements among all continual learning strategies. (4) PIPETTE is able to attain the highest accuracy, improving it by 9.37% compared to NR and 6.86% better than OTT. All these four observations necessitate a continual learning-based strategy. Furthermore, we report the detection accuracies across all classes for PIPETTE and FS in Fig. 4.11. We observe that for classes such as "bicycle", "bus", "train", and "boat", PIPETTE outperforms FS. We skip ASR because it incurs an order-of-magnitude higher delay, making it the worst baseline.

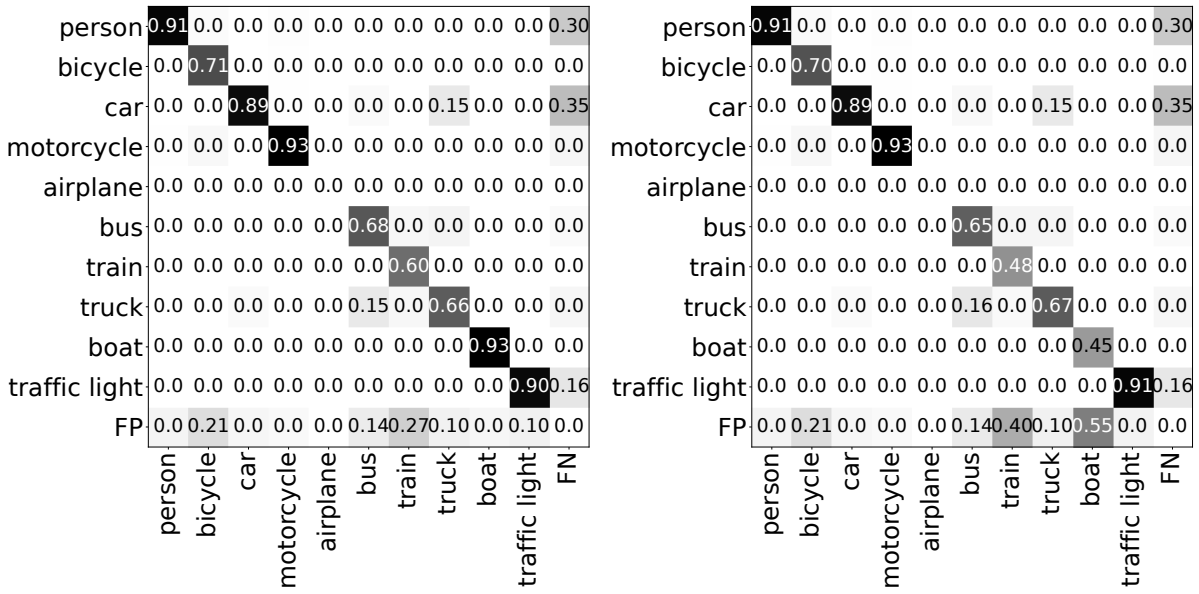


FIGURE 4.11: Confusion matrix of PIPETTE (left) and FS (right).

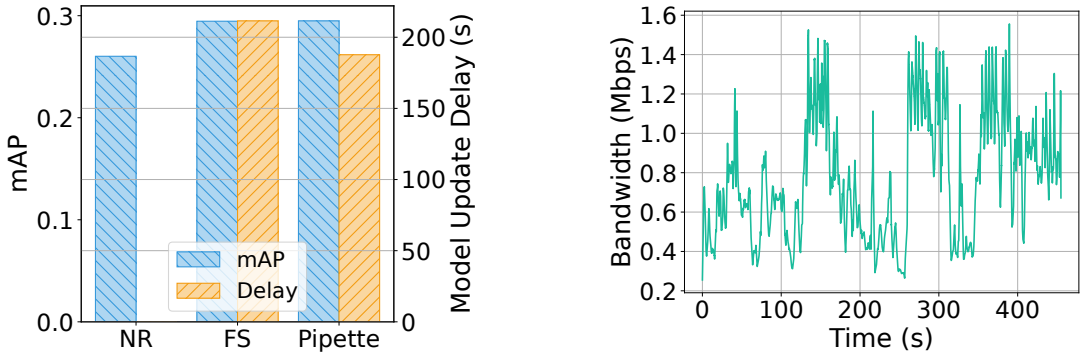
4.3.3 Impact of Drastic Scene Change on Accuracy

We study the effectiveness of all the baselines under drastic changes in environmental lighting conditions. For this, we identified five videos (lasting $\approx 5hrs$) from our dataset where the scene changes from light to dark or dark to light (Fig. 4.4).

We report the mAP of all the schemes in the Fig. 4.10. We note that under severe domain change, all the continual learning schemes (FS, ASR, and PIPETTE) improve the accuracy by at least 13%. The mAP of the pretrained models suffers severely under these conditions and remains around 0.27 on average in contrast to 0.32 on the aggregate level (all videos). The large improvements ($\approx$ 53% more) imply the benefit of continual learning under such conditions. We note that PIPETTE provides the highest accuracy in such scenarios, surpassing FS by 3.6% and ASR by $\approx$ 2%. The accuracy benefits of PIPETTE become considerable when we consider the fact that the mAP of the most accurate variants of Yolo models has only improved by 6.3% from Yolov5 to Yolov8 in the last five years [181]. Furthermore, the accuracy gains of PIPETTE also come with faster updates, incurring 43× and 7.4% less update delay than ASR and FS, respectively. This makes PIPETTE a more suitable system under drastic domain change with varying bandwidth conditions.

4.3.4 Performance Under Low Bandwidth

We further test PIPETTE under constrained BW (Fig. 4.12(b)) with a mean BW of $0.9Mbps$ on the videos with intense scene change as discussed in §4.3.3. We show the mAP and the model



(a) Accuracy and update delay

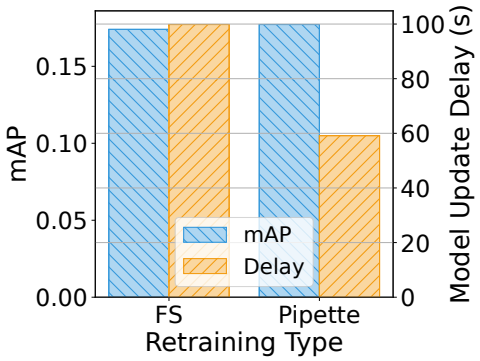
(b) Bandwidth pattern

FIGURE 4.12: (a) shows the accuracy and mean update delay of PIPETTE and FS under low bandwidth trace depicted in (b)

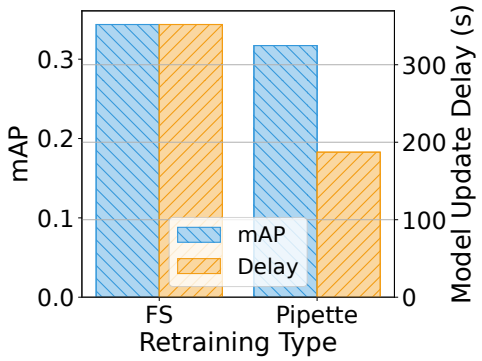
update delay of both PIPETTE and FS in Fig. 4.12(a). We discard ASR because its update delay was orders of magnitude higher, even in high BW (Fig. 4.9(a)). We observe from Fig. 4.12(a) that, compared to NR, both FS and PIPETTE improve mAP by 13.5%. However, PIPETTE achieves 12.7% lower delay in retrieving the updated model from the server. This happens as PIPETTE is designed to balance epochs and encoding frame rate. FS always uses an encoding frame rate of $1fps$ and a training epoch of 20. Therefore, during periods of extremely low BW, it fails to provide the trained model, compromising AV safety quickly. However, in such cases, PIPETTE reduces the encoding frame rate (sending less data) to minimize the update delay.

Special Cases: From all the videos in our dataset used for the results shown in the previous section, we identified two distinct cases to better understand PIPETTE's behavior and highlight specific use cases. Fig. 4.13 depicts the accuracy and model update delay for both cases using two videos each $\approx 1.11hr$ long. The first case shows a good BW condition, with a countryside scene where vehicles appear sparsely. In such cases, as shown in Fig. 4.13(a), in contrast to FS, PIPETTE significantly reduces the update delay by 68.8% while still providing a comparable accuracy. This happens because, regardless of the vehicles' presence, FS samples unnecessary frames, whereas the confidence-score-based sampling in PIPETTE identifies and discards frames with no objects. Thus, PIPETTE provides a faster model update.

In the second scenario, the BW is constrained in a challenging scene, i.e., a scene with a varying number of objects and low visibility. PIPETTE provides 1.8× faster model updates by selecting a lower encoding frame rate than FS due to poor BW. However, unlike good network characteristics in §4.3.3, due to the higher round-trip-time, it uses fewer training epochs to minimize total delay. This drops the accuracy by 8.4% compared to FS. A possible approach to address this corner case is to perform local fine-tuning or reuse a previously stored model. We intend to explore such optimizations in the future.



(a) Good BW having countryside scene with less traffic around



(b) Low BW with scenario having a challenging scene

FIGURE 4.13: Accuracy and update delay under two example cases.

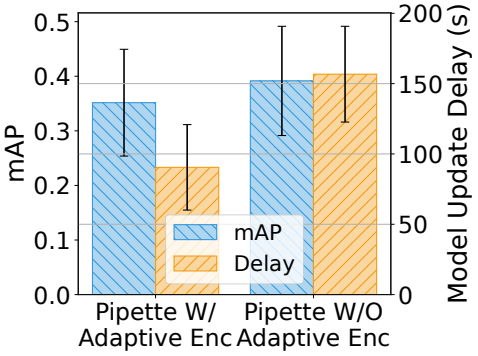


FIGURE 4.14: mAP and model update delay of PIPETTE with and without adaptive encoding frame rate selection before streaming.

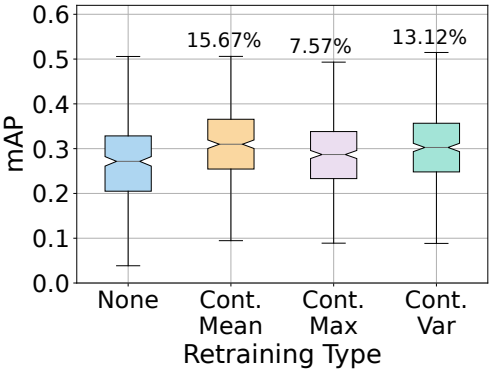


FIGURE 4.15: Sensitivity to different metrics (mean, maximum, and variance of confidence scores) to get $C(I_t)$ in for each frame Eq. (4.1).

4.3.5 Ablation Study and Sensitivity Tests

Need for Adaptive Encoding Frame Rate: An important part of PIPETTE is its adaptive encoding frame rate, which helps it adapt to the fluctuating network bandwidth (BW). To emphasize its contribution, we perform an ablation study by completely removing the adaptive encoding and fixing it at 5fps. Thus, PIPETTE has only one stage, which is confidence score-based frame selection. We report the result in Fig. 4.14. We note that removing the adaptive encoding improves the mAP by 11.34% while increasing the update delay by 70%. This implies that although adaptive encoding reduces the accuracy, it helps limit the model update delay.

Impact of different metrics in $C(I_t)$: In Fig. 4.15, we justify our design choice of using the mean of confidence scores of all the detections within a frame in Eq. (4.1) to calculate $C(I_t)$ compared to maximum and variance of confidence scores as used in works like [190] and [191]. We note that using the mean (Cont. Mean) for frame selection in continuous training is superior to using maximum (Cont. Max) and variance (Cont. Var). Using maximum has the least benefit, improving the mAP by 7.57%. Although Variance seems promising, it misses frames.

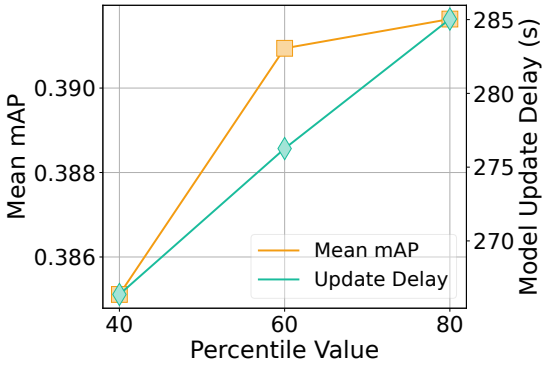


FIGURE 4.16: Sensitivity to percentile value of δ in Eq. (4.5).

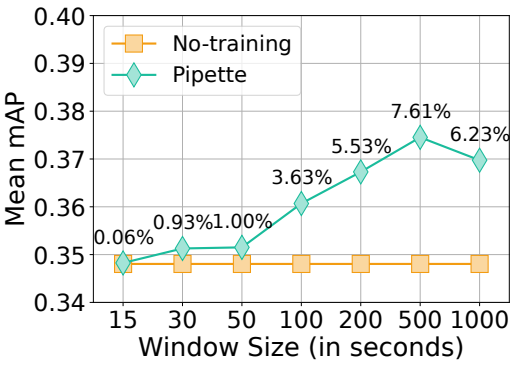


FIGURE 4.17: Effect of varying training window size.

To exemplify, consider two cases with four confidence scores $A = \{0.84, 0.88, 0.84, 0.88\}$ and $B = \{0.44, 0.48, 0.44, 0.48\}$. Here, both A and B have low variance, implying good detection, therefore, both will be discarded. But as chosen by mean but not by variance, B should be considered as a difficult sample, as all its detections have poor confidence scores.

Impact of percentile value in threshold: In Eq. (4.4) and Eq. (4.5), we select only those Δ values which are greater than the threshold δ that uses p^{th} percentile. We perform a sensitivity study for three different percentile values (shown in Fig. 4.16). We observe that increasing the percentile increases the mAP and the model update delay. We select 60 as the optimal percentile, as it offers the best tradeoff between mAP and update delay.

Impact of retraining window size: PIPETTE uses a fixed retraining window size. Changing the window size has a significant impact on the accuracy of the retrained model. We report in Fig. 4.17 the mean improvement in model accuracy (compared to no-training) after training when using different window sizes in PIPETTE. We note that increasing the size increases the accuracy (mAP) until the size reaches $500s$, after which the accuracy drops. Therefore, PIPETTE uses a window size of $500s$. Note that instead of using fixed time periods as retraining triggers, the amount of drift can be utilized as a trigger as well. PIPETTE's strategy can generalize to such triggers. We leave such evaluations for future consideration.

4.4 Related Works

Data Ingestion Cost in Video Analytics: The cost of data ingestion for video analytics has been extensively studied in the systems community. However, most works focus only on applications that rely on inference. Reducto [6] and SmartFilter [118] use adaptive frame filtering to prune similar frames. DDS [7], AccMPEG [4], and MRIM [151] encode video with varying spatial video quality. AdaMask [5] removes the background, keeping only the region of interest.

CloudSeg [144] utilizes super-resolution to upscale a low-resolution stream sent from the camera. However, none of these works addresses the high network cost associated with model training.

Continual Learning: There are a myriad of works using continual learning to handle the problem of data drift in DNNs [10, 157, 158, 166, 171, 172, 176, 178, 192, 193, 194, 195, 196, 197, 198]. RECL, Mocha, and Gecko [157, 171, 176] propose reusing a model from a large model zoo, which is continually updated. This is complementary to our work. Ekya [158], AdaInf [194], and ElasticDNN [195] investigate balancing inference and model retraining on the GPU by designing an appropriate job scheduler. DACAPO [193] designs a specialized hardware for concurrent execution of inference, labeling, and training. The works [10, 172] retrain the model under adverse conditions and across different perspectives. Momento [199] and LiveNAS [200] apply continuous training during video streaming for completion-time prediction and video upscaling, respectively. Reference [192] leverages camera correlation to train and deploy a single model at the edge, serving multiple cameras with similar visual scenes. RILOD [196] incrementally trains a model whenever it encounters a new class. None of these works examines adaptation to varying network characteristics.

Frame Sampling: A few of the continual learning works propose some form of frame sampling to limit the number of frames for retraining. Works like [166, 176, 178] use content-based adaptive sampling. EAMU [10] is closest to ours. It picks samples using confidence scores, then performs frame differencing to discard similar frames when BW is limited. However, they only experimented on stationary traffic cameras. For AV, the frame differencing struggles because the entire frame is mobile. The prior work [191] uses the variance of the confidence scores to select the hard samples. PatchLine [177] and EdgeSync [201] leverage the entropy of predictions to pick relevant samples. Momento [199] creates batches and then discards high-density batches, i.e., batches with more similar samples. AdaptAV [198] employs various sampling strategies, including adaptive, density-based, and event-based sampling. SBAD [190] depends on the maximum of the confidence scores of the predictions to pick frames for retraining. It only selects high-confidence-score frames, which we find ineffective for continuous retraining. All of these works overlook the impact of the network dynamicity on their sampling and retraining decisions.

4.5 Discussions & Limitations

Model Reuse: There are works such as Mocha [171], Gecko [176], and RECL [157] that propose reusing an optimal model from a model zoo. The zoo is updated in the background. This significantly reduces the model update time by hiding the retraining from the model update pipeline. PIPETTE can complement such works during their retraining sessions, which still require streaming frames.

Scene Dynamicity-based Sampling Rate Selection: PIPETTE dynamically selects the encoding frame rate based on available network bandwidth and then samples using confidence scores. Some prior work [166, 176, 178] analyzes consecutive frames to estimate the vehicle's speed. Then change the frame sampling rate accordingly. This technique varies the encoding frame rate based on the vehicle's speed; when the vehicle is stationary, samples are taken at a lower rate than when it is mobile. PIPETTE does not use such a strategy. However, this can complement PIPETTE in saving even more bandwidth. The current work does not explore its feasibility.

Assuming No GPU Contention on Server: While training, our setup assumes no contention on the server. However, an edge server handles multiple requests concurrently from different AVs. Therefore, the same GPU resource is typically shared among multiple retraining instances [194]. In such a case, the annotation time will no longer be linear. Moreover, the training time depends on the training scheduling policy. Currently, PIPETTE does not handle this non-linearity introduced by the contending retraining sessions. However, using the update similar to the accuracy profiler curve discussed in §4.2.4 PIPETTE can adapt to such cases.

4.6 Conclusion

In this work, we propose PIPETTE, a system that leverages an adaptive frame-sampling strategy for continual learning in autonomous vehicles (AVs). This enables AVs to adapt to unseen adverse environments by quickly having an updated model available through server-based training. Since such model retrainings require streaming the training frames from the AV to the server. It requires an appropriate sampling and encoding frame rate that adapts to current network characteristics. PIPETTE adjusts the encoding frame rate to the available bandwidth while also using the student model's confidence scores to identify frames most useful for retraining. It also adjusts the number of training epochs to minimize update delay. Compared to the baseline strategies, PIPETTE achieves the best tradeoff between the accuracy of retrained models and the delay in obtaining the updated model from the server in extreme conditions.

Chapter 5

Conclusion and Future Works

5.1 Conclusion

In this thesis, we discussed our work on real-time traffic surveillance, live video streaming, and cloud-assisted autonomous driving. All these video applications suffer from the poor, inconsistent bandwidth and variable latency of the current wireless networks. This thesis proposes novel lightweight data filtering and content-aware data allocation to improve and better adapt these modern video applications to the current wireless network infrastructure.

Specifically, in the surveillance setting, we propose a metadata-driven filtering approach that can run in real-time, even on the camera, to detect regions of video containing moving objects. Our pruning algorithm leverages the observation that the bitrates of these regions strongly correlate with the number of moving objects.

To improve the quality of experience in live video streaming, we propose using multiple devices with internet connections to collaborate synergistically and receive different spatial regions of the video from the sender on different devices. We further design a content-aware scheduler to allocate different spatial portions to different paths.

Finally, we address the high model update delay associated with retraining jobs in cloud-assisted autonomous driving. We design a lightweight frame selection strategy to pick only the frame that provides meaningful improvement after training. Our strategy adapts to network bandwidth to minimize update delay while ensuring superior post-training accuracy.

For each application, we implement our proposed adaptive filtering and content-aware solutions on inexpensive edge devices and demonstrate that they run in real-time and perform well in practice.

5.2 Future Works

In all the applications of this thesis, we primarily emphasized bandwidth and latency. However, there are other interesting aspects to explore, such as energy aspects of multi-device streaming, privacy in video analytics, and local on-device model training in AVs. We now briefly discuss these potential research ideas.

5.2.1 Designing Energy-aware Scheduler With Better BW Estimation

In live streaming, we observe that our bandwidth estimation technique provide inaccurate results. One way to address inaccurate bandwidth estimation is to incorporate feedback from the primary device. Upon receiving a complete video segment, the receiver can report the completion time and other QoE metrics as part of the feedback, which the streamer can use to offset bandwidth and scheduling inaccuracies. Such feedback has been shown to work well in works such as Converge [40] and XLINK [48]. Moreover, for multi-device collaboration in the live streaming work, introductino of the second device comes at the cost of increased energy consumption. It would be interesting to study this implication and the possibility to incorporate this energy constraint into an appropriate utility function. This an make the overall system more energy efficient.

5.2.2 On-device Training to Remove Cloud-assistance

A major problem associated with cloud-assisted model training in AVs is the high network costs and long update delays. One way to address this is to train locally on the vehicle itself. However, such training on AV is a non-trivial problem because the GPU compute is shared. During training, the GPU is shared between the inference task and the training job. This hampers the AV's perception rate, compromising its safety. Since AVs now come with a variety of processors, including a neural accelerator, inference can be offloaded to these accelerators, with the GPU largely used for training. However, the modern object detection models are less compatible with the neural processors and therefore frequently fall back to the GPU, making it essential to consider such constraints while training.

5.2.3 Privacy-aware Video Analytics

In current deployments, the camera streams the captured video directly to remote cloud/edge servers. This has the associated risk of leaking private information. For instance, in a locality, if there is only one yellow car with a black sticker. Given a video, someone can easily identify

the car's owner. Therefore, the video should be encoded in such a way before sending it to the server so that it hides distinguishable features of vehicles or people (as in [202]). One possible solution is to design a neural codec-based video compression technique that can mask vulnerable features in frames. However, this transformation should be reversible, allowing recovery of the original video if needed, given the encoding mask. Auto-encoders are a good candidate for such an application.

5.2.4 Utilizing Cross-correlation Across Cameras

In real-life deployments, multiple camera points at the same scene, therefore have overlapping field of view. This provides an opportunity to use cross-correlation among the cameras as shown by [8, 9]. Such correlation across cameras can be utilized for the reidentification and tracking of vehicles. This also opens the possibility of more bandwidth saving by sending only the non-overlapping tiles of a camera because the overlapping objects might already be detected in another camera's view.

References

[1] Creative Commons License. What is machine learning andrew ng. https://www.youtube.com/watch?v=yeII6KWenWU, Dec 27, 2021.

[2] 5G Experience in India: From Rollout to Real-World Impact | Opensignal. URL https://insights.opensignal.com/2025/12/5g-experience-in-india-from-rollout-to-real-world-impact/dt.

[3] TRAI Annual Report 1&2. URL https://www.trai.gov.in/sites/default/files/2026-01/Annual_Report_06012026.pdf.

[4] Kuntai Du, Qizheng Zhang, Anton Arapin, Haodong Wang, Zhengxu Xia, and Junchen Jiang. Accmpeg: Optimizing video encoding for accurate video analytics. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 450–466, 2022. URL https://proceedings.mlsys.org/paper_files/paper/2022/file/853f7b3615411c82a2ae439ab8c4c96e-Paper.pdf.

[5] Shengzhong Liu, Tianshi Wang, Jinyang Li, Dachun Sun, Mani Srivastava, and Tarek Abdelzaher. AdaMask: Enabling Machine-Centric Video Streaming with Adaptive Frame Masking for DNN Inference Offloading. In *Proceedings of the 30th ACM MM*, pages 3035–3044, Lisboa Portugal, October 2022. ACM. ISBN 978-1-4503-9203-7. doi: 10.1145/3503161.3548033. URL https://dl.acm.org/doi/10.1145/3503161.3548033.

[6] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. Reducto: On-Camera Filtering for Resource-Efficient Real-Time Video Analytics. In *Proceedings of the ACM SIGCOMM*, pages 359–376, Virtual Event USA, July 2020. ACM. ISBN 978-1-4503-7955-7. doi: 10.1145/3387514.3405874. URL https://dl.acm.org/doi/10.1145/3387514.3405874.

[7] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. Server-Driven Video Streaming for Deep Learning Inference. In *Proceedings of the ACM SIGCOMM*, pages 557–570, Virtual Event USA, July 2020. ACM. ISBN 978-1-4503-7955-7. doi: 10.1145/3387514.3405887. URL https://dl.acm.org/doi/10.1145/3387514.3405887.

[8] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable adaptation of video analytics. In *2018 Conference of the ACM Special Interest Group on Data Communication*, page 253–266, 2018. doi: 10.1145/3230543.3230574.

[9] Hongpeng Guo, Shuochao Yao, Zhe Yang, Qian Zhou, and Klara Nahrstedt. Crossroi: Cross-camera region of interest optimization for efficient real time video analytics at scale. In *12th ACM Multimedia Systems Conference*, page 186–199, 2021. ISBN 9781450384346. doi: 10.1145/3458305.3463381.

[10] Yuxin Kong, Peng Yang, and Yan Cheng. Edge-Assisted On-Device Model Update for Video Analytics in Adverse Environments. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9051–9060, Ottawa ON Canada, October 2023. ACM. ISBN 9798400701085. doi: 10.1145/3581783.3612585. URL `https://dl.acm.org/doi/10.1145/3581783.3612585`.

[11] Sohee Park, Arani Bhattacharya, Zhibo Yang, Samir R. Das, and Dimitris Samaras. *Mosaic* : Advancing User Quality of Experience in 360-Degree Video Streaming With Machine Learning. *IEEE Transactions on Network and Service Management*, 18(1):1000–1015, March 2021. doi: 10.1109/TNSM.2021.3053183.

[12] Sohee Park, Minh Hoai, Arani Bhattacharya, and Samir R. Das. Adaptive streaming of 360-degree videos with reinforcement learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1839–1848, January 2021.

[13] Arvind Narayanan, Eman Ramadan, Jason Carpenter, Qingxu Liu, Yu Liu, Feng Qian, and Zhi-Li Zhang. A first look at commercial 5g performance on smartphones. In *Proceedings of The Web Conference 2020*, WWW '20, page 894–905, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370233. doi: 10.1145/3366423.3380169. URL `https://doi.org/10.1145/3366423.3380169`.

[14] Md Rokebul Islam, Nafis Ibn Shahid, Dewan Tanzim ul Karim, Abdullah Al Mamun, and Md Khalilur Rhaman. An efficient algorithm for detecting traffic congestion and a framework for smart traffic control system. In *2016 18th International Conference on Advanced Communication Technology*. IEEE, 2016. doi: 10.1109/ICACT.2016.7423566.

[15] Chacha Chen, Hua Wei, Nan Xu, Guanjie Zheng, Ming Yang, Yuanhao Xiong, Kai Xu, and Zhenhui Li. Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. *AAAI Conference on Artificial Intelligence*, 34(04): 3414–3421, Apr. 2020. doi: 10.1609/aaai.v34i04.5744.

[16] K. K. Santhosh, D. P. Dogra, and P. P. Roy. Anomaly detection in road traffic using visual surveillance: A survey. *ACM Computing Surveys*, 53(6), Dec 2020. doi: 10.1145/3417989.

[17] Number of devices and connections per person worldwide 2023. URL https://www.statista.com/statistics/1190270/number-of-devices-and-connections-per-person-worldwide/.

[18] Shubham Chaudhary, Navneet Mishra, Keshav Gambhir, Tanmay Rajore, Arani Bhattacharya, and Mukulika Maity. Compact: Content-aware multipath live video streaming for online classes using video tiles. In *Proceedings of the 16th ACM MMSys conference*, MMSys '25, page 201–213, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714672. doi: 10.1145/3712676.3714451. URL https://doi.org/10.1145/3712676.3714451.

[19] Shubham Chaudhary, Aryan Taneja, Anjali Singh, Purbasha Roy, Sohum Sikdar, Mukulika Maity, and Arani Bhattacharya. TileClipper: Lightweight selection of regions of interest from videos for traffic surveillance. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 967–984, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-41-0. URL https://www.usenix.org/conference/atc24/presentation/chaudhary.

[20] Xinyue Chen, Si Chen, Xu Wang, and Yun Huang. "I was afraid, but now I enjoy being a streamer!": Understanding the Challenges and Prospects of Using Live Streaming for Online Education. *Proc. ACM Hum.-Comput. Interact.*, 4(CSCW3):237:1–237:32, January 2021. doi: 10.1145/3432936.

[21] Online Learning Statistics: The Ultimate List in 2024 | Devlin Peck, 2024-11-14.

[22] Genevieve Carlton Ph.D. 2024 Online Learning Statistics, June 2024. Section: Online Colleges.

[23] Alex Yelenevych. The future of edtech. https://www.forbes.com/sites/forbesbusinesscouncil/2022/12/26/t future-of-edtech/?sh=4ed280296c2f. Published on Dec 26, 2022; Accessed on Apr 12, 2024.

[24] 50 Online Education Statistics: 2024 Data on Higher Learning & Corporate Training, June 2020.

[25] The Evolving Landscape of Students' Mobile Learning Practices in Higher Education, 2024-11-14.

[26] Travis Faas, Lynn Dombrowski, Alyson Young, and Andrew D. Miller. Watch Me Code: Programming Mentorship Communities on Twitch.tv. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–18, November 2018. doi: 10.1145/3274319.

[27] Di (Laura) Chen, Dustin Freeman, and Ravin Balakrishnan. Integrating Multimedia Tools to Enrich Interactions in Live Streaming for Language Learning. In *Proceedings of the*

2019 CHI Conference on Human Factors in Computing Systems, CHI '19, pages 1–14, New York, NY, USA, May 2019. Association for Computing Machinery. doi: 10.1145/3290605.3300668.

[28] Man Wu and Qin Gao. Using live video streaming in online tutoring: Exploring factors affecting social interaction. *International Journal of Human-Computer Interaction*, 36(10): 964–977, 2020. doi: 10.1080/10447318.2019.1706288.

[29] Pranit Sarda. Why youtube still rules the edtech roost. https://www.forbesindia.com/article/edtech-special/why-youtube-still-rules-the-edtech-roost/57761/1, 2020. Published on Feb 18, 2020; Accessed on Apr 12, 2024.

[30] Research Article, Arbaz Khan, Mubashir Saeed, Muhammad Anwar, and Lareb Kanwal. Journal of Mass Communication & Journalism Unleashing the Potential: A Study of the Effectiveness and Impact of YouTube Educational Content on Student Learning Outcomes. *Journal of Mass Communication and Journalism*, 13:05, 2023, September 2023. doi: 10.37421/2165-7912.2023.13.538.

[31] Live streaming latency - YouTube Help. https://support.google.com/youtube/answer/7444635, 2024-04-04. Accessed on April 4, 2024.

[32] Shivang Aggarwal, Moinak Ghoshal, Piyali Banerjee, Dimitrios Koutsonikolas, and Joerg Widmer. 802.11ad in Smartphones: Energy Efficiency, Spatial Reuse, and Impact on Applications. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, Vancouver, BC, Canada, May 2021. IEEE. doi: 10.1109/INFOCOM42981.2021.9488763.

[33] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. Understanding Operational 5G: A First Measurement Study on Its Coverage, Performance and Energy Consumption. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 479–494, Virtual Event USA, July 2020. ACM. doi: 10.1145/3387514.3405882.

[34] Xinjie Yuan, Mingzhou Wu, Zhi Wang, Yifei Zhu, Ming Ma, Junjian Guo, Zhi-Li Zhang, and Wenwu Zhu. Understanding 5G performance for real-world services: a content provider's perspective. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 101–113, Amsterdam Netherlands, August 2022. ACM. doi: 10.1145/3544216.3544219.

[35] Yunze Zeng, Parth H. Pathak, and Prasant Mohapatra. A first look at 802.11ac in action: Energy efficiency and interference characterization. In *2014 IFIP Networking Conference*, pages 1–9, Trondheim, Norway, June 2014. IEEE. doi: 10.1109/IFIPNetworking.2014.6857103.

[36] Jia He, Mostafa Ammar, and Ellen Zegura. A Measurement-Derived Functional Model for the Interaction Between Congestion Control and QoE in Video Conferencing. In Anna Brunstrom, Marcel Flores, and Marco Fiore, editors, *Passive and Active Measurement*, pages 129–159, Cham, 2023. Springer Nature Switzerland. doi: 10.1007/978-3-031-28486-1_7.

[37] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. Measuring the performance and network utilization of popular video conferencing applications. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 229–244, Virtual Event, November 2021. ACM. doi: 10.1145/3487552.3487842.

[38] Hyoyoung Lim, Jinsung Lee, Jongyun Lee, Sandesh Dhawaskar Sathyanarayana, Junseon Kim, Anh Nguyen, Kwang Taik Kim, Youngbin Im, Mung Chiang, Dirk Grunwald, Kyunghan Lee, and Sangtae Ha. An Empirical Study of 5G: Effect of Edge on Transport Protocol and Application Performance. *IEEE Transactions on Mobile Computing*, 23(4):3172–3186, April 2024. doi: 10.1109/TMC.2023.3274708.

[39] Ahmad Hassan, Arvind Narayanan, Anlan Zhang, Wei Ye, Ruiyang Zhu, Shuowei Jin, Jason Carpenter, Z. Morley Mao, Feng Qian, and Zhi-Li Zhang. Vivisecting mobility management in 5G cellular networks. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 86–100, Amsterdam Netherlands, August 2022. ACM. doi: 10.1145/3544216.3544217.

[40] Sandesh Dhawaskar Sathyanarayana, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. Converge: QoE-driven Multipath Video Conferencing over WebRTC. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 637–653, New York NY USA, September 2023. ACM. ISBN 9798400702365. doi: 10.1145/3603269.3604822. URL `https://dl.acm.org/doi/10.1145/3603269.3604822`.

[41] Haiping Wang, Zhenhua Yu, Ruixiao Zhang, Siping Tao, Hebin Yu, and Shu Shi. TwinStar: A Practical Multi-path Transmission Framework for Ultra-Low Latency Video Delivery. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9234–9242, Ottawa ON Canada, October 2023. ACM. doi: 10.1145/3581783.3613443.

[42] Jinlong E, Lin He, Zongyi Zhao, Yachen Wang, Gonglong Chen, and Wei Chen. AggDeliv: Aggregating Multiple Wireless Links for Efficient Mobile Live Video Delivery. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 1173–1180, May 2024. doi: 10.1109/INFOCOM52122.2024.10621184.

[43] Jiyan Wu, Chau Yuen, Ming Wang, and Junliang Chen. Content-Aware Concurrent Multipath Transfer for High-Definition Video Streaming over Heterogeneous Wireless Networks. *IEEE Transactions on Parallel and Distributed Systems*, 27(3):710–723, March 2016. doi: 10.1109/TPDS.2015.2416736.

[44] Jiyan Wu, Bo Cheng, Chau Yuen, Yanlei Shang, and Junliang Chen. Distortion-Aware Concurrent Multipath Transfer for Mobile Video Streaming in Heterogeneous Wireless Networks. *IEEE Transactions on Mobile Computing*, 14(4):688–701, April 2015. doi: 10.1109/TMC.2014.2334592.

[45] Jiyan Wu, Bo Cheng, and Ming Wang. Improving Multipath Video Transmission With Raptor Codes in Heterogeneous Wireless Networks. *IEEE Transactions on Multimedia*, 20 (2):457–472, February 2018. doi: 10.1109/TMM.2017.2741425.

[46] Changqiao Xu, Zhuofeng Li, Jinglin Li, Hongke Zhang, and Gabriel-Miro Muntean. Cross-Layer Fairness-Driven Concurrent Multipath Video Delivery Over Heterogeneous Wireless Networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(7):1175–1189, July 2015. doi: 10.1109/TCSVT.2014.2376138.

[47] Chengke Wang, Hao Wang, Feng Qian, Kai Zheng, Chenglu Wang, Fangzhu Mao, Xingmin Guo, and Chenren Xu. Experience: A Three-Year Retrospective of Large-scale Multipath Transport Deployment for Mobile Applications. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–15, Madrid Spain, July 2023. ACM. doi: 10.1145/3570361.3592506.

[48] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, Qing An, Hai Hong, Hongqiang Harry Liu, and Ming Zhang. XLINK: QoE-driven multi-path QUIC transport in large-scale video services. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 418–432, Virtual Event USA, August 2021. ACM. ISBN 978-1-4503-8383-7. doi: 10.1145/3452296.3472893. URL `https://dl.acm.org/doi/10.1145/3452296.3472893`.

[49] Yunzhe Ni, Feng Qian, Taide Liu, Yihua Cheng, Zhiyao Ma, Jing Wang, Zhongfeng Wang, Gang Huang, Xuanzhe Liu, and Chenren Xu. {POLYCORN}: Data-driven cross-layer multipath networking for high-speed railway through composable schedulerlets. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1325–1340, 2023.

[50] Petroc Taylor. Average number devices and connections per person worldwide in 2018 and 2023. https://www.statista.com/statistics/1190270/number-of-devices-and-connections-per-person-worldwide/, Jan 19, 2023. Accessed on Nov 22, 2024.

[51] Kisten Schmitt Alexus Bazen. Cell phone statistics 2024. https://www.consumeraffairs.com/cell_phones/cell-phone-statistics.html, 2023. Published on 28 September 2023; Accessed on April 8, 2024.

[52] Multiple device ownership means more smartphone usage. https://cdne.kantar.com/north-america/inspiration/technology/multiple-device-ownership-means-more-smartphone-usage, 2024. Accessed on April 8, 2024.

[53] Teens and Internet, Device Access Fact Sheet. https://www.pewresearch.org/internet/fact-sheet/teens-and-internet-device-access-fact-sheet/, January 2024. Accessed on April 8, 2024.

[54] Katherine Schaeffer. Most U.S. teens who use cellphones do it to pass time, connect with others, learn new things. https://www.pewresearch.org/short-reads/2019/08/23/most-u-s-teens-who-use-cellphones-do-it-to-pass-time-connect-with-others-learn-new-things/, August 2019. Accessed on April 8, 2024.

[55] Xiao Zhu, Jiachen Sun, Xumiao Zhang, Y Ethan Guo, Feng Qian, and Z Morley Mao. Mpbond: efficient network-level collaboration among personal mobile devices. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 364–376, 2020.

[56] Lorenzo Keller, Anh Le, Blerim Cici, Hulya Seferoglu, Christina Fragouli, and Athina Markopoulou. MicroCast: cooperative video streaming on smartphones. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 57–70, Low Wood Bay Lake District UK, June 2012. ACM. doi: 10.1145/2307636.2307643.

[57] Shuowei Jin, Ruiyang Zhu, Ahmad Hassan, Xiao Zhu, Xumiao Zhang, Z. Morley Mao, Feng Qian, and Zhi-Li Zhang. Oasis: Collaborative neural-enhanced mobile video streaming. In *Proceedings of the 15th ACM Multimedia Systems Conference*, MMSys '24, page 45–55, New York, NY, USA, 2024. Association for Computing Machinery. doi: 10.1145/3625468.3647610.

[58] Costin Raiciu, Mark Handley, and Damon Wischik. Coupled congestion control for multipath transport protocols. Technical report, 2011.

[59] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath TCP. March 2011. URL `https://www.usenix.org/conference/nsdi11/design-implementation-and-evaluation-congestion-control-multipath-tcp`.

[60] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP.

[61] Quentin De Coninck. The packet number space debate in multipath QUIC. *ACM SIGCOMM Computer Communication Review*, 52(3):2–9, July 2022. doi: 10.1145/3561954.3561956.

[62] Quentin De Coninck and Olivier Bonaventure. Multipath QUIC: Design and Evaluation. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 160–166, Incheon Republic of Korea, November 2017. ACM. doi: 10.1145/3143361.3143370.

[63] Quentin De Coninck and Olivier Bonaventure. Multipath QUIC: Design and Evaluation. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 160–166, Incheon Republic of Korea, November 2017. ACM. ISBN 978-1-4503-5422-6. doi: 10.1145/3143361.3143370. URL `https://dl.acm.org/doi/10.1145/3143361.3143370`.

[64] Varun Singh, Saba Ahsan, and Jörg Ott. MPRTP: multipath considerations for real-time media. In *Proceedings of the 4th ACM Multimedia Systems Conference*, pages 190–201, Oslo Norway, February 2013. ACM. doi: 10.1145/2483977.2484002.

[65] René F Kizilcec, Kathryn Papadopoulos, and Lalida Sritanyaratana. Showing face in video instruction: effects on information retention, visual attention, and affect. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 2095–2102, 2014. doi: 10.1145/2556288.2557207.

[66] Jiahui Wang, Pavlo Antonenko, and Kara Dawson. Does visual attention to the instructor in online video affect learning and learner perceptions? an eye-tracking analysis. *Computers & Education*, 146:103779, 2020. doi: 10.1016/j.compedu.2019.103779.

[67] Margot Van Wermeskerken, Susanna Ravensbergen, and Tamara Van Gog. Effects of instructor presence in video modeling examples on attention and learning. *Computers in Human Behavior*, 89:430–438, 2018. doi: 10.1016/j.chb.2017.11.038.

[68] Omer F. Aladag, Deniz Ugur, Mehmet N. Akcay, and Ali C. Begen. Content-Aware Playback Speed Control for Low-Latency Live Streaming of Sports. In *Proceedings of the 12th ACM Multimedia Systems Conference*, pages 344–349, Istanbul Turkey, June 2021. ACM. doi: 10.1145/3458305.3478437.

[69] Chen-Tai Kao, Yen-Ting Liu, and Alexander Hsu. Speeda: adaptive speed-up for lecture videos. In *Proceedings of the adjunct publication of the 27th annual ACM symposium on User interface software and technology - UIST'14 Adjunct*, pages 97–98, Honolulu, Hawaii, USA, 2014. ACM Press. doi: 10.1145/2658779.2658794.

[70] Irshad Abibouraguimane, Kakeru Hagihara, Keita Higuchi, Yuta Itoh, Yoichi Sato, Tetsu Hayashida, and Maki Sugimoto. CoSummary: adaptive fast-forwarding for surgical videos by detecting collaborative scenes using hand regions and gaze positions. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pages 580–590, Marina del Ray California, March 2019. ACM. doi: 10.1145/3301275.3302284.

[71] Kai-Yin Cheng, Sheng-Jie Luo, Bing-Yu Chen, and Hao-Hua Chu. SmartPlayer: user-centric video fast-forwarding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 789–798, Boston MA USA, April 2009. ACM. doi: 10.1145/1518701.1518823.

[72] Jiangkai Wu, Yu Guan, Qi Mao, Yong Cui, Zongming Guo, and Xinggong Zhang. Zgaming: Zero-latency 3d cloud gaming by image prediction. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 710–723, New York, NY, USA, 2023. Association for Computing Machinery. doi: 10.1145/3603269.3604819.

[73] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, December 2012. doi: 10.1109/TCSVT.2012.2221191.

[74] Swetank Kumar Saha, Shivang Aggarwal, Rohan Pathak, Dimitrios Koutsonikolas, and Joerg Widmer. MuSher: An Agile Multipath-TCP Scheduler for Dual-Band 802.11ad/ac Wireless LANs. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, Los Cabos Mexico, October 2019. ACM. doi: 10.1145/3300061.3345435.

[75] Preethi Natarajan, Professor Paul D. Amer, Jonathan Leighton, and Fred Baker. Using SCTP as a Transport Layer Protocol for HTTP. Internet Draft draft-natarajan-http-over-sctp-02, Internet Engineering Task Force, July 2009. URL `https://datatracker.ietf.org/doc/draft-natarajan-http-over-sctp-00`. Num Pages: 13.

[76] Preethi Natarajan, Professor Paul D. Amer, Jonathan Leighton, and Fred Baker. Using SCTP as a Transport Layer Protocol for HTTP. Internet Draft draft-natarajan-http-over-sctp-02, Internet Engineering Task Force, July 2009. Num Pages: 13.

[77] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, Santa Clara, CA, July 2015. ISBN 978-1-931971-225. URL `https://www.usenix.org/conference/atc15/technical-session/presentation/netravali`.

[78] Jingning Han, Bohan Li, Debargha Mukherjee, Ching-Han Chiang, Adrian Grange, Cheng Chen, Hui Su, Sarah Parker, Sai Deng, Urvang Joshi, Yue Chen, Yunqing Wang, Paul Wilkins, Yaowu Xu, and James Bankoski. A Technical Overview of AV1. *Proceedings of the IEEE*, 109(9):1435–1462, September 2021. doi: 10.1109/JPROC.2021.3058584.

[79] Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J. Sullivan, and Jens-Rainer Ohm. Overview of the Versatile Video Coding (VVC) Standard and its Applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10): 3736–3764, October 2021. doi: 10.1109/TCSVT.2021.3101953.

[80] Praveen Kumar Yadav and Wei Tsang Ooi. Tile rate allocation for 360-degree tiled adaptive video streaming. In *28th ACM International Conference on Multimedia*, page 3724–3733, 2020. doi: 10.1145/3394171.3413550.

[81] Jeroen Van Der Hooft, Maria Torres Vega, Stefano Petrangeli, Tim Wauters, and Filip De Turck. Tile-based Adaptive Streaming for Virtual Reality Video. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 15(4):1–24, November 2019. doi: 10.1145/3362101.

[82] Jangwoo Son, Dongmin Jang, and Eun-Seok Ryu. Implementing Motion-Constrained Tile and Viewport Extraction for VR Streaming. In *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 61–66, Amsterdam Netherlands, June 2018. ACM. doi: 10.1145/3210445.3210455.

[83] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210, Los Angeles CA USA, August 2017. ACM. doi: 10.1145/3098822.3098843.

[84] Keshav Gambhir, Tanmay Rajore, Shubham Chaudhary, Taral Jain, Avishi Gupta, Mukulika Maity, and Arani Bhattacharya. Native: Network aggregation based tiled live video streaming. In *2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 219–221, 2023. doi: 10.1109/COMSNETS56262.2023.10041371.

[85] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 97–108, Nice France, December 2012. ACM. doi: 10.1145/2413176.2413189.

[86] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, London United Kingdom, August 2015. ACM. doi: 10.1145/2785956.2787486.

[87] Sohee Park, Arani Bhattacharya, Zhibo Yang, Samir R. Das, and Dimitris Samaras. Mosaic: Advancing user quality of experience in 360-degree video streaming with machine learning.

IEEE Transactions on Network and Service Management, 18(1):1000–1015, 2021. doi: 10.1109/TNSM.2021.3053183.

[88] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: auto-tuning video ABR algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 44–58, Budapest Hungary, August 2018. ACM. doi: 10.1145/3230543.3230558.

[89] Marko Viitanen, Ari Koivula, Ari Lemmetti, Arttu Ylä-Outinen, Jarno Vanne, and Timo D Hämäläinen. Kvazaar: open-source hevc/h. 265 encoder. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 1179–1182, 2016.

[90] Ffmpeg. https://www.ffmpeg.org/. [Online; accessed Sept-2021].

[91] Jean Le Feuvre, Cyril Concolato, and Jean-Claude Moissinac. Gpac: open source multimedia framework. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 1009–1012, 2007.

[92] Understanding and Choosing The Right Frame Rates for You. https://riverside.fm/blog/frame-rate-guide, https://riverside.fm/blog/frame-rate-guide, 2024-04-08. Accessed on April 8, 2024.

[93] javacv/samples/DeepLearningFaceDetection.java at master · bytedeco/javacv. https://github.com/bytedeco/javacv/blob/master/samples, 2024-04-01.

[94] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 183–196, New York, NY, USA, 2017. Association for Computing Machinery. doi: 10.1145/3098822.3098842.

[95] Changqiao Xu, Tianjiao Liu, Jianfeng Guan, Hongke Zhang, and Gabriel-Miro Muntean. CMT-QA: Quality-Aware Adaptive Concurrent Multipath Data Transfer in Heterogeneous Wireless Networks. *IEEE Transactions on Mobile Computing*, 12(11):2193–2205, November 2013. ISSN 1558-0660. doi: 10.1109/TMC.2012.189. Conference Name: IEEE Transactions on Mobile Computing.

[96] J.R. Iyengar, P.D. Amer, and R. Stewart. Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths. *IEEE/ACM Transactions on Networking*, 14(5):951–964, October 2006. doi: 10.1109/TNET.2006.882843.

[97] T. Daniel Wallace and Abdallah Shami. Concurrent Multipath Transfer Using SCTP: Modelling and Congestion Window Management. *IEEE Transactions on Mobile Computing*, 13 (11):2510–2523, November 2014. doi: 10.1109/TMC.2014.2307330.

[98] socat(1): Multipurpose relay - Linux man page. https://linux.die.net/man/1/socat, 2024-02-14.

[99] Yihua Ethan Guo, Ashkan Nikravesh, Z. Morley Mao, Feng Qian, and Subhabrata Sen. Accelerating Multipath Transport Through Balanced Subflow Completion. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, pages 141–153, Snowbird Utah USA, October 2017. ACM. doi: 10.1145/3117811.3117829.

[100] Rl 3: Upper confidence bound (ucb) to solve multi-armed bandit problem. https://www.youtube.com/watch?v=RPbtzWgzD9M, 2024-04-12.

[101] #2 Machine Learning Specialization [Course 1, Week 1, Lesson 1] - YouTube. https://www.youtube.com/watch?v=wiNXzydta4c, 2024-04-12.

[102] #3 Machine Learning Specialization [Course 1, Week 1, Lesson 2] - YouTube. https://www.youtube.com/watch?v=XtlwSmJfUs4, 2024-04-12.

[103] IIT Kharagpur July 2018. Prof Soumya Kanti Ghosh & Prof Sandip Chakraborty. https://www.youtube.com/watch?v=O–rkQNKqls, April 2018.

[104] Matteo Varvello, Hyunseok Chang, and Yasir Zaki. Performance characterization of video-conferencing in the wild. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 261–273, Nice France, October 2022. ACM. doi: 10.1145/3517745.3561442.

[105] Matthias A. Lee. Python Tesseract, November 2022. URL `https://github.com/madmaze/pytesseract`. original-date: 2010-10-27T23:02:49Z.

[106] 4G/LTE Bandwidth Logs. https://users.ugent.be/ jvdrhoof/dataset-4g/, 2024-11-14.

[107] Worldwide Broadband Speed League 2023. https://www.cable.co.uk/broadband/speed/worldwide-speed-league/, 2024-04-09.

[108] Vibhaalakshmi Sivaraman, Pantea Karimi, Vedantha Venkatapathy, Mehrdad Khani, Sadjad Fouladi, Mohammad Alizadeh, Frédo Durand, and Vivienne Sze. Gemino: Practical and Robust Neural Compression for Video Conferencing, October 2023. arXiv:2209.10507 [cs].

[109] Netflix Technology Blog. Toward A Practical Perceptual Video Quality Metric, April 2017.

[110] Netflix/vmaf, November 2024. original-date: 2016-02-08T18:41:38Z.

[111] Satadal Sengupta, Niloy Ganguly, Sandip Chakraborty, and Pradipta De. HotDASH: Hotspot Aware Adaptive Video Streaming Using Deep Reinforcement Learning. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 165–175, Cambridge, September 2018. IEEE. doi: 10.1109/ICNP.2018.00026.

[112] Gerui Lv, Qinghua Wu, Yanmei Liu, Zhenyu Li, Qingyue Tan, Furong Yang, Wentao Chen, Yunfei Ma, Hongyu Guo, Ying Chen, and Gaogang Xie. Chorus: Coordinating mobile multipath scheduling and adaptive video streaming. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '24, page 246–262, New York, NY, USA, 2024. Association for Computing Machinery. doi: 10.1145/3636534.3649359.

[113] Bo Han, Feng Qian, Lusheng Ji, and Vijay Gopalakrishnan. MP-DASH: Adaptive Video Streaming Over Preference-Aware Multipath. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 129–143, Irvine California USA, December 2016. ACM. doi: 10.1145/2999572.2999606.

[114] Baptiste Jonglez, Martin Heusse, and Bruno Gaujal. SRPT-ECF: challenging Round-Robin for stream-aware multipath scheduling. In *2020 IFIP Networking Conference (Networking)*, pages 719–724, June 2020.

[115] Xutong Zuo, Yong Cui, Xin Wang, and Jiayu Yang. Deadline-aware Multipath Transmission for Streaming Blocks. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pages 2178–2187, May 2022. doi: 10.1109/INFOCOM48880.2022.9796942. ISSN: 2641-9874.

[116] Tan Zhang, Aakanksha Chowdhery, Paramvir Bahl, Kyle Jamieson, and Suman Banerjee. The design and implementation of a wireless video surveillance system. In *21st Annual International Conference on Mobile Computing and Networking*, pages 426–438, 2015.

[117] Saman Naderiparizi, Pengyu Zhang, Matthai Philipose, Bodhi Priyantha, Jie Liu, and Deepak Ganesan. Glimpse: A programmable early-discard camera architecture for continuous mobile vision. In *15th Annual International Conference on Mobile Systems, Applications, and Services*, page 292–305, 2017. doi: 10.1145/3081333.3081347.

[118] Jude Tchaye-Kondi, Yanlong Zhai, Jun Shen, Dong Lu, and Liehuang Zhu. Smartfilter: An edge system for real-time application-guided video frames filtering. *IEEE Internet of Things Journal*, pages 1–1, 2022. doi: 10.1109/JIOT.2022.3188518.

[119] Yiding Wang, Weiyan Wang, Junxue Zhang, Junchen Jiang, and Kai Chen. Bridging the edge-cloud barrier for real-time advanced vision analytics. *Proceedings of the 11th USENIX Conference on Hot Topics in Cloud Computing*, page 18, 2019.

[120] GoPro Support, HEVC Explained. https://community.gopro.com/s/article/hevc. Published on Mar 22, 2023; Accessed on Aug 1, 2023.

[121] Jinwoo Hwang, Minsu Kim, Dohee Kim, Daeun Kim, Seungho Nam, Yoonsung Kim, Hardik Sharma, and Jongse Park. CoVA: Exploiting Compressed-Domain Analysis to Accelerate Video Analytics. In *2022 USENIX Annual Technical Conference*, pages 707–722. URL `https://www.usenix.org/conference/atc22/presentation/hwang`.

[122] Yongfei Zhang, Chao Zhang, Rui Fan, Siwei Ma, Zhibo Chen, and C.-C. Jay Kuo. Recent Advances on HEVC Inter-Frame Coding: From Optimization to Implementation and Beyond. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(11):4321–4339, November 2020. doi: 10.1109/TCSVT.2019.2954474.

[123] Jens-Rainer Ohm, Gary J. Sullivan, Heiko Schwarz, Thiow Keng Tan, and Thomas Wiegand. Comparison of the Coding Efficiency of Video Coding Standards—Including High Efficiency Video Coding (HEVC). *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1669–1684, December 2012. doi: 10.1109/TCSVT.2012.2221192.

[124] G.J. Sullivan and R.L. Baker. Rate-distortion optimized motion compensation for video compression using fixed or variable size blocks. In *IEEE Global Telecommunications Conference*, pages 85–90, Phoenix, AZ, USA, 1991. doi: 10.1109/GLOCOM.1991.188361.

[125] Kiran Misra, Andrew Segall, Michael Horowitz, Shilin Xu, Arild Fuldseth, and Minhua Zhou. An Overview of Tiles in HEVC. *IEEE Journal of Selected Topics in Signal Processing*, 7(6):969–977, December 2013. doi: 10.1109/JSTSP.2013.2271451.

[126] Jan De Cock, Zhi Li, Megha Manohara, and Anne Aaron. Complexity-based consistent-quality encoding in the cloud. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 1484–1488, 2016. doi: 10.1109/ICIP.2016.7532605.

[127] Weihe Li, Jiawei Huang, Shiqi Wang, Chuliang Wu, Sen Liu, and Jianxin Wang. An apprenticeship learning approach for adaptive video streaming based on chunk quality and user preference. *IEEE Transactions on Multimedia*, 25:2488–2502, 2023. doi: 10.1109/TMM.2022.3147667.

[128] Chao-Yang Kao and Youn-Long Lin. A memory-efficient and highly parallel architecture for variable block size integer motion estimation in h. 264/avc. *IEEE transactions on very large scale integration (VLSI) systems*, 18(6):866–874, 2009. doi: 10.1109/TVLSI.2009.2017122.

[129] Philipp Helle, Simon Oudin, Benjamin Bross, Detlev Marpe, M. Oguz Bici, Kemal Ugur, Joel Jung, Gordon Clare, and Thomas Wiegand. Block merging for quadtree-based partitioning in hevc. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12): 1720–1731, 2012. doi: 10.1109/TCSVT.2012.2223051.

[130] Jingning Han, Bohan Li, Debargha Mukherjee, Ching-Han Chiang, Adrian Grange, Cheng Chen, Hui Su, Sarah Parker, Sai Deng, Urvang Joshi, Yue Chen, Yunqing Wang, Paul Wilkins, Yaowu Xu, and James Bankoski. A technical overview of AV1. *Proceedings of the IEEE*, 109(9):1435–1462, 2021. doi: 10.1109/JPROC.2021.3058584.

[131] Yu-Wen Huang, Jicheng An, Han Huang, Xiang Li, Shih-Ta Hsiang, Kai Zhang, Han Gao, Jackie Ma, and Olena Chubach. Block partitioning structure in the vvc standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3818–3833, 2021. doi: 10.1109/TCSVT.2021.3088134.

[132] Miao Zhang, Fangxin Wang, and Jiangchuan Liu. Casva: Configuration-adaptive streaming for live video analytics. In *IEEE Conference on Computer Communications*, pages 2168–2177, 2022. doi: 10.1109/INFOCOM48880.2022.9796875.

[133] Prakhar Ganesh, Yao Chen, Yin Yang, Deming Chen, and Marianne Winslett. YOLO-ReT: Towards high accuracy real-time object detection on edge GPUs. In *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2022. doi: 10.1109/WACV51458.2022.00138.

[134] Ambarella, AI Envisioned, Ambarella enables artificial intelligence on a wide range of connected cameras using Amazon SageMaker Neo. https://www.ambarella.com/news/ambarella-enables-artificial-intelligence-on-a-wide-range-of-connected-cameras-using-amazon-sagemaker-neo/. Visited on April 30, 2023.

[135] Shikha Goel, Rajesh Kedia, M. Balakrishnan, and Rijurekha Sen. Infer: Interference-aware estimation of runtime for concurrent cnn execution on dpus. In *2020 International Conference on Field-Programmable Technology (ICFPT)*, pages 66–71, 2020. doi: 10.1109/ICFPT51103.2020.00018.

[136] Ultralytics yolov5. URL `https://github.com/ultralytics/yolov5`. Accessed on August 1, 2022.

[137] Christopher D Manning. *An introduction to information retrieval*. Cambridge university press, 2009.

[138] Andrea Ceccarelli and Leonardo Montecchi. Evaluating object (mis)detection from a safety and reliability perspective: Discussion and measures. *IEEE Access*, 11:44952–44963, 2023. doi: 10.1109/ACCESS.2023.3272979.

[139] Yunhao Du, Zhicheng Zhao, Yang Song, Yanyun Zhao, Fei Su, Tao Gong, and Hongying Meng. Strongsort: Make deepsort great again. *IEEE Transactions on Multimedia*, 25:8725–8737, 2023. doi: 10.1109/TMM.2023.3240881.

[140] Rand R Wilcox and HJ Keselman. Modern robust data analysis methods: measures of central tendency. *Psychological methods*, 8(3):254, 2003. doi: 10.1037/1082-989X.8.3.254.

[141] Marko Viitanen, Ari Koivula, Ari Lemmetti, Arttu Ylä-Outinen, Jarno Vanne, and Timo D. Hämäläinen. Kvazaar: Open-source hevc/h.265 encoder. In *24th ACM International Conference on Multimedia*, page 1179–1182, 2016. doi: 10.1145/2964284.2973796.

[142] Jean Le Feuvre. GPAC filters. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pages 249–254, Istanbul Turkey, May 2020. ACM. doi: 10.1145/3339825.3394929.

[143] Maxim Priymak and Richard Sinnott. Real-Time Traffic Classification through Deep Learning. In *2021 IEEE/ACM 8th International Conference on Big Data Computing, Applications and Technologies (BDCAT '21)*, pages 128–133, Leicester, United Kingdom, December 2021. doi: 10.1145/3492324.3494165.

[144] Namhyuk Ahn. Fast, Accurate, and Lightweight Super-Resolution with Cascading Residual Network, July 2022. URL `https://github.com/nmhkahn/CARN-pytorch`. original-date: 2017-10-10T20:58:44Z.

[145] Average frame rate video surveillance statistics 2021. https://ipvm.com/reports/average-frame-rate-video-surveillance-2021. Visited on Oct 9, 2023; Published on Jan 8, 2021.

[146] IBM Documentation, Camera frame rate, resolution, and video format requirements. https://www.ibm.com/docs/en/video-analytics/1.0.6?topic=requirements-camera-frame-rate-resolution-video-format, March 2021.

[147] Odroid h3-plus. https://www.hardkernel.com/shop/odroid-h3-plus/. Accessed on May 1, 2023.

[148] Vinod Nigade, Lin Wang, and Henri Bal. Clownfish: Edge and cloud symbiosis for video stream analytics. In *ACM/IEEE Symposium on Edge Computing*, pages 55–69, 2020. doi: 10.1109/SEC50012.2020.00012.

[149] C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose. Videoedge: Processing camera streams using hierarchical clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 115–131, 2018. doi: 10.1109/SEC.2018.00016.

[150] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Paramvir Bahl, and Joseph Gonzalez. Spatula: Efficient cross-camera video analytics on large camera networks. In *ACM/IEEE Symposium on Edge Computing*, pages 110–124, 2020. doi: 10.1109/SEC50012.2020.00016.

[151] Ji-Yan Wu, Vithurson Subasharan, Tuan Tran, and Archan Misra. Mrim: Enabling mixed-resolution imaging for low-power pervasive vision tasks. In *2022 IEEE PerCom*, pages 44–53, 2022. doi: 10.1109/PerCom53586.2022.9762398.

[152] Rachel Huang, Jonathan Pedoeem, and Cuixian Chen. Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2503–2510, 2018. doi: 10.1109/BigData.2018.8621865.

[153] Yuxuan Cai. *YOLObile: Real-time object detection on mobile devices via compression-compilation co-design*. PhD thesis, Northeastern University, 2020.

[154] Shuochao Yao, Yiran Zhao, Huajie Shao, ShengZhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices. In *16th ACM Conference on Embedded Networked Sensor Systems*, page 278–291, 2018. doi: 10.1145/3274783.3274840.

[155] Yakun Huang, Xiuquan Qiao, Jian Tang, Pei Ren, Ling Liu, Calton Pu, and Junliang Chen. Deepadapter: A collaborative deep learning framework for the mobile web using context-aware network pruning. In *IEEE Conference on Computer Communications*, 2020. doi: 10.1109/INFOCOM41043.2020.9155379.

[156] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529*, 2017.

[157] M Khani, G Ananthanarayanan, K Hsieh, J Jiang, R Netravali, Y Shu, M Alizadeh, and V Bahl. RECL: Responsive Resource-Efficient Continuous Learning for Video Analytics. In *USENIX NSDI 2023*, 2023.

[158] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 119–135, Renton, WA, April 2022. ISBN 978-1-939133-27-4. URL `https://www.usenix.org/conference/nsdi22/presentation/bhardwaj`.

[159] Jiong Dong, Kaoru Ota, and Mianxiong Dong. Video frame interpolation: A comprehensive survey. *ACM Transactions on Multimedia Computing and Communication Applications*, 19(2s), May 2023. ISSN 1551-6857. doi: 10.1145/3556544.

[160] *Dahua Technology, Network Video Recorder User's Manual*. Available at https://us.dahuasecurity.com/wp-content/uploads/2023/08/Network-Video-Recorder_Users-Manual_V2.3.1_20230210-Eng.pdf.

[161] *Wisenet Network Camera: User Manual*. Available at https://www.hanwhasecurity.com/wp-content/uploads/attachments/u/s/user_manual-xnp-6120h-english_web-0710.pdf.

[162] Is there any idea to achieve hevc variable bitrate mode in kvazaar encoder? https://github.com/ultravideo/kvazaar/issues/400. Published on Mar 21, 2024; Accessed on Jun 2, 2024.

[163] Sibendu Paul, Kunal Rao, Giuseppe Coviello, Murugan Sankaradas, Oliver Po, Y. C. Hu, and Srimat Chakradhar. Enhancing video analytics accuracy via real-time automated camera parameter tuning. In *20th ACM Conference on Embedded Networked Sensor Systems*, 2023. doi: 10.1145/3560905.3568527.

[164] Autonomous vehicle market, 2025. URL `https://www.nextmsc.com/report/autonomous-vehicle-market`.

[165] Global autonomous vehicle market size 2030, 2025. URL `https://www.statista.com/statistics/1224515/av-market-size-worldwide-forecast/`.

[166] Mehrdad Khani, Pouya Hamadanian, Arash Nasr-Esfahany, and Mohammad Alizadeh. Real-Time Video Inference on Edge Devices via Adaptive Model Streaming. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4552–4562, Montreal, QC, Canada, October 2021. IEEE. ISBN 978-1-66542-812-5. doi: 10.1109/ICCV48922.2021.00453. URL `https://ieeexplore.ieee.org/document/9711068/`.

[167] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016. URL `https://arxiv.org/abs/1510.00149`.

[168] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF CVPR*, pages 16091–16101, June 2023.

[169] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE CVPR*, June 2018.

[170] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5958–5968. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/li20m.html`.

[171] Maozhe Zhao, Shengzhong Liu, Fan Wu, and Guihai Chen. Responsive dnn adaptation for video analytics against environment shift via hierarchical mobile-cloud collaborations. In *Proceedings of the 23rd ACM SenSys*, pages 317–331, 2025.

[172] Xiangxiang Dai, Zeyu Zhang, Peng Yang, Yuedong Xu, Xutong Liu, and John C S Lui. AxiomVision: Accuracy-Guaranteed Adaptive Visual Model Selection for Perspective-Aware Video Analytics. 2024.

[173] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023. URL `https://github.com/ultralytics/ultralytics`.

[174] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2022. doi: 10.1109/TPAMI.2021.3057446.

[175] David Lopez-Paz and Marc' Aurelio Ranzato. Gradient episodic memory for continual learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/f87522788a2be2d171666752f97ddebb-Paper.pdf`.

[176] Liang Wang, Xiaoyang Qu, Jianzong Wang, Guokuan Li, Jiguang Wan, Nan Zhang, Song Guo, and Jing Xiao. Gecko: Resource-Efficient and Accurate Queries in Real-Time Video Streams at the Edge. In *IEEE INFOCOM 2024*, pages 481–490, May 2024. doi: 10.1109/INFOCOM52122.2024.10621399. URL `https://ieeexplore.ieee.org/document/10621399/`. ISSN: 2641-9874.

[177] Zhiqiang Cao, Yun Cheng, Zimu Zhou, Anqi Lu, Youbing Hu, Jie Liu, Min Zhang, and Zhijun Li. Patching in Order: Efficient On-Device Model Fine-Tuning for Multi-DNN Vision Applications. *IEEE Transactions on Mobile Computing*, 23(12):14484–14501, December 2024. ISSN 1558-0660. doi: 10.1109/TMC.2024.3443057. URL `https://ieeexplore.ieee.org/document/10636841/`.

[178] Liang Wang, Kai Lu, Nan Zhang, Xiaoyang Qu, Jianzong Wang, Jiguang Wan, Guokuan Li, and Jing Xiao. Shoggoth: Towards Efficient Edge-Cloud Collaborative Real-Time Video Inference via Adaptive Online Learning. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, July 2023. doi: 10.1109/DAC56929.2023.10247821. URL `https://ieeexplore.ieee.org/document/10247821/`.

[179] Iain E Richardson. *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011.

[180] Rostand A K. Fezeu, Claudio Fiandrino, Eman Ramadan, Jason Carpenter, Lilian Coelho De Freitas, Faaiq Bilal, Wei Ye, Joerg Widmer, Feng Qian, and Zhi-Li Zhang. Unveiling the 5g mid-band landscape: From network deployment to performance and application qoe. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 358–372, 2024.

[181] Ultralytics. Yolov5. URL `https://docs.ultralytics.com/models/yolov5`.

[182] Miao Zhang, Fangxin Wang, and Jiangchuan Liu. CASVA: Configuration-Adaptive Streaming for Live Video Analytics. In *IEEE INFOCOM 2022*, pages 2168–2177, London, United Kingdom, May 2022. IEEE. ISBN 978-1-66545-822-1. doi: 10.1109/INFOCOM48880.2022.9796875. URL `https://ieeexplore.ieee.org/document/9796875/`.

[183] Xiufeng Xie and Kyu-Han Kim. Source compression with bounded dnn perception loss for iot edge computer vision. In *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361699. doi: 10.1145/3300061.3345448. URL `https://doi.org/10.1145/3300061.3345448`.

[184] esnet/iperf, July 2025. URL `https://github.com/esnet/iperf`. original-date: 2014-02-25T18:42:52Z.

[185] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. *Proceedings of the 2015 USENIX Annual Technical Conference (USENIX ATC '15)*, pages 417–429, July 2015. URL `https://www.usenix.org/conference/atc15/technical-session/presentation/netravali`.

[186] T Y Lin, M Maire, S Belongie, J Hays, P Perona, D Ramanan, P Dollár, and C. L. Zitnick. Microsoft COCO: Common Objects in Context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10602-1. doi: 10.1007/978-3-319-10602-1_48.

[187] Mohammad Mehdi Rastikerdar, Jin Huang, Shiwei Fang, Hui Guan, and Deepak Ganesan. Cactus: Dynamically switchable context-aware micro-classifiers for efficient iot inference. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*, pages 505–518, 2024.

[188] Bryan Bo Cao, Lawrence O'Gorman, Michael Coss, and Shubham Jain. Few-class arena: A benchmark for efficient selection of vision models and dataset difficulty measurement. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=2ET561DyPe`.

[189] J Utah - YouTube. URL https://www.youtube.com/@jutah/featured.

[190] Dani Manjah, Davide Cacciarelli, Mohamed Benkedadra, Baptiste Standaert, Gauthier Rotsart De Hertaing, Benoît Macq, Stéphane Galland, and Christophe De Vleeschouwer. Stream-Based Active Distillation for Scalable Model Deployment. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 4999–5007, Vancouver, BC, Canada, June 2023. IEEE. ISBN 9798350302493. doi: 10.1109/CVPRW59228.2023.00528. URL https://ieeexplore.ieee.org/document/10208751/.

[191] Yulu Gan, Mingjie Pan, Rongyu Zhang, Zijian Ling, Lingran Zhao, Jiaming Liu, and Shanghang Zhang. Cloud-Device Collaborative Adaptation to Continual Changing Environments in the Real-World. In *2023 IEEE/CVF CVPR*, pages 12157–12166, Vancouver, BC, Canada, June 2023. IEEE. ISBN 979-8-3503-0129-8. doi: 10.1109/CVPR52729.2023.01170. URL https://ieeexplore.ieee.org/document/10204872/.

[192] Shubham Chaudhary, Arani Bhattacharya, Saket Anand, and Aruna Balasubramanian. Scalable and sustainable video analytics on edge using sensor clustering. In *Proceedings of the 30th ACM MobiCom*, ACM MobiCom '24, page 2239–2241, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704895. doi: 10.1145/3636534.3695902. URL https://doi.org/10.1145/3636534.3695902.

[193] Yoonsung Kim, Changhun Oh, Jinwoo Hwang, Wonung Kim, Seongryong Oh, Yubin Lee, Hardik Sharma, Amir Yazdanbakhsh, and Jongse Park. DACAPO: Accelerating Continuous Learning in Autonomous Systems for Video Analytics. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 1246–1261, June 2024. doi: 10.1109/ISCA59077.2024.00093. URL https://ieeexplore.ieee.org/document/10609643/?arnumber=10609643.

[194] Sudipta Saha Shubha and Haiying Shen. AdaInf: Data Drift Adaptive Scheduling for Accurate and SLO-guaranteed Multiple-Model Inference Serving at Edge Servers. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 473–485, New York NY USA, September 2023. ACM. ISBN 9798400702365. doi: 10.1145/3603269.3604830. URL https://dl.acm.org/doi/10.1145/3603269.3604830.

[195] Qinglong Zhang, Rui Han, Chi Harold Liu, Guoren Wang, and Lydia Y. Chen. ElasticDNN: On-Device Neural Network Remodeling for Adapting Evolving Vision Domains at Edge. *IEEE Transactions on Computers*, 73(6):1616–1630, June 2024. ISSN 1557-9956. doi: 10.1109/TC.2024.3375608. URL https://ieeexplore.ieee.org/document/10466390/?arnumber=10466390. Conference Name: IEEE Transactions on Computers.

[196] Dawei Li, Serafettin Tasci, Shalini Ghosh, Jingwen Zhu, Junting Zhang, and Larry Heck. RILOD: near real-time incremental learning for object detection at the edge. In *Proceedings*

of the 4th ACM/IEEE Symposium on Edge Computing, SEC '19, pages 113–126, New York, NY, USA, November 2019. Association for Computing Machinery. ISBN 978-1-4503-6733-2. doi: 10.1145/3318216.3363317. URL https://dl.acm.org/doi/10.1145/3318216.3363317.

[197] Yan Zhuang, Zhenzhe Zheng, Yunfeng Shao, Bingshuai Li, Fan Wu, and Guihai Chen. Nebula: An Edge-Cloud Collaborative Learning Framework for Dynamic Edge Environments. In *Proceedings of the 53rd International Conference on Parallel Processing*, ICPP '24, pages 782–791, New York, NY, USA, August 2024. Association for Computing Machinery. ISBN 9798400717932. doi: 10.1145/3673038.3673120. URL https://dl.acm.org/doi/10.1145/3673038.3673120.

[198] Yuheng Zhu, Dhruva Ungrupulithaya, Boluo Ge, and Man-Ki Yoon. AdaptAV: Continuous Adaption of Vision Models for Autonomous Vehicles Using Cloud-based Oracle. In *2024 IEEE 100th VTC2024-Fall*, pages 1–7, October 2024. doi: 10.1109/VTC2024-Fall63153.2024.10757493. URL https://ieeexplore.ieee.org/document/10757493/. ISSN: 2577-2465.

[199] Alexander Dietmüller, Romain Jacob, and Laurent Vanbever. On sample selection for continual learning: A video streaming case study. *SIGCOMM Comput. Commun. Rev.*, 54(2):10–35, August 2024. ISSN 0146-4833. doi: 10.1145/3687234.3687237. URL https://doi.org/10.1145/3687234.3687237.

[200] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 107–125, 2020.

[201] Peng Zhao, Runchu Dong, Guiqin Wang, and Cong Zhao. EdgeSync: Faster Edge-model Updating via Adaptive Continuous Learning for Video Data Drift, June 2024. URL http://arxiv.org/abs/2406.03001. arXiv:2406.03001 [cs].

[202] Linshan Jiang, Qun Song, Rui Tan, and Mo Li. PriMask: Cascadable and Collusion-Resilient Data Masking for Mobile Cloud Inference. In *Proceedings of the Twentieth ACM Conference on Embedded Networked Sensor Systems*, pages 164–178, Boston Massachusetts, November 2022. ACM. ISBN 978-1-4503-9886-2. doi: 10.1145/3560905.3568531. URL https://dl.acm.org/doi/10.1145/3560905.3568531.