Continuous Learning for Lightweight Machine Learning Inference at the Edge

by

Mehrdad Khani

B.S., Sharif University of Technology (2016)
S.M., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

September 2023

Authored by:	Mehrdad Khani Department of Electrical Engineering and Computer Science August 31, 2023
Certified by:	Mohammad Alizadeh Associate Professor of Electrical Engineering and Computer Science Thesis Supervisor
Accepted by:	Leslie A. Kolodziejski Professor of Electrical Engineering and Computer Science Chair, Department Committee on Graduate Students

Continuous Learning for Lightweight Machine Learning Inference at the Edge

by

Mehrdad Khani

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2023, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

With the proliferation of edge devices such as mobile phones, consumer robots, drones, wearables, and IoT devices, the generation of data at the edge of the internet network has been increasing exponentially. Machine Learning (ML) models, particularly Deep Neural Networks (DNNs), have the ability to process this data with remarkable accuracy. However, state-of-the-art ML models require substantial computational resources that edge devices typically lack, necessitating a shift to powerful servers in the cloud as hosts for these models. Running these models at the edge is desirable due to benefits such as low-latency results and adherence to data privacy constraints, but is limited by the available computational power and energy consumption of edge devices. Moreover, lightweight models designed for edge devices often exhibit a significant drop in accuracy. Continuous learning offers a potential solution by improving the accuracy of lightweight models by dynamically adapting them to specific scenes or narrow distributions of inputs, which is especially relevant since in practice, these models do not need to generalize to every possible sample from the distribution.

In this thesis, two key methods are introduced to tackle the challenges in continuous learning systems for edge devices: Model Streaming and Model Reuse. Model Streaming offloads the adaptation process to remote machines with greater computational capacity and updates only a critical subset of model parameters that significantly influence the lightweight model's performance, reducing the bandwidth needed for model updates. Model Reuse uses an efficient DNN model to dynamically select a suitable lightweight model from a library of historical models designed for similar input distributions, boosting the scalability, responsiveness, and accuracy of continuous learning systems. These methods are applied to practical systems, including MMNet for adaptive neural signal detection in 5G cellular communication systems, AMS for real-time video inference on edge devices, SRVC for efficient video compression, and RECL for responsive, resource-efficient continuous learning for video analytics.

We show how continuous learning can significantly improve lightweight machine learning inference on edge devices. The proposed techniques effectively address the unique challenges posed by resource-constrained edge environments. Practical applications presented in the thesis, such as MMNet, AMS, SRVC, and RECL, demonstrate the real-world effectiveness of these methods. These innovations in continuous learning have the potential to reshape the landscape of edge computing by offering more accurate and adaptable inference capabilities, enabling efficient use of computational resources, reduced latency, and better energy efficiency.

Thesis Supervisor: Mohammad Alizadeh
Title: Associate Professor of Computer Science

Acknowledgments

I would like to start by expressing my deepest gratitude to my advisor, Mohammad Alizadeh. His unwavering support and invaluable guidance have been fundamental throughout my doctoral journey. Mohammad's mentorship has not only empowered me to explore and develop my own research taste but has also significantly enriched my academic experience. I am truly privileged to have been under his guidance, and for that, I am profoundly grateful.

I am deeply thankful to my committee members, Hari Balakrishnan, Samuel Madden, and Ganesh Ananthanarayanan, for their insightful comments and feedback on this dissertation. Their critiques and perspectives have not only enhanced this dissertation but have also provided valuable direction for future work. Hari has consistently instilled a sense of purpose in me, Sam has been consistently generous, and Ganesh has been nothing short of an exemplary co-advisor, always offering me invaluable advice and insights about high-level directions.

I extend my heartfelt thanks to my mentors and collaborators: Manya Ghobadi, Jakob Hoydis, Ravi Netravali, Junchen Jiang, and Amin Vahdat, for their invaluable guidance and collaboration. Their knowledge and input have been pivotal in shaping the outcomes of my research. Manya kindly took me under her wing and introduced me to the world of optical networking, while Jakob did the same in the field of wireless communications. Ravi has been a true mentor and guide, providing both technical assistance in the early days of my PhD and career advice as I approached its conclusion. Junchen has been a constant source of energy and support throughout the entire course of the RECL project, from inception to the final manuscript edits. Working with Amin has been an absolute privilege, and I have learned so much from his vision and insights. Amin has consistently been a true inspiration and a pillar of support in every interaction we have had.

I would like to extend my warmest thanks to my collaborators: Kevin Hsieh, Yuanchao Shu, Victor Bahl, Songtao He, Pouya Hamadanian, Arash Nasr-Esfahany, Hongzi Mao, Ziyi Zhu, Madeleine Glick, Keren Bergman, Benjamin Klenk, Eiman Ebrahimi, Vibhaalakshmi Sivaraman, Pantea Karimi, Sadjad Fouladi, Frédo Durand, and Vivienne Sze. Their expertise and contributions have significantly enriched the scope and quality of our research. I will truly miss our last-minute sync-up calls as submission deadlines approached.

At MIT, I found a community within my labmates and friends at CSAIL, whose camaraderie and support I will miss dearly: Omid Abari, Abdullah Alomar, Venkat Arun, Arjun Balasingam, Favyen Bastani, Frank Cangialosi, Inho Cho, Prateesh Goyal, Pouya Hamada-

nian, Songtao He, Peter Iannucci, Pantea Karimi, Moein Khazraee, Hongzi Mao, Radhika Mittal, Akshay Narayan, Srinivas Narayana, Arash Nasr-Esfahany, Vikram Nathan, Parimarjan Negi, Amy Ousterhout, Sudarsanan Rajasekaran, Ahmed Saeed, Harsha Sharma, Anirudh Sivaraman, Vibhaalakshmi Sivaraman, Will Sussman, Deepak Vasisht, Frank Wang, Lei Yang, and Zhizhen Zhong, among many others.

I would also like to extend my sincere gratitude to our administrative staff, Sheila Marian, Angelly Arriola, Janet E. Fischer, and Alicia Duarte, whose exceptional support throughout my time at MIT allowed me to focus on my research without distraction. I am additionally grateful to Leslie Kolodziejski and Asu Ozdaglar, as well as the MIT International Students Office, for their steadfast support during the personal family emergency I experienced during my doctoral studies. Their contributions were invaluable and greatly appreciated.

I would like to offer my heartfelt thanks to my friends in Boston and the Bay Area, Maryam Akbarpour, Maryam Aliakbarpour, Yeganeh Alimohammadi, Shervine Amidi, Afshin Amidi, Mona Azadkia, Tara Boroushaki, Ali Fahimnia, Alireza Fallah, Mojtaba Forghani, Babak Fotouhi, Khashayar Gatmiri, Sara Haddad, Behzad Haghgoo, Soheil Hor, Farnaz Jahanbakhsh, Ali Kheradmand, Sepideh Mahabadi, Negar Matoorian, Morteza Moazami, Sajjad Mohammadi, Naghmeh Momeni, Amir Najafi, Setayesh Radkani, Pouya Rezazadeh, Maedeh Seyedolmohadesin, Mohammad Hossein Shafinia, Amir Tohidi, Tina Torkaman, Mahdi Torkashvand, and Ali Vakilian, who have been a constant source of strength and support. I would also like to thank my old group of friends from high school, Sepehr Abbasizadeh, Farshid Effaty, Pooyan Ehsani, and Hadi Yami, for always being there for me, regardless of distance.

My gratitude extends immeasurably to my parents, Tahereh and Houshyar, and my sister, Mahtab, who have supported me through thick and thin. Their love, support, and belief in me have been the foundation of my journey. Their sacrifices and dedication have enabled me to follow my dreams. We unexpectedly lost my father near the end of this journey. He was always an endless source of passion and inspiration for me, and along with my mother, they always supported me in becoming the best version of myself.

Finally, I wish to express my deepest love and gratitude to my wife, Zahra. Her support has been my bedrock, and her love, understanding, and encouragement have sustained me through the challenges and triumphs of this journey. She has been my constant guiding star and my greatest strength, and for that, I am eternally grateful. I am also truly blessed to be surrounded by my wonderful parents-in-law, Maryam and Abbas, and my in-law siblings Mohsen and Tara.

It is thanks to the sacrifices made by my parents, sister, and wife throughout the years

before and during my PhD that this dissertation is ready today, and I hereby dedicate it to them, my beloved family!

To my family

In loving memory of my dad

Previously Published Material

Chapter 3 revises a previous publication [1]: M. Khani, M. Alizadeh, J. Hoydis, P. Fleming. Adaptive Neural Signal Detection for Massive MIMO. IEEE TWC, 2020.

Chapter 4 revises a previous publication [2]: M. Khani, P. Hamadanian, A. Nasr-Esfahany, M. Alizadeh. Real-time Video Inference on Edge Devices via Adaptive Model Streaming. IEEE/CVF ICCV, 2021.

Chapter 5 revises a previous publication [3]: M. Khani, V. Sivaraman, M. Alizadeh. Efficient Video Compression via Content-Adaptive Super-Resolution. IEEE/CVF ICCV, 2021.

Chapter 6 revises a previous publication [4]: 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. USENIX NSDI, 2023.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Devices at the edge of the internet network, such as mobile phones, consumer robots, drones, wearables, and various Internet of Things (IoT) devices, continuously generate an immense volume of data. Machine Learning (ML), with its ability to process this data with remarkable accuracy, has reshaped data inference, enhancing existing applications and paving the way for new ones.

From voice assistants to autonomous driving, these applications rely on data inference using Deep Neural Networks (DNNs) at their core. DNN models have continually pushed the boundaries of accuracy for fundamental tasks across various signal modalities, such as image detection [7], natural language processing [8], speech recognition [9], Lidar perception [10], and wireless detection [11].

However, achieving this accuracy often comes at a substantial computational cost. State-of-the-art ML models have been consistently growing in size faster than Moore's law for several years now, necessitating a shift from desktop computer stations to powerful servers in the cloud as hosts for these models. For example, language models like GPT-3 [12] have few hundred billions of parameters these days, making them computationally expensive (if not infeasible) to run on any device at the edge.

Concurrently, there is an escalating demand to run these ML models at the edge, close to their data sources. This proximity ensures real-time, low-latency results while adhering to data privacy constraints [13–16]. Moreover, offloading the inference to powerful remote machines, such as those in the cloud [17–19], may require high network bandwidth, especially for video and Lidar data, and is vulnerable to network outages, a common characteristic of wireless connections [20].

Nonetheless, edge devices offer substantially less computational power than cloud accelerators. Figure 1-1 illustrates the number of images that can be processed using a typical vision model across devices ranging from running in the cloud to the edge. The figure clearly depicts a substantial gap in processing throughput and power across these devices. This discrepancy becomes increasingly problematic when attempting to achieve state-of-the-art accuracy on edge devices. For instance, large semantic segmentation and object detection models can take several seconds to process on mobile phones [21, 22]. Even compact accelerators like Coral Edge TPU [23] and NVIDIA Jetson [24], commonly used in small drones and robots [25–27], are unable to run these models in real-time, processing 30 frames per second without batching multiple frames. Another crucial factor to consider is energy consumption, particularly for many edge devices that run on batteries. For example, an iPhone 14 Pro operating at full processing speed as in Figure 1-1 will exhaust its battery in approximately an hour.

On the other hand, lightweight neural network models crafted for efficient execution on edge devices often exhibit a considerable drop in accuracy when compared to state-of-the-art models. This gap is exemplified by the performance of various object detection models in relation to their inference latency on the COCO dataset, as portrayed in Figure 1-2. The accuracy metric (mean Average Precision) of the best model is almost double what can be accomplished in real-time on the edge devices. This drop in accuracy of lightweight ML inference at the edge provides a strong motivation for exploring and developing continuous learning methods for lightweight inference at the edge.

Continuous learning aims to improve the accuracy of lightweight models by dynamically adapting them to each specific scene or narrow distribution of inputs. A critical observation is that the main constraint of such lightweight models is their lack of generalization to wide data distributions. However, in practice, it is not necessary for these models to generalize to every possible sample from the distribution. Consider two examples depicted in Figure 1-3: a Roomba camera running at home and a Dash camera driving on city streets. For inference on the camera data in each of these scenarios, our model only needs to perform well on images captured within some specific home or some specific city. In fact, it is sufficient for the model to perform well in a particular room at home or a specific neighborhood in the city at any given time. The core idea behind the continuous learning framework is to enhance the accuracy of such lightweight models by dynamically adapting them to each specific scene or narrow distribution of inputs. This highlights the potential of continuous learning, where lightweight models can be continuously adapted to specific scenes or narrow input distributions, leading to improved accuracy.

However, accomplishing continuous adaptation demands thoughtful consideration of

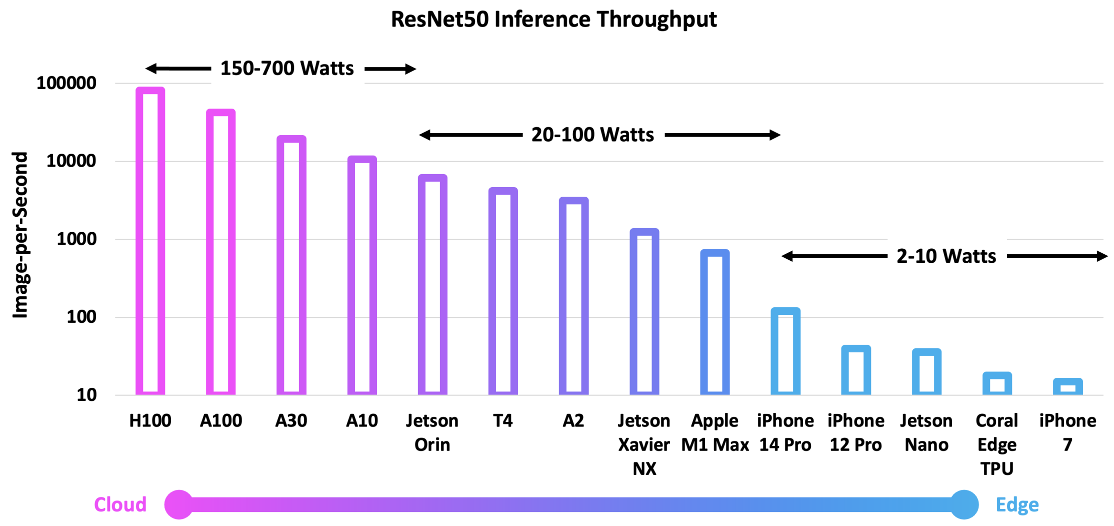


Figure 1-1: Processing throughput of a standard vision model (ResNet50) across a range of devices, spanning from cloud servers to edge devices, highlighting the gap in compute power between the cloud and the edge.

system resource requirements for practical implementations. In this thesis, we put forth techniques that enable new design points in the ML inference systems space, aiming to render continuous adaptations beneficial across a diverse range of applications.

1.2 Primary Contributions

Running continuous learning on edge devices without careful optimization can lead to increased latency, reduced responsiveness, and even system failures due to excessive computational and memory overhead. The limited processing capabilities of edge devices often make it challenging to meet the computational requirements of continuous learning algorithms. Additionally, these devices usually operate under strict energy constraints, further limiting the amount of computation that can be performed on the device.

In this thesis, we present new methods and system designs that effectively tackle these challenges, paving the way for practical implementation of continuous learning. We introduce the concept of continuous learning tailored specifically for resource-constrained edge environments. This represents a novel system design approach with important implications for achieving lightweight and accurate machine learning (ML) inference at the edge.

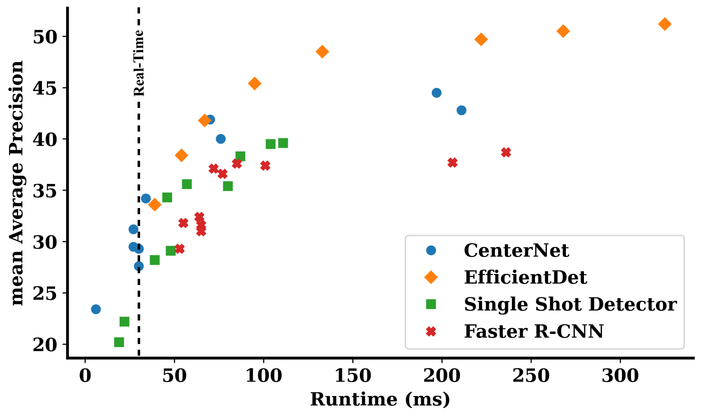
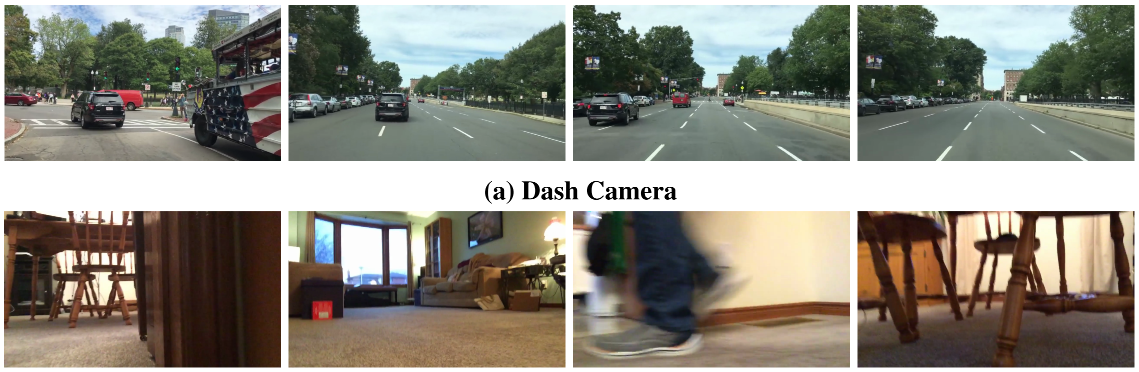


Figure 1-2: Object detection accuracy vs. inference time for various deep models. Accuracy decreases when the compute footprint is reduced.



(a) Dash Camera

(b) Roomba Camera

Figure 1-3: Two example video camera samples in real-world. Cameras usually capture samples from a specific scene at each point in time.

1.2.1 Methods

We introduce two key methods in this thesis to tackle the challenges in continuous learning systems, facilitating their application in diverse scenarios:

- **Model Streaming:** To mitigate the substantial computational and memory overhead caused by dynamic model updates on edge devices, we introduce Model Streaming. This approach offloads the adaptation process to remote machines with greater computational capacity. Model Streaming updates only a critical subset of model parameters that significantly influence the lightweight model's performance. This strategy reduces the bandwidth needed for model updates, enhancing the operational efficiency of continuous learning systems.

- **Model Reuse:** Capitalizing on the spatio-temporal localities among different data streams at various times, we introduce Model Reuse. This method uses an efficient DNN model to dynamically select a suitable lightweight model from a library of historical models designed for similar input distributions. Model Reuse boosts the scalability, responsiveness, and accuracy of continuous learning systems.

1.2.2 Systems & Designs

This thesis introduces several continuous learning systems and designs, demonstrating the real-world applicability of our methods:

MMNet: Adaptive Neural Signal Detection for Massive MIMO. MMNet [1] is a continuous learning system tailored for 5G cellular communication systems. It addresses the challenge of signal detection in Massive Multiple-Input Multiple-Output (MIMO) environments, where a base station with many antennas serves multiple single-antenna devices simultaneously. Existing symbol detection algorithms either perform poorly or are impractical to deploy in this setting due to their computational complexity. MMNet overcomes these hurdles by using a lightweight signal detector that exploits temporal and spectral (frequency) locality in real-world channels. By continually adapting the signal detection model to these channel characteristics, MMNet outperforms existing detection approaches on realistic channels, offering a scalable and efficient solution for next-generation cellular systems.

AMS: Real-Time Video Inference on Edge Devices via Adaptive Model Streaming. AMS [2] embodies a practical application of the Model Streaming method for real-time video inference on edge devices. This system continuously adapts a lightweight model on the edge device to improve its performance on live video feeds. The adaptation process is offloaded to remote servers that have superior computational resources. AMS incorporates several practical techniques to minimize communication costs, including adaptive sampling of training frames, fractional model updates, and strategies to avoid overfitting. This results in a significant reduction in bandwidth usage and improved accuracy, allowing for real-time, low-latency video inference at the edge, which is crucial for many applications such as surveillance and navigation.

SRVC: Efficient Video Compression via Content-Adaptive Super-Resolution. SRVC [3] leverages Model Streaming for efficient video compression, presenting a unique take on continuous learning in the realm of video coding. The system combines a conventional codec with a lightweight, content-adaptive super-resolution model to enhance video quality. SRVC encodes videos into two streams: a content stream, generated by compressing a low-resolution version of the video, and a model stream, which contains periodic updates to the super-resolution model. Upon decoding, SRVC uses the super-resolution model to upsample the low-resolution frames, reconstructing high-quality video in real-time. SRVC's decoder outperforms existing learning-based compression schemes in terms of speed and image quality, demonstrating the effectiveness of continuous learning for video compression.

RECL: Responsive Resource-Efficient Continuous Learning for Video Analytics. RECL integrates the Model Reuse method into a video analytics framework, demonstrating the benefits of combining model retraining with model reuse. This system is designed to be responsive and resource-efficient, making it suitable for large-scale deployments across multiple edge devices. To achieve this, RECL maintains a library of previously trained models, or a "model zoo," and uses a DNN to select the most suitable model for a given input distribution. By reusing models in this way, RECL can respond quickly to changes in the scene while minimizing the computational cost of retraining. Furthermore, RECL dynamically optimizes GPU allocation between model retraining, model selection, and updates to the model zoo, ensuring efficient resource utilization. As the model zoo grows richer over time, RECL's performance and scalability continue to improve.

1.3 Key Takeaways

There are three key takeaways from this thesis. First, continuous learning methods offer a practical approach to improve lightweight machine learning inference at the edge. By dynamically adapting models to specific scenes or narrow input distributions, continuous learning can enhance accuracy without requiring models to generalize to every possible sample from the distribution. Second, the proposed techniques of Model Streaming and Model Reuse effectively address the main challenges of implementing continuous learning in resource-constrained edge environments. Model Streaming optimizes the adaptation process by offloading computational tasks to remote machines, reducing overhead on edge devices. Model Reuse leverages spatio-temporal localities to select suitable lightweight models, improving scalability and responsiveness.

Third, the practical applications showcased in this thesis demonstrate the effectiveness of the proposed methods in real-world scenarios from wireless signal detection to video analytics. These applications highlight the potential of continuous learning for enhancing inference performance at the edge.

Moving forward, the findings of this thesis suggest that continuous learning and the optimization techniques presented can have broader implications for edge computing. They can influence the design and deployment of edge applications, enabling more efficient use of computational resources, improved latency, and enhanced energy efficiency. Furthermore, continuous learning methods can shape the development of new edge computing applications by providing accurate and adaptable inference capabilities close to the data source.

Chapter 2

Background

Current machine learning inference heavily relies on robust cloud-based servers, which are characterized by significant computational and storage capabilities. However, cloud-based inference can experience latency and increased network congestion due to data communication with the cloud. To mitigate these issues and adhere to data privacy preferences, edge computing has emerged as a promising alternative, bringing computation and data storage closer to the source of data generation. However, edge devices are often more resource-constrained than conventional cloud servers. This necessitates the development of streamlined ML methods designed specifically for edge implementation. In this chapter, we will explore various strategies for performing ML inference at the edge, with a primary focus on reducing resource requirements.

2.1 Reducing the Compute Overhead at the Edge

Leveraging ML at the edge is a growing concern in various areas of research. The goal is to offer advanced ML insights in resource-limited environments, which often lack the necessary computational power or network connectivity for persistently running state-of-the-art ML models.

2.1.1 Crafting Lightweight Models

To run ML models efficiently at the edge, system-wide optimizations are employed. Optimizations have been proposed at various layers of the system stack. In practice, several such optimizations are often performed jointly to further enhance performance.

DNN architectures are usually tailored to each specific class of edge devices for maximum efficiency. The key insight behind this approach is that the performance (latency) of the same operator can vary across different devices, necessitating a customized network architecture to achieve the ideal balance between accuracy and efficiency. Prior work has shown the promise of such customization both through manual [28] and automated methods [29, 30].

A well-know example of manually tailored model design is in MobileNetV2 [28]. This model is developed for mobile devices, and it employs specialized convolutional layers, specifically the inverted residuals with linear bottlenecks, that are notably appropriate for mobile applications. This design considerably reduces memory usage during inference by preventing the generation of large intermediate tensors in their entirety. As a result, the requirement for main memory access is decreased, a benefit for many embedded hardware designs that provide only limited amounts of high-speed, software-controlled cache memory.

More generally, neural architecture search methods are formulated to automatically explore and ascertain the optimal architecture for a neural network for a given device [29, 30]. Recent studies incorporate these methods to automatically traverse the design space, often introducing innovative algorithms (e.g., reinforcement learning) that guide the search process. This process involves sampling possible architectures within the limitations of the hardware's capabilities and constraints and iteratively identifying the most efficient one.

Computational efficiency of ML models can be further improved by model compression techniques such as quantization and pruning. These strategies aim to decrease the model size in memory and number of computational operations while maintaining a high level of predictive accuracy. Specifically, quantization reduces the numerical precision of the model's parameters, and pruning eliminates unnecessary weights or neurons from a neural network, effectively reducing model complexity [31–33]. As evidenced in recent studies [34], both model quantization and weight pruning serve as effective approaches in lessening the computational footprint of models, with only a marginal impact on accuracy.

There are task-specific optimizations available to reduce the overhead of ML applications. For instance, in video analytics, optical flow methods help amortize inference costs by forgoing inference for specific frames and focusing on a select subset, with motion information bridging the gaps [35–37]. In the context of large language models, caching the key and value matrices for intermediary tokens within the self-attention layers can notably enhance performance [38].

2.1.2 Offloading Computations

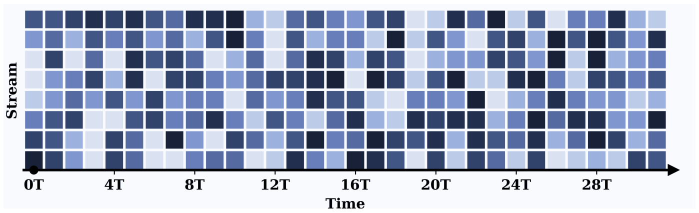
An alternative solution to resource limitations at the edge is to expand the total available computational resources by leveraging the network. Various proposals suggest offloading all or part of the computation to a remote machine [17–19, 39–41]. This approach allows access to greater computational resources, resulting in improved model performance. Moreover, the cloud's ability for resource pooling significantly enhances resource utilization. However, such methods typically require significant network bandwidth, add to latency, and are vulnerable to network disruptions [18, 42]. To offset these drawbacks, some proposals advocate the use of on-premise edge computing servers [43–45] that place the remote machine closer to the edge devices. While this approach does alleviate some of the challenges, it introduces significant additional costs related to infrastructure provision and maintenance.

2.2 Model Adaptation at the Edge

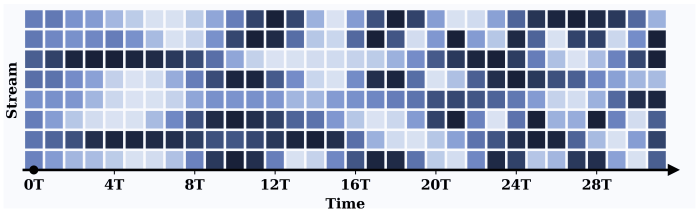
In most real-world deployments of ML at the edge, we are presented with a plethora of data sources, each exhibiting its own unique characteristics. This diversity, while being a boon for model training, also introduces the challenge of encountering data that the model has never been exposed to before, often termed as unseen or out-of-distribution data. Such scenarios can arise due to a myriad of reasons: changing environmental conditions, evolving user behaviors, introduction of new devices, or even just the sheer unpredictability of the real world.

In traditional, centralized ML models, regular updates and retraining can address this by frequently ingesting new data. However, the edge setting is intrinsically different. Here, models are deployed on devices with constrained resources, often operating in environments with limited connectivity. This makes regular model updates more challenging, and sometimes, even unfeasible. Furthermore, the immediacy of decision-making at the edge demands rapid adaptation to maintain model accuracy and utility.

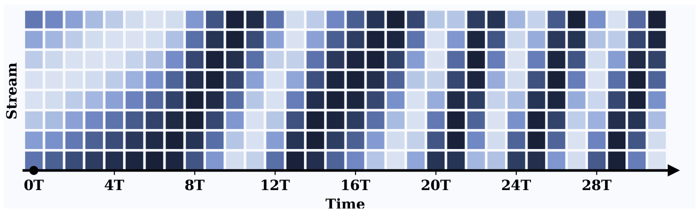
Consequently, in the edge setting, there's a heightened importance of building inference systems that are not just performant out-of-the-box but are also capable of learning and adapting over time. This could mean incorporating mechanisms like online learning, few-shot learning, or transfer learning to allow models to adapt to new data patterns without the need for complete retraining. In essence, the dynamic nature of data at the edge necessitates a paradigm shift in how we approach, design, and maintain ML inference systems, ensuring they remain relevant and effective throughout their deployment lifecycle.



(a) Random images as input streams



(b) Video cameras as input streams



(c) MIMO channels as input streams

Figure 2-1: Input data sources display various types of localities over time and across streams, depending on their nature. The data distribution in each segment is mapped to a scalar combination ratio and is represented by a weighted sum of yellow and dark blue at the respective ratios. Variations in color represent discrepancies in data distribution.

2.2.1 Continuous Learning

In order to harness the full potential of ML at the edge, models must not only be lightweight but also capable of adapting to changing data distributions. This introduces the concept of continuous learning—wherein an ML model refines its predictions over time, specializing for each local distribution.

Framework

Sensors typically generate data in sequences, also known as streams. These data streams usually exhibit a correlation structure. In this context, temporal locality pertains to the correlation of data points within a single stream over time, while spatial locality relates to the correlation across different streams. Figure 2-1 demonstrates the color-coded mapping of the sensory data samples into scalar space for three real-world streams. In this figure, differences in slot colors indicate discrepancies between their data distributions. Although most machine learning models are primarily designed and evaluated for random

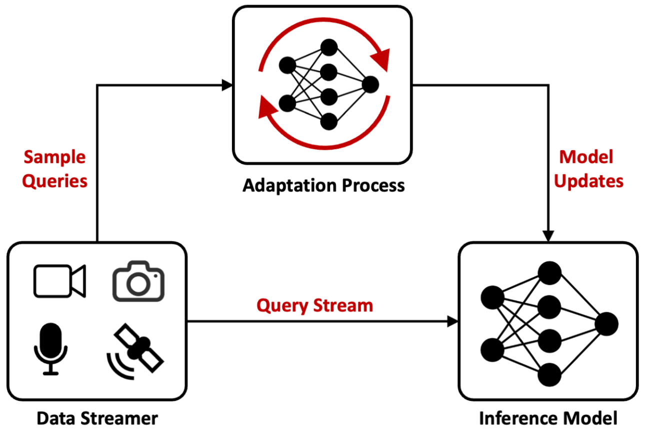


Figure 2-2: Typical components in a continuous learning system.

data streams assuming independent and identically distributed (i.i.d.) samples drawn from the overall data distribution like in Figure 2-1a, many real-world streams inherently display more accentuated locality structures. Examples include video cameras (Figure 2-1b) that exhibit temporal locality over time, and MIMO channels (Figure 2-1c) that illustrate both temporal locality and spatial locality across streams of adjacent frequency channels.

Central to a continuous learning system is a data streamer, which periodically sends new data samples to an adaptation service. The adaptation service then fine-tunes a lightweight model based on the recent queries and uses this updated model for future queries.

The core idea in continuous learning is to leverage such temporal localities to boost the accuracy of the lightweight models by specializing them for each specific local distribution in time. Figure 2-2 can be used to illustrate the high-level components of a continuous learning system. It includes a data streamer that periodically sends new samples from the query stream for the adaptation process. The adaptation service uses the recent queries to fine-tune (a copy of) the inference lightweight model for the current queries, and uses the updated lightweight model for inference on future queries from the data streamer. The adaptation process is central to continuous learning, dynamically specializing a lightweight model designed for low computational complexity and fast inference, based on the current queries.

Adaptation Techniques

Our work is related to online learning [46] algorithms for minimizing dynamic or tracking regret [47–49]. Dynamic regret compares the performance of an online learner to a sequence of optimal solutions. In our case, the goal is to track the performance of the best lightweight model at each point in a video. Several theoretical works have studied online gradient descent algorithms in this setting with different assumptions about the loss functions [50, 51]. Other work has focused on the "experts" setting [52–54], where the learner maintains multiple models and uses the best of them at each time. Our approach is based on online gradient descent because tracking multiple models per video at a server is expensive.

Online Knowledge Distillation. Knowledge distillation is a technique where a smaller model (student) is trained using knowledge gleaned from a larger, pre-trained model (teacher). Online knowledge distillation extends this concept; as new data becomes available, the student model is continually updated, enhancing its predictive capacity and effectively mimicking the teacher model's performance for the specific locality in time.

Inverse Learning. Inverse learning is another model adaptation approach where a model is trained to solve an inverse problem by learning the mapping from outputs back to inputs. This strategy is particularly useful when direct inference from input to output is complex or non-deterministic.

Unsupervised Domain Adaptation. Domain adaptation methods [55, 56] aim to address the discrepancies between training and test data distributions. In a common approach, an unsupervised algorithm fine-tunes the model using the entire test dataset. However, the accuracy attained through this adaptation method is typically lower than that achieved by supervised methods.

The subsequent chapters will delve deeper into these concepts, discussing the methodologies used for continuous learning and knowledge distillation, their implementation, and their performance. We will also explore the application of these methodologies in different scenarios, examining the associated benefits and challenges.

2.2.2 Interplay between Data Locality and Updates Timing

Optimizing a model's accuracy requires a balance between adapting it to current data and avoiding overfitting, which can lower the model's performance when data changes. This balancing act hinges on the rate of data change and the model's capacity. For instance, a smaller model might benefit from frequent updates even if it risks overfitting. We demonstrate in this section that smaller models, indeed, gain more from continuous retraining.

To delve into these nuances, let's refer to the adaptation process in the continuous learning framework from §2.2.1. We have two parameters to tweak:

1. T_{update}: the *model update interval*. The model retrains every T_{update} seconds.

2. $T_{horizon}$: the *training horizon*. Each update trains the model using frames from the

previous $T_{horizon}$ seconds of video.

There's an interrelation between these parameters. If $T_{horizon}$ is small, the model might overfit, necessitating more frequent updates (a smaller T_{update}). However, with a larger $T_{horizon}$, models could better generalize, allowing for less frequent updates. Yet, an overly large $T_{horizon}$ can be detrimental; a model might struggle to generalize across diverse frames, leading to decreased accuracy.

Figure 2-3 exemplifies this dynamic for the task of video semantic segmentation. We look at two model variations: (i) DeeplabV3 with a MobileNetV2 backbone; (ii) A condensed version of the above with half the channels in each convolutional layer.

Selecting 50 uniformly spaced points over a driving scene video from Los Angeles, we train both models on frames from interval $[t - T_{horizon}, t)$, then test them on frames in $[t, t + T_{update})$ (setting T_{update} at 16 seconds).

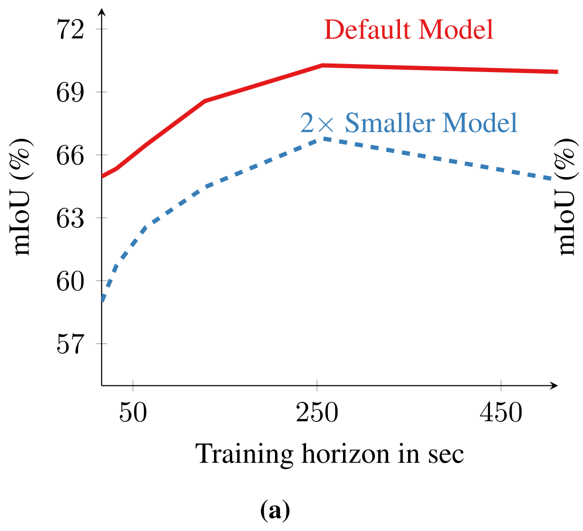
Figure 2-3a plots the average accuracy of both models against different $T_{horizon}$ values. The smaller model reaches its best accuracy around $T_{horizon} \approx 256$ seconds, then declines with a larger $T_{horizon}$ due to limited model capacity. The standard model follows a similar trend, but its accuracy drop is more gradual for bigger $T_{horizon}$ values.

Lastly, Figure 2-3b displays how different training horizons influence the necessary model update frequency for retaining high accuracy. Using the same video, we chart accuracy against the model update interval (T_{update}) for default model trainings at $T_{horizon} = 16, 64, 256$ seconds. Predictably, more regular updates boost accuracy across the board. However, the accuracy for models with a short training horizon ($T_{horizon} = 16$ seconds) falls off significantly with longer T_{update} intervals.

2.2.3 Other Adaptation Frameworks

Lifelong Learning. The goal of lifelong learning [57] is to accumulate knowledge over time [58]. Hence the main challenge is to improve the model based on new data over time, while not forgetting data observed in the past [59, 60]. However, lightweight models often do not have enough capacity to perform well on broad distribution of data and, as we discuss later in this chapter, some forgetting will be helpful if we have the systems support for continuous adaptation. However, as lightweight models have limited capacity, in continuous we aim to track the best model at each point in time, and these models are allowed not to have the same performance on the old data.

Federated Learning. Another body of work on improving edge models over time is federated learning [61], in which the training mainly happens on the edge devices and device updates are aggregated at a server. The server then broadcasts the aggregated model back



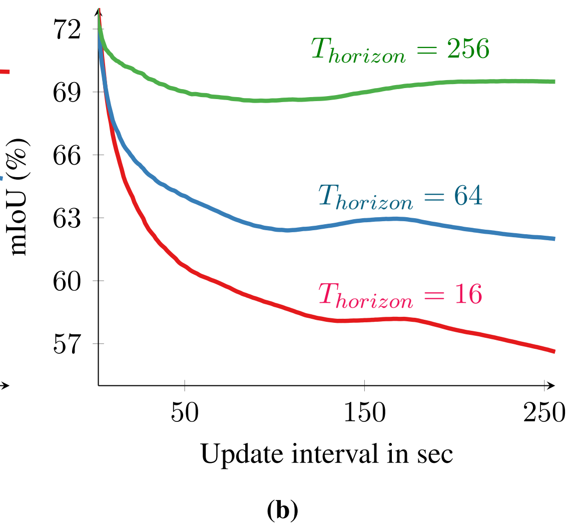


Figure 2-3: Impact of training horizon and model update interval on the mean-intersection-over-union (mIoU) accuracy for semantic segmentation.

to all devices. Such updates happen at a time scale of hours to days [62], and they aim to learn better generalizable models that incorporate data from all edge devices. In contrast, the trainings in our framework takes place at a time scale of a couple of seconds and intends to improve the accuracy of an individual edge device's model for its particular data distribution at the time.

Meta Learning. Meta learning [63–65] algorithms aim to learn models that can be adapted to any target task in a set of tasks, given only few samples (shots) from that task. Meta learning is not a natural framework for continual model specialization for a wide range of signals. First, as data streams have temporal coherence, there is little benefit in handling an arbitrary order of task arrival. Indeed, it is more natural to adapt the latest model over time instead of always training from a meta-learned initial model.[1] Second, training such a meta model usually requires two nested optimization steps [63], which would significantly increase the adaptation compute overhead. Finally, most meta learning work considers a finite set of tasks but this notion is not well-suited to video.

2.3 Challenges of Continuous Learning at the Edge

Continuous learning on edge devices introduces several computational challenges that need to be addressed to achieve efficient and effective model adaptation. These challenges include memory and compute overhead, teacher model constraints, and frequent retraining

[1] An exception is sudden changes in the data stream that we discuss in chapter 6.

requirements.

Continuous learning often relies on gradient descent algorithms, which can be computationally and memory-intensive. This poses a challenge for edge devices with limited computational capabilities and memory resources. Executing resource-intensive processes on edge devices may lead to performance degradation, increased energy consumption, and limitations in the size of models that can be accommodated. Therefore, it is crucial to develop techniques that optimize memory usage and computational efficiency during continuous learning on edge devices.

In online knowledge distillation methods, the teacher model, which provides guidance to the student model, may be too large to fit on edge devices. The constraints imposed by the limited resources of edge devices necessitate careful resource management and efficient utilization of available computational resources. It is essential to develop strategies that enable effective knowledge transfer from the teacher model to the student model while considering the resource limitations of edge devices.

In scenarios where data distributions change frequently, such as in video streams, frequent retraining is necessary to adapt the model to the evolving environment. However, performing frequent retraining on edge devices can impose a significant computational cost, further highlighting the need for efficient computational strategies. The challenge lies in finding a balance between the frequency of model updates and the associated computational overhead. Techniques that enable adaptive and efficient retraining strategies are required to maintain model accuracy while minimizing the computational burden on edge devices.

Chapter 3

MMNet: Adaptive Neural Signal Detection for Massive MIMO

3.1 Overview

The fifth generation of cellular communication systems (5G) promises an order of magnitude higher spectral efficiency (measured in bits/s/Hz) than legacy standards such as Long Term Evolution (LTE) [66]. One of the key enablers of this better efficiency is Massive Multiple-Input Multiple-Output (MIMO) [67], in which a base station (BS) equipped with a very large number of antennas (around 64–256) simultaneously serves multiple single-antenna user equipments (UEs) on the same time-frequency resource.

Legacy systems already use MIMO [68], but this is the first time it will be deployed on such a large scale, creating significant challenges for *signal detection*. The goal of signal detection is to infer the transmitted signal vector $\mathbf{x}$ from the vector $\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}$ received at the BS antennas, where $\mathbf{H}$ is the channel matrix and $\mathbf{n}$ is Gaussian noise. Traditional MIMO signal detection schemes with strong performance [69–72] are feasible only for small systems and have prohibitive complexity for massive MIMO deployments. Thus, there is a need for low-complexity signal detection schemes that can both perform well and scale to large system dimensions.

In recent work, researchers have proposed several learning approaches for MIMO signal detection. Samuel *et al.* [11, 73] achieved impressive results with a deep neural network architecture called DetNet, e.g., matching the performance of a semidefinite relaxation (SDR) baseline for i.i.d. Gaussian channel matrices while running 30$\times$ faster. He *et al.* [74] introduced OAMPNet, a model inspired by the Orthogonal AMP algorithm [75], and demonstrated strong performance on both i.i.d. Gaussian and small-sized correlated channel matrices based on the Kronecker model [76]. DetNet and OAMPNet are both trained offline: they try to learn a single detector during training for a family of channel matrices (e.g., i.i.d. Gaussian channels).

In this chapter we show that neither approach is effective in practice. We conduct extensive experiments using a dataset of channel realizations from the 3GPP 3D MIMO channel [77], as implemented in the QuaDRiGa channel simulator [78]. Our results show that DetNet's training is unstable for realistic channels, while OAMPNet suffers a large performance gap (4–7dB at symbol error rate of 10^{-3}) compared to the optimal Maximum-Likelihood detector on these channels. Both models (as well as several classical baselines) perform well in simpler settings used for evaluation in prior work (e.g., i.i.d. Gaussian channels, low-order modulation schemes). Our results demonstrate the difficulty of learning a single detector that generalizes across a wide range of realistic channel matrices (esp. poorly-conditioned channels that are difficult to invert).

Motivated by these findings, we revisit MIMO detection from an online learning perspective. We ask: *Can a receiver optimize its detector for every realization of the channel matrix?* Intuitively, such an approach could perform better than using a fixed detector for a wide variety of channel matrices. However, conventional wisdom suggests that training a MIMO detector online is impossible because of the stringent performance requirements [11].

Our design, MMNet, overcomes this challenge with two key ideas. First, it uses a neural network architecture that strikes a balance between flexibility and complexity. Prior neural network architectures for MIMO detection are poorly suited to online training. DetNet is a large model with 1-10 million parameters depending on the system size and modulation scheme, making it prohibitively expensive to train online. OAMPNet, on the other hand, is very restrictive, adding only 2 trainable parameters per iteration to the OAMP algorithm. Since the OAMP algorithm requires strong assumptions about channel matrices (unitarily-invariant channels [75]), OAMPNet, even with online training, performs poorly on channels that deviate from the assumptions.

MMNet's neural network is based on iterative soft-thresholding algorithms, a popular class of solutions to "linear-inverse" problems [79–81] like MIMO detection. These algorithms repeatedly refine an estimate of the signal by alternating between a linear detector and a non-linear denoising step. By preserving the core components of these algorithms in MIMO detection, such as a simple denoiser that is optimal for uncorrelated Gaussian noise, MMNet avoids the pitfalls of overly general neural network architectures like DetNet. At the same time, unlike OAMPNet, MMNet provides adequate flexibility in the architecture through trainable parameters that can be optimized for each channel realization.

MMNet's second key idea is an online training algorithm that exploits the locality of channel matrices at a receiver in both the frequency and time domains. By leveraging spectral and temporal locality, MMNet accelerates training by more than two orders of magnitude compared to naively retraining the neural network from scratch for each channel realization.

Taken together, these ideas enable MMNet to achieve performance within $\sim$2dB of the optimal Maximum-Likelihood detector with 10-15$\times$ less computational complexity than the second-best scheme, OAMPNet. On random i.i.d. Gaussian channels, an even simpler version of MMNet (MMNet-iid) with 100$\times$ less complexity than OAMPNet and DetNet achieves near-optimal performance without any retraining.

We empirically analyze the dynamics of errors across different layers of MMNet and OAMPNet to understand how MMNet achieves higher detection accuracy. Our analysis reveals that MMNet "shapes" the distribution of noise at the input of the denoisers to ensure they operate effectively. In particular, as signals propagate through the MMNet neural network, the noise distribution at the input of the denoiser stages approaches a Gaussian distribution, create precisely the conditions in which the denoisers can attenuate noise maximally.

The rest of this chapter is organized as follows. Section 3.2 provides background on classical and learning-based detection schemes, and introduces a general iterative framework that can express many of these algorithms. Section 3.4 introduces the MMNet design in addition to a simple variant for i.i.d. channels. Section 3.5 shows performance results of detection algorithms on i.i.d. Gaussian and 3GPP MIMO channels for different modulations. Section 3.6 discusses the error dynamics of MMNet and empirically studies why it performs better than OAMPNet. Section 3.7 introduces MMNet's online training algorithm. The code to reproduce our results is available at https://github.com/mehrdadkhani/MMNet.

3.2 Background

3.2.1 Notation

We will use lowercase symbols for scalars, bold lowercase symbols for column vectors and bold uppercase symbols to denote matrices. Symbols $\{\theta, \boldsymbol{\theta}, \boldsymbol{\Theta}\}$ are used to represent the parameters of trainable models. The transpose and pseudo-inverse of matrix $\boldsymbol{A}$ are denoted by $\boldsymbol{A}^H$ and $\boldsymbol{A}^+ = (\boldsymbol{A}^H\boldsymbol{A})^{-1}\boldsymbol{A}^H$ respectively. $\mathbf{I}_n$ stands for identity matrix of size n.

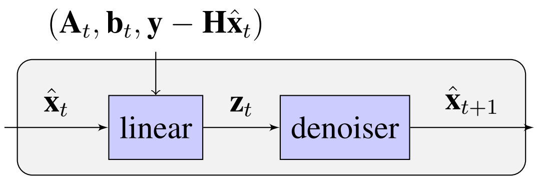

$(\mathbf{A}_t, \mathbf{b}_t, \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_t)$
$\hat{\mathbf{x}}_t$
linear
$\mathbf{z}_t$
denoiser
$\hat{\mathbf{x}}_{t+1}$

Figure 3-1: A block of an iterative detector in our general framework. Each block contains a linear transformation followed by a denoising stage.

3.2.2 The MIMO Signal Detection Problem

We consider a communication channel from N_t single-antenna transmitters to a receiver equipped with N_r antennas. The received vector $\mathbf{y} \in \mathbb{C}^{N_r}$ is given as

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}, \tag{3.1}$$

where $\mathbf{H} \in \mathbb{C}^{N_r \times N_t}$ is the channel matrix, $\mathbf{n} \sim \mathcal{CN}(0, \sigma^2 \mathbf{I}_{N_r})$ is complex Gaussian noise, and $\mathbf{x} \in \mathcal{X}^{N_t}$ is the vector of transmitted symbols. $\mathcal{X}$ denotes the finite set of constellation points. We assume that each transmitter chooses a symbol from $\mathcal{X}$ uniformly at random, and all transmitters use the same constellation set. Further, as is standard practice, we assume that the constellation set $\mathcal{X}$ is given by a quadrature amplitude modulation (QAM) scheme [82]. All constellations are normalized to unit average power (e.g., the QAM4 constellation is $\{\pm\frac{1}{\sqrt{2}} \pm j\frac{1}{\sqrt{2}}\}$).

The channel matrix $\mathbf{H}$ is assumed to be known at the receiver. The goal of the receiver is to compute the maximum likelihood (ML) estimate $\hat{\mathbf{x}}$ of $\mathbf{x}$:

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x} \in \mathcal{X}^{N_t}} ||\mathbf{y} - \mathbf{H}\mathbf{x}||_2. \tag{3.2}$$

The optimization problem in eq. (3.2) is NP-hard due to the finite-alphabet constraint $\mathbf{x} \in \mathcal{X}^{N_t}$ [83]. Therefore, over the last three decades, researchers have proposed a variety of detectors with differing levels of complexity. We refer the interested reader to [70, 71] for a comprehensive overview of MIMO detection schemes.

3.2.3 An iterative framework for MIMO detection

We focus on a class of iterative estimation algorithms for solving eq. (3.2) shown in Figure 3-1. Each iteration of these algorithms comprises the following two steps:

$$\text{General Iteration:} \qquad \begin{aligned} \mathbf{z}_t &= \hat{\mathbf{x}}_t + \mathbf{A}_t(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_t) + \mathbf{b}_t \\ \hat{\mathbf{x}}_{t+1} &= \eta_t\left(\mathbf{z}_t\right). \end{aligned} \tag{3.3}$$

The first step takes as input $\hat{\mathbf{x}}_t$, a current estimate of $\mathbf{x}$, and the *residual* error $\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_t$, and applies a *linear* transformation to obtain an intermediate signal $\mathbf{z}_t$. In the second step, a *non-linear* "denoiser" is applied to $\mathbf{z}_t$ to produce $\hat{\mathbf{x}}_{t+1}$, a new estimate of $\mathbf{x}$ that is used as the input for the next iteration. Together, the linear and denoising steps aim to improve the quality of the estimate $\hat{\mathbf{x}}_t$ from one iteration to the next. In this chapter we use the terms *iteration*, *layer*, and *block* interchangeably to refer to one complete iteration (the linear step followed by the non-linear denoiser). All algorithms discussed assume $\hat{\mathbf{x}}_0 = 0$.

The denoiser is a non-linear function $\eta_t \colon \mathbb{C}^{N_t} \to \mathbb{C}^{N_t}$ in general, however, most algorithms apply the same denoising function $\beta_t \colon \mathbb{C} \to \mathbb{C}$ to each element of $\mathbf{z}_t$. A natural choice for the denoising function is the minimizer of $\mathbb{E}[\|\hat{\mathbf{x}} - \mathbf{x}\|_2 | \mathbf{z}_t]$, which is given by:

$$\eta_t(\mathbf{z}_t) = \mathbb{E}[\mathbf{x}|\mathbf{z}_t]. \tag{3.4}$$

Optimal denoiser for Gaussian noise: Several existing MIMO detection schemes assume that the noise at the input of the denoiser $\mathbf{z}_t - \mathbf{x}$ has an i.i.d. Gaussian distribution with diagonal covariance matrix $\sigma_t^2 \mathbf{I}_{N_t}$. In this case, the optimal element-wise denoising function derived from eq. (3.4) has the form

$$\beta_t(z; \sigma_t^2) = \frac{1}{Z} \sum_{x_i \in \mathcal{X}} x_i \exp\left(-\frac{\|z - x_i\|^2}{\sigma_t^2}\right), \tag{3.5}$$

where $Z = \sum_{x_j \in \mathcal{X}} \exp\left(-\frac{\|z - x_j\|^2}{\sigma_t^2}\right)$. The standard deviation of input noise at the denoisers, σ_t, generally varies from iteration to iteration, and depends on the linear steps in each iteration. Different algorithms use different methods to estimate σ_t. In the rest of this chapter, $\eta_t(\cdot; \sigma_t)$ refers to a denoiser which applies eq. (3.5) to each element of its input vector.

3.3 Related Work

In this section we briefly describe several algorithms for MIMO detection. We begin with traditional, non-learning approaches (§3.3.1) and then discuss recent deep learning proposals (§3.3.2). We show how many of these algorithms can be expressed in the iterative framework discussed above.

3.3.1 Classical MIMO detection algorithms

Linear

The simplest method to approximately solve eq. (3.2) is to relax the constraint of $\mathbf{x} \in \mathcal{X}^{N_t}$ to $\mathbf{x} \in \mathbb{C}^{N_t}$ and then round the relaxed solution to the closest point on the constellation:

$$\text{Linear:} \qquad \begin{aligned} \mathbf{z} &= \arg\min_{\mathbf{x}\in\mathbb{C}^{N_t}} \|\mathbf{y} - \mathbf{H}\mathbf{x}\|_2 = \mathbf{H}^{+}\mathbf{y} \\ \hat{\mathbf{x}} &= \arg\min_{\mathbf{x}\in\mathcal{X}^{N_t}} \|\mathbf{x} - \mathbf{z}\|_2. \end{aligned} \tag{3.6}$$

Rounding each component of $\mathbf{z}$ to the closest point in the constellation set $\hat{\mathbf{x}}$ leads to the well-known zero-forcing (ZF) detector, which is equivalent to a single step of eq. (3.3) with initial condition of $\hat{\mathbf{x}}_0 = 0$, $\mathbf{A}_0 = \mathbf{H}^{+}$, $\mathbf{b}_0 = 0$, and a hard-decision denoiser with respect to the points in the constellation. Other widely-used single-step linear detectors include the matched filter and the minimum mean square error (MMSE) detectors [67] with $\mathbf{A}_0 = \mathbf{H}^H$ and $\mathbf{A}_0 = (\mathbf{H}^H\mathbf{H} + \sigma^2\mathbf{I}_{N_t})^{-1}\mathbf{H}^H$, respectively. Linear detectors are attractive for practical implementation because of their low complexity, but they perform substantially worse than the optimal detector.

We can also perform the optimization in eq. (3.6) in multiple iterations using gradient descent. The gradient of the objective function in the first equation of eq. (3.6) with respect to $\mathbf{x}$ is $-2\mathbf{H}^H(\mathbf{y} - \mathbf{H}\mathbf{x})$. Hence, if we set $\mathbf{A}_t$ to $2\alpha\mathbf{H}^H$ and $\mathbf{b}_t = 0$, the linear step of eq. (3.3) is equivalent to minimizing $||\mathbf{y} - \mathbf{H}\mathbf{x}||_2$ using gradient descent with step size α. This is followed by a projection onto the constellation set in the denoising step. If we had a compact convex constellation set, this projected gradient descent procedure is guaranteed to converge to the global optimum. Discrete constellation sets, however, are not compact convex. Nonetheless, solving the linear least squares problem in eq. (3.6) iteratively may be desirable to avoid the cost of computing the pseudo-inverse of the channel matrix.

Approximate Message Passing (AMP)

MIMO detection can, in principle, be solved through belief propagation (BP) if we consider a bipartite graph representation of the model in eq. (3.1) [84]. BP on this graph requires $\mathcal{O}(N_rN_t)$ update messages in each iteration, which would be prohibitive for large system dimensions. In the large system limit, Jeon *et al.* [81] introduce approximate message passing (AMP) as a lower complexity inference algorithm for solving eq. (3.2) for i.i.d. Gaussian channels. AMP reduces the number of messages in each iteration to $\mathcal{O}(N_r + N_t)$.

The algorithm performs the following sequence of updates:

$$\text{AMP:} \qquad \begin{aligned} \mathbf{z}_t &= \hat{\mathbf{x}}_t + \mathbf{H}^H(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_t) + \mathbf{b}_t \\ \mathbf{b}_t &= \alpha_t \left(\mathbf{H}^H(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_{t-1}) + \mathbf{b}_{t-1}\right) \\ \hat{\mathbf{x}}_{t+1} &= \eta_t \left(\mathbf{z}_t; \sigma_t\right). \end{aligned} \tag{3.7}$$

To express AMP in our iterative framework, use $\mathbf{A}_t = \mathbf{H}^H$ as the linear operator. The vector $\mathbf{b}_t$ is known as the *Onsager* term. The scalar sequences σ_t and α_t are computed analytically given the SNR and system parameters (constellation, number of transmitters and receivers); see [85] for details. The denoising function $\eta_t(\cdot; \sigma_t)$ applies the optimal denoiser for Gaussian noise in eq. (3.5) to each element of the vector $\mathbf{z}_t$. Jeon *et al.* [81] prove that AMP is asymptotically optimal for large i.i.d. Gaussian channel matrices.

Orthogonal AMP (OAMP) [75] is another scheme proposed to relax the i.i.d. Gaussian channel assumption in the original AMP algorithm. OAMP assumes unitarily-invariant channel matrices [86] and operates as follows:

$$\text{OAMP:} \qquad \begin{aligned} \mathbf{z}_t &= \hat{\mathbf{x}}_t + \gamma_t \mathbf{H}^H \left(v_t^2 \mathbf{H}\mathbf{H}^H + \sigma^2 \mathbf{I}\right)^{-1} (\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_t) \\ \hat{\mathbf{x}}_{t+1} &= \eta_t \left(\mathbf{z}_t; \sigma_t^2\right) \end{aligned} \tag{3.8}$$

where $\gamma_t = N_t/\text{trace}\left(v_t^2 \mathbf{H}^H \left(v_t^2 \mathbf{H}\mathbf{H}^H + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{H}\right)$ is a normalizing factor and v_t^2 is proportional to the average noise power at the denoiser output at iteration t and can be computed given the SNR and system dimensions [75]. Notice that OAMP requires a matrix inverse operation in each iteration, making it more expensive computationally than AMP.

Other techniques

Several detection schemes relax the lattice constraint ($\mathbf{x} \in \mathcal{X}^{N_t}$) in eq. (3.2). For example, Semi-Definite Relaxation (SDR) [72] formulates the problem as a semi-definite program. Sphere decoding [69] conducts a search over solutions $\hat{\mathbf{x}}$ such that $||\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}||_2 \le r$. Increasing r covers a larger set of possible solutions, but this comes at the cost of increased complexity, approaching that of brute-force search. There is a large body of work on improvements to this idea which can be found in [70, 71]. While these approaches can perform well, their computational complexity is prohibitive for Massive MIMO systems with currently available hardware.

Another class of detector applies several stages of linear detection followed by interference subtraction from the observation $\mathbf{y}$. The V-BLAST scheme [87] does this by detecting the strongest symbols, which are then successively removed from $\mathbf{y}$. The drawbacks of this approach are error propagation of early symbol decisions and high complexity due to the N_t required stages, as well as the necessary reordering of transmitters after each step. Parallel interference cancellation (PIC) has been proposed to circumvent these problems. PIC jointly detects all transmitted symbols and then attempts to create an interference-free channel for each transmitter through the cancellation of all other transmitted symbols [88, 89]. A large system approximation of this approach was recently developed in [90] based on [91]. However, it is currently limited to binary phase shift keying (BPSK) modulation and leads to unsatisfactory performance for realistic system dimensions.

In summary, most existing techniques are too complex to be implemented at the scale required by next-generation Massive MIMO systems. On the other hand, light-weight techniques like AMP cannot handle correlated channel matrices. These limitations have motivated a number of learning-based proposals for MIMO detection, which we discuss next.

3.3.2 Learning-based MIMO detection schemes

DetNet

Inspired by iterative projected gradient descent optimization, Samuel *et al.* [11, 73] propose DetNet, a deep neural network architecture for MIMO detection. This architecture performs very well in case of i.i.d. complex Gaussian channel matrices and achieves the performance of state-of-the-art algorithms for lower-order modulation schemes, such as BPSK and QAM4. However, it is far more complex. The neural network is described by the following equations:

$$\text{DetNet:} \qquad \begin{aligned} \mathbf{q}_t &= \hat{\mathbf{x}}_{t-1} - \theta_t^{(1)}\mathbf{H}^H\mathbf{y} + \theta_t^{(2)}\mathbf{H}^H\mathbf{H}\hat{\mathbf{x}}_{t-1} \\ \mathbf{u}_t &= \left[\mathbf{\Theta}_t^{(3)}\mathbf{q}_t + \mathbf{\Theta}_t^{(4)}\mathbf{v}_{t-1} + \boldsymbol{\theta}_t^{(5)}\right]_+ \\ \mathbf{v}_t &= \mathbf{\Theta}_t^{(6)}\mathbf{u}_t + \boldsymbol{\theta}_t^{(7)} \\ \hat{\mathbf{x}}_t &= \mathbf{\Theta}_t^{(8)}\mathbf{u}_t + \boldsymbol{\theta}_t^{(9)} \end{aligned} \tag{3.9}$$

where $[x]_+ = \max(x, 0)$, which is also known as ReLU activation function [92], is applied element-wise [11].

Although DetNet's performance is promising, it has two main limitations. First, its heuristic nature makes it difficult to reason about how the neural network works, and how to extend its architecture, for example, to support spatially correlated channel matrices. Second, DetNet's architecture does not incorporate known properties of iterative methods and is thus unnecessarily complex. For example, similar to other iterative schemes, DetNet's neural network can be viewed to be performing a linear transformation followed by a non-linear projection in each iteration. DetNet's linear step computes $\boldsymbol{q}_t$, and the non-linear projection computes $\hat{\mathbf{x}}_t$ from $\boldsymbol{q}_t$.[1] However, unlike other iterative schemes that use the simple non-linear denoiser in eq. (3.5), DetNet's non-linear projection is a fully-connected 2-layer neural network that operates on an N_t-dimensional input vector. In fact, DetNet uses parameter matrices $\boldsymbol{\Theta}_t^{(3)}$ and $\boldsymbol{\Theta}_t^{(4)}$ to map the input of the projection to an even larger space before mapping it back to a N_t-dimensional vector.[2].

OAMPNet

He *et al.* [74] designed a learning-based iterative scheme based on the OAMP algorithm. OAMPNet adds two tuning parameters per iteration to the OAMP algorithm, as follows:

$$\text{OAMPNet:} \qquad \begin{aligned} \mathbf{z}_t &= \hat{\mathbf{x}}_t + \theta_t^{(1)}\mathbf{H}^H\left(v_t^2\mathbf{H}\mathbf{H}^H + \sigma^2\mathbf{I}\right)^{-1}(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_t) \\ \hat{\mathbf{x}}_{t+1} &= \eta_t\left(\mathbf{z}_t; \sigma_t^2\right). \end{aligned} \tag{3.10}$$

In the above equations, OAMPNet uses the second trainable parameter , $\theta_t^{(2)}$, in order to balance the estimate of denoisers input noise variance σ_t^2. OAMPNet uses the same denoisers used by AMP.

OAMPNet shows very good performance in the case of i.i.d. Gaussian channels, but it does not generalize to realistic channels with spatial correlations, as our experiments in section 3.5 show. The reason is that OAMPNet bases its architecture on OAMP, which requires a strict assumption about the system: unitarily-invariant channel matrices. Therefore, its performance degrades on realistic channel matrices that do not conform to this assumption. Further, like OAMP, OAMPNet must compute a matrix pseudo-inverse in each iteration and, therefore, its complexity is still quite high compared to schemes like AMP.

3.4 MMNet Design

MMNet is a neural-network-based signal detection scheme inspired by the iterative framework described in Section 3.2.3. Unlike prior approaches that use a single model for all channel matrices, MMNet is designed to be trained online for each realization of $\mathbf{H}$. In this approach, the receiver continually adapts its parameters as it observes new channel matrices. We demonstrate that online training is feasible in practice with a suitable neural

[1]The role of $\mathbf{v}_t$ in DetNet is unclear, and is not explained in [11].

[2]In the specific networks evaluated in [11], the $\mathbf{u}_t$ vector has up to $6N_t$ dimensions depending on the constellation.

network architecture by exploiting the fact that realistic channels exhibit locality in both the frequency and time domains. We introduce the neural network architecture in this section and discuss the training algorithm in Section 3.7.

The main idea behind MMNet's architecture is to strike a balance between flexibility and complexity in the linear and denoising components of each layer of the neural network. In the following, we present two different neural network architectures for (1) i.i.d. Gaussian and (2) arbitrary channels.

i.i.d. Gaussian channels: In the i.i.d. Gaussian case, the model is extremely simple:

$$\text{MMNet-iid:} \qquad \begin{aligned} \mathbf{z}_t &= \hat{\mathbf{x}}_t + \theta_t^{(1)}\mathbf{H}^H(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_t) \\ \hat{\mathbf{x}}_{t+1} &= \eta_t\left(\mathbf{z}_t; \sigma_t^2\right). \end{aligned} \tag{3.11}$$

Here, the denoiser is the optimal denoiser for Gaussian noise given in (3.5). MMNet-iid assumes the same distribution of noise at the input of the denoiser for all transmitted symbols and estimates its variance σ_t^2 according to

$$\begin{aligned} \sigma_t^2 = \frac{\theta_t^{(2)}}{N_t}\Bigg(& \frac{\|\mathbf{I} - \mathbf{A}_t\mathbf{H}\|_F^2}{\|\mathbf{H}\|_F^2}\left[\|\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_t\|_2^2 - N_r\sigma^2\right]_+ \\ & + \frac{\|\mathbf{A}_t\|_F^2}{\|\mathbf{H}\|_F^2}\sigma^2\Bigg), \end{aligned} \tag{3.12}$$

where $\mathbf{A}_t = \theta_t^{(1)}\mathbf{H}^H$. The intuition behind (3.12) is that the noise at the input of the denoiser at step t is comprised of two parts: (i) the residual error caused by deviation of $\hat{\mathbf{x}}_t$ from the true value of $\mathbf{x}$, and (ii) the contribution of the channel noise $\mathbf{n}$. The first component is amplified by the linear transformation $(\mathbf{I} - \mathbf{A}_t\mathbf{H})$, and the second component is amplified by $\mathbf{A}_t$. See [75, 81] for further details on this method for estimating noise variance.

This model has only two parameters per layer: $\theta_t^{(1)}$ and $\theta_t^{(2)}$. We discuss this model merely to illustrate that, for the i.i.d. Gaussian channel matrix case (which most prior work focused on for evaluation), a simple model that adds a small amount of flexibility to existing algorithms like AMP can perform very well. In fact, our results will show that, in this case, we do not even need to train the parameters of the model online for each channel realization; training offline over randomly sampled i.i.d. Gaussian channel suffices.

Arbitrary channels: The MMNet neural network for arbitrary channel matrices is as follows:

$$\text{MMNet:} \qquad \begin{aligned} \mathbf{z}_t &= \hat{\mathbf{x}}_t + \boldsymbol{\Theta}_t^{(1)}(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_t) \\ \hat{\mathbf{x}}_{t+1} &= \eta_t\left(\mathbf{z}_t; \boldsymbol{\sigma}_t^2\right) \end{aligned} \tag{3.13}$$

where $\mathbf{A}_t = \mathbf{\Theta}_t^{(1)}$ is an $N_t \times N_r$ complex-valued trainable matrix. In order to enable the model to handle cases in which different transmitted symbols have different levels of noise, we estimate the noise variance at the input of the denoiser corresponding to each transmitted signal as:

$$\boldsymbol{\sigma}_t^2 = \frac{\boldsymbol{\theta}_t^{(2)}}{N_t}\left(\frac{\|\mathbf{I} - \mathbf{A}_t\mathbf{H}\|_F^2}{\|\mathbf{H}\|_F^2}\left[\|\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}_t\|_2^2 - N_r\sigma^2\right]_+ \right. \\ \left. + \frac{\|\mathbf{A}_t\|_F^2}{\|\mathbf{H}\|_F^2}\sigma^2\right), \tag{3.14}$$

where the parameter vector $\boldsymbol{\theta}_t^{(2)}$ of size $N_t \times 1$ scales the noise variance by different amounts for each symbol.

MMNet concatenates T layers of the above form. We use the average L2-loss over all T layers in order to train the model, which is given by

$$L = \frac{1}{T}\sum_{t=1}^{T}\|\hat{\mathbf{x}}_t - \mathbf{x}\|_2^2. \tag{3.15}$$

MMNet's architecture provides many more degrees of freedom than the highly-constrained OAMPNet, but it is also much simpler than DetNet. In particular, MMNet uses a flexible linear transformation (which does not need to be linear in $\mathbf{H}$) to construct the intermediate signal z_t, but it applies the standard optimal denoiser for Gaussian noise in (3.5). Further, unlike OAMPNet, MMNet does not require any matrix inverse operations.

3.5 Experiments

In this section, we evaluate and compare the performance of MMNet with state-of-the-art schemes for both i.i.d. Gaussian and realistic channel matrices. These are our main findings:

1. On i.i.d. Gaussian channels, most schemes perform very well. MMSE, SDR, V-BLAST and DetNet are 1–3dB worse than the Maximum-Likelihood solution. AMP performance degrades for higher-order modulations in high SNRs. MMNet-iid and OAMPNet are both very close to Maximum-Likelihood in all experiment on these channels. MMNet-iid, however, has two orders of magnitude lower complexity than other learning-based schemes, OAMPNet and DetNet.

2. On realistic, spatially-correlated channel matrices, the performance of all existing

learning-based approaches degrades significantly. MMNet consistently achieves the smallest gap with Maximum-Likelihood. MMSE has a 8–10dB gap with Maximum-Likelihood on 64×16 channels. OAMPNet reduces this gap to 5dB, and MMNet closes the gap further to less than 2.2dB. DetNet and AMP perform poorly on realistic channels; with QAM4, for example, AMP achieves an SER of only 0.36 at 7dB while MMSE can bring down the SER to 0.033 in the same setting. The best SER that we achieve in this setting with DetNet before running into stability problems is 0.045, which is worse than MMSE. Finally, MMNet's performance is robust and degrades more gracefully to channel estimation errors than OAMPNet and MMSE

3.5.1 Methodology

We first briefly discuss the details of detection schemes used for comparison. Since some of these schemes (including MMNet) require training, we then discuss the process of generating data and training/testing on this data.

Compared Schemes

In our experiments, we compare the following schemes on QAM modulation:

- **MMSE:** Linear decoder that applies the SNR-regularized channel's pseudo inverse and rounds the output to the closest point on the constellation.
- **SDR:** Semidefinite programming using a rank-1 relaxation interior point method [93].
- **V-BLAST:** Multi-stage interference cancellation BLAST algorithm using Zero-Forcing as the detection stage introduced in [89].
- **AMP:** AMP algorithm for MIMO detection from Jeon *et al.* [81]. AMP runs 50 iterations of the updates described in (3.7). We verified that adding more iterations does not improve the results.
- **DetNet:** The deep learning approach introduced in [11]. The DetNet paper describes instantiations of the architecture for BPSK, QAM4 and QAM16; these neural networks have, on the order of 1–10M, trainable parameters depending on the size of the system and constellation set.
- **OAMPNet:** The OAMP-based architecture [74] implemented in 10 layers with 2 trainable parameters per layer and an inverse matrix computation at each layer.

- **MMNet-iid:** The simple mode described in (3.11). This scheme has only 2 scalar parameters per layer and does not require any matrix inversions. We implement this neural network with 10 layers.
- **MMNet:** Our design described in (3.13). It has 10 layers, and the total number of trainable parameters is $2N_t(N_r + 1)$ per layer, independent of constellation size. In the systems evaluated, this results in 20K-41K trainable parameters.
- **Maximum-Likelihood:** The optimal solver for (3.2) using a highly-optimized Mixed Integer Programming package Gurobi [94].

Dataset

Training and test data are generated through the model described in (3.1). In this model, there are three sources of randomness: the signal **x**, the channel noise **n** and the channel matrix **H**. Each transmitted signal **x** is generated randomly and uniformly over the corresponding constellation set. All transmitters are assumed to use the same modulation. The channel noise **n** is sampled from a zero-mean i.i.d. normal distribution with a variance that is set according to the operating SNR, defined as $\mathrm{SNR(dB)} = 10\log\left(\mathbb{E}[\|\mathbf{Hx}\|_2^2]/\mathbb{E}[\|\mathbf{n}\|_2^2]\right)$. For every training batch, the SNR(dB) is chosen uniformly at random in the desired operating SNR interval. This interval depends on the modulation scheme. For each modulation in each experiment, the SNR regime is chosen such that the best scheme other than Maximum-Likelihood can achieve a SER of 10^{-3}–10^{-2}.

The channel matrices **H** are either sampled from an i.i.d. Gaussian distribution (i.e., each column of **H** is a complex-normal $\mathcal{CN}(0, (1/N_r)\mathbf{I}_{N_r})$), or they are generated via the realistic channel simulation described below.

We study two system size ratios (N_t/N_r) of 0.25 and 0.5, with the total number of receivers fixed at $N_r = 64$. These are typical values for 4G/5G base stations in urban cellular deployments. For the case of realistic channels, we generate a dataset of channel realizations from the 3GPP 3D MIMO channel model [77], as implemented in the QuaDRiGa channel simulator [78].[3] We consider a base station (BS) equipped with a rectangular planar array consisting of 32 dual-polarized antennas installed at a height of $25\,\mathrm{m}$. The BS is assumed to cover a $120°$-cell sector of radius $500\,\mathrm{m}$ within which $N_t \in \{16, 32\}$ single-antenna users are dropped randomly. A guard distance of $10\,\mathrm{m}$ from the BS is kept. Each user is then assumed to move along a linear trajectory with a speed of $1\,\mathrm{m/s}$. Channels are sampled every $\lambda/4\,\mathrm{m}$ at a center frequency of $2.53\,\mathrm{GHz}$ to obtain sequences of length 100.

[3]The simulation results were generated using QuaDRiGa Version 2.0.0-664.

Each channel realization is then converted to the frequency domain assuming a bandwidth of $20\,\text{MHz}$ and using 1024 sub-carriers from which only every fourth is kept, resulting in $F = 256$ effective sub-carriers. We gather a total of 40 user drops, resulting in 40×256 length 100 sequences of channel matrices (i.e., 1M channel matrices in total). Since the path loss can vary dramatically between different users, we assume perfect power control, which normalizes the average received power across antennas and sub-carriers to one. Denote $\mathbf{H}[f, k]$ as the kth column of $\mathbf{H}$ on sub-carrier f. Our normalization ensures that

$$\frac{1}{N_r N_t F} \sum_{k=1}^{N_t} \sum_{f=1}^{F} \|\mathbf{H}[f, k]\|^2 = 1.$$

Training

MMNet, DetNet, and OAMPNet require training and were implemented in TensorFlow [95]. We have converted eq. (3.2) to its equivalent real-valued representation for TensorFlow implementations (cf. [96, Sec. II]). DetNet and OAMPNet are both trained as described in the corresponding publications (i.e., batch size of 5K samples). We trained each of the latter two algorithms for 50K iterations.

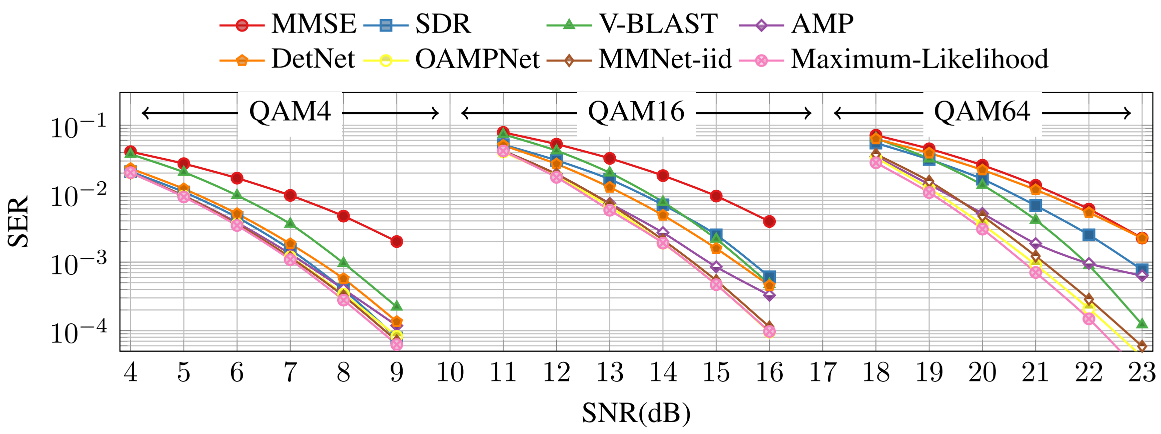
To train MMNet, we use the Adam optimizer [97] with a learning rate of 10^{-3}. Each training batch has a size of 500 samples. We train MMNet for 1K iterations on each realization of $\mathbf{H}$ in the naive implementation. In Section 3.7.2, we exploit frequency and time domain correlations to reduce the training requirement to 9 iterations per channel matrix.

In spatially correlated channels, we perform an additional 5K iterations of training with a batch size of 5K samples for each realization of $\mathbf{H}$ on the pre-trained OAMPNet model, in order to have a fair comparison against MMNet's online training. However, as we found that this extra training does not meaningfully improve the performance of OAMPNet, we do not count this re-training overhead in the complexity of OAMPNet algorithm in section 3.7.

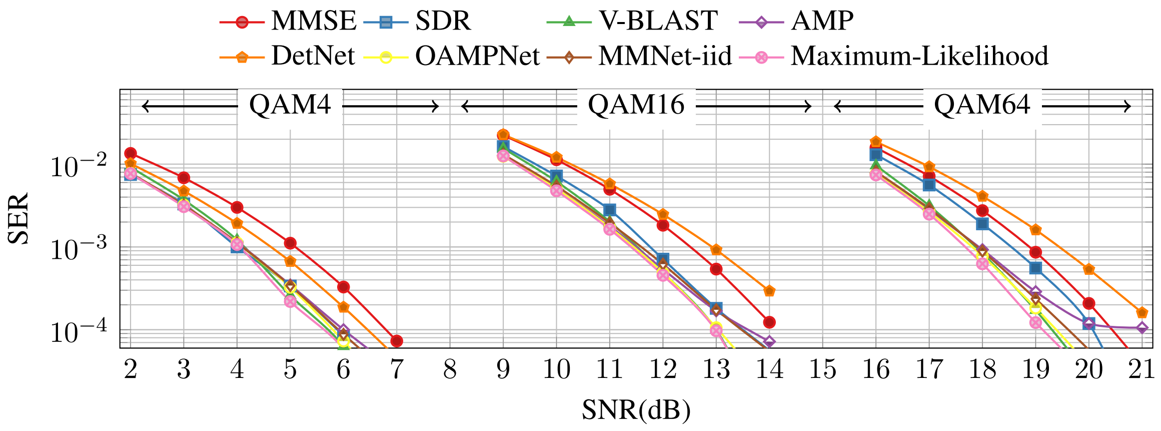
For i.i.d. Gaussian channels, MMNet-iid is not trained per channel realization $\mathbf{H}$. Instead, we use 10K iterations with a batch size of 500 samples to train a single MMNet-iid neural network, which we then test on new channel samples.

3.5.2 Results

We compare different schemes along two axes: performance and complexity. In this section, we focus on performance, leaving a comparison of complexity to Section 3.7. We use the SNR required to achieve an SER of 10^{-3} as the primary performance metric. In practice, most error correcting schemes operate around an SER of 10^{-3}–10^{-2}, so this is the



(a) 32 Transmit, 64 Receive Antennas



(b) 16 Transmit, 64 Receive Antennas

Figure 3-2: SER vs. SNR of different schemes for three modulations (QAM4, QAM16 and QAM64) and two system sizes (32 and 16 transmitters, 64 receivers) with i.i.d. Gaussian channels.

relevant regime for MIMO detection.

i.i.d. Gaussian channels

Figure 3-2 shows the SER vs. SNR of the state-of-the-art MIMO schemes on i.i.d. channels for two system sizes: 32 and 16 transmitters (Figure 3-2a and Figure 3-2b, respectively), and 64 receivers.

We make the following observations:

1. The SNR required to achieve a certain SER increases by 2–3dB as we double the number of transmitters (notice the different range of x-axes). The reason is that there is higher interference with more transmitters.

2. There is a 2–3dB performance gap between Maximum-Likelihood and MMSE across all modulations for $N_t = 32$. However, this gap decreases to 1dB for $N_t = 16$, because of the lower interference in this case.

3. Multiple schemes perform similarly to Maximum-Likelihood, especially at lower-order modulations like QAM4. As we move to QAM64, the performance of several schemes degrades compared to Maximum-Likelihood.

4. SDR performs better than MMSE, but its gap with Maximum-Likelihood increases with modulation order.

5. V-BLAST achieves almost the optimal performance across all modulations when we have 16 transmit antennas. However, its performance is sensitive to system size and degrades when we increase the number of transmitters to 32.

6. AMP is near-optimal in many cases (recall that, theoretically, AMP is asymptotically optimal for i.i.d. Gaussian channels as the system size increases). However, it suffers from robustness issues at higher SNR levels, especially with higher-order modulations like QAM64.

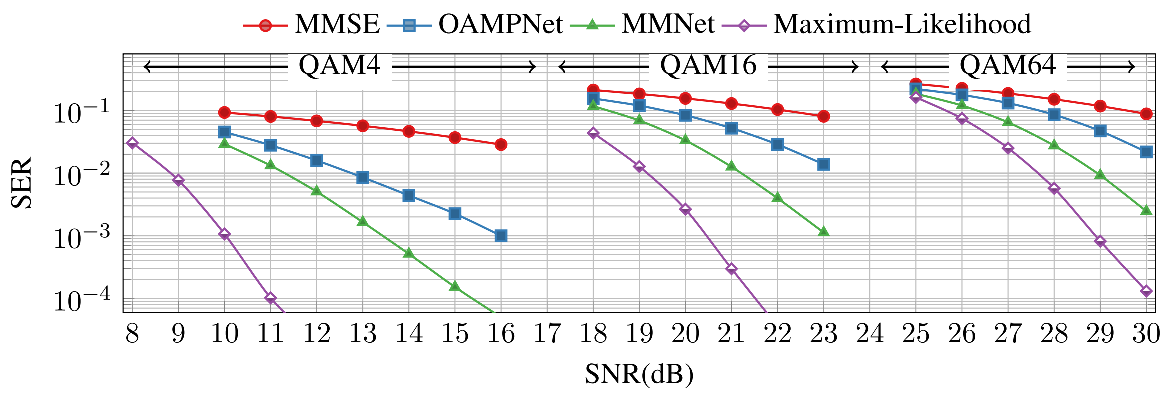
7. DetNet has a good performance on QAM4, but its gap with Maximum-Likelihood increases as we move to QAM16 and QAM64. With QAM64, DetNet performs even worse than MMSE for $N_t = 16$.

8. OAMPNet and the simple MMNet-iid approach are both very close to Maximum-Likelihood across different modulations over a wide range of SNRs, even though these models have only two parameters per layer.

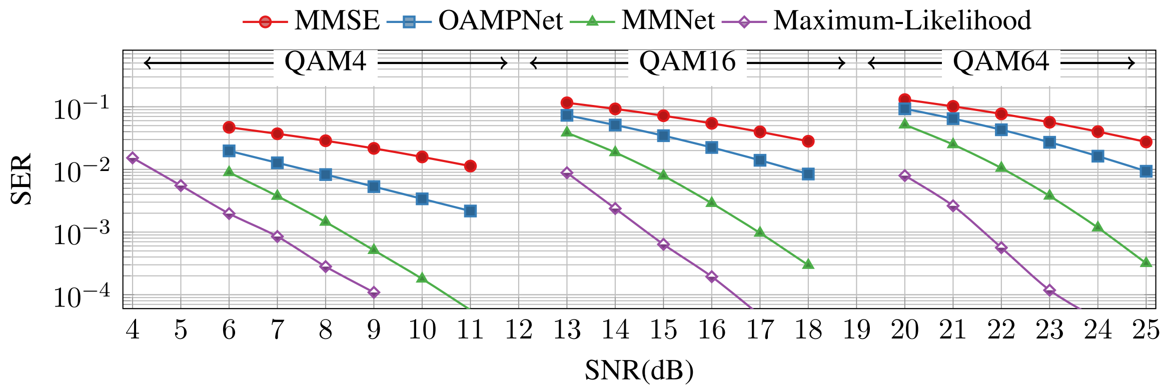
In summary, these results show that for i.i.d. Gaussian channel matrices, adding just a small amount of flexibility via tuning parameters to existing iterative schemes like AMP can result in equivalent or improved performance over much more complex deep learning models like DetNet. Further, these models even outperform classical algorithms like SDR.

Realistic channels

Figure 3-3 shows the results for the realistic channels derived using the 3GPP 3D MIMO channel model. We consider only MMSE (as a baseline), OAMPNet, MMNet and Maximum-Likelihood. As shown in the i.i.d. Gaussian case, schemes like SDR, V-BLAST and DetNet



(a) 32 Transmit, 64 Receive Antennas



(b) 16 Transmit, 64 Receive Antennas

Figure 3-3: SER vs. SNR of different schemes for three modulations (QAM4, QAM16 and QAM64) and two system sizes (32 and 16 transmitters, 64 receivers) with 3GPP MIMO channels.

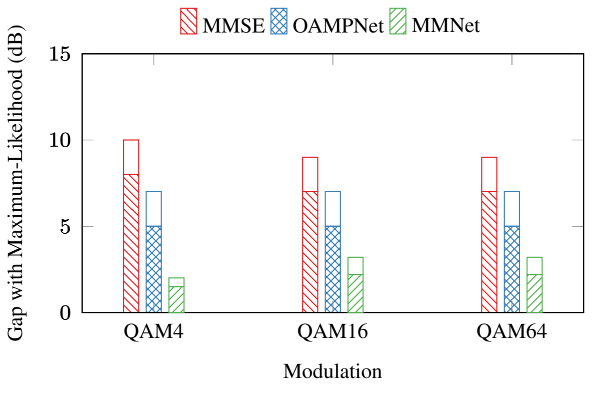
do not perform as well as the OAMPNet baseline.[4] Also, AMP is not designed for correlated channels and is known to perform poorly on them (see discussion in section 3.6).

We make the following observations:

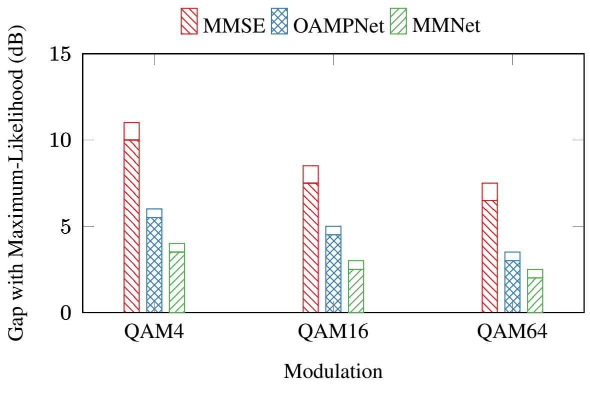
1. There is a much larger gap with Maximum-Likelihood for all detection schemes on these channels compared to the i.i.d. case.

2. To achieve comparable SER, we require 4–7dB increase in SNR relative to the i.i.d. case in Figure 3-2. Also, doubling the number of transmitters from 16 to 32 incurs about a 5dB penalty in SNR for each scheme in this case (compare with 2–3dB in i.i.d. case.)

[4]We tried to train DetNet for realistic channels and ran into significant difficulty with stability and convergence in training.



(a) 3GPP MIMO $N_t = 16$



(b) 3GPP MIMO $N_t = 32$

Figure 3-4: SNR requirement gap with Maximum-Likelihood at SER of 10^{-3}. The total bar height shows the 90th-percentile gap (over different channels) while the hatched section depicts the average.

3. MMSE exhibits a relatively flat SER vs. SNR curve. For example, it requires 4dB SNR increase on QAM16 to go from an SER of 20% to 10% on 64×32 channels.

4. OAMPNet's performance improves faster than MMSE as the SNR increases. Compared to MMSE, OAMPNet can achieve a similar SER at 2–3dB lower SNR.

5. MMNet outperforms MMSE and OAMNet schemes for both system sizes and in all modulations.

In Figure 3-4, we plot the performance gap with Maximum-Likelihood for these three detection schemes. For this purpose, we measure the difference in the minimum SNR level that is required to have SER of 10^{-3}. In this figure, we also show the 90th-percentile of the SNR gap for different channels. We observe that MMNet reduce the SNR requirement by up to 5dB and 8dB, respectively, over OAMPNet and MMSE for realistic channels.

Robustness to channel estimation errors

In order to evaluate MMNet's robustness against channel estimation errors [98], we consider a least-squares channel estimator which is equivalent to adding complex Gaussian noise to the channel matrix, i.e., : $\hat{\mathbf{H}}_{ij} = \mathbf{H}_{ij} + \boldsymbol{\delta}_{ij}$, where $\boldsymbol{\delta}_{ij} \sim \mathcal{CN}(0, \sigma_c^2)$. We provide $\hat{\mathbf{H}}$ to the signal detector instead of $\mathbf{H}$. Figure 3-5 compares the impact of imperfect channel estimation on 3GPP MIMO channels for QAM16 modulation at different values of σ_c. Here we have 32 transmit and 64 receive antennas operating at 23dB SNR. We observe that MMNet is able to achieve the same SER as OAMPNet at 10dB higher channel estimation error at a fixed SNR

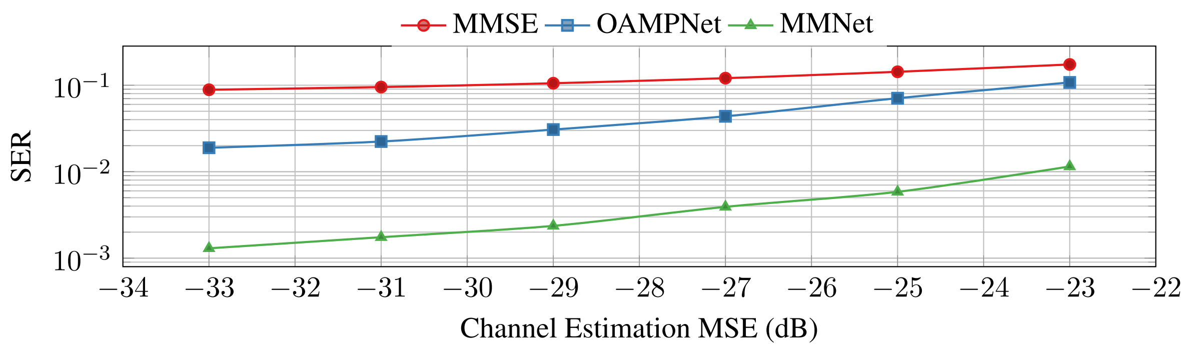


Figure 3-5: SER for QAM16 versus channel estimation MSE.

3.6 Why MMNet works

In this section, we examine why MMNet performs better than the best previous learning-based scheme, OAMPNet. By analyzing the dynamics of the error ($\hat{\mathbf{x}}_t - \mathbf{x}$), we find that MMNet's denoisers are significantly more effective than those in OAMPNet. We show that this occurs because the linear stages learned by MMNet create favorable conditions for the denoisers. Specifically, we find that the distribution of noise at the input of MM-Net's denoisers is nearly Gaussian, whereas the noise distribution for OAMPNet is far from Gaussian. Since the denoisers in both architectures are optimized for Gaussian noise, they perform much better in MMNet.

3.6.1 Error dynamics

Define the error at the outputs of the linear and denoiser stages at iteration t as $\mathbf{e}_t^{lin} = \mathbf{z}_t - \mathbf{x}$ and $\mathbf{e}_t^{den} = \hat{\mathbf{x}}_{t+1} - \mathbf{x}$, respectively. For algorithms such as MMNet and OAMPNet with $\mathbf{b}_t = 0$, we can rewrite the update equations of (3.3) in terms of these two errors in the form:

$$\mathbf{e}_t^{lin} = (\mathbf{I} - \mathbf{A}_t\mathbf{H})\mathbf{e}_{t-1}^{den} + \mathbf{A}_t\mathbf{n} \quad (3.16a)$$
$$\mathbf{e}_t^{den} = \eta_t(\mathbf{x} + \mathbf{e}_t^{lin}) - \mathbf{x}. \quad (3.16b)$$

Equation (3.16a) includes two terms, corresponding to two sources of error that contribute to $\mathbf{e}_t^{lin}$: (1) the error in the output signal from the previous iteration, and (2) the channel noise. Different choices of $\mathbf{A}_t$ impact these two terms differently. For example, if we set $\mathbf{A}_t$ to $\mathbf{H}^+$ (the pseudo-inverse of the channel matrix), we are only left with the term $\mathbf{H}^+\mathbf{n}$ in $\mathbf{e}_t^{lin}$, thus eliminating the first error term entirely. However, this comes at a price: we are left with Gaussian noise with covariance matrix $\sigma^2\mathbf{H}^+\mathbf{H}^{+H}$. This presents two potential problems:

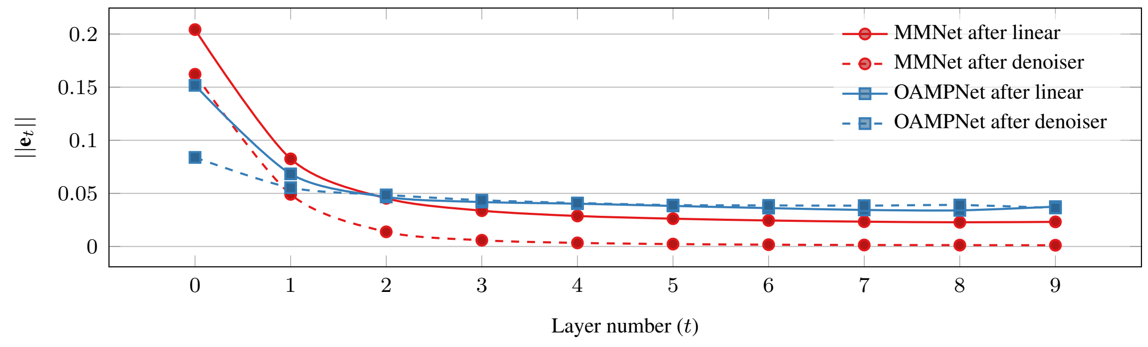


Figure 3-6: Noise power after the linear and denoiser stages at different layers of OAMPNet and MMNet. The OAMPNet denoisers become ineffective after the third layer on 3GPP MIMO channels.

(i) if $\mathbf{H}$ is ill-conditioned, it might amplify the channel noise (e.g., inversely proportional to the smallest singular value of $\mathbf{H}$ in some directions); (ii) the resulting noise, $\mathbf{e}_t^{lin}$, may have correlated elements, and therefore applying an element-wise denoising function to it as per Equation (3.16b) may be suboptimal.

Another naive option is to eliminate the channel noise term entirely by setting $\mathbf{A}_t$ to zero. This would effectively remove the linear stages from the architecture, reducing it to a cascade of denoising stages. However, applying a (well-chosen) denoiser multiple times in a cascade should be no better than applying it once.

It is also instructive to consider the error dynamics in the special case of i.i.d. channels. In this case, if we set $\mathbf{A}_t = \mathbf{H}^H$, the factor $\mathbf{I} - \mathbf{A}_t\mathbf{H}$ asymptotically goes to zero as we increase N_r [85], and the auto-covariance of $\mathbf{A}_t\mathbf{n}$, $\sigma^2\mathbf{H}^H\mathbf{H}$, is approximately equal to $\sigma^2\mathbf{I}_{N_t}$. This means that the linear stage does not amplify the channel noise or make it correlated. On the other hand, the error from the previous iteration, $\mathbf{e}_{t-1}^{den}$, is attenuated significantly via $\mathbf{I} - \mathbf{A}_t\mathbf{H}$. These calculations explain why AMP has great performance on i.i.d. Gaussian channels. However, in case of correlated channels, neither $\mathbf{I} - \mathbf{A}_t\mathbf{H}$ is close to zero, nor is $\mathbf{A}_t\mathbf{n}$ uncorrelated, and therefore AMP does not perform well on realistic channels.

3.6.2 Analysis

Based on the above discussion of the error dynamics, we identify two desirable properties for picking $\mathbf{A}_t$:

1. *Noise reduction property:* $\mathbf{A}_t$ must reduce the magnitude of $\mathbf{e}_t^{lin}$. This requires striking a balance between the two terms in (3.16a), because attenuating one term may amplify the other and vice-versa.

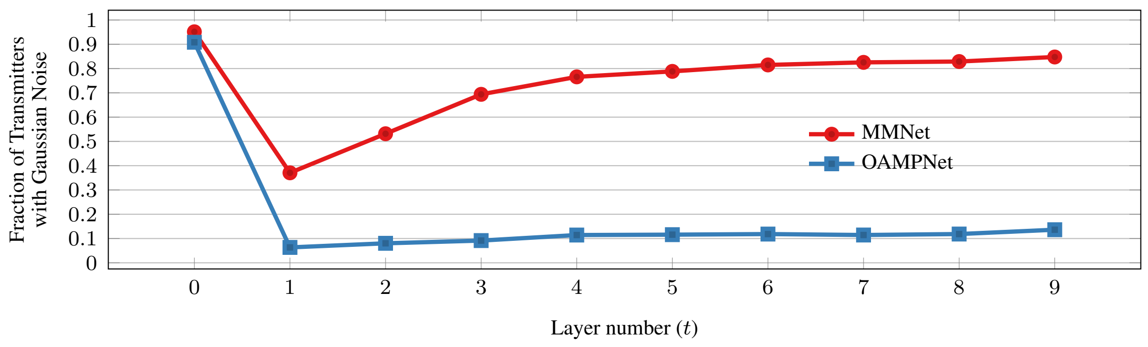


Figure 3-7: Fraction of transmitters that have Gaussian error distribution after the linear block for each layer with significance level of 5%. MMNet produces Gaussian distributed errors at the output of linear blocks, while OAMPNet fails to achieve the Gaussian property.

2. *Uncorrelated Gaussian noise property:* $\mathbf{A}_t$ must "shape" the distribution of $\mathbf{e}_t^{lin}$ to make it suitable for the subsequent denoising stage. In particular, the denoisers in most iterative schemes (e.g., MMNet and OMAPNet) are specifically designed for uncorrelated Gaussian noise. Thus, ideally, the linear stage should avoid outputting correlated or non-Gaussian noise.

Figure 3-6 shows the average noise power at the output of the linear and denoiser stages across different layers, for both OAMPNet and MMNet on 64×16 3GPP MIMO channels. We observe that for OAMPNet, the average noise power after the linear stage (before the denoiser) and after the denoiser is the same from the third layer ($t = 2$) onwards. In other words, in OAMPNet, the denoisers are unable to reduce the noise in their input signal after a few layers. However, with MMNet, the noise power is significantly lower at the output of the denoisers compared to their input at all layers.

We hypothesize that the reason OAMPNet's denoisers become ineffective is that the noise distribution for OAMPNet is not Gaussian, whereas MMNet is able to provide near-Gaussian noise to its denoisers. We evaluate how close the noise distribution is to Gaussian for both schemes using the Anderson test [99]. In order to measure this score, we generate 10,000 samples of $\mathbf{x}$ and $\mathbf{y}$ per channel realization $\mathbf{H}$. We run each sample through both MMNet and OAMPNet, and we calculate the Anderson score for the noise distribution at the output of the linear stages, for each transmitter and channel matrix. If this score is below a threshold of 0.786, it indicates that the noise comes from a Gaussian distribution with a significance of 5%, i.e. the probability of false rejection of a Gaussian distribution is less than 5%. In Figure 3-7, we plot the average fraction of transmitters that have Gaussian distributed noise at the output of the linear stage according to this test. Since in both schemes we start with $\hat{\mathbf{x}}_0 = 0$, the output of the linear stage at layer $t = 0$ is $\mathbf{A}_0\mathbf{n}$, which is Gaussian. Thus, the fraction of transmitters with Gaussian noise is 1 in layer $t = 0$ for both schemes. However, both schemes deviate from Gaussian noise in layer $t = 1$, while at the same time sharply reducing the total noise power as seen in Figure 3-6. However, the noise for fewer transmitters in MMNet deviate from a Gaussian distribution. Unlike OAMPNet, in which the noise for 95% of the transmitters is not Gaussian at layer $t = 1$, for MMNet nearly 40% of the transmitters exhibit Gaussian noise. On the other hand, MMNet reduces the noise power slightly less than OAMPNet in layer $t = 1$.

In subsequent layers, the noise distribution for MMNet becomes increasingly Gaussian, with nearly 90% of transmitter passing the Anderson test by layer $t = 9$. By contrast, most transmitters in OAMPNet continue to exhibit non-Gaussian noise in subsequent layers, though the fraction of transmitter with Gaussian noise marginally increases.

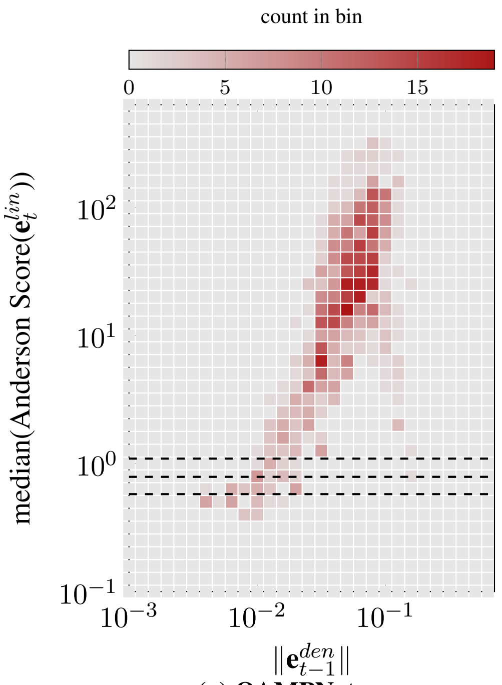
Next, we measure the effect of input noise power on the distribution of noise at the output of the linear stages in both schemes. In other words, we are interested to know how $\|\mathbf{e}_{t-1}^{den}\|$ impacts the Gaussian distribution property of $\mathbf{e}_t^{lin}$. For this purpose, we choose the median of Anderson scores as a measure of the linear stage's ability to maintain the Gaussian property at its output. In Figure 3-8, we show the 2D histogram of this median score for different values of $\|\mathbf{e}_{t-1}^{den}\|$. For reference, we also plot three thresholds corresponding to 1%, 5% and 15% significance for the normality test as dashed horizontal lines. To be Normally distributed with 1%, 5%, or 15% confidence, the Anderson scores must fall below the respective line.

We notice that the median score in both schemes increases with the norm of the error from the previous iteration. In other words, the linear stages that have a higher input noise power produce outputs that deviate more significantly from a Gaussian. However, MMNet is $100\times$ better in terms of the median Andersen score for large values of $\|\mathbf{e}_{t-1}^{den}\|$. This figure also suggests that OAMPNet's inability to achieve the Gaussian property is not limited to the first couple of layers, in which it aggressively reduces the noise power. The later linear stages, for which $\|\mathbf{e}_{t-1}^{den}\|$ is fairly small, are also not very good at achieving the Gaussian property.

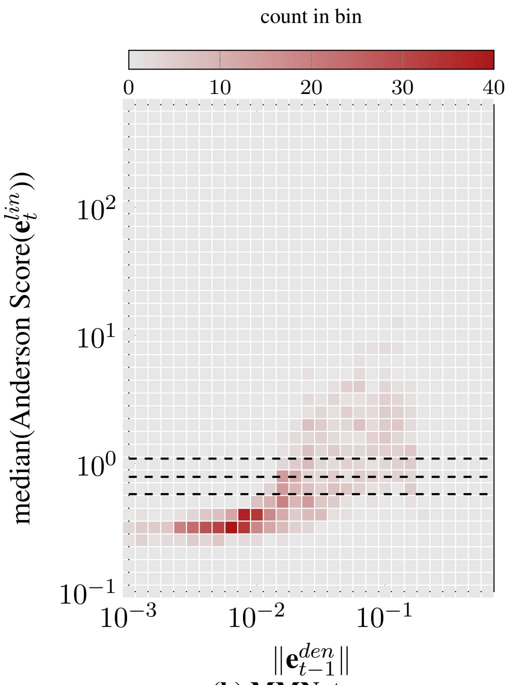
3.6.3 Impact of channel condition number

Finally, we evaluate the impact of the channel *condition number* on MMNet and OAMPNet. A channel's condition number is defined as the ratio of its largest singular value to the smallest. It is well-known that symbol detection is difficult for channel matrices with higher condition number.

In Figure 3-9a, we show a scatterplot depicting the fraction of transmitters with the Gaussian noise property at output of the linear stage in layer $t = 4$ vs. channel condition

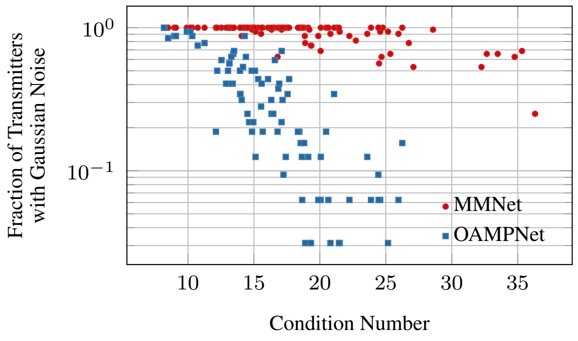


(a) OAMPNet

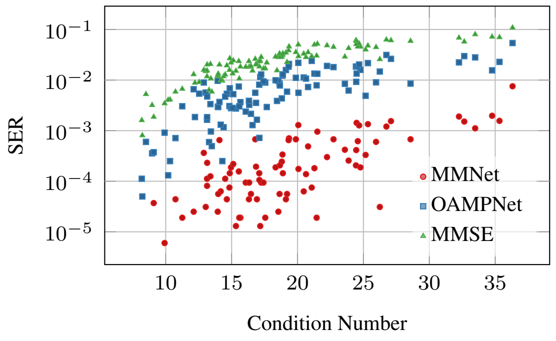


(b) MMNet

Figure 3-8: Median of Anderson score for the noise at the output of linear stage vs. the power of noise at the input of the stage. MMNet achieves the Gaussian property much more effectively for all noise power levels. Dashed horizontal lines show the thresholds for 1%, 5% and 15 significance level.



(a) Fraction of transmitters with Gaussian noise at layer $t = 4$



(b) Symbol error rate

Figure 3-9: Effect of channel condition number on performance in the 3GPP MIMO dataset. (a) MMNet is more robust in maintaining the Gaussian noise property for channels with large condition number. (b) SER is directly affected by the condition number.

number for the 3GPP MIMO dataset. We show the behavior at layer $t = 4$ as an example of what occurs in the first few iterations of the detection process. We see that, for OAMPNet, the fraction of transmitters satisfying the Gaussian noise property decreases sharply for channels with higher condition number. By contrast, MMNet maintains the Gaussian noise property for a much broader range of channels. The consequence of the failure to achieve the Gaussian property on SER is shown in Figure 3-9b. Although the performance of all schemes degrades as the channel condition number increases, MMNet is able to achieve an SER of less than 10^{-3} for a wider range of channel conditions.

3.7 Online Training Algorithm

Training deep learning models is a computationally intensive task, often requiring hours or even days for large models. The computation overhead depends on two factors: (i) the total number of required training samples, and (ii) the size of the model. For example, to train a model like DetNet with about 1M parameters, we need 50K iterations with a batch size of 5K samples, i.e., 250M training samples. If we assume each parameter of the model shows up in at least one floating-point operation per training sample, we require a minimum of 2.5×10^{14} floating-point operations for the entire training process. This computational complexity makes training such a large model online for each realization of $\mathbf{H}$ impossible.[5]

In comparison, MMNet has only $\sim$40K parameters, and training it from scratch requires about 1000 iterations with batch size of 500. Further, we show that by taking advantage of locality of the channels observed at a receiver, we can further reduce training cost to 9 iterations (with batch size 500) on average per channel realization. All in all, training MMNet has six orders of magnitude lower computational overhead than DetNet, making online training for each realization of $\mathbf{H}$ practical.

In this section, we first discuss the temporal and spectral locality of realistic channels. Then, we show how we can exploit these localities to accelerate online training.

3.7.1 Channel locality

The channel matrices measured at a base station exhibit two forms of locality:

- *Temporal:* Channel matrices change gradually over time as user devices move within a cell or the multipath environment changes. The samples of $\mathbf{H}$ at nearby points in time are thus correlated.

- *Spectral*: The base station needs to recover signals from several frequency subcarriers (1024 in our 3GPP MIMO model). The channels for subcarriers in nearby

[5]Of course, DetNet was specifically designed for offline training, where computational complexity is less of a concern.

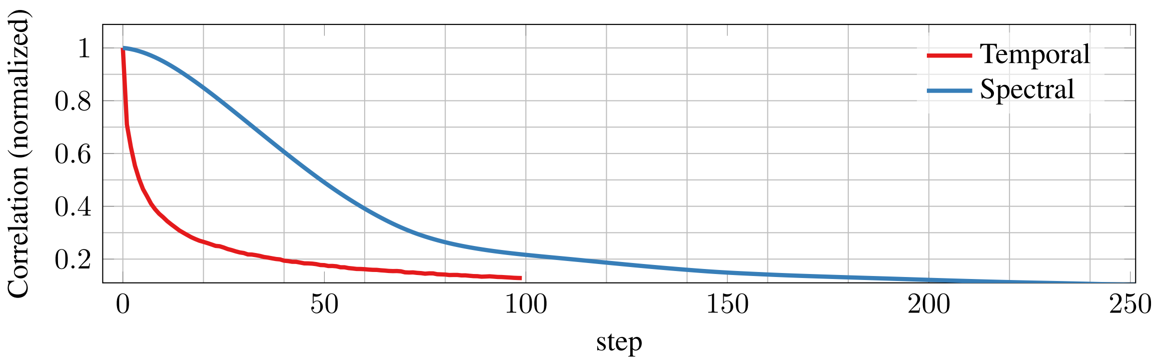


Figure 3-10: Correlation of channel samples across time and frequency dimensions. The correlation decays relatively quickly in the time dimension, but the channel matrices show strong locality across sub-carriers in frequency dimension.

frequencies are also strongly correlated. The frequency affects the phase of the multipath signal components incident on receiver antennas. For a path of length l, the phase difference for two subcarriers Δf apart in frequency is $\Delta\phi = \frac{2\pi l \Delta f}{c}$. Therefore the received signal and the channels for adjacent subcarriers will be similar at each receiver antenna.

Both forms of locality reduce the complexity of training for each channel realization, because (i) the cost of channel-specific computations can be amortized across multiple correlated channel realizations across time and frequency, (ii) the trained model for one channel realization can serve as strong initialization for training for adjacent channel realizations in the time-frequency plane.

Figure 3-10 shows both forms of locality by plotting the correlation among the 3GPP MIMO channel samples across time and frequency (subcarriers). To compute these correlations, we take the average of the inner-product of each channel matrix with its neighboring channel matrices separated in time or frequency by different number of steps. A shift of one step in the time domain corresponds to two channel matrices at the same subcarrier frequency that are 118ms apart in time. A shift of 1 step in the frequency domain corresponds to two channel matrices at adjacent subcarriers (78.1KHz apart) at the same time. We normalize the inner-product by the norm of the matrices, such that the correlation of a channel matrix with itself is 1. As the figure show, we observe a stronger locality in the frequency domain than in the time domain in the 3GPP MIMO channels.

3.7.2 Training algorithm and results

In this section, we show how channel locality can help reduce the total number of operations MMNet needs to decode each received signal $\mathbf{y}$ at the BS. The computational complexity

Algorithm 1 MMNet online training

```
1: $\mathcal{M} \leftarrow$ Construct MMNet with parameters $\vartheta = \{\mathbf{\Theta}_t^{(1)}, \boldsymbol{\theta}_t^{(2)}\}_{t=1}^{T}$
2: $\vartheta \leftarrow$ Initialize model parameters randomly
3: for $n \in \{1,\ 2,\ ...\}$ {$n$ keeps the time step} do
4:     for $f \in \{1,\ 2,\ ...,\ F\}$ do
5:         $\mathbf{H}[f] \leftarrow$ Measured channel at time step $n$ and frequency $f$
6:         #TrainIterations $\leftarrow$ Set to $\Phi$ if $f \neq 1$ and $\Psi$ otherwise
7:         for it$\in \{1, 2, ...,$#TrainIterations$\}$ do
8:             $\mathbf{D} \leftarrow$ Generate random $(\mathbf{x}, \mathbf{y})$ batch on $\mathbf{H}[f]$ using eq. (3.1)
9:             $L \leftarrow$ Find the loss in eq. (3.15) for $\mathcal{M}$ over the samples in $\mathbf{D}$
10:            $\vartheta \leftarrow$ Compute the model updates using $\nabla_{\vartheta} L$
11:        end for
12:        $\mathcal{M}_n[f] \leftarrow \mathcal{M}$.copy( ) {Store the parameters $\vartheta$}
13:    end for
14: end for
```

of MMNet is mostly dominated by the cost of online training for each new realization of the channel $\mathbf{H}$. This cost in turn depends on the channel *coherence time*. In the case of a quasi-static channel, as expected for instance in fixed-wireless access or backhaul solutions such as 5G home wireless (see Section 7.6.2 in [67]), the channel between the transmitter and receiver does not change for extended periods of time. In such cases, MMNet does not require frequent retraining and can reuse the same model until the communication channel changes significantly. However, MMNet can also operate at reasonable computation cost when the channel is changing fairly frequently. For example, our 3GPP MIMO channel samples were generated assuming all devices constantly move at a speed of 1 m/s, and after about 500ms, the channel correlation is less than 0.5. However, even in this scenario, we require only 9 training iterations on average per channel realization, as explained next.

To see how, note that a receiver at a BS must simultaneously decode signals from different subcarriers. Since channels exhibit strong correlations across sub-carriers (Figure 3-10), training the MMNet detector on $\mathbf{H}$ for one subcarrier produces a detector that will achieve very similar performance on adjacent subcarriers. The performance of this detector will however decay for more distant subcarriers in the frequency domain.

Based on this observation, we propose the online training scheme in Algorithm 1. We start from a random initialization of the MMNet neural network model $\mathcal{M}$. We define n as an index for time intervals in which we can assume that channels do not change substantially. For each interval n, we measure a channel matrix $\mathbf{H}[f]$ for each subcarrier frequency f. The basic idea in the algorithm is to train the model for Ψ iterations (with a batch size of 500) for the first subcarrier ($f = 1$), then retrain the model using only Φ additional training iterations per subcarrier for all subsequent subcarriers. Typical values

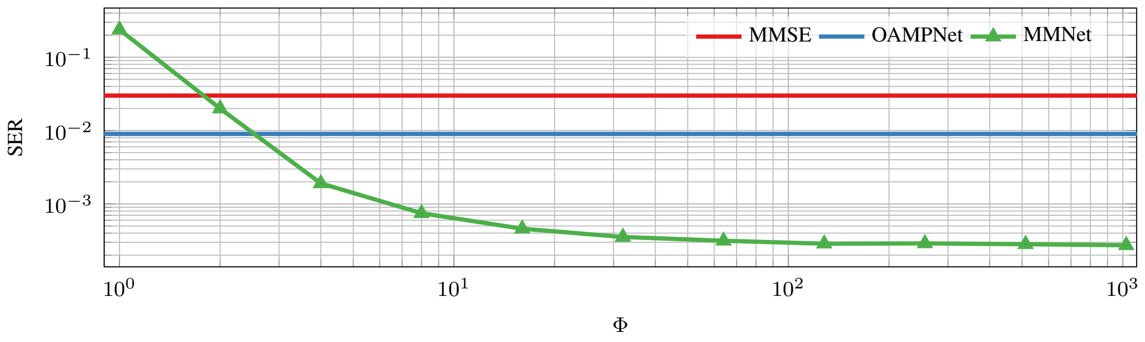


Figure 3-11: SER vs. Φ using Algorithm 1 with $\Psi = 10^3$ for training MMNet on QAM16 modulation. MMNet requires only 9 overall iterations of batch size 500 per channel realization to train to a reasonable performance with $(\Psi, \Phi) = (10^3, 8)$.

for the hyperparameters Ψ and Φ are respectively 1000 and 3–10 in our experiments. For each channel matrix $\mathbf{H}[f]$, we generate $(\mathbf{x}, \mathbf{y})$ training data pairs using eq. (3.1). Once the model has trained for subcarrier f, we save a copy of the model as $\mathcal{M}_n[f]$ for detecting all signals received in time interval n on that subcarrier. We repeat the entire training algorithm in each time interval. In Figure 3-11, we show the result of our online training method on 3GPP MIMO channels for the QAM16 modulation with 16 transmit and 64 receive antennas at 18dB SNR. In this experiment, we set $\Psi = 1000$ and compare MMNet with other schemes on a range of Φ values. We see that while MMSE and OAMPNet achieve a SER of 0.03 and 0.009 respectively, MMNet can bring the SER down to 0.0007 (below the 10^{-3} threshold). With this approach, MMNet performs 9.19K iterations of training with batch size 500 in order to learn a detector for all 1024 subcarriers in total at each time interval n. Therefore the cost of online training is less than 9 iterations on average per channel realization, yet MMNet delivers better performance than other schemes, like OAMPNet and MMSE.

3.7.3 Computational complexity

One iteration of training MMNet on a batch of size of b has a complexity of $\mathcal{O}(bN_r^2)$, as detection takes $\mathcal{O}(N_r^2)$ in MMNet. To put this in perspective, a light-weight algorithm like AMP has a complexity of $\mathcal{O}(N_r^2)$ dominated by the multiplication of the channel matrix and residual vectors. The MMSE scheme has a higher complexity of $\mathcal{O}(N_r^3)$ because it needs to invert a matrix. OAMPNet similarly requires a matrix inversion, resulting in a complexity of $\mathcal{O}(N_r^3)$.

Moving beyond $\mathcal{O}(\cdot)$ analysis, Figure 3-12 shows the average number of multiplication operations required per signal detection on 3GPP MIMO channels for learning-based algorithms in addition to two classic baselines, MMSE and AMP. For these numbers, all al-

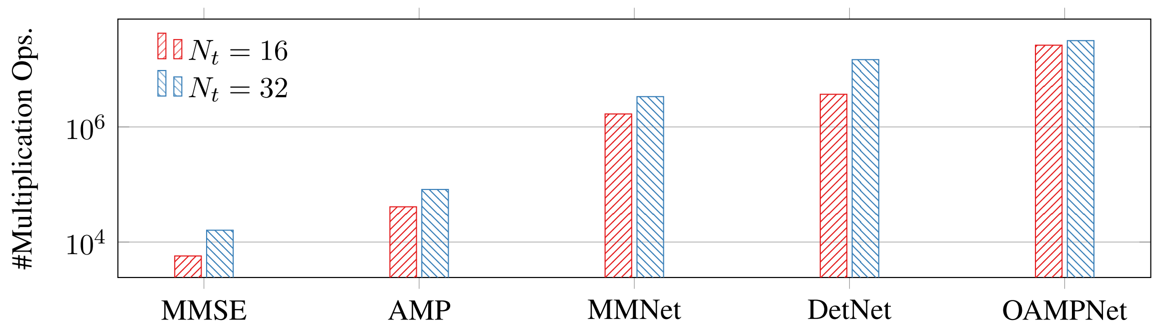


Figure 3-12: Number of multiplication operations per signal detection for different algorithms on QAM16 with $N_r = 64$ receive antennas in 3GPP MIMO model. Detection with MMNet, including its online training process, requires fewer multiplication operations than detection with pre-trained DetNet and OAMPNet models.

gorithms reuse computations whenever feasible. In particular, in every channel coherence interval in the 3GPP MIMO model, each algorithm receives $\sim$100 signals to detect [67, Definition 2.2 on page 220]. MMSE calculates the required channel matrix inverse only once for all 100 received signals in the coherence interval. This reduces computation complexity for MMSE by a factor of 100$\times$, resulting in 5–7$\times$ fewer multiplications than AMP, which cannot reuse computation but has modest complexity. MMNet reuses the weights it trains (with 9 iterations of batch size 500 on average) for all 100 received signals in a coherence interval. MMNet, with its online training and detection operations combined, places after AMP with 2–5$\times$ fewer multiplications than pre-trained DetNet. However, as we have seen, neither AMP nor DetNet extend to realistic, spatially correlated channels. OAMPNet's computational complexity is higher than the other models, because it has to calculate a new matrix inverse in each layer for every received signal, as v_t^2 in eq. (3.10) depends on $\hat{\mathbf{x}}_t$.

Consequently, the cost of MMNet with its online training algorithm is 10–15$\times$ less than OAMPNet depending on the system size. MMNet has 41$\times$ higher computational complexity than a light-weight iterative approach like AMP, which only works for i.i.d. Gaussian channels.

3.8 Conclusion

This chapter proposed a deep learning architecture for Massive MIMO detection, and an online training algorithm to optimize it for every realization of the channel matrix at a base station. MMNet outperforms state-of-the-art detection algorithms on realistic channels with spatial correlation. We designed MMNet as an iterative algorithm and showed that a carefully chosen degree of flexibility in the model, in addition to leveraging the channel's spectral and temporal correlation, can enable online training at a less or equal computation complexity than other deep-learning based schemes. MMNet is 4–8dB better overall than the classic MMSE detector and it requires 2.5dB lower SNR at the same SER, relative to the second-best detection scheme, OAMPNet, at 10–15$\times$ less computational complexity. Many extensions of MMNet are possible to support, for example, a varying number of transmitters with possibly different modulation schemes.

From a hardware perspective, implementing MMNet has its own challenges and requires an in-depth study. For example, the sequential online training algorithm introduced in this chapter incurs latency, which may be traded off with parallel training of multiple sub-carriers at the cost of more training iterations and hence increased complexity. The optimal trade-off depends on the channel coherence time.

Our results show that evaluations based on simple channel models such as i.i.d. Gaussian channels can lead to misleading conclusions for MIMO detection performance. Future work should therefore evaluate on realistic channel models, from either simulation, ray-tracing, or measurements. We have released the simulated 3GPP MIMO channel dataset used in this work in hope that it will serve as a useful benchmark for the community.

Chapter 4

AMS: Real-Time Video Inference on Edge Devices via Adaptive Model Streaming

4.1 Overview

Real-time video inference is a core component for many applications, such as augmented reality, drone-based sensing, robotic vision, and autonomous driving. These applications use Deep Neural Networks (DNNs) for inference tasks like object detection [7], semantic segmentation [100], and pose estimation [101]. However, state-of-the-art DNN models are too expensive to run on low-powered edge devices (e.g., mobile phones, drones, consumer robots [21, 22]), and cannot run in real-time even on accelerators such as Coral Edge TPU and NVIDIA Jetson [25–27].

A promising approach to improve inference efficiency is to specialize a lightweight model for a specific video and task. The basic idea is to use distillation [102] to transfer knowledge from a large "teacher" model to a small "student" model. For example, Noscope [103] trains a student model to detect a few object classes on specific videos offline. Just-In-Time [104] extends the idea to live, dynamic videos by training the student model online, specializing it to video frames as they arrive. These approaches provide significant speedups for scenarios that perform inference on powerful machines (e.g., server-class GPUs), but they are impractical for on-device inference at the edge. The offline approach isn't desirable since videos can vary significantly from device to device (e.g., different locations, lighting conditions, etc.), and over time for the same device (e.g., a drone flying over different areas). On the other hand, training the student model online on edge devices

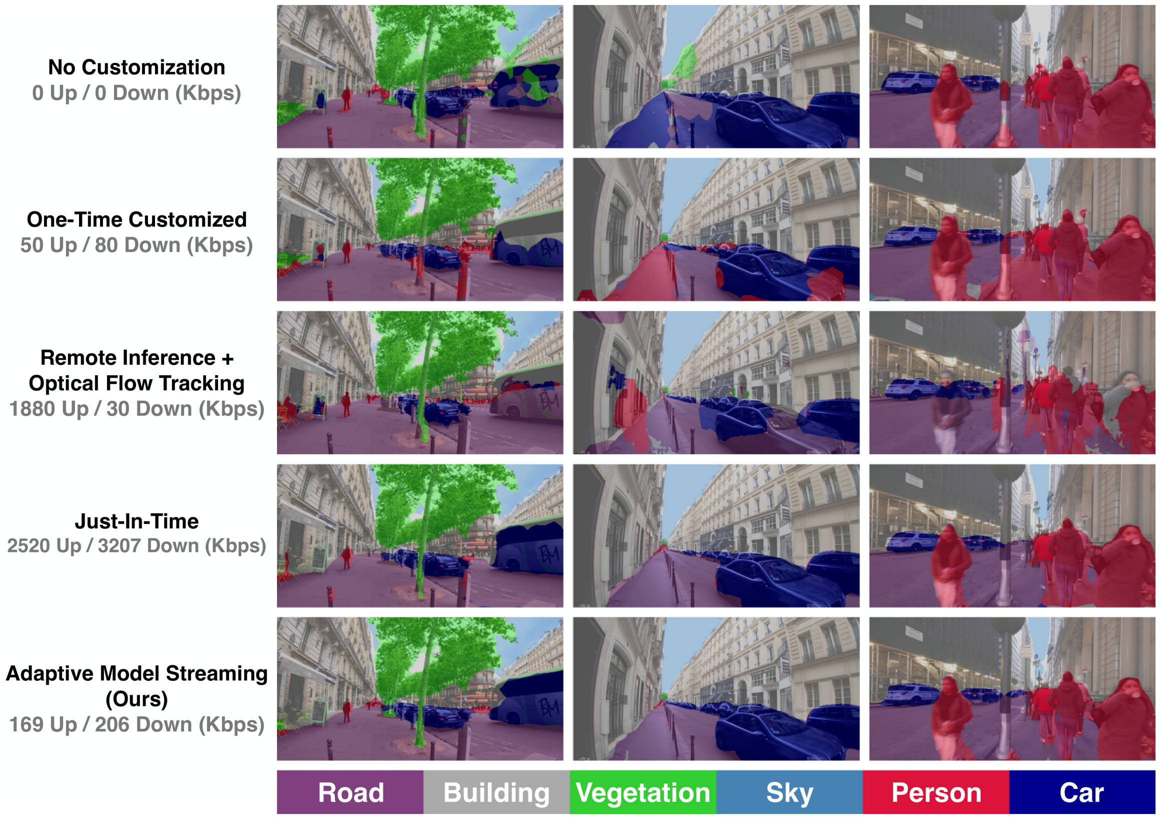


Figure 4-1: Semantic segmentation results on real-world outdoor videos: rows from top to bottom represent No Customization, One-Time, Remote+Tracking, Just-In-Time, and AMS. Uplink and downlink bandwidth usage are reported below each variant. AMS provides better accuracy with limited bandwidth and reduces artifacts (e.g., see the car/person detected in error by the no/one-time customized models and remote tracking in the second column).

is computationally infeasible.

In this chapter we propose *Adaptive Model Streaming* (AMS), a new approach to real-time video inference on edge devices that offloads knowledge distillation to a remote server communicating with the edge device over the network. AMS *continually* adapts a small student model running on the edge device to boost its accuracy for the specific video in real time. The edge device periodically sends sample video frames to the remote server, which uses them to fine-tune (a copy of) the edge device's model to mimic a large teacher model, and sends (or "streams") the updated student model back to the edge device.

Performing knowledge distillation over the network introduces a new challenge: communication overhead. Prior techniques such as Just-In-Time aggressively overfit the student model to the most recent frames, and therefore must frequently update the model to sustain high accuracy. We show, instead, that training the student model over a suitably chosen horizon of recent frames — not too small to overfit narrowly, but not too large to surpass the generalization capacity of the model — can achieve high accuracy with an order of magnitude fewer model updates compared to Just-In-Time training.

Even then, a naïve implementation of over-the-network model training would require significant bandwidth. For example, sending a (small) semantic segmentation model such as DeeplabV3 with MobileNetV2 [28] backbone with $\sim$2 million (float16) parameters every 10 seconds would require over 3 Mbps of bandwidth. We present techniques to reduce both downlink (server to edge) and uplink (edge to server) bandwidth usage for AMS. For the downlink, we develop a coordinate-descent [105, 106] algorithm to train and send a small fraction of the model parameters in each update. Our method identifies the subset of parameters with the most impact on model accuracy, and is compatible with optimizers like Adam [97] that maintain a state (e.g., gradient moments) across training iterations. For the uplink, we present algorithms that dynamically adjust the frame sampling rate at the edge device based on how quickly scenes change in the video. Taken together, these techniques reduce downlink and uplink bandwidth to only 181–225 Kbps and 57–296 Kbps respectively (across different videos) for a challenging semantic segmentation task. To put AMS's bandwidth requirement in perspective, it is less than the YouTube recommended bitrate range of 300–700 Kbps to live stream video at the lowest (240p) resolution [107].

We evaluate our approach for real-time semantic segmentation using a lightweight model (DeeplabV3 with MobileNetV2 [28] backbone). This model runs at 30 frames-per-second with 40 ms camera-to-label latency on a Samsung Galaxy S10+ phone (with Adreno 640 GPU). Our experiments use four datasets with long (10 minutes+) videos spanning a variety of scenarios (e.g., city driving, outdoor scenes, and sporting events). Our results show:

1. Compared to pretraining the same lightweight model without video-specific customization, AMS provides a 0.4–17.8% boost (8.3% on average) in mIoU, computed relative to the labels from a state-of-the-art DeeplabV3 with Xception65 [108] backbone model. It also improves mIoU by 4.3% on average (up to 39.1%) compared to customizing the model once using the first 60 seconds of each video.
2. Compared to a remote inference baseline accompanied by on-device optical flow tracking [36, 109], AMS provides an average improvement of 5.8% (up to 24.4%) in mIoU.
3. AMS requires 15.7× less downlink bandwidth on average (up to 44.5×) to achieve similar accuracy compared to Just-In-Time [104] (with similar reductions in uplink bandwidth).

Figure 4-1 shows three visual examples comparing the accuracy of AMS with these baseline approaches.

Our code and video datasets are available online at https://github.com/modelstreaming/ams.

4.2 Related Work

We described prior work on knowledge distillation for video in §4.1. Here, we discuss other related work.

On-device inference. Small, mobile-friendly models have been designed both manually [28] and using neural architecture search [29, 30]. Model quantization and weight pruning [31–34] have further been shown to reduce the computational footprint of such models with a small loss in accuracy. Specific to video, some techniques amortize the inference cost by using optical flow methods to skip inference for some frames [35–37]. Despite this progress, there remains a large gap in the performance of lightweight models and state-of-the-art solutions [110, 111]. AMS is complementary to on-device optimization techniques and would also benefit from them.

Remote inference. Several proposals offload all or part of the computation to a remote machine [17–19, 39–41], but these schemes generally require high network bandwidth, incur high latency, and are susceptible to network outages [18, 42]. Proposals like edge computing [43–45] that place the remote machine close to the edge devices lessen these barriers, but do not eliminate them and incur additional infrastructure and maintenance costs. AMS requires much less bandwidth than remote inference, and is less affected by network delay or outages since it performs inference locally on the device.

Online learning. Our work is also related to online learning [46] algorithms for minimizing dynamic or tracking regret [47–49]. Dynamic regret compares the performance of an online learner to a sequence of optimal solutions. In our case, the goal is to track the performance

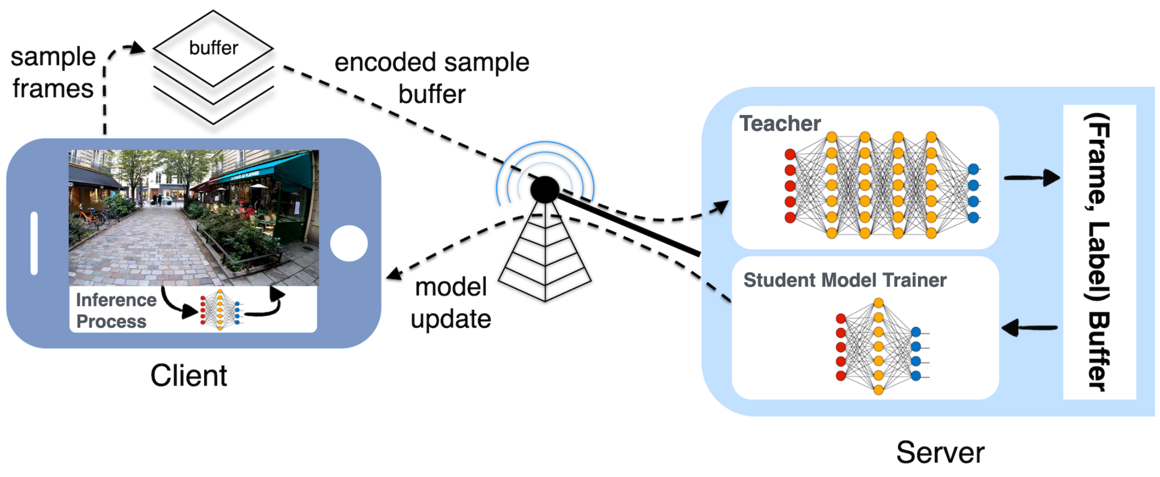


Figure 4-2: AMS system overview.

of the best lightweight model at each point in a video. Several theoretical works have studied online gradient descent algorithms in this setting with different assumptions about the loss functions [50, 51]. Other work has focused on the "experts" setting [52–54], where the learner maintains multiple models and uses the best of them at each time. Our approach is based on online gradient descent because tracking multiple models per video at a server is expensive.

Other paradigms for model adaptation include lifelong/continual learning [57], meta-learning [63, 64], federated learning [61], and unsupervised domain adaptation [55, 56]. Our here work is only tangentially related to these efforts, and for further details, please refer to their discussion in section 2.2.

4.3 Adaptive Model Streaming (AMS)

Figure 4-2 provides an overview of AMS. Each edge device buffers sampled video frames for T_{update} seconds, then compresses and sends the buffered frames to the remote server. The server uses these frames to train a copy of the edge device's model using supervised knowledge distillation [102], and sends the model changes to the edge device. For concreteness, we describe our design for semantic segmentation, but the approach is general and can be adapted to other tasks.

Server. Algorithm 2 shows the server procedure for serving a single edge device (we discuss multiple edge devices in Appendix C). The AMS algorithm at the server runs iteratively on each new batch of frames received from the edge device. It consists of two phases: inference and training.

Inference phase: To train, the server first needs to label the incoming video frames. It obtains these labels using a state-of-the-art segmentation model (like DeeplabV3 [100]

Algorithm 2 Adaptive Model Streaming Server

```
 1: Initialize the student model with pre-trained parameters $\mathbf{w}_0$
 2: Send $\mathbf{w}_0$ and the student model architecture for the edge
 3: $\mathcal{B} \leftarrow$ Initialize a time-stamped buffer to store (sample frame, teacher prediction) tuples
 4: for $n \in \{1, 2, ...\}$ do
 5: ↑                  $\mathcal{R}_n \leftarrow$ Set of new sample frames from the edge device
 6:                    for $\mathbf{x} \in \mathcal{R}_n$ do
 7:    Infer. Phase        $\tilde{\mathbf{y}} \leftarrow$ Use the teacher model to infer the label of $\mathbf{x}$
 8:                        Add $(\mathbf{x}, \tilde{\mathbf{y}})$ to $\mathcal{B}$ with time stamp of receiving $\mathbf{x}$
 9: ↓                  end for
10: ↑                  $\mathcal{I}_n \leftarrow$ Select a subset of model parameter indices
11:                    for k $\in \{1, 2, ..., K\}$ do
12:    Training Phase      $\mathbf{S}_k \leftarrow$ Uniformly sample a mini-batch of data points from $\mathcal{B}$ over the last $T_{horizon}$ seconds
13:                        Candidate updates $\leftarrow$ Calculate Adam optimizer updates w.r.t the empirical loss on $\mathbf{S}_k$
14:                        Apply candidate updates to model parameters indexed by $\mathcal{I}_n$
15: ↓                  end for
16:                    $\tilde{\mathbf{w}}_n \leftarrow$ New value of model parameters which are indexed by $\mathcal{I}_n$
17:                    Send $(\tilde{\mathbf{w}}_n, \mathcal{I}_n)$ for the edge device
18:                    Wait for $T_{update}$ seconds
19: end for
```

with Xception65 [108] backbone), which serves as the "teacher" for supervised knowledge distillation. The server runs the teacher on new frames, and adds the frames, their timestamps, and labels to a training data buffer $\mathcal{B}$.

Training phase: The server trains the student model to minimize the loss over the sample frames in its buffer from the last $T_{horizon}$ seconds of video. To reduce bandwidth usage, the server selects a small subset (e.g., 5%) of parameters for each model update, and trains them for K iterations on randomly-sampled mini-batches of frames. We discuss how the server chooses the parameters to train in §4.3.1.

The server also dynamically adapts the frame sampling rate used by the edge device based on the video characteristics (how fast scenes changes) as described in §4.3.2.

Edge device. The edge device deploys the new models as soon as they arrive to perform local inference. To switch models without disrupting inference, the edge device maintains an inactive copy of the running model in memory and applies the model update to that copy. Once ready, it swaps the active and inactive models. The edge device also samples frames at the rate specified by the server and sends them to the server every T_{update} seconds.

4.3.1 Reducing Downlink Bandwidth

The downlink (server-to-edge) bandwidth depends on (i) how frequently we update the student model, (ii) the cost of each model update. We discuss each in turn.

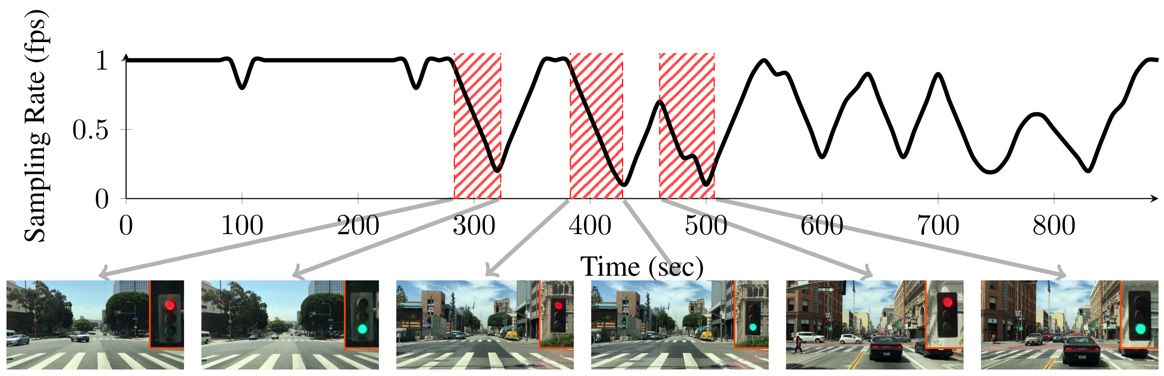


Figure 4-3: Adaptive frame sampling for a driving video. The sampling rate decreases every time the car slows down for the red traffic light and increases as soon as the light turns green.

How Frequently to Train?

The training frequency required depends crucially on the training *horizon* ($T_{horizon}$) for each model update. Prior work, Just-In-Time [104], trains the student model whenever it detects the accuracy has dropped below a threshold, and it trains only on the most recent frame (until the accuracy exceeds the threshold). This approach tends to overfit on recent frames, and therefore requires frequent retraining to maintain the desired accuracy. While this is possible when training and inference occur on the same machine, it is impractical for AMS (§4.4).

Although lightweight models (e.g., those customized for mobile devices) have less capacity than large models, they can still generalize to some extent (e.g., over video frames captured in the same street, a specific room in a home, etc.). Therefore, rather than overfitting narrowly to one or a few frames, AMS uses a training horizon of several minutes. This reduces the required model update frequency, and helps mitigate sharp drops in accuracy when the model lags behind during scene changes (see Figure 4-5).

For semantic segmentation using DeeplabV3 with MobileNetV2 [28] backbone as the student model, we find that $T_{horizon} = 4$ minutes and $T_{update} = 10$ seconds work well across a wide variety of videos (§4.4). However, the optimal values of these parameters can depend on both the model capacity and the video. For example, a lower-capacity student model might benefit from a shorter $T_{horizon}$ and T_{update}, and a stationary video with little scene change could use a longer T_{update}. Appendix B describes a simple technique to dynamically adapt T_{update} to minimize bandwidth consumption (especially for stationary videos).

Which Parameters to Update?

Naïvely sending the entire student model to the edge device can consume significant bandwidth. For example, sending DeeplabV3 with MobileNetV2 backbone, which has 2 million (float16) parameters, every 10 seconds would require 3.2 Mbps of downlink bandwidth. To reduce bandwidth, we employ *coordinate descent* [105, 106], in which we train a small subset (e.g., 5%) of parameters, $\mathcal{I}_n$, in each training phase n and send only those parameters to the edge device.

To select $\mathcal{I}_n$, we use the model gradients to identify the parameters (coordinates) that provide the largest improvement in the loss function when updated. A standard way to do this, called the *Gauss-Southwell selection rule* [112], is to update the parameters with the largest gradient magnitude. We could compute the gradient for the entire model but only update the coordinates with the largest gradient values, leaving the rest of the parameters unmodified. This method works well for simple stateless optimizers like stochastic gradient descent (SGD), but optimizers like Adam [97] that maintain some internal state across training iterations require a more nuanced approach.

Adam keeps track of moving averages of the first and second moments of the gradient across training iterations. It uses this state to adjust the learning rate for each parameter dynamically based on the magnitude of "noise" observed in the gradients [97]. Adam's internal state updates in each iteration depend on the point in the parameter space visited in that iteration. Therefore, to ensure the internal state is correct, we cannot simply compute Adam's updates for K iterations, and then choose to keep only the coordinates with the largest change at the end. We must know *beforehand* which coordinates we intend to update, so that we can update Adam's internal state consistently with the actual sequence of points visited throughout training.

Our approach to coordinate descent for the Adam optimizer computes the subset of parameters that will be updated at the start of each training phase, based on the coordinates that changed the most in the *previous* training phase. This subset is then fixed for the K iterations of Adam in that training phase.

The pseudo code in Algorithm 3 describes the procedure in the n^{th} training phase. Each training phase includes K iterations with randomly-sampled mini-batches of data points from the last $T_{horizon}$ seconds of video. In iteration k, we update the first and second moments of the optimizer ($\mathbf{m}_{n,k}$ and $\mathbf{v}_{n,k}$) using the typical Adam rules (Lines 7–10). We then calculate the Adam updates for all model parameters $\mathbf{u}_{n,k}$ (Lines 11–12). However, we only apply the updates for parameters determined by the binary mask $\mathbf{b}_n$ (Line 13). Here, $\mathbf{b}_n$ is a vector of the same size as the model parameters, with ones at indices that are in $\mathcal{I}_n$

Algorithm 3 *Gradient-Guided Method for Adam Optimizer*

```
1: $\mathcal{I}_n \leftarrow$ Indices of $\gamma$ fraction with largest absolute values in $\mathbf{u}_{n-1}$ {*Entering $n^{th}$ Training Phase*}
2: $\mathbf{b}_n \leftarrow$ binary mask of model parameters; 1 iff indexed by $\mathcal{I}_n$
3: $\mathbf{w}_{n,0} \leftarrow \mathbf{w}_{n-1}$ {Use the latest model parameters as the next starting point}
4: $\mathbf{m}_{n,0} \leftarrow \mathbf{m}_{n-1,K}$ {Initialize the first moment estimate to its latest value}
5: $\mathbf{v}_{n,0} \leftarrow \mathbf{v}_{n-1,K}$ {Initialize the second moment estimate to its latest value}
6: **for** k $\in \{1, 2, ..., K\}$ **do**
7:     $\mathbf{S}_k \leftarrow$ Uniformly sample a mini-batch of data points from $\mathcal{B}$ over the last $T_{horizon}$ seconds
8:     $\mathbf{g}_{n,k} \leftarrow \nabla_{\mathbf{w}} \tilde{\mathcal{L}}(\mathbf{S}_k; \mathbf{w}_{n,k-1})$ {Get the gradient of all model parameters w.r.t. loss on $\mathcal{S}_k$}
9:     $\mathbf{m}_{n,k} \leftarrow \beta_1 \cdot \mathbf{m}_{n,k-1} + (1-\beta_1) \cdot \mathbf{g}_{n,k}$ {Update first moment estimate}
10:    $\mathbf{v}_{n,k} \leftarrow \beta_2 \cdot \mathbf{v}_{n,k-1} + (1-\beta_2) \cdot \mathbf{g}^2_{n,k}$ {Update second moment estimate}
11:    $i \leftarrow i + 1$ {Increment Adam's global step count}
12:    $\mathbf{u}_{n,k} \leftarrow \alpha \cdot \frac{\sqrt{1-\beta_2^i}}{1-\beta_1^i} \cdot \frac{\mathbf{m}_{n,k}}{\sqrt{\mathbf{v}_{n,k}+\epsilon}}$ {Calculate the Adam updates for all model parameters}
13:    $\mathbf{w}_{n,k} \leftarrow \mathbf{w}_{n,k-1} - \mathbf{u}_{n,k} * \mathbf{b}_n$ {Update the parameters indexed by $\mathcal{I}_n$ ($*$ is elem.-wise mul.)}
14: **end for**
15: $\mathbf{u}_n \leftarrow \mathbf{u}_{n,K}$
16: $\mathbf{w}_n \leftarrow \mathbf{w}_{n,K}$
```

and zeros otherwise. We select the $\mathcal{I}_n$ to index the γ fraction of parameters with the largest absolute value in the vector $\mathbf{u}_{n-1}$ (Line 1). We update $\mathbf{u}_n$ at the end of each training phase to reflect the latest Adam update for all parameters (Line 15). In the first training phase, $\mathcal{I}_n$ is selected uniformly at random.

At the end of each training phase, the server sends the updated parameters $\mathbf{w}_n$ and their indices $\mathcal{I}_n$. For the indices, it sends a bit-vector identifying the location of the parameters. As the bit-vector is sparse, it can be compressed and we use gzip [113] in our implementation to carry this out. All in all, using gradient-guided coordinate descent to send 5% of the parameters in each model update reduces downlink bandwidth by 13.3× with negligible loss in performance compared to updating the complete model (§4.4.2).

4.3.2 Reducing Uplink Bandwidth

AMS adjusts the frame sampling rate at edge devices dynamically based on the extent and speed of scene change in a video. This helps reduce uplink (edge-to-server) bandwidth and server load for stationary or slowly-changing videos.

To obtain a robust signal for scene change, we define a metric, *ϕ-score*, that tracks the rate of change in the *labels* associated with video frames. Compared to raw pixels, labels typically take values in a much smaller space (e.g., a few object classes), and therefore provide a more robust signal for measuring change. The server computes the ϕ-score using the teacher model's labels. Consider a sequence of frames $\{\mathbf{I}_k\}_{k=0}^n$, and denote the teacher's output on these frames by $\{\mathcal{T}(\mathbf{I}_k)\}_{k=0}^n$. For every frame $\mathbf{I}_k$, we define ϕ_k using the same loss function that defines the task, but computed using $\mathcal{T}(\mathbf{I}_k)$ and $\mathcal{T}(\mathbf{I}_{k-1})$ respectively to be the prediction and ground-truth labels. In other words, we set ϕ_k to be the loss (error) of the teacher model's prediction on I_k with respect to the label $\mathcal{T}(\mathbf{I}_{k-1})$. Hence, the smaller the ϕ_k score, the more alike are the labels for $\mathbf{I}_k$ and $\mathbf{I}_{k-1}$, i.e., stationary scenes tend to achieve lower scores.

The server measures the average ϕ-score over recent frames, and periodically (e.g., every $\delta t = 10$ sec) updates the sampling rate at the edge device to try to maintain the ϕ-score near a target value ϕ_{target}:

$$r_{t+1} = \Big[r_t + \eta_r \cdot \big(\bar{\phi}_t - \phi_{target} \big) \Big]_{r_{min}}^{r_{max}}, \tag{4.1}$$

where η_r is a step size parameter, and the notation $[\cdot]_{r_{min}}^{r_{max}}$ means the sampling rate is limited to the range $[r_{min}, r_{max}]$. We use $r_{min} = 0.1$ fps (frames-per-second) and $r_{max} = 1$ fps in our implementation.

Figure 4-3 shows an example of adaptive sampling rate for a driving video. Notice how the sampling rate decreases when the car stops behind a red traffic light, and then increases once the light turns green and the car starts moving.

Compression. The edge device does not send sampled frames immediately. Instead it buffers samples corresponding to one model update interval (T_{update}, which the server communicates to the edge), and it runs H.264 [114] video encoding on this buffer to compress it before transmission. The time taken at the edge device to fill the compression buffer and transmit a new batch of samples is hidden from the server by overlapping it with the training phase of the previous step. Performance isn't overly sensitive to the latency of delivering training data. As a result, it is possible to operate H.264 in a slow mode, achieving significant compression. Compressing the buffered samples in our experiments took at most 1 second.

4.4 Evaluation

4.4.1 Methodology

Datasets. We evaluate AMS on the task of semantic segmentation using four video datasets: Cityscapes [115] driving sequence in Frankfurt (1 video, 46 mins long)[1], LVS [104] (28 videos, 8 hours in total), A2D2 [116] (3 videos, 36 mins in total), and Outdoor Scenes (7 videos, 1.5 hours in total), which we collected from Youtube to cover a range of scene

[1]This video sequence is not labeled and was the only long video sequence available from Cityscapes (upon request).

variability, including fixed cameras and moving cameras at walking, running, and driving speeds (see Appendix A for details and samples from Outdoor Scenes videos).

Metric. To evaluate the accuracy of different schemes, we compare the inferred labels on the edge device with labels extracted for the same video frames using the teacher model. For Cityscapes, A2D2, and Outdoor Scenes datasets we use DeeplabV3 [100] model with Xception65 [108] backbone (2048×1024 input resolution) trained on the Cityscapes dataset [115] as the teacher model. For LVS, we follow Mullapudi *et al.* [104] in using Mask R-CNN [117] trained on the MS-COCO dataset [118] as the teacher model. Labeling each frame using the teacher models takes 200–300ms on a V100 GPU. We report the mean Intersection-over-Union (mIoU) metric relative to the labels produced by this reference model. The metric computes the Intersection-over-Union (defined as the number of true positives divided by the sum of true positives, false negatives and false positives) for each class, and takes a mean over the classes. We manually select a subset of most common output classes in each of these videos as summarized in table A.1 in appendix A.

Dataset	**Metric**	**No Custom.**	**One-Time**	**Remote+Tracking**	**Just-In-Time**	**AMS**
Outdoor Scenes	mIoU (%)	63.68	69.73	69.05	73.14	**74.26**
	Up/Down BW (Kbps)	0/0	63.1/91.4	1949/54.6	2735/3109	189/205
A2D2 [116]	mIoU (%)	62.05	50.78	63.25	69.23	**69.31**
	Up/Down BW (Kbps)	0/0	56.9/100	1927/40.5	2487/2872	158/203
Cityscapes [115]	mIoU (%)	73.08	63.90	66.53	**75.75**	75.66
	Up/Down BW (Kbps)	0/0	8.2/49.2	1667/50.8	2920/3294	164/226
LVS [104]	mIoU (%)	59.32	64.88	61.52	65.70	**67.38**
	Up/Down BW (Kbps)	0/0	48.1/77.4	1865/21.6	2456/3264	165/207

Table 4.1: Comparison of mIoU (in percent), Uplink and Downlink bandwidth (in Kbps) for different methods across 4 video datasets.

Schemes. On the edge device, we use the DeeplabV3 with MobileNetV2 [28] backbone at a 512×256 input resolution, which runs smoothly in real-time at 30 frame-per-second (fps) on a Samsung Galaxy S10+ phone's Adreno 640 GPU with less than 40 ms camera-to-label latency. We use a single NVIDIA Tesla V100 GPU at the server for all schemes.

We compare the following schemes:

- **No Customization:** We run a pre-trained model on the edge device without video-specific customization. For the LVS dataset, we use a checkpoint pre-trained on PASCAL VOC 2012 dataset [119]. For the rest of the datasets we used a checkpoint pre-trained for Cityscapes [115].
- **One-Time:** We fine-tune the entire model on the first 60 seconds of the video at the server and send it to the edge. This adaptation happens only once for every video. Comparing this scheme with AMS will show the benefit of *continuous* adaptation.

- **Remote+Tracking:** We use the teacher model at a remote server to infer the labels on sample frames (one frame every second), which are then sent to the device. The device locally interpolates the labels to 30 frames-per-second using optical flow tracking [36, 109]. For tracking, we use the OpenCV implementation of Farneback optical flow estimation [120] with 5 iterations, Gaussian filters of size 64, 3 pyramid levels, and a polynomial degree of 5 at 1024×512 resolution. Although it takes 700 ms to compute the flows for each frame in our tests on a Linux CPU machine, we assumed an optimized implementation with edge hardware support can run in real-time [121] in favor of this approach. We set the sampling rate to 1 fps, which matches the the maximum sampling rate for AMS. Note that, unlike AMS, this approach cannot apply the "buffer compression" method (see §4.3.2) as the buffering latency would make the labels stale. To avoid accuracy loss, we send the samples at full quality with this scheme; this requires about 2 Mbps of uplink bandwidth.[2]
- **Just-In-Time:** We deploy the online distillation algorithm proposed by [104] at the sever. This scheme trains the student model on the most recent sample frame until its training accuracy meets a threshold. Using the default parameters, it increases the sampling/training frequency (up to one model update every 266 ms) if it cannot meet the threshold accuracy within a maximum number of training iterations. Mullapudi *et al.* [104] also propose a specific lightweight model, JITNet. However, their Just-In-Time adaptation algorithm is general and can be used with any model. We evaluated Just-In-Time training with both our default student model (DeeplabV3 with MobileNetV2 backbone) and the JITNet architecture, and found they achieve similar performance in terms of both accuracy (less than 2% difference in mIoU) and number of model updates.[3] Hence, we report the results of this approach for the same model as AMS for a more straightforward comparison. Similar to AMS, we use the gradient-guided strategy (§4.3.1) for this scheme to adapt 5% of the model parameters in each update, which actually achieves a slightly better overall performance (e.g., 1.2% mIoU increase on Outdoor Scenes dataset) than updating the entire model. We also tried using ASR for Just-In-Time. While adding ASR reduced the uplink bandwidth requirement by a factor of 2, it was still 7× larger than AMS's uplink bandwidth and dropped the mIoU by 1.74% as Just-In-Time overfits very aggressively. Thus we use Just-In-Time with its original sampling strategy for a more fair comparison. The accuracy threshold is a controllable hyper-parameter that determines the frequency of model updates. A higher

[2]For reference, sending one frame per second with a good JPEG quality (quality index of 75) at this resolution requires ∼700 Kbps of bandwidth.

[3]Our implementation of the JITNet model on Samsung Galaxy S10+ mobile CPU runs 2× slower at the same input resolution compared to DeeplabV3 with MobileNetV2 backbone.

Description	No Cust.	One-Time	Rem.+Trac.	JIT	AMS
Interview	71.91	87.40	**89.98**	86.47	87.75
Dance recording	72.80	84.26	**86.41**	84.40	83.88
Street comedian	54.49	65.06	58.81	69.79	**72.03**
Walking in Paris	69.94	67.63	69.59	75.22	**75.87**
Walking in NYC	49.05	54.96	54.49	56.54	**59.74**
Driving in LA	66.26	66.30	66.48	70.95	**71.01**
Running	61.32	62.51	57.57	68.64	**69.55**

Table 4.2: Impact of the scene variations pace on mIoU (in percent) for different methods across the videos in Outdoor Scenes dataset.

threshold achieves better accuracy at the cost of higher downlink bandwidth for sending model updates. We set the accuracy threshold to achieve roughly the same accuracy as AMS on each video, allowing us to compare their bandwidth usage at the same accuracy. Using Just-In-Time's default threshold (75%) improves overall accuracy by 1.0% at the cost of 3.3× higher bandwidth. Following [104], we use the Momentum Optimizer [122] with a momentum of 0.9.

- **AMS:** We use Algorithm 2 at the server with $T_{horizon} = 240$ sec, and $K = 20$ iterations. We set the ASR parameters r_{min} and r_{max} to 0.1 and 1 frames-per-seconds respectively, with $\delta t = 10$ sec. Unless otherwise stated, 5% of the model parameters are selected using the gradient-guided strategy. In the uplink, we compress and send the buffer of sampled frames described in §4.3.2 using H.264 in the two-pass mode at medium preset speed and a target bitrate of 200 Kbps. We used AMS with the same set of hyper-parameters for all 39 videos across the four datasets. For training, we use Adam optimizer [97] with a learning rate of 0.001 ($\beta_1 = 0.9$, $\beta_2 = 0.999$).

4.4.2 Results

Comparison to baselines. Table 4.1 summarizes the results across the four datasets. We report the mIoU, uplink and downlink bandwidth, averaged over the videos in each dataset. We also report per-video results for the Outdoor Scenes dataset in Table 4.2. The main takeaways are:

1. Adapting the edge model provides significant mIoU gains. AMS achieves 0.4–17.8% (8.3% on average) better mIoU score than No Customization.
2. One-Time is sometimes better and sometimes worse than No Customization. Recall that One-Time specializes the model based on the first minute of a video. When the first minute is representative of the entire video, One-Time can improve accuracy. However, on videos that vary significantly over time (e.g., driving scenes in A2D2 and Cityscapes), customizing the model for the first minute can backfire. By contrast, AMS consis-

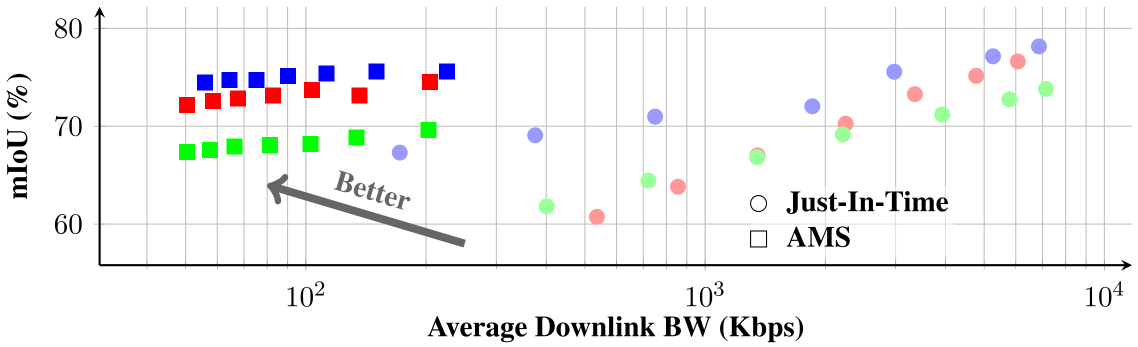


Figure 4-4: mIoU vs. downlink bandwidth for AMS and Just-In-Time with different parameters. Each color represents one dataset and each marker's shape/tint represents the scheme.

tently improves accuracy (up to 39.1% for some videos, 4.3% on average compared to One-Time) since it continually adapts the model to video dynamics. Continuous training may overfit the model when the scene does not change for a long time, which is why One-Time marginally outperforms it in the Dance recording video. We discuss a simple mechanism for adaptation of the training rate in Appendix B.

3. Remote+Tracking performs better on static videos since optical flow tracking works better in these cases. However, it struggles on more dynamic videos and performs worse than AMS (up to 24.4% on certain videos, 5.8% on average). For example, note that in table 4.2, Remote+Tracking performs no better than No Customization (which does not use the network) on the Driving in LA, Walking in Paris, and Running videos. Remote+Tracking requires much less bandwidth in the downlink compared to Just-In-Time and AMS as it downloads labels rather than model updates. However, in the uplink it requires about 2Mbps of bandwidth since it cannot buffer and compress frames to ensure it receives labels with low latency (unlike AMS).
4. Just-In-Time achieves the closest overall mIoU score to AMS, but it requires 4.4–44.5$\times$ more downlink bandwidth (15.7$\times$ on average), and 5.2–37.1$\times$ more uplink bandwidth (16.8$\times$ on average) across all videos. Across all videos, AMS requires only 181–225 Kbps downlink bandwidth and 57–296 Kbps uplink bandwidth.

Impact of AMS and Just-In-Time parameters. Both AMS and Just-In-Time have parameters that affect their accuracy and model-update frequency. To compare these schemes more comprehensively, we sweep these parameters and measure the mIoU and downlink bandwidth they achieve at each operating point. For Just-In-Time, we vary the target accuracy threshold in the interval 55–85 percent, and for AMS, we vary T_{update} between 10 to 40 seconds. Figure 4-4 shows the results for 3 datasets (Cityscapes, A2D2, and Outdoor Scenes).[4] Comparing the data points of the same color (same dataset) for the two

[4]We omit the LVS dataset from these results to reduce cost of running the experiments in the cloud.

	Fraction			
Strategy	20%	10%	5%	1%
Last Layers	-5.98	-6.58	-8.98	-10.99
First Layers	-2.63	-5.54	-8.37	-15.45
First&Last Layers	-1.0	-2.29	-3.54	-7.30
Random Selection	-0.21	-0.70	-2.90	-5.29
Gradient-Guided	+0.13	-0.13	-0.73	-2.87
BW (Kbps)	715	384	205	46
Full model BW (Kbps)		**3302**		

Table 4.3: Average difference in mIoU relative to full-model training (in percent) for different coordinate descent strategies on the Outdoor Scenes dataset.

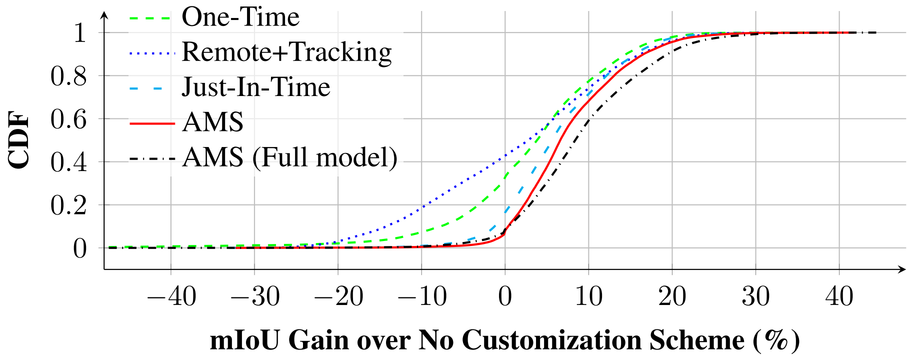


Figure 4-5: CDF of mIoU gain relative to No Customization across all frames for different schemes.

schemes, we observe that Just-In-Time requires about 10× more bandwidth to achieve the same accuracy as AMS. Note that we apply our gradient-guided parameter selection to Just-In-Time; without this, it would have required 150× more bandwidth than AMS. AMS is less sensitive to limited bandwidth than Just-In-Time (notice the difference in slope of mIoU vs. bandwidth for the two schemes). As discussed in §4.3.1, the reason is that AMS trains the student model over a longer time horizon (as opposed to a single recent frame). Thus it generalizes better and can tolerate fewer model updates more gracefully.

Impact of the gradient-guided method. Table 4.3 compares the gradient-guided method descibed in §4.3.1 with other approaches for selecting a subset of parameters (coordinates) in the training phase on the Outdoor Scene dataset. The *First*, *Last*, and *First&Last* methods select the parameters from the initial layers, final layers, and split equally from both, respectively. *Random* samples parameters uniformly from the entire network. Gradient-guided performs best, followed by Random. Random is notably worse than gradient-guided when training a very small fraction (1%) of model parameters. The methods that update only the

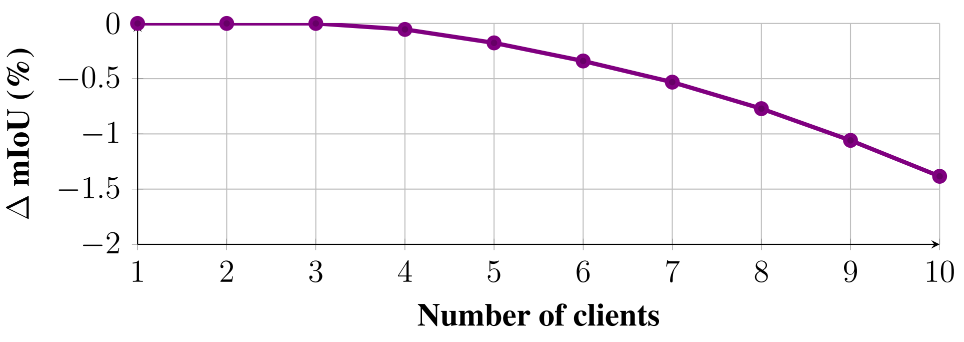


Figure 4-6: Average multiclient mIoU degradation compared to single-client performance on Outdoor Scenes dataset.

first or last model layers are substantially worse than the other approaches.

Overall, Table 4.3 shows that AMS's gradient-guided method is very effective. Sending only 5% of the model parameters results in only 0.73% loss of accuracy on average (on the Outdoor Scenes dataset), but it reduces the downlink bandwidth requirement from 3.3 Mbps for full-model updates to 205 Kbps. Moreover, in a similar experiment, gradient-guided outperforms using SGD with the Gauss-Southwell selection rule at all fractions of model updates, with their gap reaching 1.94% in mIoU for a 5% fraction.

Robustness to scene changes. Does AMS consistently improve accuracy across all frames or are the benefits limited to certain segments of video with stationary scenes? Figure 4-5 plots the cumulative distribution of mIoU improvement relative to No Customization across all frames (more than 1 million frames across the four datasets) for all schemes. AMS consistently outperforms the other schemes. Surprisingly, Just-In-Time has worse accuracy than AMS, despite updating its model much more frequently. AMS achieves better mIoU than No Customization in 93% of frames, while Just-In-Time and One-Time customization are only better 82% and 67% of the time. This shows that AMS's training strategy, which avoids overfitting to a few recent frames, is more robust and handles scene variations better.

Multiple edge devices. Figure 4-6 show the accuracy degradation (w.r.t. single edge device) when multiple edge device share a single GPU at the server in round-robin manner. By giving more GPU time to videos with more scene variation, AMS scales to supporting up to 9 edge device on a single V100 GPU at the server with less than 1% loss in mIoU (see Appendix B for more details).

Uplink sampling rate. Figure 4-7 shows the distribution of ASR's average sampling rate across different videos in four datasets. Notice that we set the ASR's maximum sampling rate (see §4.3.2), r_{max}, to 1 fps as our results show sampling faster than 1 frame-per-second provides negligible improvement in accuracy along increasing bandwidth usage and server inference overhead. We use $r_{min} = 0.1$ fps.

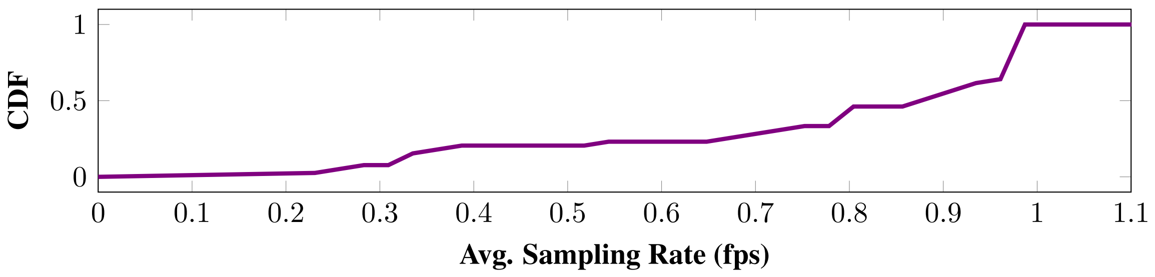


Figure 4-7: Cumulative distribution of average ASR sampling rate across all videos.

4.5 Conclusion

We presented AMS, an approach for improving the performance of real-time video inference on low-powered edge devices that uses a remote server to continually train and stream model updates to the edge device. Our design centers on reducing communication overhead: avoiding excessive overfitting, updating a small fraction of model parameters, and adaptively sampling training frames at edge devices. AMS makes over-the-network model adaptation possible with a few 100 Kbps of uplink and downlink bandwidth, levels easily sustainable on today's (wireless) networks. Our results showed that AMS improves accuracy of semantic segmentation using a mobile-friendly model by 0.4–17.8% compared to a pretrained (uncustomized) model across a variety of videos, and requires 15.4$\times$ less bandwidth to achieve similar accuracy to recent online distillation methods.

Chapter 5

SRVC: Efficient Video Compression via Content-Adaptive Super-Resolution

5.1 Overview

Recent years have seen a sharp increase in video traffic. It is predicted that by 2022, video will account for more than 80% of all Internet traffic [123, 124]. Video delivery is so bandwidth-intensive that during surge periods such as the initial months of the pandemic, Netflix and Youtube were forced to throttle video quality to reduce overheads [125, 126]. Further, while mobile devices support 1080p resolutions these days, cellular networks are still plagued by low bandwidth and frequent fluctuations in most parts of the world. Hence efficient video compression to reduce bandwidth consumption without compromising on quality is more critical than ever.

While the demand for video content has increased over the years, the techniques used to compress and transmit video have largely remained the same. Ideas such as applying Discrete Cosine Transforms (DCTs) to video blocks and computing motion vectors [114, 127] , which were developed decades ago, are still in use today. Even the latest H.265 codec improves upon these same ideas by incorporating variable block sizes [128]. Recent efforts [6, 129, 130] to improve video compression have turned to deep learning to capture the complex relationships between the components of a video compression pipeline. These approaches have had moderate success at outperforming current codecs, but they are much less compute- and power-efficient.

We present SRVC, a new approach particularly useful for cellular networks and low bitrate-scenarios, that combines existing compression algorithms with a lightweight, content-adaptive super-resolution (SR) neural network that significantly boosts performance with

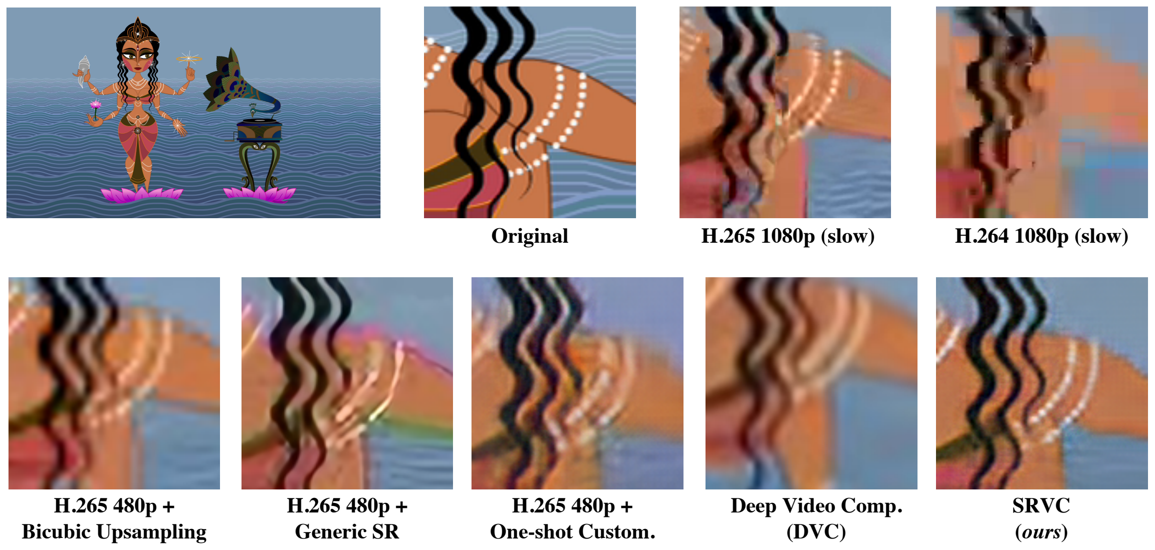


Figure 5-1: Comparing different video compression schemes at a 200 Kbps bitrate (except for DVC) on the 1560th frame of Sita Sings the Blues video in Xiph [5] dataset. DVC [6] is encoding at its lowest available bitrate that requires 4.97 Mbps in this example.

low computation cost. SRVC compresses the input video into two bitstreams: a *content stream* and a *model stream*, each with a separate bitrate that can be controlled independently of the other stream. The content stream relies on a standard codec such as H.265 to transmit low-resolution frames at a low bitrate. The model stream encodes a *time-varying* SR neural network, which the decoder uses to boost the quality of decompressed frames derived from the content stream. SRVC uses the model stream to specialize the SR network for short segments of video dynamically (e.g., every few seconds). This makes it possible to use a small SR model, consisting of just a few convolutional and upsampling layers.

Applying SR to improve the quality of low-bitrate compressed video isn't new. AV1 [131], for instance, has a mode (typically used in low-bitrate settings) that encodes frames at low resolution and applies an upsampler at the decoder. While AV1 relies on standard bicubic [132] or bilinear [133] interpolation for upsampling, recent proposals have shown that learned SR models can significantly improve the quality of these techniques [134, 135].

However, these approaches rely on *generic* SR neural networks [136–138]) that are designed to generalize across a wide range of input images. These models are large (e.g., 10s of millions of parameters) and can typically reconstruct only a few frames per second even on high-end GPUs [139]. But in many usecases, generalization isn't necessary. In particular, we often have access to the video being compressed ahead of time (e.g, for on-demand video). Our goal is to dramatically reduce the complexity of the SR model in such applications by specializing it (in a sense, overfitting it) to short segments of video.

To make this idea work, we must ensure that the overhead of the model stream is low. Even with our small SR model (with 2.22M parameters), updating the entire model every few seconds would consume a high bitrate, undoing any compression benefit from lowering the resolution of the content stream. SRVC tackles this challenge by carefully selecting a small fraction (e.g., 1%) of parameters to update for each segment of the video, using a "gradient-guided" coordinate-descent [105] strategy that identifies parameters that have the most impact on model quality. Our primary finding is that a SR neural network adapted in this manner over the course of a video can provide such a boost to quality, that including a model stream along with the compressed video is more efficient than allocating the entire bitstream to content.

In summary, we make the following contributions:

- We propose a novel dual-stream approach to video streaming that combines a time-varying SR model with compressed low-resolution video produced by a standard codec. We develop a coordinate descent method to update only a fraction of model parameters for each few-second segment of video with low overhead.

- We propose a lightweight model with spatially-adaptive kernels, designed specifically for content-specific SR. Our model runs in real-time, taking only 11 ms (90 fps) to generate a 1080p frame on an NVIDIA V100 GPU. In comparison, DVC [6] takes 100s of milliseconds at the same resolution.

- We show that, in low bitrate regimes, to achieve the same PSNR, SRVC requires only 20% of the bitrate as H.265 in its *slow* encoding mode [1], and 3% of DVC's bits-per-pixel. SRVC's quality improvement extends across all frames in the video.

Figure 5-1 shows visual examples comparing the SRVC with these baseline approaches at competitive or higher bitrates. Our datasets and code are available at `https://github.com/AdaptiveVC/SRVC.git`

5.2 Related Work

Standard codecs. Prior work has widely studied video encoder/decoders (codecs) such as H.264/H.265 [140, 141], VP8/VP9 [142, 143], and AV1 [131]. These codecs rely on hand-designed algorithms that exploit the temporal and spatial redundancies in video pixels, but cannot adapt to specific videos. Existing codecs are particularly effective when used in

[1]To the authors' knowledge, this is the first learning-based scheme that compares to H.265 in its slow mode

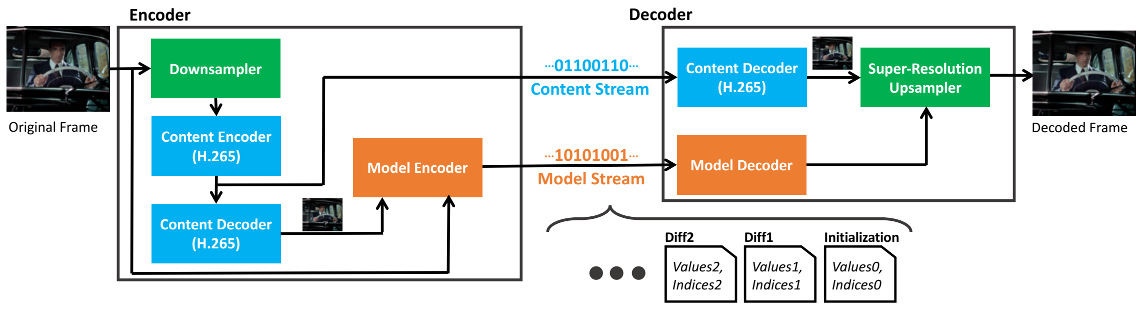


Figure 5-2: SRVC encodes video into two bitstreams.Content stream encodes downsampled low-resolution video with the existing codec. Model stream encodes periodic updates to a lightweight super-resolution neural network customized for short segments of the video.

slow mode for offline compression. Nevertheless, SRVC's combination of a low-resolution H.265 stream with a content-adaptive SR model outperforms H.265 at high resolution, even in its *slow* mode. Some codecs like AV1 provide the option to encode at low resolution and upsample using bicubic interpolation [132]. But, as we show in §5.4, SRVC's learned model provides a much larger improvement in video quality compared to bicubic interpolation.

Super resolution. Recent work on single-image SR [137, 138] and video SR [134, 135, 144, 145] has produced a variety of CNN-based methods that outperform classic interpolation methods such as bilinear [133] and bicubic [132]. Accelerating these SR models has been of interest particularly due to their high computational complexity at higher resolutions [146]. Our design adopts the idea of subpixel convolution [147], keeping the spatial dimension of all layers identical to the low-resolution input until the final layer. Fusing the information from several video frames has been shown to further improve single-image SR models [148]. However, to isolate the effects of using a content-adaptive SR model, we focus on single-image SR in this work.

Learned video compression. End-to-end video compression techniques [6, 129, 130, 149, 150] follow a compression pipeline similar to standard codecs but replace some of the core components with DNN-based alternatives, e.g., flow estimators [151] for motion compensation and auto-encoders [152] for residue compression. However, running these models in real time is challenging. For example, even though the model in [130] is explicitly designed for low-latency video compression, it decodes only 10 frames-per-second (fps) for 640×480 resolution on an NVIDIA Tesla V100 [130]. In contrast, H.264 and H.265 process a few hundred frames a second at the same resolution. Moreover, existing learned video compression schemes are designed to generalize and not targeted to specific videos. Few approaches have proposed overfitting [153] and updating only specific layers [154] of the SR model, yet do not go as far as presenting a holistic solution and an extensive evaluation. In this work, we show that augmenting existing codecs with content-adaptive SR achieves better quality and compression than end-to-end learned compression schemes.

Lightweight models. Lightweight models intended for phones and compute-constrained devices have been designed manually [28] and using neural architecture search techniques [29, 30]. Model quantization and weight pruning [31–34] have helped reduce the computation footprint of models with a small loss in accuracy. Despite the promise of these optimizations, the accuracy of these lightweight models falls short of state-of-the-art solutions. SRVC is complementary to such optimization techniques and would benefit from them.

5.3 Methods

Figure 5-2 shows an overview of SRVC's compression pipeline. SRVC compresses video into two bitstreams:

1. **Content stream:** The encoder downsamples the input video frames by a factor of k in each dimension (e.g., k=4) to generate low-resolution (LR) frames using area-based downsampling. It then encodes the LR frames using an off-the-shelf video codec to generate the content bitstream (our implementation uses H.265 [128]). The decoder decompresses the content stream using the same codec to reconstruct the LR frames. Since video codecs are not lossless, the LR frames at the decoder will not exactly match the LR frames at the encoder.

2. **Model stream:** A second bitstream encodes the SR model that the decoder uses to upsample the each decoded LR frame. We partition the input video into N fixed-length segments, each τ seconds long (e.g., $\tau = 5$). For each segment $t \in \{0, ..., N-1\}$, we adapt the SR model to the frames in that segment during encoding. Specifically, the encoder trains the SR model to map the LR decompressed frames within a segment to high-resolution frames. Let $\mathbf{\Theta}_t$ denote the SR model parameters obtained for segment t. The model adaptation is sequential: the training procedure for segment t initializes the model parameters to $\mathbf{\Theta}_{t-1}$. The model stream encodes the sequence $\mathbf{\Theta}_t$ for $t \in \{0, ..., N-1\}$. It starts with the full model $\mathbf{\Theta}_0$, and then encodes the *changes* in the parameters for each subsequent model update, i.e., $\mathbf{\Delta}_t = \mathbf{\Theta}_t - \mathbf{\Theta}_{t-1}$. The decoder updates the parameters every τ seconds, using the last model parameters $\mathbf{\Theta}_{t-1}$ to find $\mathbf{\Theta}_t = \mathbf{\Theta}_{t-1} + \mathbf{\Delta}_t$.

The model stream adds overhead to the compressed bitstream. To reduce this overhead, we develop a small model that is well-suited to *content-specific* SR (§5.3.1), and design an algorithm that significantly reduces the overhead of model adaptation by training only a small fraction of the model parameters that have the highest impact on the SR quality in each segment (§5.3.2).

5.3.1 Lightweight SR Model Architecture

Existing SR models typically use large and deep neural networks (e.g., typical EDSR has 43M parameters across more than 64 layers [139]), making them difficult to use in a real-time video decoder. Moreover, adapting a large DNN model to specific video content and transmitting it to the decoder would incur high overhead.

We propose a new lightweight architecture that keeps the model small and shallow, and yet, is very effective at content-based adaptation (§5.4.2). Our model is inspired by classical algorithms like bicubic upsampling [155], which typically use only one convolutional layer and a fixed kernel for upsampling the entire image. It uses this basic architecture but replaces the fixed kernel with spatially-adaptive kernels that are customized for different regions of the input frame. Our model partitions each frame into patches, and uses a shallow CNN operating on the patches to generate different (spatially-adaptive) kernels for each patch (Fig. 5-3).

More formally, the model first partitions an input frame into equal-sized patches of $P \times P$ pixels (e.g. $P = 5$ pixels) using a common space-to-batch operation. For each patch, a patch-specific block (Adaptive Conv Block in Fig. 5-3) computes a 3×3 convolution kernel with 3 input and F output channels ($27F$ parameters) using a two-layer CNN, and applies this kernel (pink box) to the patch. The forward pass of the adaptive conv block with input patch $\mathbf{x} \in \mathbb{R}^{P\times P\times 3}$ and output features $\mathbf{y} \in \mathbb{R}^{P\times P\times F}$ is summarized as follows:

$$\begin{aligned} \mathbf{w} &= f(\mathbf{x}), \\ \mathbf{y} &= \sigma(\mathbf{w} * \mathbf{x}). \end{aligned}$$

We use a two-layer CNN to model $f(\cdot)$ in our architecture. We finally reassemble the feature patches (batch-to-space) and compute the output using another two-layer CNN followed by a pixel shuffler (depth-to-space) [147] that brings the content to the higher resolution. All convolutions have a kernel height and width of 3, except for the first layer of the regular block that uses kernel size of 5.

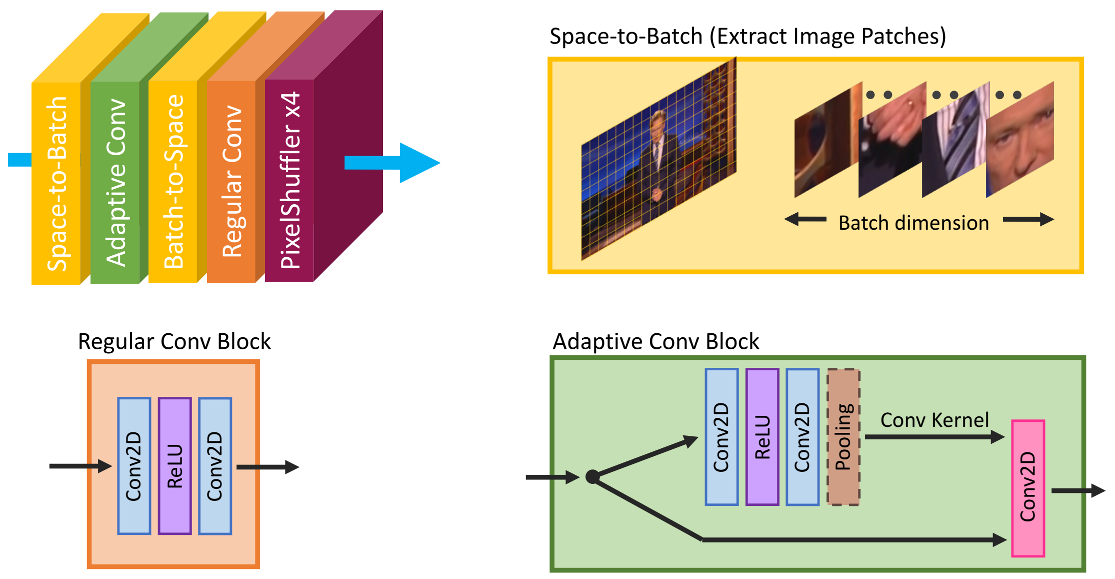


Figure 5-3: SRVC lightweight SR model architecture.

5.3.2 Model Adaptation and Encoding

Training algorithm. We use the L2-loss between the SR model's output and the corresponding high-resolution frame (input to the encoder), over all the frames in each segment to train the model for that segment. Formally, we define the loss as

$$L(\Theta_t) = \frac{1}{n|F_t|} \sum_{i=1}^{n} \sum_{j=1}^{|F_t|} ||Y_{ij} - X_{ij}||^2$$

where $|F_t|$ is the number of frames in the t^{th} segment, each with n pixels, and Y_{ij} and X_{ij} denote the value of the i^{th} pixel in the j^{th} frame of the decoded high-resolution output frame and the original high-resolution input frame respectively. During the training, we randomly crop the samples at half of their size in each dimension. We use Adam optimizer [97] with learning rate of 0.0001, and first and second momentum decay rates of 0.9 and 0.999.

To reduce the model stream bitrate, we update only a fraction of the model parameters across video segments. Our approach is to update only those parameters that have the most impact on the model's accuracy. Specifically, we update the model parameters with *the largest gradient magnitude* for each new segment as follows. First, we save a copy of the model at the beginning of a new segment and perform one iteration of training over all the frames in the new segment. We then choose the fraction η of the parameters with the largest magnitude of change in this iteration, and reset the model parameters to the starting saved copy. We apply the Adam updates for only the selected paramaters and discard the updates for the rest of the model (keeping those parameters fixed).

Encoding the model stream. To further compress the model stream, we only transmit changes to the model parameters at each update. We encode the model updates into a bitstream by recording the indices and associated change in values of the model parameters (Fig. 5-2). SRVC's model encoding is lossless: the encoder and decoder both update the same subset of parameters during each update. To update a fraction η of the parameters for a model with M float16 parameters, we need an average bitrate of at most $(16+\log(M))\times\eta M/\tau$ to express the deltas and the indices every τ seconds. For example, with model size $M = 2.22$ million parameters (F=32, see Table 5.2), $\tau = 10$ seconds, and $\eta = 0.01$, we only require 82 Kbits/sec to encode the model stream required to generate 1080p video. To put this number into perspective, Netflix recommends a bandwidth of 5 Mbits/sec at 1080p resolution [156]. The model stream can be compressed further using lossy compression techniques or by dynamically varying η or the model update frequency based on scene changes.

Training the SR model for 1080p resolution and encoding the updates into the model stream takes about 12 minutes for each minute worth of video with our un-optimized implementation. However, given the small compute overhead of our lightweight model, we shared a V100 GPU between five simultaneous model training (encoding) processes without any significant slow down to any process. Hence, the overall throughput of the encoding on V100 GPU is about 2.5 minutes of training per minute of content. We consider this duration feasible for offline compression scenarios where videos are available to content providers well ahead of viewing time. We believe that there is significant room to accelerate the encoding process too with standard techniques (e.g., training on sampled frames rather than all frames) and further engineering. We leave an exploration of these opportunities to future work.

5.4 Experiments

5.4.1 Setup

Dataset. Video datasets like JCT-VC [157], UVG [158] and MCL-JCV [159], consisting of only a few hundred frames ($\sim$10 sec) per video, are too short to evaluate SRVC's content-adaptive SR. Hence, we train and test the efficacy of SRVC on a custom dataset consisting of 28 downloadable videos from Vimeo (short films) and 4 full-sequence videos from the Xiph Dataset [5]. We trim all videos to 10 minutes and resize them to 1080p resolution in RAW format from their original 4K resolution and MPEG-4 format using area-based

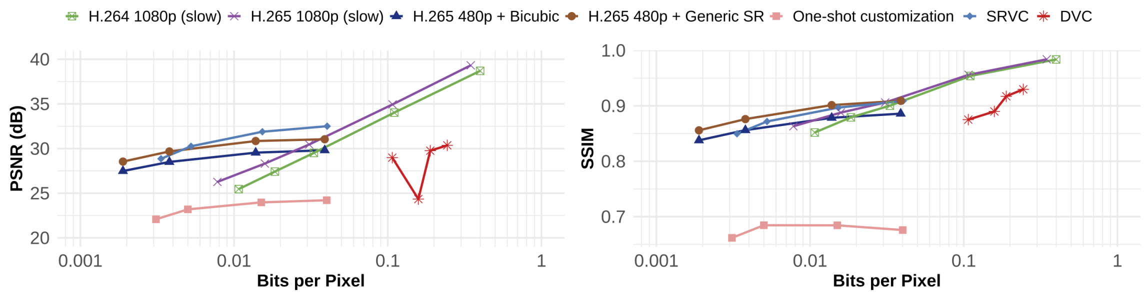


Figure 5-4: Tradeoff between video quality and bits-per-pixel for different approaches on three long videos from the Xiph dataset. SRVC with content-adaptive streaming reduces the bitrate consumption to 16% of current codecs and ∼2% of end-to-end compression schemes like DVC. Though comparable in video quality to SRVC, the generic SR approach does not run in real-time.

interpolation [160]. We use the resulting 1080p frames as our high-resolution source frames in our pipeline. We re-encode each video's raw frames at different qualities or Constant Rate Factors (CRFs) on H.264/H.265 to control the bitrate. We also use area-interpolation to downsample the video to 480p and encode the low-resolution (LR) video using H.265 at different CRFs to achieve different degrees of compression. The SR model in SRVC is then trained to learn the mapping from each LR video at a particular CRF to the original 1080p video at its best quality.

Baselines. We compare the following approaches. The first four only use a content stream while the next three use both a content stream and a model stream. The last approach is an end-to-end neural compression scheme.

- **1080p H.264/H.265:** We use ffmpeg and the libx264/libx265 codec to re-encode each of the 1080p videos at different CRFs using the *slow* preset.

- **480p H.265 + Bicubic upsampling:** We use ffmpeg and the libx265 codec to downsample the 1080p original video to LR 480p at different CRFs using area-interpolation and the *slow* preset. This approach's bitrate comes only from its content stream: the downsampled 480p frames encoded using H.265. We use bicubic interpolation to upsample the 480p videos back to 1080p. This isolates the bitrate reduction from just encoding at lower resolutions.

- **480p H.265 + Generic SR:** Instead of Bicubic upsampling, we use a more sophisticated DNN-based super-resolution model (EDSR [139] with 16 residual blocks) to upsample the 480p frames to 1080p. The upsampling takes about 50ms for each frame (about 5× worse than SRVC). We use a pre-trained checkpoint that has been trained on a generic corpus of images [161]. Since we expect all devices to be able to pre-fetch such a model, this approach only has a content stream at 480p encoded using H.265. Thus, its bits-per-pixel value is identical to the Bicubic case.

- **480p H.265 + One-shot Customization:** We evaluate a version of SRVC that uses a lightweight SR model (§5.3.1) without the model adaptation procedure. For this, we train our SR model exactly once (one-shot) using the entire 1080p video and encode it in the model stream right at the beginning before any LR content. The content stream for this approach comprises of the 480p H.265 video while the model stream consists of a single initial model customized to the entire video duration. The overhead of the model is amortized over the entire video and added to the content bitrate when computing the total bits-per-pixel value.

- **480p H.265 + SRVC:** We evaluate SRVC which uses the same initial SR model as One-shot Customization but is periodically adapted to the most recent 5 second segment of the video. To train this model, we use random crops (half the frame size in each dimension) from each reference frame within a video segment. The content stream for SRVC relies on standard H.265. The model stream, on the other hand, is updated every 5 seconds and is computed using our gradient-guided strategy, which only encodes the change to those parameters that have the largest gradients in each video segment (§5.3.2). To compute the total bits-per-pixel, we add the model stream's bitrate (computed as described in §5.3.2) to the content stream's bitrate. We also add the overhead of sending the initial model in full to the model stream's bitrate.

- **DVC:** An official checkpoint [162] of Deep Video Compression [6], an end-to-end neural network based compression algorithm. To evaluate DVC, we compute the PSNR and SSIM metrics, and use Lu *et al.*'s[6] estimator to measure their required bits-per-pixel for every frame at four different bitrate-distortion trade-off operating points ($\lambda \in \{256, 512, 1024, 2048\}$).

Model and training procedure. Our model uses 32 output feature channels in the adaptive convolution block, resulting in 2.22 million parameters. However, only 1% of them are updated by the model stream and that too, only every 5 seconds. We vary the number of output feature channels, the fraction of model parameters updated, and the update interval to understand its impact on SRVC's performance.

Metrics and color space. We compute the average Peak Signal-To-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) across all frames at the output of the decoder (after upsampling). We report PSNR based on the mean square error across all pixels in the video (over all frames) where the pixel-wise error itself is computed on the RGB

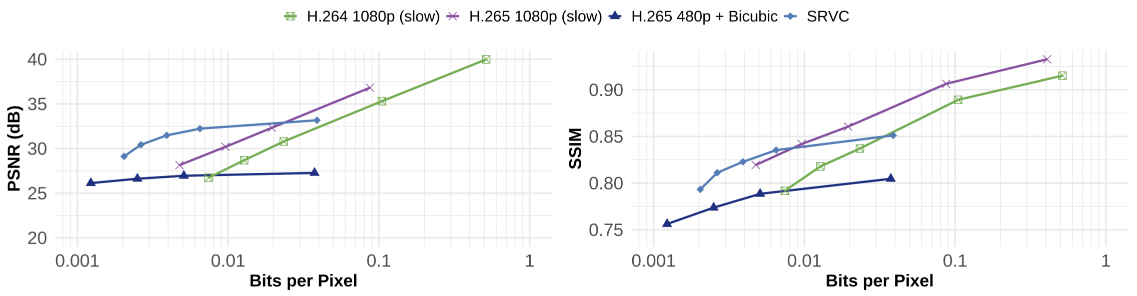


Figure 5-5: Tradeoff between video quality and bits-per-pixel for different approaches on 28 videos from Vimeo. To achieve 30dB PSNR, SRVC requires 10% and 25% of the bits-per-pixel required by H.264 and H.265 in their *slow* modes.

space. SSIM is computed as the average SSIM between the decoded frames and their corresponding high-resolution original counterparts. However, since variations in frame quality over the course of a video can have significant impact on users' experience, we also show a CDF of both PSNR and SSIM across all frames in the video.

We compute the content bitrate for the all approaches relying on H.264/5 at both 1080p and 480p using ffmpeg. For approaches that stream a model in addition to video frames, we compute the model stream bitrate based on the total number of model parameters, the fraction of them that are streamed in each update interval, and the frequency of updates (§5.3.2). The content and model stream bitrates are combined to compute a single bits-per-pixel metric. Note that the bits-per-pixel range in our evaluations is an order-of-magnitude lower than results reported in prior work [6, 129] because our approach is designed for low-bitrate scenarios and we compare to the *slow mode* in H.264/5 which is more efficient than the "fast" and "medium" modes We plot PSNR and SSIM metrics at different bits-per-pixel to compare different schemes. Since SRVC runs inference on decoded frames as they are rendered to users, its SR model needs to run in real-time. To evaluate its feasibility, we also compare SRVC's speed in frames per second to other learning-based approaches.

5.4.2 Results

Compression performance. Fig. 5-1 shows a visual comparison of the different schemes for similar bits-per-pixel values. For DVC in this figure, we show the results for the lowest bitrate model available that ends up using 4.97 Mbps, which is significantly larger than the 200 Kbps bitrate of other schemes in this example. To compare the compression provided by different approaches across a wide range of bits-per-pixel values, we analyze the PSNR and SSIM achieved by different methods on three long Xiph [5] videos in Fig. 5-4. Tab. 5.1 summarizes the BD-Rate and BD-PSNR [163] metrics for the same experiment. Note that

Method	**BD-PSNR (dB)**	**BD-Rate (%)**
DVC	-10.04	598.76
H.264 1080p (slow)	-1.38	45.5
H.265 1080p (slow)	0	0
H.265 480p (slow) + Bicubic	+0.67	-55.81
H.265 480p (slow) + Generic SR	+2.61	-75.31
SRVC (Ours)	**+3.41**	**-80.09**

Table 5.1: BD-PSNR and BD-Rate of different approaches on Xiph dataset relative to H.265 1080p (slow).

the bits-per-pixel metric captures both the contribution of the content and the model for those approaches that use a model stream for SR. We do not report the bitrate distortion metrics for One-shot customization as its PSNR hardly overlaps with H.265.

As seen in Fig. 5-4, SRVC achieves PSNR comparable to today's H.265 standard (in *slow* mode) with far less bits-per-pixel. For instance, to achieve a PSNR of 30 dB, SRVC requires only 0.005 bits-per-pixel while H.265 and H.264 codecs, even in their slowest settings, require more than 0.03 bits-per-pixel. In BD-Rate and BD-PSNR terms (Tab. 5.1), SRVC on average achieves a 3.41dB improvement relative to H.265 slow preset at 1080p at the same bitrate, or requires only 20% of the bitrate to achieve the same PSNR. However, One-shot Customization's performs poorer than a simple bicubic interpolation. This is because SRVC's custom SR model is not large enough to generalize to the entire video, but has enough parameters to learn a small segment. It is worth noting that to achieve the same PSNR, SRVC requires only 3% of the bits-per-pixel required by DVC [6], the end-to-end neural compression scheme. SRVC's SSIM is comparable but 0.01-0.02 better than current codecs for the same level of bits-per-pixel, particularly at higher bitrates. SRVC also outperforms a generic SR approach (EDSR) by 0.8dB and 4.8% respectively on BD-Rate and BD-PSNR metrics.

Fig. 5-4 suggests that a 480p stream augmented with a generic SR model performs just as well as SRVC in terms of its PSNR and SSIM for a given bits-per-pixel level. However, typical SR models are too slow to perform inference on a single frame (about 5$\times$ slower in this case), making them unfit for real-time video delivery. To evaluate the performance of viable schemes on real-world video, we evaluate the bits-per-pixel vs. video quality trade-off on 28 videos publicly available on Vimeo. As Fig. 5-5 suggests, SRVC outperforms all other approaches on the PSNR achieved for a given bits-per-pixel value. In particular, to achieve 30dB PSNR, SRVC requires 25% and 10% of the bits-per-pixel required by H.265 and H.264 respectively.

A key takeaway from Tab. 5.1, and Figures. 5-4 and 5-5 is that for a given bitrate

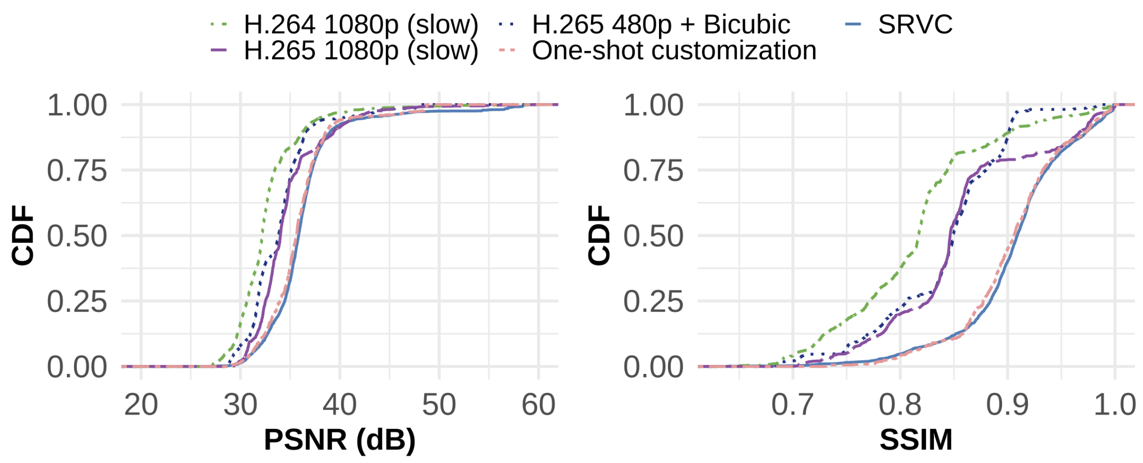


Figure 5-6: CDF of PSNR and SSIM improvements with SRVC across all video frames at a bits-per-pixel of 0.002. The quality enhancement from SRVC is not limited to only those frames that follow a model update.

budget, SRVC achieves better quality than standard codecs. This suggests that beyond a baseline bitrate for the content, it is better to allocate bits to streaming a SR model than to dedicate more bits to the content. We describe this trade-off between model and content bitrates in more detail in Fig. 5-7.

Robustness of quality improvements. To see if SRVC's improvements come from just producing a few high-quality frames right after the model is updated, we plot a CDF of the PSNR and SSIM values across all frames of the Meridian video in Fig. 5-6. We compare schemes at a bits-per-pixel value of ∼0.002. Since DVC [6] has a much higher bits-per-pixel and EDSR [139] performs poorly on this video, we exclude both approaches [2]. Firstly, we notice that both One-shot Customization and SRVC perform better than other schemes. Further, this improvement occurs over all of the frames in that no frame is worse off with SRVC than it is with the defacto codec. In fact, over 50% of the frames experience a 2–3 dB improvement in PSNR and a 0.05–0.0075 improvement in SSIM with both versions of SRVC.

Impact of number of Output Feature Channels. Since SRVC downsamples frames at the encoder and then streams a model to the receiving client who resolves the decoded frames, it is important that SRVC performs inference fast enough to run at the framerate of the video on an edge-device with limited processing power. Viewers need at least 30 fps for good quality. Consequently, the inference time on a single frame cannot afford to be longer than 33ms. In fact, the Meridian [164] video has a frame rate of 60 fps, so running low-latency inference is even more critical.

[2]Figures. 5-4, 5-5 and 5-6 cover different videos, and thus, their results cannot be directly compared.

#Feature Channels (F)	**8**	**16**	**32**	**64**	**128**
PSNR(dB)	38.49	38.69	39.87	39.89	39.90
SSIM	0.942	0.944	0.946	0.947	0.949
Inference Time (ms)	7	9	11	17	25
Num. of Parameters	0.59M	1.14M	2.22M	4.39M	8.72M

Table 5.2: Impact of number of output feature channels in SRVC's adaptive convolutional block on inference time and quality metrics for a video snippet on an NVIDIA V100 GPU.

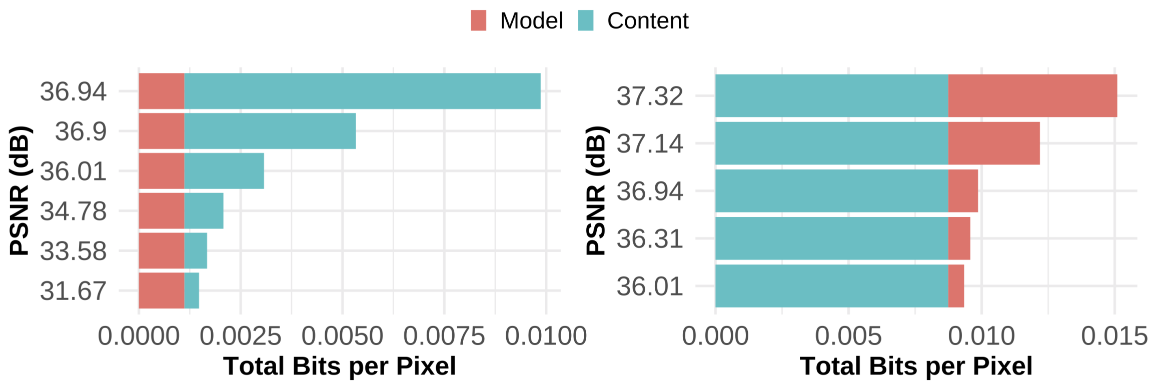


Figure 5-7: Impact of varying bits-per-pixel for the content stream for a fixed model bitrate and vice-versa. Increasing the bits-per-pixel for the low-resolution H.265 content stream improves PSNR, especially at low bitrates. At higher content bitrates, increasing the model bitrate by transmitting more model parameters further improves PSNR.

To evaluate the practicality of SRVC's lightweight model, we evaluate the end-to-end inference time per frame on an NVIDIA V100 GPU as we vary the number of the output feature channels in the adaptive convolution block (F) in Tab. 5.2. While increasing F improves the PSNR and SSIM values due to better reconstruction of the fine details, it comes at a cost. With $F = 64$ and $F = 128$, the inference times of 17 ms and 25 ms respectively causing the frame rate to drop below the input 60 fps. Further, the number of parameters increases to nearly $10M$, a steep number for the model to stream periodically. Hence, we design SRVC's model to use 32 output feature channels, ensuring it takes only 11 ms to run inference on a single frame. In comparison, the EDSR generic SR model is about 5$\times$ slower to perform inference on a single frame. Even the end-to-end neural video compression approach DVC [6] takes over hundreds of milliseconds to infer a single frame at 1080p.

Trade-off between model bitrate and content bitrate in SRVC. The presence of dedicated model and content streams in SRVC implies that the bitrate for each stream can be controlled independent of the other, to achieve different compression levels. Fig. 5-7 shows

Update Interval (s)	**5**	**10**	**15**	**20**	∞
PSNR(dB)	37.25	36.52	36.57	36.45	35.32
SSIM	0.92	0.91	0.91	0.91	0.91
Bits-per-pixel	0.006	0.003	0.002	0.0015	0

Table 5.3: Impact of SRVC's model update interval on the bits-per-pixel consumed by model updates and the associated gains in video quality. We find that an update interval of 5 seconds strikes a good trade-off between bits-per-pixel and quality.

the impact of altering the content bitrate for a fixed model bitrate and vice-versa, when encoding the Meridian video using SRVC. The content bits-per-pixel is varied by changing the quality (CRF) of the 480p H.265 stream. In contrast, the contribution from the model bits-per-pixel is controlled by the fraction of model parameters transmitted during each update.

As anticipated, for a fixed amount of model bits-per-pixel (updating 1% of the model parameters), PSNR improves as the content bitrate is increased. This is because as the quality of the underlying low-resolution H.265 frames improves, it becomes easier for the model to resolve them to their 1080p counterparts. Increasing the content bitrate from the lowest quality level of CRF 35 (with 0.0014 bits-per-pixel) to CRF 20 (with 0.003 bits-per-pixel) improves PSNR from 31 dB to 36 dB. However, increasing the bits-per-pixel for the content beyond that yields diminishing returns on PSNR (also illustrated in Fig. 5-5). At higher quality levels, Fig. 5-7 suggest that modest increases in the bits-per-pixel allocated to the model result in large improvements to the PSNR. For instance, adapting 10% of the model parameters consumes 0.006 bits-per-pixel, 6x more bits-per-pixel than adapting 0.5% of the model parameters, but results in a PSNR improvement of 1dB from 36.31 dB to 37.32 dB.

Impact of SRVC's update interval. SRVC can also control the bits-per-pixel consumed by the model stream by varying the interval over which updates to the SR model are performed. Frequent updates increase the model bitrate, but ensure better reconstruction since the model is trained on frames very similar to the current frame. An extreme scenario is an update interval of ∞ that corresponds to the One-shot Customization. Tab. 5.3 captures the impact of varying the update interval on the average quality of decoded frames from the Meridian video. We find that an update interval of 5 seconds achieves good performance without compromising much on bits-per-pixel. The fact that the PSNR does not degrade significantly for modest increases to the update interval suggests further optimizations atop SRVC that only update the model after a drastic scene change.

5.5 Conclusion

In this work, we present SRVC, an approach that augments existing video codecs with a lightweight and content-adaptive super-resolution model. SRVC achieves video quality comparable to modern codecs with better compression. Our design is a first step towards leveraging super-resolution as a video compression technique. Future work includes further optimizations to identify the pareto frontier for the model vs. content bitrate trade-off, more sophisticated techniques to detect scene changes and optimize update intervals, and the design of more effective lightweight super-resolution neural network architectures.

Chapter 6

RECL: Responsive Resource-Efficient Continuous Learning for Video Analytics

6.1 Overview

Video analytics with deep neural networks (DNNs) is a promising technology adopted in a wide range of applications such as enterprise security, retail, traffic management, and transportation [165, 166]. Across these applications, it is often imperative to run analytics tasks directly on edge devices (e.g., using on-premises edge servers [167, 168]) to ensure that the system can deliver real-time results with low latency and in compliance with data privacy constraints [13–16]. However, the edge has limited compute resources, which cannot match the unrelenting growth of video analytics workloads, DNN models, and video streams [169, 170]. Even for applications that can be deployed in resourceful environments such as public clouds, the cost of running video analytics remains exorbitant despite recent advancements in DNN resource efficiency [171–173]. For example, a high-end NVIDIA V100 GPU can only support two video streams running the state-of-the-art YOLOv5-L model [173] at 30 FPS, which translates to a steep cost of $1,100/month/stream on public clouds [174].

One common approach to reducing the resource requirements for video analytics is to use specialized and compressed DNNs [175–178]. However, owing to their inherent limits on the number of object appearances and scenes they can learn in their condensed structures, such specialized DNNs require *continuous retraining* to cope with dynamic scenes (data drifts) in order to maintain high inference accuracy. Recent work in the computer vision and systems communities [2, 179, 180] has shown the effectiveness of this approach for edge video analytics, delivering both high resource efficiency and accuracy in results.

Though promising, continuous retraining and deploying specialized DNNs has two fundamental limitations. First, continuous retraining consumes the vast majority of *compute resources* in these video analytics systems (70%–90% in our study) [2, 180], making model retraining the key bottleneck in scaling video analytics to more video streams with limited compute resources. Our study (Fig. 6-2) shows that accuracy drops sharply (by 40% in object detection) as 4× more cameras share the GPU cycles to retrain their models (§6.2.2). Second, it takes *time* to retrain specialized DNNs, and abrupt video scene changes inevitably lead to drastic accuracy drops until the retraining is completed (see Fig. 6-3 for an example). Hence, it is fundamentally challenging to uphold the accuracy lower tail during the retraining.

Our goal in this work is to address the above two fundamental limitations so that video analytics are scalable with more consistent accuracy. As retraining specialized DNNs requires resources and takes time, we aim to minimize the necessity of retraining by judiciously *reusing* historical specialized DNNs that are trained with past video segments. The intuition behind our approach is that video streams typically exhibit spatio-temporal correlations (e.g., a car drives back on the same street or another car has been on the same street before) [181]. Thus, it is likely that the current video segment bears some resemblance to historical video segments, and the corresponding historical specialized DNNs can be *reused* for the current scene. Indeed, our study in §6.2 shows that an idealized model reusing scheme can consistently deliver high accuracy (35% mAP) with limited compute resources. In comparison, existing continuous retraining systems (e.g., [2]) cannot keep up with the compute demand of more cameras, with their accuracy dropping to a low 24% mAP.

Technical challenges: Harnessing the potential of model reuse for video analytics faces two challenges. First, we need to quickly and accurately find the specialized DNN that works well for the current video segment so that we can reuse the DNN in real-time. This is difficult because it is unclear how to compare the similarity of high-dimensional and unstructured data such as video segments [182], and comparing the current video segment with all the historical video segments is not practically feasible. Second, we need to keep the cost of enabling model reuse much lower than the cost of model retraining. This is challenging as the cost of seeking through historical models grows with the size of history, while model retraining only requires fixed expenditure for each video segment. Recent video analytics solutions that reuse historical models (e.g., ODIN [182]) cannot address these challenges because they are not designed for resource efficiency.

Solution: We present RECL, a new video analytics solution that leverages historical specialized DNNs to improve scalability, responsiveness, and accuracy consistency in a resource-constrained environment. RECL is the first end-to-end system that integrates model reusing with model retraining for resource-efficient video analytics, entailing three main ideas:

• We design a *fast and robust model selection* procedure to quickly select a suitable model from the *model zoo*, a large collection of historical specialized models (§6.3.1). Our model selector is inspired by sparse gating networks in the mixture of experts (MoEs) approach [183–185], and we make it resource-efficient by decoupling the training of the gating network from the training of underlying experts. This allows RECL to select a model based on the characteristic of video analytic tasks and video scenes (e.g., detecting cars on a sunny day), which is superior to existing solutions that only consider the similarity of video frames (e.g., rainy or sunny days) [182].

• RECL *shares* the model zoo across different edge devices to enable more *model reusing* and dynamically adds new experts to the model zoo with a lightweight process to update the model selector (§6.3.3).

• RECL shares GPU resources across the retraining jobs using an *iterative training scheduler* that dynamically prioritizes retraining jobs that progress faster (§6.3.2). As a result, it spends little retraining resources on expert models that are already a good match with the current video segments.

We implement and evaluate RECL on two computer vision tasks: object detection and object classification. We compare RECL against three state-of-the-art video analytics systems (Ekya [180], AMS [2], and ODIN [182]) over a total of 71 hours of driving videos. Given the same compute resource, our evaluation shows that RECL improves the object detection mAP and image classification accuracy over the state-of-the-art solutions by up to 9.0% and 7.4%, respectively. To put these accuracy gains in perspective, the state-of-the-art mAP score for the object detection task on the PASCAL dataset has only improved by less than 8 percent in the past 6 years [186]. Moreover, the baseline systems need at least $3.2\times$ more compute resources to match RECL's accuracy. Our ablation study shows that RECL's superior performance mostly comes from effective integration of model reuse in our design. Compared to Ekya as a prior continuous training approach, RECL achieves the same accuracy up to 91 seconds faster on average. We also show that the compute overhead of RECL declines gracefully over time as more expert models are learned and added to the model zoo.

6.2 Background and Motivation

We first introduce the background of continuous retraining and deploying specialized DNNs for video analytics (§6.2.1). We then discuss the fundamental limitations of this approach and how reusing historical specialized DNNs can address these limitations effectively (§6.2.2).

6.2.1 Continuous Retraining for Video Analytics

State-of-the-art generic DNNs are often too expensive to run for video analytics all the time in resource-constrained environments such as a mobile edge computing (MEC) network [187]. A common approach is to deploy specialized and compressed DNNs (or "expert" models) that are trained using the knowledge of the generic and expensive DNNs (or the "teacher" model). The idea is to use knowledge distillation [102] to transfer the knowledge from a large teacher model to a small expert model for a specific video segment or video stream. On a matching video segment, an expert model can save compute resources by orders of magnitude while achieving similar model accuracy as the large teacher model [175, 176, 188]. This approach has been widely adopted in modern systems such as Microsoft's Rocket [177] and Google's Learn2Compress [178].

As an expert model only recognizes a limited set of object appearances and video scenes, a static expert model cannot achieve high accuracy on dynamic live videos where objects and scenes inevitably change over time (e.g., different locations, lighting conditions, object classes, etc.) [180]. A promising approach to employing expert models on dynamic live videos is to *continuously retrain* the expert model with the most recent video frames. Recent work [2, 179, 180, 189] has established that continuous retraining and deploying small expert models can simultaneously achieve high accuracy and resource efficiency on dynamic video content. Furthermore, continuous retraining has shown superior performance compared to running the large teacher model on a subset of frames and interpolating the labels (e.g., using optical flow tracking methods) [2].

Figure 6-1 can be used to illustrate the high-level components of a video analytics system that continuously retrains and deploys expert models. They include: (i) *camera service:* periodically sends new sample video frames to the adaptation service; (ii) *adaptation service:* uses the recently sampled frames to fine-tune (a copy of) the camera's expert model to mimic a larger teacher model for the current scene, and sends (or "streams") the updated expert model to the inference service; and (iii) *inference service:* uses the received lightweight expert model for real-time inference on video frames from the camera service.

This chapter focuses on the adaptation service. As retraining an expert model takes

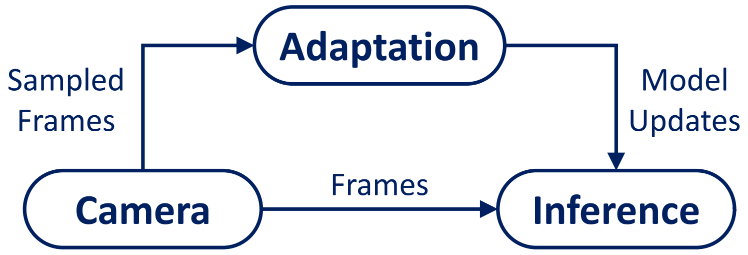


Figure 6-1: Overview of a video analytics system utilizing continuous learning. A typical adaptation module continuously retrains expert models or selects them from an existing collection of models trained in the past.

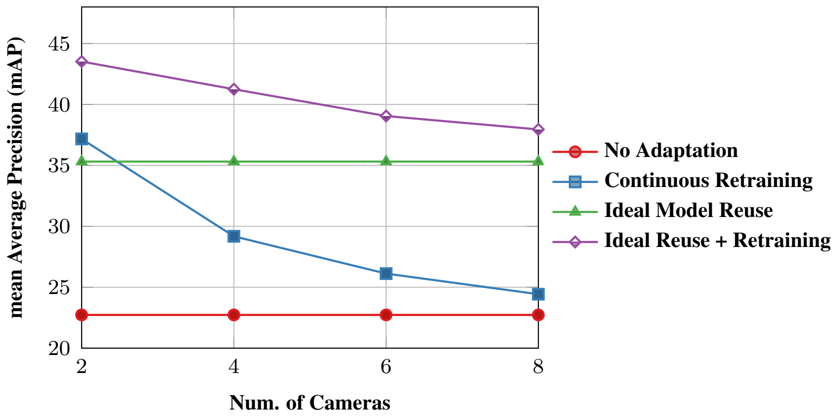


Figure 6-2: Object detection accuracy (mAP) of different designs under different numbers of cameras. Model reuse has the potential to significantly improve the accuracy in resource-constrained regimes (4, 6, and 8 cameras), and when combining model reuse and model retraining, performance *could* be greatly improved.

significant compute resources and time (§6.1), the adaptation service becomes a key bottleneck in resource efficiency and accuracy consistency. We observe that these systems [2, 180] need to spend 70%–90% of the overall compute resource on retraining their expert models. This is because model training is much more expensive than model inference. Besides, knowledge distillation needs to run the large teacher model to generate data labels on the sampled frames. In order to address this fundamental challenge, we need an effective approach to minimize the necessity of invoking expert retraining.

6.2.2 The Case of Reusing Historical Expert Models

It is well known that a video deployment usually exhibits temporally and spatially recurrent patterns [181, 182, 190]. Similar video scenes reoccur on the same camera at a similar time of day (e.g., morning or night), weather (e.g., sunny or raining), and location (e.g., a drone revisits the same street). More importantly, a video scene from one camera can also appear on *other* cameras, especially those in the same geographical vicinity, such as a self-driving car visiting a place that other cars in the same fleet have seen. These temporal and spatial correlations imply that some expert models trained on video scenes in the past could perform reasonably well on the current video scene, and we can potentially leverage these historical expert models to minimize the necessity of retraining.

To empirically show the potential of reusing historical expert models, we use a total of 71 hours of driving videos collected from YouTube (more details in §6.5.1). The large teacher model is a state-of-the-art object detector DNN, YOLOX-X (282 GFLOPs), and the expert model is a much smaller variant YOLOX-Nano (1 GFLOPs) [173]. Similar to existing continuous retraining solutions, we train one expert model for each 30-second video segment. We create a model zoo using all the expert models trained on the first 30 hours of the videos ("training data"), and we use the remaining 41 hours of the videos ("test data") to report the object detection accuracy.

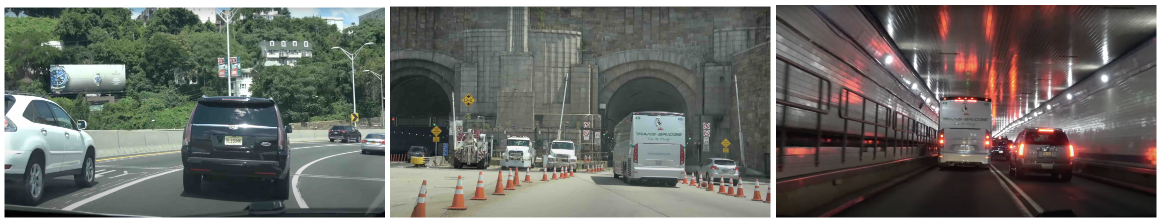
We evaluate four designs:

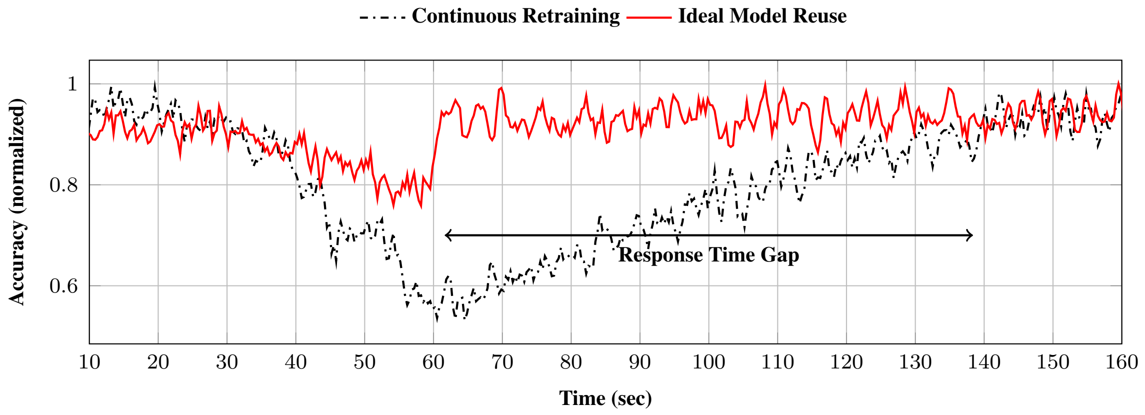
1. **No Adaptation:** trains a single expert model based on all training data and deploys this expert on the test data.
2. **Continuous Retraining:** periodically retrains an expert model for each camera using the most recent video segments. This serves as a reference point of recent model-retraining systems, such as AMS [2] and Ekya [180].
3. **Ideal Model Reuse:** deploys the *best* expert model from a given model zoo created based on video segments in the first 30 hours (ignoring the model-selection overhead). This can be seen as a strictly better version of ODIN [182], recent model reusing baseline.
4. **Ideal Reuse with Retraining:** combines 2 & 3 (retraining the reused model selected by 3.) This shows how much an ideal model reusing scheme can improve in a continuous retraining framework.

All designs are given the same amount of GPU resources to continuously retrain expert models, while No Adaptation (Design 1) and Ideal Model Reuse (Design 3) do not use this resource for retraining.

Benefits in resource efficiency: Figure 6-2 shows the mean Average Precision (mAP) score on the test data while varying the number of cameras. The observations are two-fold.

First, model reuse is a promising direction in minimizing retraining. The benefits of model reuse become more evident when the compute resource is not enough to retrain the expert models for more cameras (4, 6, and 8 cameras). Even when the compute resource *is* enough for model retraining (2 cameras), Ideal Model Reuse can still achieve a similar mAP as Continuous Retraining. This observation is encouraging because reusing history models


(a) t=10s (b) t=30s (c) t=40s



(d) Accuracy

Figure 6-3: Example of a scene change when a car camera enters a tunnel and how fast Ideal Model Reuse and Continuous Retraining respond: Model updates arrive every 30 seconds. At t=60 sec, both schemes can access sample frames from the tunnel scene. Ideal Model Reuse switches to a good model for the new scene immediately at $t = 60$ sec, whereas Continuous Retraining takes about 80 sec to retrain the model till the accuracy bounces back.

does not require the resources (not shown here) to retrain any new expert models, and at the same time, the best expert model in the past already achieves comparable accuracy with the expert models trained on the most recent video data.

Second, model reuse has a promising synergy with continuous retraining—Ideal Reuse with Retraining achieves the highest mAP across the board. This is because the reused model provides a strong starting point for retraining, which reduces the compute resource needed by retraining (i.e., faster convergence) *and* improves the inference accuracy of the resultant expert models.

Benefits in accuracy consistency: Another key benefit of model reuse is that we do not need to wait for an expert model to finish retraining. This is particularly important when a camera has experienced a sudden scene change and is in urgent need of a new model. For example, when a car drives into a tunnel, we can select and change the expert model quickly without the latency of training a new expert (Fig. 6-3 shows a concrete example). We demonstrate this benefit with the CDF of mAP across all video segments for the 8 camera setting (Figure 6-4). As the figure shows, Ideal Model Reuse has a much better

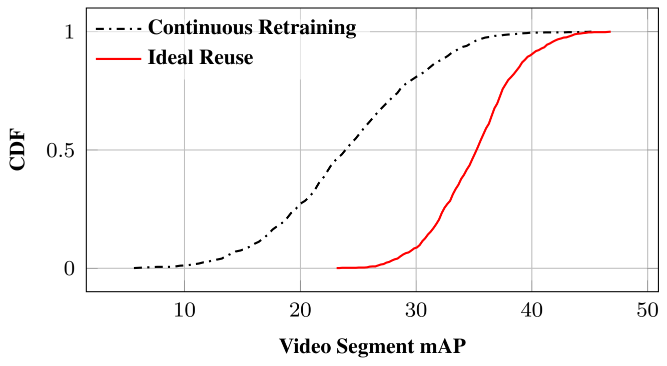


Figure 6-4: Ideal Reuse improves both average and tail accuracy (mAP) across video segments.

tail mAP than Continuous Retraining. For instance, at the 1st percentile, Ideal Model Reuse retains 24% mAP while Continuous Retraining drops to an unacceptable 7% mAP. Figure 6-3 illustrates a concrete example. As the car drives into the tunnel ($t = 40s$), Ideal Model Reuse switches to a matching expert much faster ($t = 60s$) than Continuous Retraining ($t = 120s$), which leads to a much more moderate drop in model accuracy.

Challenges of model reuse: Several technical challenges need to be addressed to fully realize the benefits of model reuse. Ideal Model Reuse assumes that it always selects the best expert model with no compute cost or delay in searching through all experts in the model zoo, which is not practical. Recent model reuse solutions in the database community (e.g., ODIN [182]) cannot address these challenges either, because they are not designed for resource efficiency, when sharing the compute resource among the functions of model selection and model retraining for many edge devices. To unleash the potential of model reuse in practice, we need a mechanism to find the best expert model quickly and accurately. We also need to rein in the cost and latency of model selection, so that it does not grow indefinitely with the number of videos or cameras.

In summary, reusing historical expert models is a promising complement to model retraining, and when used jointly, it leads to better resource efficiency and more stable and accurate model adaptation. That said, to make model reuse practical, several technical challenges remain, which we will tackle in the next section.

6.3 Design of RECL

This chapter presents RECL, a new end-to-end design of model adaptation for continuous learning on edge devices. At a high level, RECL is given an accurate-yet-expensive model (the "teacher") and a set of edge devices, and it automatically adapts the deployed

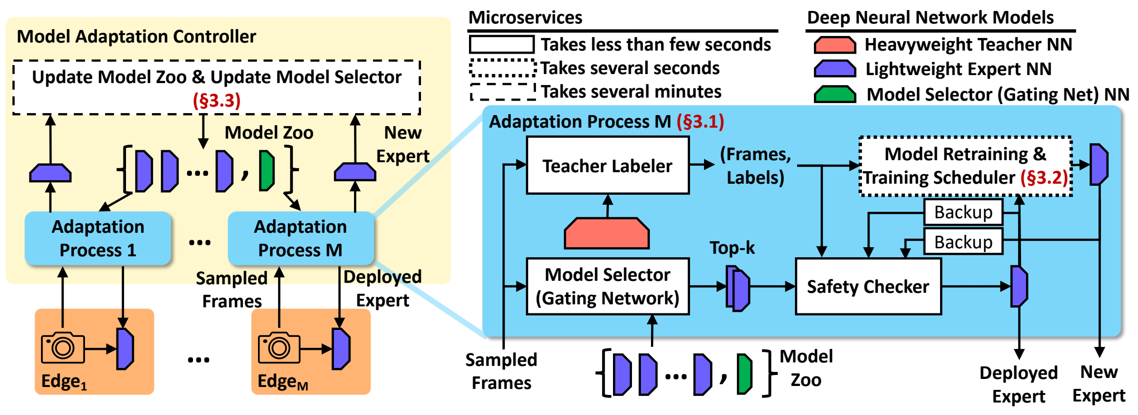


Figure 6-5: RECL system architecture. Edge devices (cameras) run real-time inference using lightweight models, and the model-adaptation controller manages a model zoo of expert models trained on history frames from edge devices (cameras) and, on receiving a model-update query, quickly selects a suitable model from the model zoo and recently trained models (the light-blue box). New models are also continuously retrained (optimized by a custom training scheduler) and then incrementally added to the model zoo over time. (The figure does not show optimizations to speedup inference on edge devices or the controller-device communication, as they are orthogonal to RECL.)

lightweight ("expert") models, each dynamically tailored to an edge device's particular distribution of video frames at any point in time, allowing each edge device to obtain results similar to running the teacher model.

Overall architecture (Fig. 6-5): RECL launches a *model-adaptation controller* on a server machine (e.g., in the cloud, edge compute cluster, etc.), which manages a set of daemons running on edge devices. The controller selects and deploys lightweight models on edge devices, which run local fast inference using the lightweight model. This work focuses on the adaptation controller, and the optimizations inside the edge devices or on the communication between the controller and the edge device are orthogonal to RECL. Furthermore, we assume the interactions between the server and edge do not interfere with any other processes running on the edge device (including the local inference).

In each model-update window (by default, every 30 seconds),[1] each edge device sends sampled frames to the controller to query if a new model should be used. (Note that the RECL controller only updates models for edge devices, which then use models to run inference on video streams.) The frame sampling rate is set dynamically based on the extent of scene change (similar to the technique used in AMS [2]). AMS takes the drift rate of the labels measured at the server as a signal for setting the frame sampling rate. As

[1]We use fixed update windows, similar to Ekya [180]. Dynamic window size is orthogonal to RECL. In general, an update window can be triggered by an edge device when it detects substantial changes in its video stream, and there are several prior efforts on scene change detection.

labels are usually in a lower dimension than input images, their variation rate is a less noisy proxy for detecting the scene change pace.

Based on the sampled frames, the controller performs two basic functions—*model selection* (§6.3.1), which selects a suitable expert model from a collection of history expert models to quickly respond to the edge device's query, and *model retraining* (§6.3.2), which fine-tunes the selected model based on the sampled frames and manages GPU resources to many edge devices to retrain their models. Furthermore, retrained models are periodically added to the model zoo shared with other edge devices (§6.3.3). The rest of the section will present their designs and rationales.

6.3.1 Model Selection

RECL's model selection module, on receiving a query from an edge device, should *quickly* select a high-quality (accuracy) expert model from a collection of models. RECL achieves this goal by: (i) maintaining a large (potentially growing) *model zoo* of history expert models that are previously trained for any edge device; and (ii) using a fast and robust selection procedure to navigate the large model zoo.

Sharing model zoo across video sessions: RECL's model zoo consists of a set of lightweight expert models, each trained for a specific scene distribution previously seen by some edge device managed by the controller. For example, if the controller manages several driving video sensors in an area, the model zoo might contain experts for different streets or neighborhoods, different weather conditions, etc. It is crucial to note that RECL does *not* directly rely on any priors about the features (e.g., weather conditions) of video content as a signal for creating new models; rather, an expert model is created based on frames of an edge device in an update time window, and then added to the model zoo if it improves performance (see §6.3.3).

An important design choice of RECL is that rather than caching the history models of different devices separately, RECL *shares* the model zoo and its gating network across devices, enabling model *reuse* across similar video sessions of different devices that might share similar temporal-spatial correlations (e.g., in the same geographical vicinity) [191]. This reduces the need for online model retraining and improves system responsiveness when an edge device experiences a sudden scene change for which a previously trained model (probably of another device) with good accuracy is available. For example, cars in the same city would observe the same scenery over time, even though the frames observed throughout one driving session may vary significantly. In such an application, the model zoo would eventually include an expert for most scene distributions encountered, significantly reducing the need for per-session model training.

Fast, robust online model selection: Figure 6-5 (right-hand side) describes RECL's online procedure to select a model from the model zoo. One strawman solution to the model selection problem is running an exhaustive search over all experts in the zoo. However, the number of models in the zoo can grow large over time, and it would become prohibitively expensive to select models by testing all of them on the sampled frames in each update window. To scale model selection to a large model zoo, RECL uses a *gating network* [185] to directly infer which models in the zoo better fit a given video content. The gating network is a lightweight DNN that given an image, assigns a score to each model in the model zoo. Logically, the gating network is similar to an image classifier, except that the labels are not object classes but models in the model zoo. A higher score indicates the model likely has higher accuracy on the image. (§6.3.3 will explain how to update the gating network to handle the changing model zoo.)

An alternative approach [182, 192] to model selection is to map video content to an embedding space (via an autoencoder), partition the embedding space, and map each partition to a specific expert model. We found that this approach works poorly in practice (§6.5.2). The intuitive reason is that auto-encoders are trained to learn the distributions of only *input* data (e.g., which video frames look similar), rather than simply learning which frames can share a good expert model. The former task is too generic, and therefore, it is significantly hard to learn an efficient embedder to deploy in practice. We refer readers to [193] for further details. In contrast, RECL's gating network directly predicts the quality (accuracy) of each expert model and avoids the need to have a good auto-encoder.

That said, it is hard to train a gating network that always picks the best model from the model zoo. Instead, RECL runs the gating network on the edge device's latest sampled frames and selects the top-K models (e.g., $K = 10$) with the highest average scores. The intuition is that the performance of the best of the top-K models improves quickly with larger K (see §6.5.3). In short, the top-K filtering approach strikes a decent balance between leveraging a large model zoo and fast model selection.

The *safety checker* then tests the accuracy of these top K models, along with the current model of the edge device and the last model retrained on video frames from the same edge device (explained in §6.3.2). The testing is based on the sampled frames and their "ground truth" labeled by the more accurate and more expensive "teacher" model. Finally, among these models (top-K from the model zoo, current model, and the last retrained model), RECL selects the one with the highest empirical accuracy on the labeled images and sends it to the device. This online model selection process is fully automatic and has a low compute cost. For the object detection task, for example, we use YOLOX-nano for the lightweight experts and ResNet18 as the gating network. These two models have a close inference cost per sample (1.1 vs. 1.8 GFLOPs). However, the gating network only runs on a significantly smaller subset of frames (e.g., 1/30th of frames).

6.3.2 Model Retraining

So far, we discussed how to reuse the previously trained models. Like other continuous learning frameworks [2, 180, 182], RECL also retrains models online for each edge device. The edge device periodically queries the controller in every model-update window. For each query, RECL will initiate a retraining job using the sampled frames sent by the device (similar to [180]), after the model selection process described above is finished.

However, to scale to more edge devices, many of which need new models, RECL must carefully allocate its GPU resource to model retraining jobs. The basic idea of RECL is to closely monitor how accuracy improves on each training job and dynamically share more GPU resources to the jobs that benefit more from additional GPU cycles.

RECL time-shares the GPU among multiple retraining jobs by micro-windows—in a micro-window, we let one of the retraining jobs use all GPU cycles and may switch to a different job at the boundary of micro-windows based on the logic described next. Each micro-window is long enough for one retraining job to complete one epoch (i.e., going through all sampled frames once). A typical micro-window size is about one second. (We will explain the reason for timesharing GPU shortly.)

Retraining scheduling algorithm: Targeting a fixed maximum accuracy gap with the teacher model for each video scene can become quickly intractable as it can be pretty challenging for the student model to track the same target performance for all real-world scenes. However, as our results show later, we can still target a fixed maximum gap on the average accuracy. Hence, having a system that uses the resources efficiently, we can always add more resources as the number of cameras grows till we are happy with the overall accuracy.

Consider C concurrent training jobs (one for each edge device). We define $I_c(\tau)$ as the improvement achieved from training the model corresponding to camera c for τ seconds. Our objective is:

$$\begin{aligned} &\max_{\tau_1,\tau_2,\ldots,\tau_C} \sum_{c=1}^{C} I_c(\tau_c) \\ &s.t. \sum_{c=1}^{C} \tau_c = T \end{aligned} \tag{6.1}$$

Algorithm 4 RECL GPU Sharing Algorithm

```
1: Input: training requests $\mathcal{R}$, micro-window number of seconds $\mu$, window size of $T$ sec
2: $budget \leftarrow T$ {Total time budget}
3: procedure ProcessRequest($r$)
4:     $acc_i \leftarrow r$.Eval()
5:     Train the model for request $r$ for $\mu$ seconds
6:     $acc_f \leftarrow r$.Eval()
7:     $budget \leftarrow budget - \mu$
8:
9:     return $(acc_f$ - $acc_i)/\mu$ {Returns the accuracy gain}
10: end procedure{Initialize the gain estimates:}
11: for r in R do
12:     $gain[r] \leftarrow$ ProcessRequest($r$)
13: end for{Schedule the most promising:}
14: while $budget > \mu$ do
15:     $r \leftarrow \arg\max gain$ {Find the request with max gain}
16:     $gain[r] \leftarrow$ ProcessRequest($r$)
17: end while
```

That is, given a time budget T (e.g., the update window duration), we want to time-share GPU resources to maximize the total improvement of accuracy across all models.

To solve this optimization problem, RECL uses the following *iterative* scheduler (Algorithm 4). At the beginning of each update window of size T, the scheduler receives a set of training requests, $\mathcal{R}$. Each training request corresponds to a set of labeled frames (already labeled by the teacher model as part of safety checking), and an expert model checkpoint (selected by the safety checker at the beginning of the window). In each micro-window of μ seconds, the ProcessRequest procedure (Lines 3-9) takes one of the requests r as input and evaluates *how much its accuracy improves between before and after a micro-window*. Notes that the cost of these accuracy evaluations is ignored as they only require a lightweight forward pass on the test subset of the data.

The main loop of the algorithm first spends one micro-window to process each request and initialize its accuracy improvement (Lines 10-12). Then it iteratively picks the training request with the largest accuracy improvement as our next model to train till we run out of time (Lines 13-16).

Since DNN training curves $I_c(\tau)$ are usually concave (i.e., accuracy improves quickly and then slows down), then this iterative algorithm effectively minimizes the maximum speed of these models' training curves ($\frac{\partial I_i}{\partial \tau_i}$). It can be shown that this iterative process converges to a near-optimal partition of the total time budget that maximizes the total accuracy improvement across the training jobs [194].

Design choices: We highlight two design choices behind the retraining scheduler.

To find the best GPU allocation, both Ekya and RECL predict each retraining job's training speed (accuracy improvement vs. epochs) but with different approaches. Ekya periodically runs extra ("out-of-band") micro-profiling on each camera: running a few epochs of training on a subset of history images to build a profile of the training curve of each camera. Such upfront micro-profiling has extra compute overhead and fails when the training curve changes over time. In other words, they inherently trade off between the profile accuracy and their overhead. In contrast, RECL uses an "in-band" profiler—it measures the *actual* learning progress (accuracy improvement) of each job on the fly and dynamically determines which one progresses faster. This scheme avoids the micro-profiling overhead of Ekya without losing accuracy. Note that RECL requires fast switching between models, which will be discussed next. Our scheduler's iterative algorithm is similar to [195] which is designed to achieve fairness among the cluster-level training jobs which compete at a significantly longer time scale.

Instead of splitting GPU cycles spatially across concurrent training jobs, RECL time-shares the GPU cycles by switching among concurrent retraining jobs every micro-window. While it is logically equivalent to spatially sharing, RECL's GPU timesharing is based on three practical considerations. (1) The delay to context switch between GPU-loaded models (usually less than tens of milliseconds) is negligible compared to a micro-window. Since a lightweight model in RECL has a small memory footprint, we can load it to GPU memory and not swap it out (even when switching between retraining jobs) until the model retraining completes. (2) Unlike Ekya, RECL does not have to finish model training very quickly (it responds to each edge device by first selecting a good model from the model zoo or the most recently trained model has been good enough). (3) It does not rely on any GPU library to dynamically reallocate GPU across different jobs.

6.3.3 Updating the Model Zoo & Selector

Admission of new models to model zoo: RECL does not add every retrained model to the model zoo. A recently trained model is considered *promising* if the safety checker finds this retrained model's accuracy is α higher than the rest of the candidate models (the top-K experts selected by gating network and edge device's current model). These promising models are put in a queue. When the promising model's queue grows larger than a fixed threshold, γ, we empty the queue by adding them to the model zoo and update the gating network (explained next) to consider the recently added models. Hence, α and γ control the frequency of model selector updates. We later study the impact of the zoo admission rate on the system performance (§6.5.3).

Incrementally update of gating network: Recall that the gating network predicts the accuracy of each expert in the model zoo on the input frame. Hence, when updating the gating network to handle new expert models, we need to first label the accuracy of all experts on both the new frames that were used to train the new expert models as well as a sub-sampled set of history frames (those used to update gating network before). To this end, we label the new samples with existing experts in the zoo and label the existing samples with the new experts added to the zoo. This way, we track the performance of all experts on a sampled set of frames so far. Note that when we create the training frame set of the gating network, we sub-sample frames used before and mix them with the new frames in order to keep the same training size over time.

As the zoo size increases, the output size of our gating network must change as well. Since the accuracy prediction logic does not change for most of the models in the zoo, we only need to add corresponding neurons for the new models to the final layer without changing the connectivity weights for existing expert models. This way, we transfer as much knowledge as possible from one gating network to the next. To further speed up the training of the gating network, we use the mean and variance of the most recent model selector in order to initialize the connectivity weights corresponding to the new experts in the final layer.

Pruning the model zoo: Though updating the gating network is usually fast, the overhead of retraining for updating the gating network grows proportionally with the size of the model zoo. To prevent the model zoo to grow indefinitely, we deem an expert in the zoo ineffective if other experts always have a preferred accuracy. In particular, we remove the experts that are chosen less than η times in the last q model selection calls. We set $q = 3000$, which is about one day's worth of video streaming in our system and study the impact of the η parameter later in §6.5.2.

6.4 Implementation

We have implemented RECL in Python and used Pytorch [196] for inference and training of ML tasks. For communication between the services, we use the gRPC [197] framework for remote procedure calls.

Microservices: We implement several microservices to prototype RECL. These microservices are designed to generalize to different continuous adaptation design choices in prior work. RECL runs a camera streaming service on each camera device to send the sub-sampled video frames to the teacher labeler service running on the adaptation server. We use TensorRT [198] and half-precision computation to further speed up the inference processes. One runner microservice manages the coordination of different components across all video streams that share resources in the server.

Hooks: Each microservice can register a hook in other microservices. These hooks are specific functions to run at predefined time events in the system. Our time events are a combination of before/after, window/microwindow, and first/last time. For example, if a model zoo update strategy requires to have information about the training gain of each model at the microwindow level, it registers accuracy evaluation hooks in the training scheduler before and after each window.

Adaptation state: There is an adaptation state shared across all microservices that register to the same runner. All microservices have read and write access to the adaptation state to optimize their decisions, possibly share the hooks results, and keep track of possible global events like the beginning of a new window.

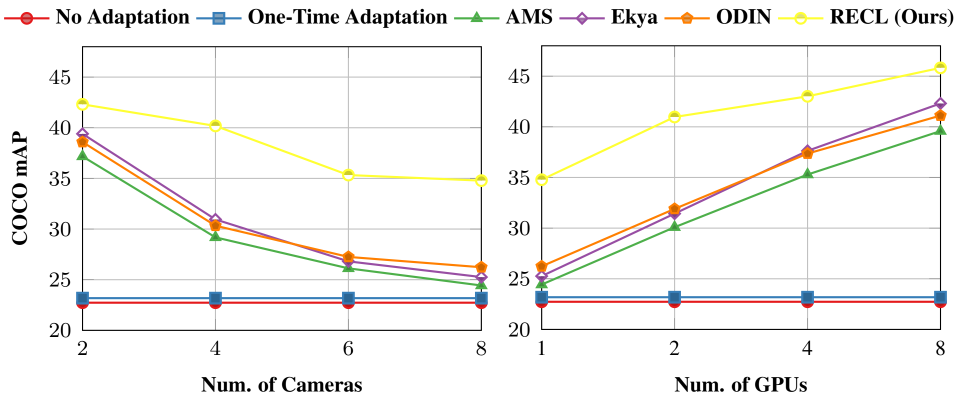
Training strategy: Our training scheduler service relies on a sharing strategy abstraction. Each strategy has access to the adaptation state, can register or subscribe to a hook, and has a run method to decide which camera model should train next in each microwindow. If an adaptation scheme is not microwindow-based, it only has to register hooks for the first and last microwindow.

Performance monitoring: For tracking the system performance metrics, we implement logging hooks to track system-level metrics like compute times and resource utilization in addition to RECL-specific performance metrics like zoo admission rate and model reuse rates.

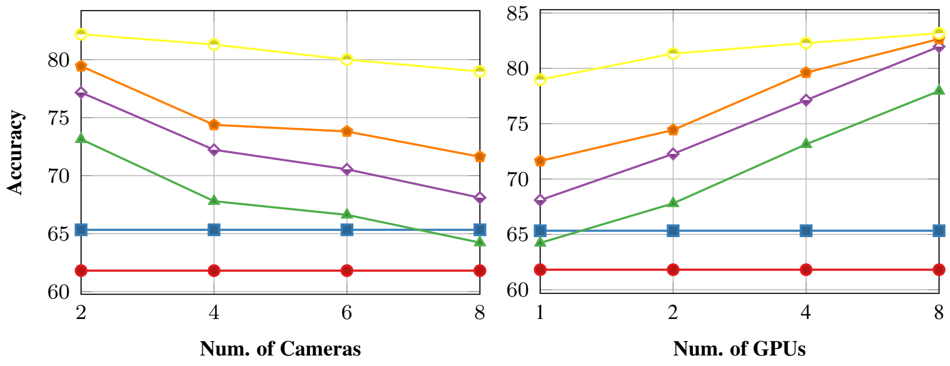
6.5 Evaluations

Finally, we evaluate RECL on two video-analytics tasks using real-world driving videos. Our key findings include:

- Given the same compute resource, RECL improves the object detection mAP and image classification accuracy over state-of-the-art baselines by up to 9.0% and 7.4%, respectively. The baselines need to use at least $3.2\times$ more compute resources to match RECL's accuracy.
- The superior performance of RECL comes primarily from our distinctive design of model reuse. RECL's fast gating network and safety checker outperform the state-of-the-art model selection mechanism in terms of accuracy and efficiency by a large margin.
- RECL is highly responsive to a model-update request. On average, the time RECL needs



(a) Object Detection



(b) Image Classification

Figure 6-6: End-to-end scaling of the average accuracy across different schemes for two typical vision tasks.

to adapt models to the same accuracy is 11–91 seconds faster than that of the baselines, with the gap growing both at the tail of the distribution (by almost $2\times$) and with the total number of cameras.

- RECL's retaining scheduler also makes better use of GPU. In contrast to round-robin and out-of-band profiling used in several recent continuous learning systems, RECL's in-band profiling provides a 2.0% higher mAP at up to $6.1\times$ lower overhead.
- Compute overhead of RECL decreases gracefully over time as more models are trained and added to the zoo.

6.5.1 Methodology & Setup

Dataset: We evaluate RECL on two computer-vision tasks—image classification and object detection—using 151 driving videos collected from YouTube. Since we would like our video sessions to include meaningful data drifts, we adopt videos that have a length of at

Model	Params	FLOPs	Throughput (FPS)
MobileNetV2	3.5M	0.32G	1.5K
ResNet50	25.6M	4.12G	153
YOLOX-Nano	0.91M	1.1G	312
YOLO-X	99.1M	282G	58
ShuffleNetV2	2.28M	0.15G	3.7K
ResNet18	11.7M	1.8G	490

Table 6.1: Specifications of the models used for the evaluation. Throughputs are reported for NVIDIA V100 GPU with a batch size of 1.

least a few minutes (up to a couple of hours)[2] with a total length of 71 hours. Furthermore, our dataset covers a wide range of cities and driving situations in North America, including weather conditions, time of day, and driving speed. Note that in each experiment, we do *not* play the exact same video segment twice on *any* edge devices, since it might artificially amplify the gain from model reusing. Driving video is a remarkably challenging workload for evaluating our system as the scenes change more widely and frequently. This workload brings a variety of situations where exact matching is impossible and requires more than a few models to cover the wide range of possible scenarios. Responsiveness is also more challenging for driving cameras compared to fixed cameras. For example, traffic light cameras mostly need only to update every few hours when the lighting/weather change, significantly stressing the compute power at the adaptation server.

Models: For object detection, we use YOLOX-Nano and YOLOX-X [173] for the student and teacher models, respectively. For image classification, we use MobileNetV2 [200] and ResNet50 [201] for the student and teacher models. Details of these models are shown in table 6.1. Our models are pre-trained on ImageNet [202] and COCO [203] datasets for classification and detection, respectively. For fast model selection, we use ResNet18 as the gating network architecture by default, unless otherwise stated.

Metrics: To evaluate the accuracy of different schemes, we compare the inference results on the edge device with labels extracted for the same video frames using the teacher model (similar to prior work [2, 180]). We use *mean Average Precision (mAP)* for the detection task, while for classification, we report *accuracy* by the proportion of correct predictions (both true positives and true negatives) among the total number of cases examined. We calculate these metrics across *all* 80 and 1000 classes of MS COCO and ImageNet datasets for detection and classification, respectively.

Setup: In our setting, model selection and training of the adaptation controller happen in the cloud, and each edge device only runs inference by the lightweight expert model on

[2]Video sessions in other similar video datasets like Berkeley Driving Dataset (BDD) [199] were not long enough for our purpose. For example, each driving episode in BDD is only 40 seconds.

the local video stream. All experts can run at a real-time inference speed (30 frames-per-second) even on lower compute power edge devices such as NVIDIA Jetson Nano [24] and Coral Edge TPU [23], and we do not evaluate any optimization on the edge device, as it is orthogonal to RECL. We use NVIDIA V100 GPUs for the adaptation server. The adaptation processes of different edge devices share the same pool of GPU resources.

Baselines. We compare RECL against the following continuous learning methods:

- **No Adaptation:** We run the pre-trained model on the edge device without any adaptation.
- **One-Time Adaptation:** We fine-tune the entire model on the first half of the videos and test on the rest. This adaptation happens only once. Comparing RECL with this scheme will show the benefit of having a continuous adaptation system in place.
- **AMS:** We implement Adaptive Model Streaming (AMS) as in [2], which uses a remote server to continually adapt lightweight expert models running on edge devices. As the update intervals are longer and our lightweight models are smaller than AMS, network bandwidth consumption is less of a concern in our setup. As such, we relax the bandwidth constraint of AMS and allow for full model parameter updates in this scheme. AMS uses a simple round-robin mechanism for GPU sharing. Comparison with AMS mainly highlights the gains of model reuse and optimized GPU sharing. As AMS reasonably outperforms Just-In-Time [179] and remote server inference in prior work [2], we no longer compare with these schemes.
- **Ekya:** Ekya enables both retraining and inference to co-exist on the edge node without any model reuse. Since RECL shares the server GPU resource only among model retraining and selection jobs (inference is on edge devices), for a fair comparison, we compare RECL with applying Ekya's microprofiler and thief scheduler (released in Ekya [180]) to model retraining jobs. Despite the more sophisticated resource-sharing mechanisms compared to AMS, Ekya, however, incurs the out-of-band profiling overhead and cannot reuse models compared to RECL. Moreover, since Ekya shows how continuous retraining significantly outperforms naive model reuse methods (e.g., reuse models from the same time of the day) [180, §6.4], we do not compare RECL with these naive reuse heuristics.
- **ODIN:** ODIN [182] is a video analytics system that can detect and recover from data drift by building expert models based on the similarity of video scenes. We use the autoencoder-based method proposed in ODIN for model selection. Specifically, the average of embedding vectors of the sampled frames in a window is used as the embedding vector of that window. Also, each trained expert is assigned an embedding vector the same as its training data. We use the L2 distance between the embedding vector of a window and the models in the zoo as a measure of similarity, and the model selector re-

turns the model with the least distance from the samples in each window as in the ODIN paper [182].

6.5.2 Results

End-to-end performance: We first compare the end-to-end accuracy of RECL with the baselines over a range of provisioned GPUs and a varying number of concurrent cameras replaying videos from our dataset. Whenever a video ends, we continue with another video from our dataset. Note that we never repeat the same video twice as it favorably impacts the accuracy gain of model reuse. We use NVIDIA V100 GPU as the adaptation server. In measuring the impact of the number of cameras on the accuracy, we fix the number of GPUs to 1. For the varying number of GPUs experiment, we run a workload consisting of 8 cameras. As shown in Figure 6-6:

1. Continuous adaptation significantly improves mAP and accuracy. Gains from continuous learning grow with more resources provisioned per camera.
2. Overall, RECL outperforms all baselines by a large margin. In object detection, for instance, RECL improves mAP by up to 9.0% (8 cameras, 1 GPU) compared to the second best approach. In terms of resource consumption, RECL supports $2.6\times$ more cameras on one GPU, and requires $3.2\times$ fewer GPU cycles to maintain an mAP of 35%.
3. mAP/accuracy improvements from model reuse are significant. Compared with Ekya and AMS which do not reuse historical models, RECL brings up to 9.8% and 10.7% improvement in mAP and accuracy for object detection and image classification, respectively.
4. In image classification, ODIN performs the best among the baselines due to its model reuse and specialization design. Nonetheless, ODIN's auto-encoder-based model selector (and the lack of optimized resource sharing) performs poorly on relatively complicated tasks like object detection. In contrast, we see better performance of RECL across all settings in both tasks due to our unique model selector and retraining scheduler design.

Model selection performance: To directly examine the model selector performance, in Figure 6-7a, we plot the accuracy of the selected models vs. the system's wall-clock for 8 GPUs, i.e., within each hour on the wall-clock, the system ingests 8 hours of video. In the figure, we also include the performance of ODIN (a recent model selector) over the same zoo created by RECL, as well as the accuracy of an oracle model that exhaustively searches over all models in the zoo at each point in time (while ignoring the oracle's compute overhead). It is not surprising that the accuracy of the model selected by RECL improves over

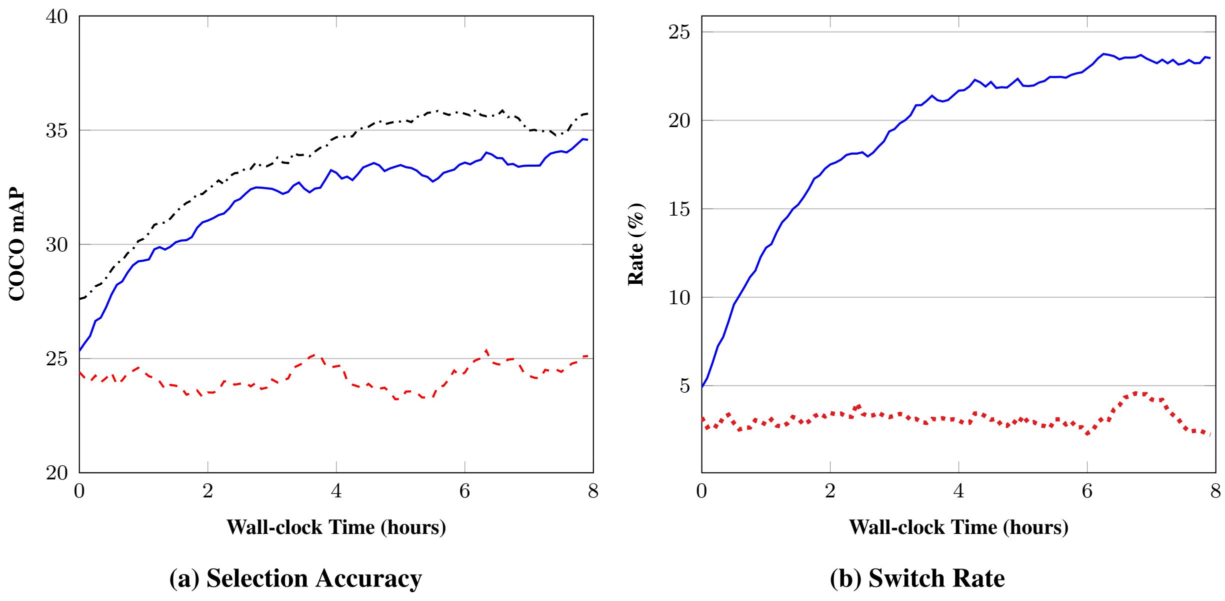


Figure 6-7: An example of RECL model selection performance over time. As the model zoo grows, (a) accuracy of the RECL-selected models gradually improves, and (b) the model selected by RECL's gating network has higher accuracy than models selected by ODIN (as evidenced by the fact that the safety checker would more frequently pick the model by the gating network than it would pick the model selected by ODIN).

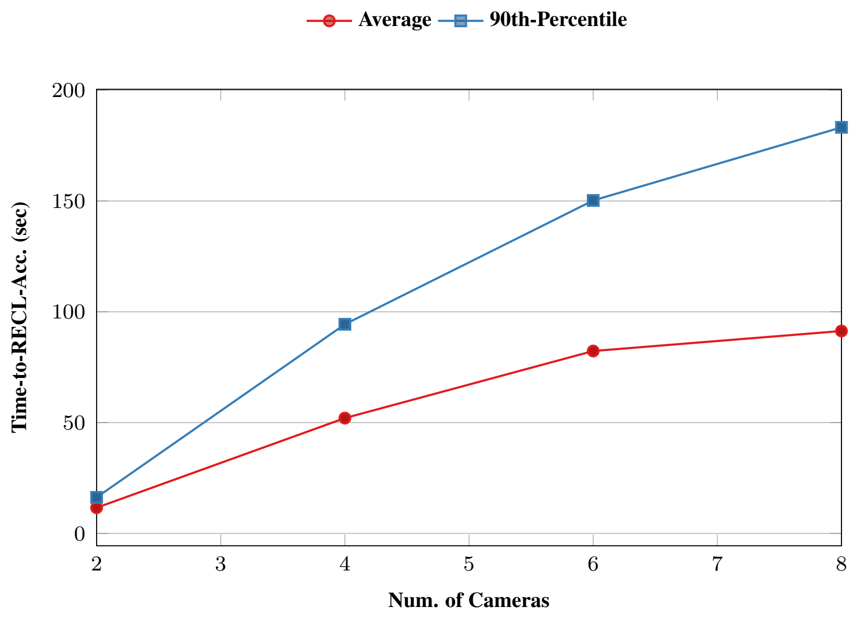


Figure 6-8: Model reuse impact on improving the response time.

time as more models are being added to the zoo. Furthermore, we observe that RECL performs closely to the oracle selector, while ODIN struggles to select a good model from the same zoo. Notice that the cost of running the oracle model is prohibitively expensive as,

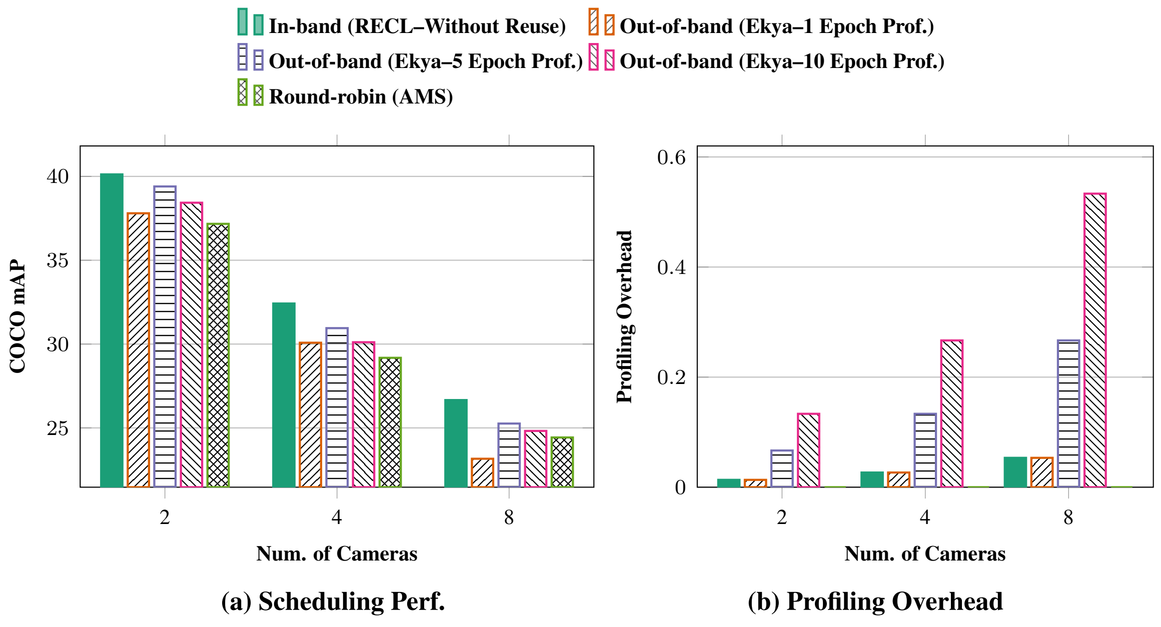


Figure 6-9: Impact of profiling on retraining performance: RECL's retraining scheduler (which uses a low-overhead in-band profiling) outperforms Ekya (which relies on out-of-band profiling on each job) and AMS (whic uses a round-robin scheduler).

after a couple of hours, it requires testing the accuracy of thousands of experts in the zoo for each sample frame. On the contrary, RECL uses ResNet18 as the underlying gating architecture that runs at 490 frames per second (see table 6.1).

We further notice that, in Figure 6-7a, the model zoo roughly converges to a desirable accuracy after four hours, totaling 32 hours of video stream ingestion. This observation shows an opportunity to reduce model zoo update frequency (and thus its cost) after enough representative experts are collected in the system. With the growing model zoo, model reuse becomes more favorable over time as well. Figure 6-7b depicts the percentage of the time that the safety checker prefers the selected model over the rest (e.g., a recently trained model). For a fair comparison of the effectiveness of model reuse, we run RECL and ODIN end-to-end independently (i.e., they are not sharing the same model zoo). As can be seen in Figure 6-7b, RECL's model hit ratio increases with a larger zoo, making our system both more accurate and efficient than ODIN.

Impact of model reuse on responsiveness: Model reuse improves the response time of the adaptation server by not needing to retrain a new expert model first. To directly evaluate this effect, we first profile the accuracy of 102 models generated in Ekya (in a video of 51 minutes long) against 2 minutes of offline training on a single V100 GPU. Using these profiles, we then measure the time it would take Ekya, as a continuous retraining approach, to adapt each model to the same accuracy level of the RECL's selected model for reuse on

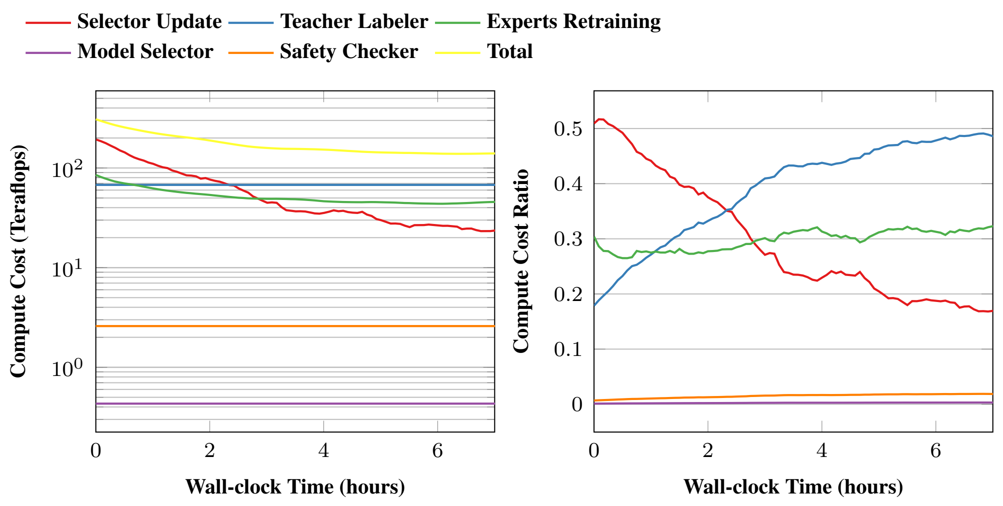


Figure 6-10: Breakdown of compute cost by the components of RECL controller. The total cost drops over time as the extra-cost of maintaining the model zoo significantly reduces, allowing RECL to enjoy the benefit of model reuse without much additional overhead.

the same window. We refer to this metric as Time-to-RECL-Accuracy. Figure 6-8 shows the mean and 90th percentile of the Time-to-RECL-Accuracy for the object detection task across a varying number of cameras sharing one GPU. We observe that Ekya takes up to 90 seconds longer than RECL, on average, to achieve the same level of accuracy. More importantly, this gap grows significantly large with increasing the number of cameras and at the tail scenarios.

Scheduler performance: We now evaluate RECL's retraining scheduler with its "in-band" profiling (§6.3.2) and compare its performance with Ekya's out-of-band micro-profiler and AMS's round-robin scheduling method. For a fair comparison between Ekya and RECL, we let Ekya adapt its early stop parameter, which has a similar effect to the micro-window-based scheduling in RECL. To run Ekya's profiler, we set its early stop parameter to 1, 5, and 10 epochs. Figure 6-9 compares the accuracy and profiling overhead (ratio of the time spent on profiling in each window) of these schedulers vs. the number of cameras. We observe that Ekya's out-of-band profilings are either too costly to run (e.g., Ekya with an early stop of 10 epochs), or too noisy to identify a good early stop parameter, which results in low accuracy. For example, an early stop at 1 epoch has the same cost as RECL's in-band method but performs worse than round-robin when resource allocation becomes more challenging with 8 cameras.

Breakdown of compute cost: Figure 6-10 shows the cost of different components in RECL over the course of 7 hours. Initially, model selector update has the dominant cost in the system. However, as the zoo grows over time, the need for updating the model zoo (and consequently the selector) reduces to the extent that after a while, the teacher labeler

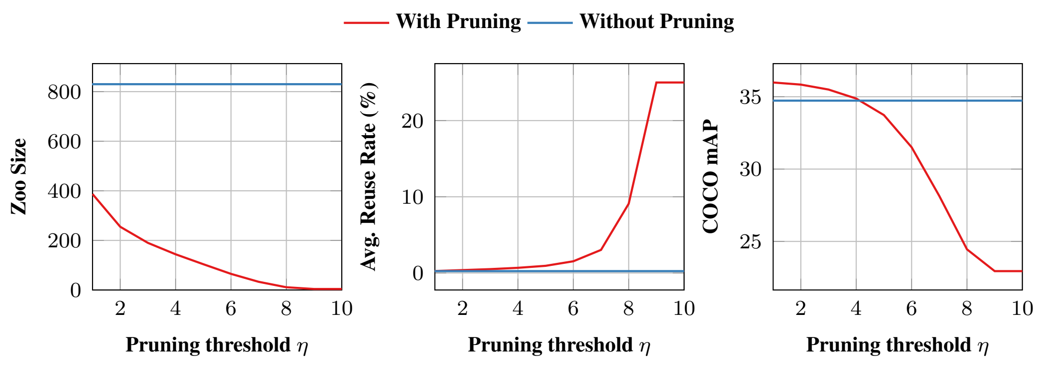


Figure 6-11: Pruning policy impact on RECL accuracy.

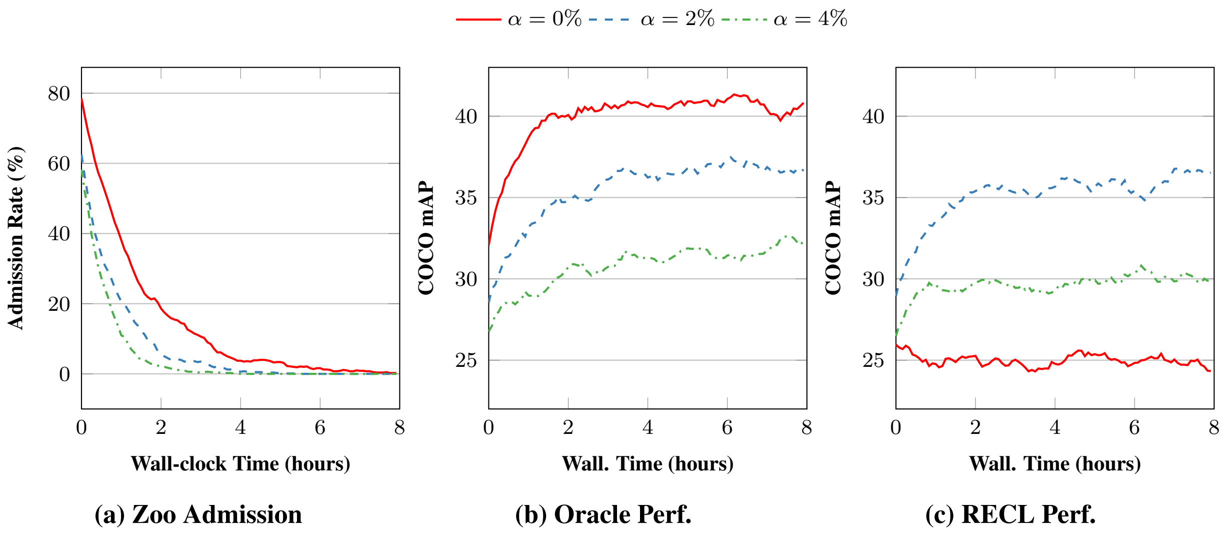
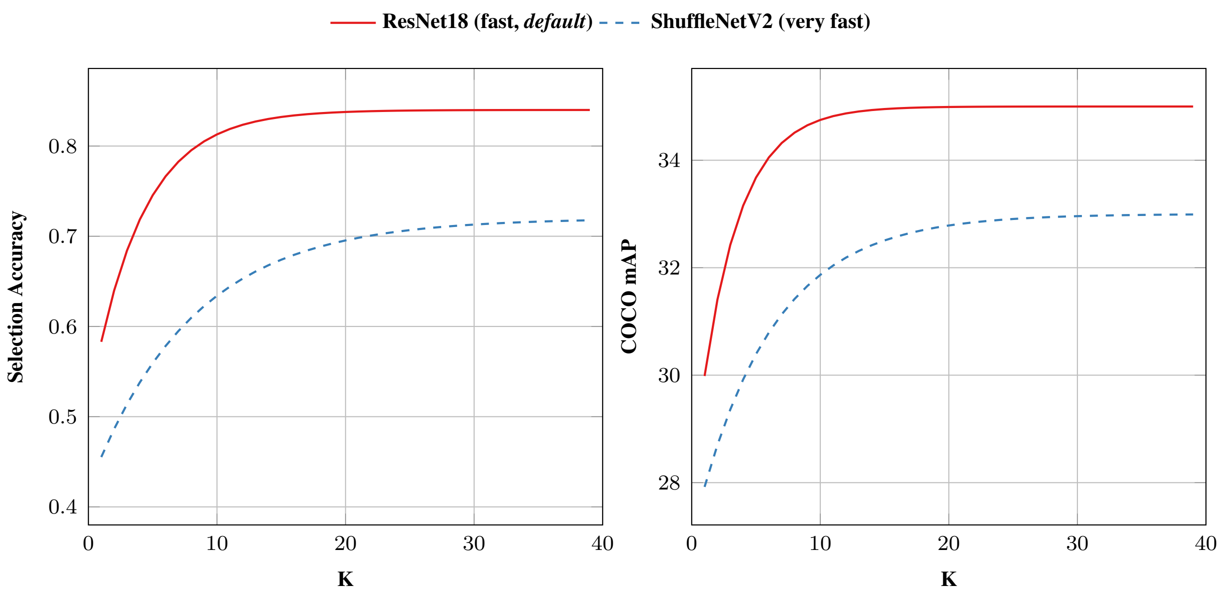


Figure 6-12: Impact of changing the admission rate through the α-promise threshold on RECL model selection performance.

and training scheduler become the dominant cost of the system, but these "base cost" is the same as a typical continuous retraining system (such as Ekya and AMS). In short, the extra overhead for RECL to enable model reusing (model selector and maintaining a growing model zoo) significantly reduces over time.

6.5.3 Ablation Studies

Model zoo pruning: In Figure 6-11, we compare RECL accuracy across various levels of pruning intensity over the course of nearly 60 hours. Naturally, reducing the value of η leads to a significant drop in model zoo size without much accuracy sacrifice. For instance, a balanced choice of the pruning threshold, $\eta = 4$ provides the same accuracy despite efficiently shrinking the size of the model zoo by a factor of 5.6×, from 830 down to about 150 experts.



(a) Gating Network Accuracy **(b) Selected Model Performance**

Figure 6-13: Impact of using top-k models suggested by the gating network for the default gating network and a faster gating network model.

Zoo admission rate impact (α-promise margin): In order to evaluate the impact of the admission rate, we turn off the zoo pruning mechanism and measure the selected model accuracy. Figure 6-12 shows this accuracy for three levels of α for both the ideal oracle selector and RECL's selector. As decreasing α allows for admitting more models to the zoo, the oracle-based scheme can choose among more models. However, it gets harder for the gating network model to select from an arbitrarily large model zoo. Hence, we observe a diminishing return in increasing the admission rate beyond $\alpha = 2\%$, which seems to be a good balance between the zoo size and the model selection complexity.

It should be noticed that the exact values of these parameters (η, α) largely depend on the dynamics of video content. The message from Fig. 6-12 and 6-11 is that there are sweet spots for them that, on a large set of videos, strike a desirable tradeoff between the cost of maintaining a reasonably sized model zoo and the quality (accuracy) of the selected models.

Model selector top-k: As discussed in §6.3.1, we pass the top-k selected model to the safety checker (instead of 1) in order to find a better model for reuse. In Figure 6-13, we show the accuracy of model selection and the performance of the selected model for our default and a faster gating network model (see table 6.1 for speed comparison). Given this observation, we find $K = 10$ is a good default operating point for RECL. Notice that while a higher K increases the cost of the safety checker, as shown in Figure 6-10, our safety checker still has a negligible overhead compared to the other components in the system.

6.6 Discussion

Safety-critical applications: Predicting the feasibility of minimum accuracy thresholds is not a trivial problem in non-convex ML training tasks. Therefore, as we cannot guarantee a minimum accuracy level using continuous adaptation for safety critical problems, the solution might come at the cost of provisioning enough resources to run the large state-of-the-art model for inference. However, if the problem is not safety-critical, one solution might be to set minimum accuracy thresholds with timeouts to achieve them, which we leave to future work.

Data residency: RECL requires sharing training samples with the adaptation server. While there are recent solutions in computation over encrypted data for secure AI [204], our current evaluation has been based on having access to the actual video frames. Depending on the data residency policies, such data sharing may constrain how far the adaptation server can be taken from the cameras.

6.7 Related work

Optimization of video-analytics systems: To maintain high inference accuracy with low resource usage and fast response, video-analytics systems have explored many approaches, including model distillation [2, 176, 180], model architecture pruning [205, 206], configuration adaptation [190, 207], frame selection [208, 209], and DNN feature reusing [210, 211]. The closest to RECL is model distillation—creating lightweight models (i.e., experts in RECL) that are small and fast yet accurate on a specific video scene [176, 212]. The challenge is that as the video scene evolves, the system must create new expert models on the fly to fit new video content. Existing solutions rely on either of two approaches—model retraining techniques train the lightweight models on the latest video frames [2, 179, 180] or on the most relevant images from the training set [189], and model selection techniques maintain, and then select a model from, a collection of history models [182] or a cascade of models with increasing capacities [189, 213].

In contrast, RECL uses both techniques—model retraining and model selection—as building blocks in an end-to-end framework. In particular, when an edge device queries for a model update, RECL can respond faster than Ekya [180] and AMS [2] by selecting a model from a large collection of history models used by all edge devices which might have seen a similar scene and object distribution. RECL also shares GPU cycles to enable more concurrent model retraining jobs, refreshing new models for more edge devices.

Model selection under data drifts: In the ML literature, model selection in a collection of expert models, or Mixture-of-Experts (MoE), has attracted much attention, especially after Shazeer et al. [185] demonstrated that using a sparsely gating network with an MoE of many expert models can drastically reduce the compute cost of DNNs. Recent work has obtained accuracy comparable to state-of-the-art expensive models with a fraction of compute cost [214]. One key distinction between RECL and MoE applications is that in MoE, all or a subset of the experts work together on each input. However, in RECL, there only works one expert on each input. For example, the recent MoE approach [214] operating on *tokenized* images requires access to 768 experts for inference on each input image. To implement such an approach, one must either load all experts in the accelerator's memory or quickly swap the experts on the accelerator per patch per image, introducing significant challenges for even more resourceful settings such as entirely cloud-based applications [215]. That said, many techniques in MoE also assume that the MoE consists of a static set of models. To handle MoEs that gradually incorporate new models (as in RECL), the gating network or the model selector must be retrained over time [216, 217]. To avoid retraining model selectors or saving training data, recent works leverage an autoencoder that projects input data to a latent space and map new models to a region in the latent space [182, 192].

RECL's model selection strategy (§6.3.1) builds on the literature on gating networks [185], but reduces the delay and compute overhead when adding new expert models. Instead of jointly training the new experts and the gating network [185], RECL freezes the new expert models already trained to fit the edge devices' recent videos and only reshapes and fine-tunes the last layer of the gating network. Compared to recent autoencoder-based model selectors [182], RECL's gating network enjoys better algorithmic intuition (see §6.3.1) and better empirical performance (§6.5.2).

Resource allocation for DNNs: Resource sharing for DNN-related jobs has been extensively studied in the systems literature, including sharing of GPU and network resources among multiple concurrent DNN training jobs (e.g., [195, 218]), inference tasks of video analytics (e.g., [207, 219]), and between inference and training jobs [180]. The common challenge facing these settings is to predict how much each job's accuracy can improve with the same amount of compute/network resources. This is usually profiled offline [207], periodically [180], or by reusing compute data [195].

RECL is a custom design of GPU sharing for continuous learning across many edge devices. Compared to Ekya [180], the most recent related work on edge continuous learning, RECL avoids profiling the training curve of each model; instead, it tracks the actual training progress of each retraining job on the fly, similar to SLAQ's quality-driven scheduler [195] proposed for large-scale DL clusters.

6.8 Conclusion

Resource efficiency is one of the most important problems in modern video analytics applications, and continuous retraining and deploying expert models is a promising direction. We show that reusing historical expert models has a large potential to improve resource efficiency and response time for continuous retraining, but this approach comes with its own challenges. We present RECL, the first end-to-end system that integrates model reusing with model retraining for resource-efficient video analytics. We show that RECL achieves significantly better resource efficiency and higher accuracy simultaneously than state-of-the-art baselines with *(i)* a fast and robust model selection procedure, *(ii)* a model zoo that shares across multiple edge devices, and *(iii)* an iterative training scheduler. We hope that our findings and designs can stimulate further research in unleashing the full potential of the synergy between model reusing and model retraining.

Chapter 7

Conclusion

This thesis addresses the challenges of machine learning inference on resource-constrained edge devices. The surge in edge computing and IoT devices, coupled with the massive growth in machine learning models' size and computational demands, has necessitated new techniques for efficient inference at the edge. This thesis introduces two key methods—Model Streaming and Model Reuse—that enable continuous learning on edge devices, improving the accuracy of lightweight models by adapting them to specific scenes or narrow input distributions. Through a range of practical applications, including MMNet for signal detection in Massive MIMO environments, AMS for real-time video inference, SRVC for efficient video compression, and RECL for responsive video analytics, we demonstrate the effectiveness and potential of these methods.

By addressing the computational and memory overhead challenges associated with continuous learning on edge devices, our proposed techniques open the door to more accurate and responsive applications on the edge. Model Streaming and Model Reuse provide innovative solutions for offloading adaptation processes and reusing models, resulting in improved operational efficiency and scalability for continuous learning systems. The applications showcased in this thesis highlight the potential of continuous learning to enhance inference performance at the edge, in scenarios ranging from wireless signal detection to video analytics.

The insights from this thesis offer a new paradigm for designing and deploying edge applications. Continuous learning and the optimization techniques presented herein can enable more efficient utilization of computational resources, improved latency, and enhanced energy efficiency for edge devices. Furthermore, by offering accurate and adaptable inference capabilities closer to the data source, continuous learning methods have the potential to shape the development of new edge computing applications, paving the way for more intelligent, efficient, and adaptive systems at the edge of the network.

7.1 Future Work

While the continuous learning methods and optimization techniques introduced in this thesis have demonstrated considerable potential for improving lightweight machine learning inference at the edge, there are several avenues for future research to further advance the field and address new challenges:

Improved Adaptation Techniques: Improved adaptation methods might enable faster and more precise continuous learning at the edge. Currently, most adaptation techniques depend on gradient descent algorithms for optimizing the model parameters. However, this approach is computationally demanding and time-consuming. RECL chapter 6 takes a promising step in this direction. Nevertheless, many redundancies exist in data processing as we do not fully understand the functionalities of different model components. Thus, discovering faster optimization methods or gaining deeper insights into our models could both lead to enhanced adaptation techniques.

Privacy-Preserving Continuous Learning: When edge devices offload the adaptation process to remote servers, there may be privacy concerns. Future work could focus on methods that ensure continuous learning while preserving data privacy. Depending on the primary goals and concerns, promising ideas worth exploring further include moving the adaptation servers closer to the edge (e.g., within the building), running neural networks on encrypted data, offering partial privacy, secure multi-party computation, and filtering sensitive parts of the data before sharing with remote servers.

Heterogeneous Edge Devices: Future work could investigate continuous learning methods optimized for heterogeneous edge devices with varying computational capabilities. Less powerful edge devices may need to run smaller models, which require more frequent updates to maintain the same accuracy. Moreover, not all devices offer consistent connection quality at all times. New notions of service quality and fairness might be needed to be developed for continuous learning systems to be effectively implemented in such heterogeneous real-world edge scenarios.

Other Modalities: This thesis mainly focused on vision and video applications. Future research can extend continuous learning techniques to other modalities, such as text and audio, expanding the potential applications of continuous learning at the edge.

In conclusion, continuous learning offers a promising approach for improving the accuracy of lightweight machine learning models at the edge. The methods and optimization techniques introduced in this thesis have laid the foundation for further research and development in this field. By exploring the avenues outlined above, we may be able to push the boundaries of machine learning inference at the edge to new levels, enabling more efficient, accurate, and adaptable applications across various domains.

Appendix A

Video Datasets Information

A.1 Outdoor Scenes Dataset

The Outdoor Scenes video dataset we release includes seven publicly available videos from Youtube, with 7–15 minutes in duration. These videos span different levels of scene variability and were captured with four types of cameras: Stationary, Phone, Headcam, and DashCam. For each video, we manually select 5–7 classes that are detected frequently by our best semantic segmentation model (DeeplabV3 with Xception65 backbone trained on Cityscapes data) at full resolution. Table A.1 shows summary information about these videos. In Figure A-1 we show six sample frames for each video. For viewing the raw and labeled videos, please refer to https://github.com/modelstreaming/ams.

A.2 Prior Work Videos

In our experiments, we also evaluate AMS on three long video datasets from prior work: Cityscapes [115] driving sequence in Frankfurt (1 video, 46 mins long)[1], LVS [104] (28 videos, 8 hours in total), A2D2 [116] (3 videos, 36 mins in total). Table A.1 shows the summary information of the classes present in each video in these datasets. Overall, LVS includes fewer classes per video, and A2D2 and Cityscapes only include driving scenes. Hence, we introduced the Outdoor Scenes dataset that includes more diverse scenes and more classes.

[1]This video sequence is not labeled and was the only long video sequence available from Cityscapes (upon request).

Dataset	Description	Classes
Outdoor Scenes	Interview	Building, Vegetation, Terrain, Sky, Person, Car
	Dance Recording	Sidewalk, Building, Vegetation, Sky, Person
	Street Comedian	Road, Sidewalk, Building, Vegetation, Sky, Person
	Walking in Paris	Road, Building, Vegetation, Sky, Person, Car
	Walking in NY	Road, Building, Vegetation, Sky, Person, Car
	Driving in LA	Road, Sidewalk, Building, Vegetation, Sky, Person, Car
	Running	Road, Vegetation, Terrain, Sky, Person
A2D2 [116]	Driving in Gaimersheim	Road, Sidewalk, Building, Sky, Person, Car
	Driving in Munich	Road, Sidewalk, Building, Sky, Person, Car
	Driving in Ingolstadt	Road, Sidewalk, Building, Sky, Person, Car
Cityscapes [115]	Driving in Frankfurt	Road, Sidewalk, Building, Sky, Person, Car
LVS [104]	Badminton	Person
	Squash	Person
	Table Tennis	Person
	Softball	Person
	Hockey	Person
	Soccer	Person
	Tennis	Person
	Volleyball	Person
	Ice Hockey	Person
	Kabaddi	Person
	Figure Skating	Person
	Drone	Person
	Birds	Bird
	Dogs	Car, Dog, Person
	Horses	Horse, Person
	Ego Ice Hockey	Person
	Ego Basketball	Car, Person
	Ego Dodgeball	Person
	Ego Soccer	Person
	Biking	Bicycle, Person
	Streetcam1	Car, Person
	Streetcam2	Car, Person
	Jackson Hole	Car, Person
	Murphys	Bicycle, Car, Person
	Samui Street	Bicycle, Car, Person
	Toomer	Car, Person
	Driving	Bicycle, Car, Person
	Walking	Bicycle, Car, Person

Table A.1: Summary of the video datasets and their target classes in evaluations.

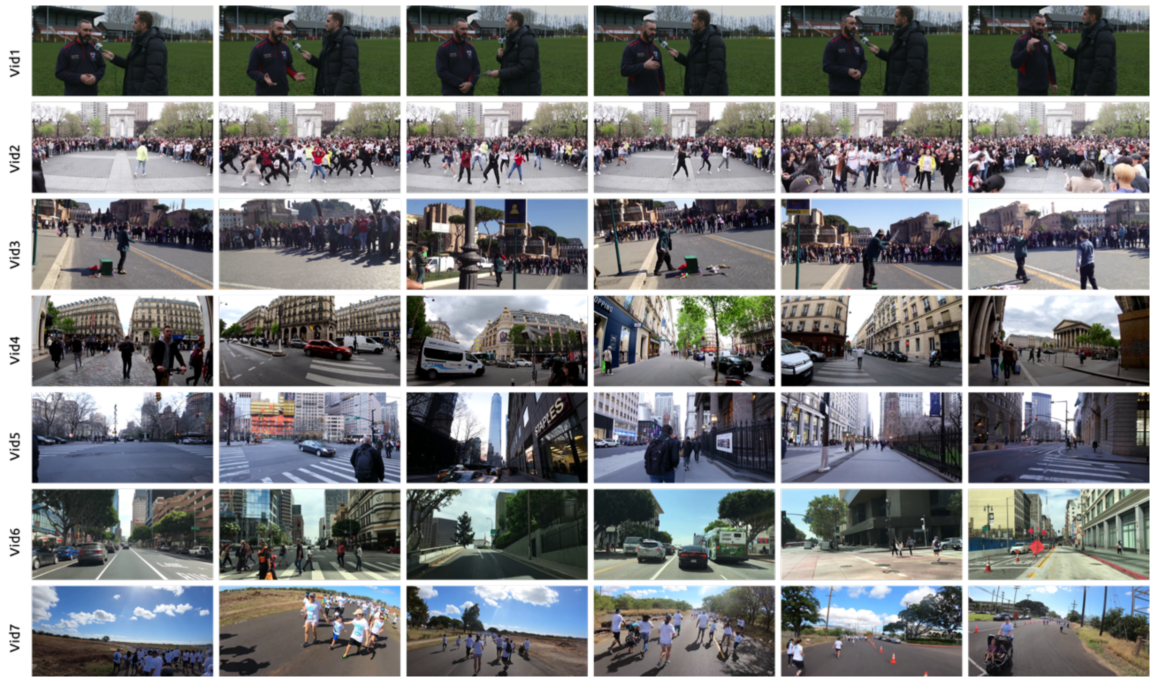


Figure A-1: Sample video frames. Rows (from top to bottom) correspond to Interview, Dance Recording, Street Comedian, Walking in Paris, Walking in NY, Driving in LA, and Running.

Appendix B

AMS Adaptive Training Rate (ATR)

We dynamically update the model update interval T_{update} for each device based on its video characteristics. For this purpose, for each interval n, we look at ASR's sampling rate decision r_n (see §4.3.2). A small sampling rate typically implies the scenes are changing slowly, and conversely a large sampling rate suggests fast variations.

We introduce a *slowdown* mode to capture relatively stationary scenes. We enter the slowdown mode if the scenes are highly similar, $r_n < \gamma_0$, and exit this mode as variations increase, $r_n > \gamma_1$. Our implementation uses $\gamma_0 = 0.25$ fps and $\gamma_1 = 0.35$ fps. We start at the maximum training rate ($T_{update} = \tau_{min}$), and update the training interval every δt seconds according to:

$$T_{update}(n+1) = \begin{cases} T_{update}(n) + \Delta, & \text{in slowdown mode} \\ \tau_{min}, & \text{otherwise} \end{cases} \tag{B.1}$$

This rule gradually increases T_{update} by a fixed Δ (e.g., $\Delta = 2$ sec) in slowdown mode, and

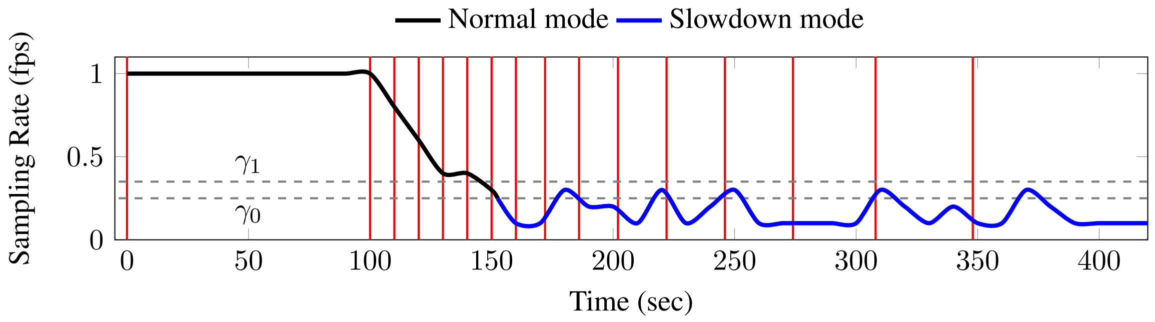


Figure B-1: ATR updates of the model update intervals w.r.t. the average sampling rate over time for Vid1. Vertical lines represent model updates. Model updates become distant after entering the slowdown mode.

aggressively resets it to τ_{min} as soon as we exit the slowdown mode to catch up with scene changes.

The sever communicates the newest T_{update} (and sampling rate) with the edge so that the edge device can accordingly synchronize its sample buffering and upload process (see §4.3.2).

In Figure B-1, we plot the behavior of ATR algorithm for the Interview video with relatively stationary scenes from Outdoor Scenes dataset. We observe that ATR enters the slowdown mode after 150 seconds as the average sampling rate goes below the entrance threshold γ_0, and it stays in this mode as the scenes rarely change after this point. We denote the model updates using the vertical lines in this plot. ATR increases the distance between the model updates in the slowdown mode to save the training cycles for other videos.

Appendix C

AMS Server Utilization

Every user that joins a cloud server requires its own share of GPU resources for inference and training operations. GPUs are expensive. At the current time, renting a GPU like the NVIDIA Tesla V100 in the cloud costs at least $1 per hour. Hence, it is important to use server GPU resources efficiently and serve multiple edge devices per GPU to keep per-user cost low.

In our prototype, we use a simple strategy that iterates in a round-robin fashion across multiple video sessions, completing one inference and training step per session. By allowing only one process to access the GPU at a time, we minimize context switching overhead. In Figure C-1 we show the decrease (w.r.t. single client) in average mIoU when different number of clients share a GPU. We observe that even with a simple round-robin scheduling algorithm, AMS scales to up to 7 edge devices on a single V100 GPU with less than 1% loss in mIoU without adaptive training rate (ATR) enabled. Enabling ATR (see Appendix B) increases the number of supported edge devices to 9. Note that these results depend on the distribution of the videos and for this purpose, we have assumed a uniform sampling of videos from the Outdoor Scenes dataset and reported the average result of multiple runs

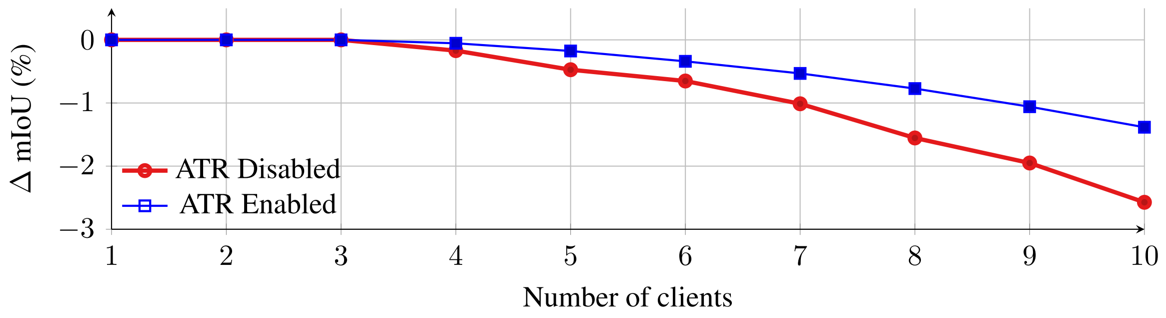


Figure C-1: Average multiclient mIoU degradation compared to single-client performance.

here. As most of the videos in this dataset (5 out of 7) tend to experience relatively high levels of scene dynamic, majority of videos get high training frequency. Hence, we expect to be able to support at least equal or even more devices by prioritizing certain videos that need more frequent model updates over the stationary ones in real-world distribution of videos.

Furthermore, we note that ASR (see §4.3.2) also significantly reduces the overhead of running teacher inference over redundant frames at the server. The impact is particularly pronounced because the teacher model usually runs at high input resolution and consumes a significant amount of GPU time (up to 200 ms for labeling each frame on an NVIDIA V100 GPU for the task of semantic segmentation).

Bibliography

[1] Mehrdad Khani, Mohammad Alizadeh, Jakob Hoydis, and Phil Fleming. Adaptive neural signal detection for massive mimo. *IEEE Transactions on Wireless Communications*, 19(8):5635–5648, 2020.

[2] Mehrdad Khani, Pouya Hamadanian, Arash Nasr-Esfahany, and Mohammad Alizadeh. Real-time video inference on edge devices via adaptive model streaming. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[3] Mehrdad Khani, Vibhaalakshmi Sivaraman, and Mohammad Alizadeh. Efficient video compression via content-adaptive super-resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4521–4530, 2021.

[4] Mehrdad Khani, Ganesh Ananthanarayanan, Kevin Hsieh, Junchen Jiang, Ravi Netravali, Yuanchao Shu, Mohammad Alizadeh, and Victor Bahl. RECL: Responsive Resource-Efficient Continuous Learning for Video Analytics. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, Boston, MA, April 2023. USENIX Association.

[5] Xiph.org Video Test Media. `https://media.xiph.org/video/derf/`.

[6] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. Dvc: An end-to-end deep video compression framework. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11006–11015, 2019.

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato,

R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[9] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182, 2016.

[10] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018.

[11] Neev Samuel, Tzvi Diskin, and Ami Wiesel. Learning to detect. *IEEE Transactions on Signal Processing*, 67(10):2554–2564, 2019.

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[13] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. Real-time video analytics: The killer app for edge computing. *Computer*, 50(10), 2017.

[14] Si Young Jang, Yoonhyung Lee, Byoungheon Shin, and Dongman Lee. Application-aware iot camera virtualization for video analytics edge computing. In *Symposium on Edge Computing (SEC)*, 2018.

[15] European Parliament. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46 (general data protection regulation). *Official Journal of the European Union (OJ)*, 59, 2016.

[16] Behrouz Jedari, Gopika Premsankar, Gazi Karam Illahi, Mario Di Francesco, Abbas Mehrabi, and Antti Ylä-Jääski. Video caching, analytics, and delivery at the wireless edge: A survey and future directions. *IEEE Commun. Surv. Tutorials*, 23(1), 2021.

[17] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*, 2017.

[18] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 613–627, 2017.

[19] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving. In *2019 {USENIX} Annual Technical Conference ({USENIX} {ATC} 19)*, pages 1049–1062, 2019.

[20] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 459–471, 2013.

[21] Tensorflow. Deelab semantic segmentation model zoo. `https://github.com/tensorflow/models/blob/master/research/deeplab/g3doc/model_zoo.md`, 2020.

[22] Tensorflow. Tensorflow object detection model zoo. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`, 2020.

[23] Coral. Coral Edge TPU datasheet. `https://coral.ai/docs/dev-board/datasheet/`, 2020.

[24] NVIDIA. NVIDIA Jetson Nano datasheet. `https://developer.nvidia.com/embedded/jetson-nano`, 2020.

[25] Coral. Edge TPU performance benchmarks. `https://coral.ai/docs/edgetpu/benchmarks/`, 2020.

[26] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. Bandwidth-efficient live video analytics for drones via edge computing. pages 159–173, 10 2018.

[27] Qianlin Liang, Prashant J. Shenoy, and David Irwin. AI on the Edge: Rethinking AI-based IoT Applications Using Specialized Edge Architectures. *ArXiv*, abs/2003.12488, 2020.

[28] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[29] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[30] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[31] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[32] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[33] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

[34] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *ArXiv*, abs/1710.09282, 2017.

[35] Zheng Zhu, Qiang Wang, Bo Li, Wei Wu, Junjie Yan, and Weiming Hu. Distractor-aware siamese networks for visual object tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 101–117, 2018.

[36] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2349–2358, 2017.

[37] Yuan-Ting Hu, Jia-Bin Huang, and Alexander Schwing. Maskrnn: Instance level video object segmentation. In *Advances in neural information processing systems*, pages 325–334, 2017.

[38] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5, 2023.

[39] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

[40] Sandeep P Chinchali, Eyal Cidon, Evgenya Pergament, Tianshu Chu, and Sachin Katti. Neural networks meet physical networks: Distributed inference between edge devices and the cloud. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, pages 50–56, 2018.

[41] Sandeep Chinchali, Apoorva Sharma, James Harrison, Amine Elhafsi, Daniel Kang, Evgenya Pergament, Eyal Cidon, Sachin Katti, and Marco Pavone. Network offloading policies for cloud robotics: a learning-based approach. *arXiv preprint arXiv:1902.05703*, 2019.

[42] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 267–282, 2018.

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

[44] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. *SIGCOMM Comput. Commun. Rev.*, 45(5):37–42, September 2015.

[45] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. Association for Computing Machinery.

[46] Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

[47] Eric C Hall and Rebecca M Willett. Dynamical models and tracking regret in online convex programming. In *Proceedings of the 30th International Conference on International Conference on Machine Learning-Volume 28*, pages I–579, 2013.

[48] Lijun Zhang, Shiyin Lu, and Zhi-Hua Zhou. Adaptive online learning in dynamic environments. In *Advances in neural information processing systems*, pages 1323–1333, 2018.

[49] Aryan Mokhtari, Shahin Shahrampour, Ali Jadbabaie, and Alejandro Ribeiro. Online optimization in dynamic environments: Improved regret rates for strongly convex problems. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 7195–7201. IEEE, 2016.

[50] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pages 928–936, 2003.

[51] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.

[52] Mark Herbster and Manfred K Warmuth. Tracking the best expert. *Machine learning*, 32(2):151–178, 1998.

[53] Tianbao Yang, Lijun Zhang, Rong Jin, and Jinfeng Yi. Tracking slowly moving clairvoyant: optimal dynamic regret of online learning with true and noisy gradient. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, pages 449–457, 2016.

[54] Elad Hazan and Comandur Seshadhri. Efficient learning algorithms for changing environments. In *Proceedings of the 26th annual international conference on machine learning*, pages 393–400, 2009.

[55] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *Advances in neural information processing systems*, pages 137–144, 2007.

[56] Guoliang Kang, Lu Jiang, Yi Yang, and Alexander G Hauptmann. Contrastive adaptation network for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4893–4902, 2019.

[57] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

[58] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, pages 348–358, 2019.

[59] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[60] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[61] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[62] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[63] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[64] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *International Conference on Learning Representations (ICLR)*, 2017.

[65] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. *arXiv preprint arXiv:1902.08438*, 2019.

[66] Afif Osseiran, Jose F Monserrat, and Patrick Marsch. *5G Mobile and Wireless Communications Technology*. Cambridge University Press, 2016.

[67] Emil Björnson, Jakob Hoydis, and Luca Sanguinetti. Massive MIMO networks: Spectral, energy, and hardware efficiency. *Foundations and Trends® in Signal Processing*, 11(3-4):154–655, 2017.

[68] Emre Telatar. Capacity of Multi-antenna Gaussian Channels. *Transactions on Emerging Telecommunications Technologies*, 10(6):585–595, 1999.

[69] Emanuele Viterbo and Joseph Boutros. A universal lattice code decoder for fading channels. *IEEE Transactions on Information theory*, 45(5):1639–1642, 1999.

[70] Shaoshi Yang and Lajos Hanzo. Fifty years of mimo detection: The road to large-scale mimos. *IEEE Communications Surveys & Tutorials*, 17(4):1941–1988, 2015.

[71] Erik G Larsson. MIMO detection methods: How they work. *IEEE signal processing magazine*, 26(3), 2009.

[72] Ami Wiesel, Yonina C Eldar, and Shlomo Shamai. Semidefinite relaxation for detection of 16-qam signaling in mimo channels. *IEEE Signal Processing Letters*, 12(9):653–656, 2005.

[73] Neev Samuel, Tzvi Diskin, and Ami Wiesel. Deep mimo detection. In *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2017.

[74] Hengtao He, Chao-Kai Wen, Shi Jin, and Geoffrey Ye Li. A model-driven deep learning network for mimo detection. *arXiv preprint arXiv:1809.09336*, 2018.

[75] Junjie Ma and Li Ping. Orthogonal amp. *IEEE Access*, 5:2020–2033, 2017.

[76] Sergey L Loyka. Channel capacity of mimo architecture using the exponential correlation matrix. *IEEE Communications letters*, 5(9):369–371, 2001.

[77] 3GPP TR 36.873. 2015. “Study on 3D channel model for LTE”. Technical report.

[78] Stephan Jaeckel, Leszek Raschkowski, Kai Börner, and Lars Thiele. QuaDRiGa: A 3-D multi-cell channel model with time evolution for enabling virtual field trials. *IEEE Trans. Antennas Propag.*, 62(6):3242–3256, 2014.

[79] Ingrid Daubechies, Michel Defrise, and Christine De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457, 2004.

[80] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[81] Charles Jeon, Ramina Ghods, Arian Maleki, and Christoph Studer. Optimality of large mimo detection via approximate message passing. In *Information Theory (ISIT), 2015 IEEE International Symposium on*, pages 1227–1231. IEEE, 2015.

[82] Lajos Hanzo, Soon Xin Ng, WT Webb, and T Keller. *Quadrature amplitude modulation: From basics to adaptive trellis-coded, turbo-equalised and space-time coded OFDM, CDMA and MC-CDMA systems*. IEEE Press-John Wiley, 2004.

[83] Alberto Del Pia, Santanu S Dey, and Marco Molinaro. Mixed-integer quadratic programming is in np. *Mathematical Programming*, 162(1-2):225–240, 2017.

[84] Amine Mezghani and Josef A Nossek. Belief propagation based mimo detection operating on quantized channel output. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 2113–2117. IEEE, 2010.

[85] Mohsen Bayati and Andrea Montanari. The dynamics of message passing on dense graphs, with applications to compressed sensing. *IEEE Transactions on Information Theory*, 57(2):764–785, 2011.

[86] Antonia M Tulino, Sergio Verdú, et al. Random matrix theory and wireless communications. *Foundations and Trends® in Communications and Information Theory*, 1(1):1–182, 2004.

[87] Peter W Wolniansky, Gerard J Foschini, GD Golden, and Reinaldo A Valenzuela. V-blast: An architecture for realizing very high data rates over the rich-scattering wireless channel. In *Signals, Systems, and Electronics, 1998. ISSSE 98. 1998 URSI International Symposium on*, pages 295–300. IEEE, 1998.

[88] Mahesh K Varanasi and Behnaam Aazhang. Multistage detection in asynchronous code-division multiple-access communications. *IEEE Transactions on communications*, 38(4):509–519, 1990.

[89] WH Chin, AG Constantinides, and DB Ward. Parallel multistage detection for multiple antenna wireless systems. *Electronics Letters*, 38(12):597–599, 2002.

[90] Ori Shental, Sivarama Venkatesan, Alexei Ashikhmin, and Reinaldo A Valenzuela. Massive blast: An architecture for realizing ultra-high data rates for large-scale mimo. *arXiv preprint arXiv:1708.05405*, 2017.

[91] Toshiyuki Tanaka and Masato Okada. Approximate belief propagation, density evolution, and statistical neurodynamics for cdma multiuser detection. *IEEE Transactions on Information Theory*, 51(2):700–706, 2005.

[92] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[93] Wing-Kin Ma, Chao-Cheng Su, Joakim Jaldén, and Chong-Yung Chi. Some results on 16-qam mimo detection using semidefinite relaxation. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2673–2676. IEEE, 2008.

[94] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2016.

[95] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[96] Wing-Kin Ma, Chao-Cheng Su, Joakim Jaldén, Tsung-Hui Chang, and Chong-Yung Chi. The equivalence of semidefinite relaxation mimo detectors for higher-order qam. *IEEE Journal of Selected Topics in Signal Processing*, 3(6):1038–1052, 2009.

[97] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[98] Hengtao He, Chao-Kai Wen, Shi Jin, and Geoffrey Ye Li. Model-driven deep learning for joint mimo channel estimation and signal detection. *arXiv preprint arXiv:1907.09439*, 2019.

[99] Theodore W Anderson, Donald A Darling, et al. Asymptotic theory of certain" goodness of fit" criteria based on stochastic processes. *The annals of mathematical statistics*, 23(2):193–212, 1952.

[100] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[101] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7291–7299, 2017.

[102] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[103] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. No-scope: optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529*, 2017.

[104] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. Online model distillation for efficient video inference. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3573–3582, 2019.

[105] Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151:3–34, 2015.

[106] Yurii Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), CORE Discussion Papers*, 22, 01 2010.

[107] Youtube. Choose live encoder settings, bitrates, and resolutions. `https://support.google.com/youtube/answer/2853702?hl=en`. Accessed: 2020-06-01.

[108] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[109] Steven S. Beauchemin and John L. Barron. The computation of optical flow. *ACM computing surveys (CSUR)*, 27(3):433–466, 1995.

[110] Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V Le, and Xiaodan Song. Spinenet: Learning scale-permuted backbone for recognition and localization. *arXiv preprint arXiv:1912.05027*, 2019.

[111] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019.

[112] Julie Nutini, Mark Schmidt, Issam H. Laradji, Michael Friedlander, and Hoyt Koepke. Coordinate descent converges faster with the gauss-southwell rule than random selection. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1632–1641. JMLR.org, 2015.

[113] P. Deutsch. Rfc1952: Gzip file format specification version 4.3, 1996.

[114] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.

[115] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016.

[116] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi, Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. A2D2: Audi Autonomous Driving Dataset. 2020.

[117] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[118] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. *ArXiv*, abs/1405.0312, 2014.

[119] Mark Everingham, S. Eslami, Luc Van Gool, Christopher Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111, 01 2014.

[120] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003.

[121] Aruna Medhekar, Vishal Chiluka, and Abhijit Patait. Accelerate OpenCV: Optical Flow Algorithms with NVIDIA Turing GPUs. `https://developer.nvidia.com/blog/opencv-optical-flow-algorithms-with-nvidia-turing-gpus/`, 2019.

[122] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999.

[123] "cisco annual internet report (2018–2023) white paper". `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`.

[124] `https://www.forbes.com/sites/markbeech/2020/03/25/covid-19-pushes-up-internet-use-70-streaming-more-than-12-first-figu`
`?sh=4335cc443104`.

[125] `https://abcnews.go.com/Technology/netflix-youtube-throttle-streaming`
`story?id=69754458`.

[126] `https://www.theverge.com/2020/3/20/21188072/amazon-prime-video-reduce-stream-quality-broadband-netflix-youtube-c`

[127] Guy Cote, Berna Erol, Michael Gallant, and Faouzi Kossentini. H. 263+: Video coding at low bit rates. *IEEE Transactions on circuits and systems for video technology*, 8(7):849–866, 1998.

[128] x265 HEVC Encoder / H.265 Video Codec. `http://x265.org/`.

[129] Eirikur Agustsson, David Minnen, Nick Johnston, Johannes Balle, Sung Jin Hwang, and George Toderici. Scale-space flow for end-to-end optimized video compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8503–8512, 2020.

[130] Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G Anderson, and Lubomir Bourdev. Learned video compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3454–3463, 2019.

[131] Yue Chen, Debargha Murherjee, Jingning Han, Adrian Grange, Yaowu Xu, Zoe Liu, Sarah Parker, Cheng Chen, Hui Su, Urvang Joshi, et al. An overview of core coding tools in the av1 video codec. In *2018 Picture Coding Symposium (PCS)*, pages 41–45. IEEE, 2018.

[132] Robert Keys. Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160, 1981.

[133] Egor Zakharov, Aleksei Ivakhnenko, Aliaksandra Shysheya, and Victor Lempitsky. Fast bi-layer neural synthesis of one-shot realistic head avatars. In *European Conference on Computer Vision*, pages 524–540. Springer, 2020.

[134] Hongwei Lin, Xiaohai He, Linbo Qing, Qizhi Teng, and Songfan Yang. Improved low-bitrate hevc video coding using deep learning based super-resolution and adaptive block patching. *IEEE Transactions on Multimedia*, 21(12):3010–3023, 2019.

[135] Longtao Feng, Xinfeng Zhang, Xiang Zhang, Shanshe Wang, Ronggang Wang, and Siwei Ma. A dual-network based super-resolution for compressed high definition video. In *Pacific Rim Conference on Multimedia*, pages 600–610. Springer, 2018.

[136] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[137] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[138] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[139] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.

[140] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. Overview of the scalable video coding extension of the h. 264/avc standard. *IEEE Transactions on circuits and systems for video technology*, 17(9):1103–1120, 2007.

[141] 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, 2012.

[142] Jim Bankoski, Paul Wilkins, and Yaowu Xu. Technical overview of vp8, an open source video codec for the web. In *2011 IEEE International Conference on Multimedia and Expo*, pages 1–6. IEEE, 2011.

[143] Debargha Mukherjee, Jingning Han, Jim Bankoski, Ronald Bultje, Adrian Grange, John Koleszar, Paul Wilkins, and Yaowu Xu. A technical overview of vp9—the latest open-source video codec. *SMPTE Motion Imaging Journal*, 124(1):44–54, 2015.

[144] Muhammad Haris, Gregory Shakhnarovich, and Norimichi Ukita. Recurrent back-projection network for video super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3897–3906, 2019.

[145] Sheng Li, Fengxiang He, Bo Du, Lefei Zhang, Yonghao Xu, and Dacheng Tao. Fast spatio-temporal residual network for video super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10522–10531, 2019.

[146] Zhang, Zhengdong and Sze, Vivienne. FAST: A Framework to Accelerate Super-Resolution Processing on Compressed Videos. In *CVPR Workshop on New Trends in Image Restoration and Enhancement*, 2017.

[147] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.

[148] Xintao Wang, Kelvin CK Chan, Ke Yu, Chao Dong, and Chen Change Loy. Edvr: Video restoration with enhanced deformable convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[149] Ren Yang, Fabian Mentzer, Luc Van Gool, and Radu Timofte. Learning for video compression with hierarchical quality and recurrent enhancement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6628–6637, 2020.

[150] Chao-Yuan Wu, Nayan Singhal, and Philipp Krahenbuhl. Video compression through image interpolation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 416–431, 2018.

[151] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox.

Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[152] Amirhossein Habibian, Ties van Rozendaal, Jakub M Tomczak, and Taco S Cohen. Video compression with rate-distortion autoencoders. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7033–7042, 2019.

[153] Gang He, Chang Wu, Lei Li, Jinjia Zhou, Xianglin Wang, Yunfei Zheng, Bing Yu, and Weiying Xie. A video compression framework using an overfitted restoration neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 148–149, 2020.

[154] Yat-Hong Lam, Alireza Zare, Francesco Cricri, Jani Lainema, and Miska M Hannuksela. Efficient adaptation of neural network filter for video compression. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 358–366, 2020.

[155] Robert Keys. Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160, 1981.

[156] `https://help.netflix.com/en/node/306`.

[157] Frank Bossen et al. Common test conditions and software reference configurations. In *JCTVC-L1100*, volume 12, 2013.

[158] Alexandre Mercat, Marko Viitanen, and Jarno Vanne. Uvg dataset: 50/120fps 4k sequences for video codec analysis and development. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pages 297–302, 2020.

[159] Haiqiang Wang, Weihao Gan, Sudeng Hu, Joe Yuchieh Lin, Lina Jin, Longguang Song, Ping Wang, Ioannis Katsavounidis, Anne Aaron, and C-C Jay Kuo. Mcl-jcv: a jnd-based h. 264/avc video quality assessment dataset. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 1509–1513. IEEE, 2016.

[160] Ping Wah Wong and Cormac Herley. Area based interpolation for image scaling, March 30 1999. US Patent 5,889,895.

[161] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 126–135, 2017.

[162] `https://github.com/GuoLusjtu/DVC/tree/master/TestDemo/VideoCodec/model`.

[163] Gisle Bjontegaard. Calculation of average psnr differences between rd-curves. *VCEG-M33*, 2001.

[164] Xiph Dataset Meridian Video. `https://media.xiph.org/video/derf/meridian/MERIDIAN_SHR_C_EN-XX_US-NR_51_LTRT_UHD_20160909_OV/`.

[165] Iyiola E Olatunji and Chun-Hung Cheng. Video analytics for visual surveillance and applications: An overview and survey. *Machine Learning Paradigms*, pages 475–515, 2019.

[166] MarketsAndMarkets. Video analytics market with covid-19 impact, by component, application (intrusion management, incident detection, people/crowd counting, traffic monitoring), deployment model (on-premises and cloud), type, vertical, and region - global forecast to 2026. `https://www.marketsandmarkets.com/Market-Reports/intelligent-video-analytics-market-778.html`, 2021.

[167] Amazon outposts. `https://aws.amazon.com/outposts/`.

[168] Azure stack edge. `https://azure.microsoft.com/en-us/services/databox/edge/`.

[169] Ion Stoica. The future of computing is distributed. `https://www.datanami.com/2020/02/26/the-future-of-computing-is-distributed/`, 2020.

[170] Shadi A. Noghabi, Landon P. Cox, Sharad Agarwal, and Ganesh Ananthanarayanan. The emerging landscape of edge computing. *GetMobile Mob. Comput. Commun.*, 23(4), 2019.

[171] Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. Searching for mobilenetv3. In *International Conference on Computer Vision (ICCV)*, 2019.

[172] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[173] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YOLOX: exceeding YOLO series in 2021. *CoRR*, abs/2107.08430, 2021.

[174] Azure linux virtual machine pricing. `https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/`.

[175] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. MCDNN: an approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2016.

[176] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: Optimizing neural network queries over video at scale. *Proc. VLDB Endow.*, 10(11):1586–1597, aug 2017.

[177] Microsoft Rocket for live video analytics. https://www.microsoft.com/en-us/research/project/live-video-analytics/, 2021.

[178] Sujith Ravi. Custom on-device ML models with Learn2Compress. https://ai.googleblog.com/2018/05/custom-on-device-ml-models.html, 2018.

[179] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. Online model distillation for efficient video inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[180] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2022.

[181] Samvit Jain, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, and Joseph Gonzalez. Scaling video analytics systems to large camera deployments. In *Proceedings of the International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2019.

[182] Abhijit Suprem, Joy Arulraj, Calton Pu, and Joao Ferreira. Odin: Automated drift detection and recovery in video analytics. *Proc. VLDB Endow.*, 13(12):2453–2465, jul 2020.

[183] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Comput.*, 6(2), 1994.

[184] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *CoRR*, abs/2106.05974, 2021.

[185] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[186] Paperswithcode leaderboard of object detection on PASCAL VOC 2007 dataset. https://paperswithcode.com/sota/object-detection-on-pascal-voc-2007. Accessed: September 2022.

[187] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled Ben Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Commun. Surv. Tutorials*, 19(4), 2017.

[188] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodík, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[189] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. Fast video classification via adaptive cascading of deep models. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 3646–3654, 2017.

[190] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018.

[191] 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 *IEEE/ACM Symposium on Edge Computing (SEC)*, 2020.

[192] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3366–3375, 2017.

[193] Geoffrey Hinton. Introduction to neural networks and machine learning, lecture 15.

[194] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[195] Haoyu Zhang, Logan Stafman, Andrew Or, and Michael J Freedman. Slaq: quality-driven scheduling for distributed machine learning. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC)*, 2017.

[196] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035, 2019.

[197] gRPC. `https://grpc.io/about/`.

[198] NVIDIA. TensorRT. `https://developer.nvidia.com/tensorrt`.

[199] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. BDD100K: A diverse driving dataset for heterogeneous multitask learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[200] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

[201] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[202] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[203] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[204] CIPHERMODE Labs. `https://www.ciphermode.tech/solutions-secureai`. Accessed: September 2022.

[205] Ran Xu, Rakesh Kumar, Pengcheng Wang, Peter Bai, Ganga Meghanath, Somali Chaterji, Subrata Mitra, and Saurabh Bagchi. Approxnet: Content and contention-aware video object classification system for embedded clients. *ACM Transactions on Sensor Networks (TOSN)*, 18(1):1–27, 2021.

[206] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[207] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and {Delay-Tolerance}. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017.

[208] 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 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM)*, 2020.

[209] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2015.

[210] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. Deepcache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2018.

[211] Angela H Jiang, Daniel L-K Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A Kozuch, Padmanabhan Pillai, David G Andersen, and Gregory R Ganger. Mainstream: Dynamic Stem-Sharing for Multi-Tenant video processing. In *USENIX Annual Technical Conference (USENIX ATC)*, 2018.

[212] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics. *Proc. VLDB Endow.*, 13(4):533–546, dec 2019.

[213] Jiashen Cao, Ramyad Hadidi, Joy Arulraj, and Hyesoon Kim. Thia: Accelerating video analytics using early inference and fine-grained query planning. *arXiv preprint arXiv:2102.08481*, 2021.

[214] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[215] Tutel: An efficient mixture-of-experts implementation for large DNN model training. `https://www.microsoft.com/en-us/research/blog/tutel-an-efficient-mixture-of-experts-implementation-for-large-dnn-m` Accessed: September 2022.

[216] Jeremy Z Kolter and Marcus A Maloof. Using additive expert ensembles to cope with concept drift. In *Proceedings of the 22nd international conference on Machine learning (ICML)*, pages 449–456, 2005.

[217] J Zico Kolter and Marcus A Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research*, 8:2755–2790, 2007.

[218] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys)*, 2018.

[219] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. Nexus: A GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, 2019.