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

Mehrdad Khani, Pouya Hamadanian, Arash Nasr-Esfahany, Mohammad Alizadeh
MIT CSAIL
{khani,pouyah,arashne,alizadeh}@csail.mit.edu

Abstract

Real-time video inference on edge devices like mobile phones and drones is challenging due to the high computation cost of Deep Neural Networks. We present Adaptive Model Streaming (AMS), a new approach to improving the performance of efficient lightweight models for video inference on edge devices. AMS uses a remote server to continually train and adapt a small model running on the edge device, boosting its performance on the live video using online knowledge distillation from a large, state-of-the-art model. We discuss the challenges of over-the-network model adaptation for video inference and present several techniques to reduce communication the cost of this approach: avoiding excessive overfitting, updating a small fraction of important model parameters, and adaptive sampling of training frames at edge devices. On the task of video semantic segmentation, our experimental results show 0.4–17.8 percent mean Intersection-over-Union improvement compared to a pre-trained model across several video datasets. Our prototype can perform video segmentation at 30 frames-per-second with 40 milliseconds camera-to-label latency on a Samsung Galaxy S10+ mobile phone, using less than 300 Kbps uplink and downlink bandwidth on the device.

1. Introduction

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 [52], semantic segmentation [7], and pose estimation [6]. However, state-of-the-art DNN models are too expensive to run on low-powered edge devices (e.g., mobile phones, drones, consumer robots [57, 58]), and cannot run in real-time even on accelerators such as Coral Edge TPU and NVIDIA Jetson [12, 59, 38].

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 [30] to transfer knowledge from a large "teacher" model to a small "student" model. For example, Noscope [33] trains a student model to detect a few object classes on specific videos offline. Just-In-Time [46] 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 is computationally infeasible.

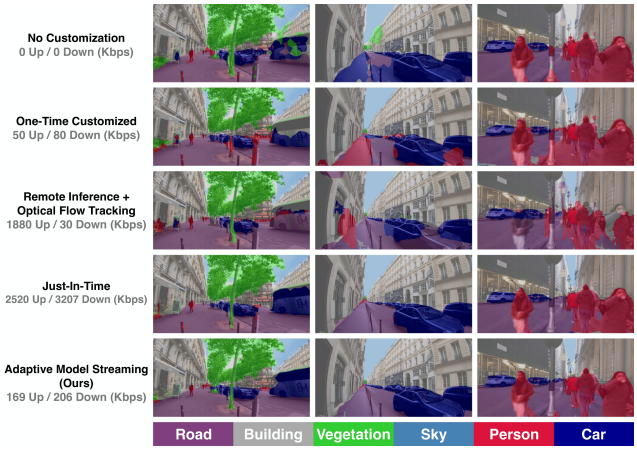

Figure 1: Semantic segmentation results on real-world outdoor videos: rows from top to bottom represent No Customization, One-Time, Remote+Tracking, Just-In-Time, and AMS. Uplink and downlink bandwidth usage are reported below each variant. AMS provides better accuracy with limited bandwidth and reduces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

In this paper 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 [54] backbone with ∼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 [61, 47] 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 [36] 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 [64].

We evaluate our approach for real-time semantic segmentation using a lightweight model (DeeplabV3 with MobileNetV2 [54] 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 mean Intersection-over-Union (mIoU), computed relative to the labels from a state-of-the-art DeeplabV3 with Xception65 [11] 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 [67, 1], 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 [46] (with similar reductions in uplink bandwidth).

Figure 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.

2. Related Work

We described prior work on knowledge distillation for video in §1. Here, we discuss other related work.

On-device inference. Small, mobile-friendly models have been designed both manually [54] and using neural architecture search [70, 62]. Model quantization and weight pruning [28, 39, 3, 8] 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 [68, 67, 32]. Despite this progress, there remains a large gap in the performance of lightweight models and state-of-the-art solutions [16, 31]. 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 [35, 10, 9, 49, 14, 65], but these schemes generally require high network bandwidth, incur high latency, and are susceptible to network outages [21, 14]. Proposals like edge computing [56, 22, 5] 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 [55] algorithms for minimizing dynamic or tracking regret [24, 66, 45]. 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 [69, 25]. Other work has focused on the "experts" setting [29, 63, 26], 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 life-

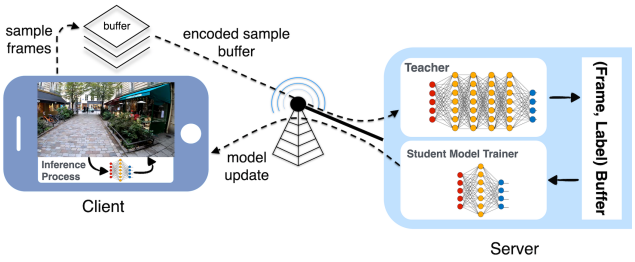

Figure 2: AMS system overview.

long/continual learning [41], meta-learning [19, 51], federated learning [43], and unsupervised domain adaptation [2, 34]. As our work is only tangentially related to these efforts, we defer their discussion to Appendix B.

3. Adaptive Model Streaming (AMS)

Figure 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 [30], 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 1 shows the server procedure for serving a single edge device (we discuss multiple edge devices in Appendix E). 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 [7] with Xception65 [11] 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 §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 §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.

Algorithm 1 Adaptive Model Streaming Server

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

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.

3.1.1 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 [46], 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).

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 5).

For semantic segmentation using DeeplabV3 with MobileNetV2 [54] 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). However, the optimal values of these parameters can depend on both the model ca-

pacity 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 C discusses the interplay between model capacity, $T_{horizon}$, and T_{update} in more detail, and Appendix D describes a simple technique to dynamically adapt T_{update} to minimize bandwidth consumption (especially for stationary videos).

3.1.2 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* [61, 47], 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* [48], 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 [36] 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 [36]. 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 2 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 $\mathcal{I}_n$ 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.

Algorithm 2 *Gradient-Guided Method for Adam Optimizer*

1: $\mathcal{I}_n \leftarrow$ Indices of γ 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: $\quad \mathbf{S}_k \leftarrow$ Uniformly sample a mini-batch of data points from $\mathcal{B}$ over the last $T_{horizon}$ seconds
8: $\quad \mathbf{g}_{n,k} \leftarrow \nabla_{\mathbf{w}} \tilde{\mathcal{L}}(\mathbf{S}_k; \mathbf{w}_{n,k-1})$ {Get the gradient of all model parameters w.r.t. loss on $\mathcal{S}_k$}
9: $\quad \mathbf{m}_{n,k} \leftarrow \beta_1 \cdot \mathbf{m}_{n,k-1} + (1-\beta_1) \cdot \mathbf{g}_{n,k}$ {Update first moment estimate}
10: $\quad \mathbf{v}_{n,k} \leftarrow \beta_2 \cdot \mathbf{v}_{n,k-1} + (1-\beta_2) \cdot \mathbf{g}_{n,k}^2$ {Update second moment estimate}
11: $\quad i \leftarrow i + 1$ {Increment Adam's global step count}
12: $\quad \mathbf{u}_{n,k} \leftarrow \alpha \cdot \frac{\sqrt{1-\beta_2^i}}{1-\beta_1^i} \cdot \frac{\mathbf{m}_{n,k}}{\sqrt{\mathbf{v}_{n,k}+\epsilon}}$ {Calculate the Adam updates for all model parameters}
13: $\quad \mathbf{w}_{n,k} \leftarrow \mathbf{w}_{n,k-1} - \mathbf{u}_{n,k} * \mathbf{b}_n$ {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: $\mathbf{w}_n \leftarrow \mathbf{w}_{n,K}$

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 [15] 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 (§4.2).

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.

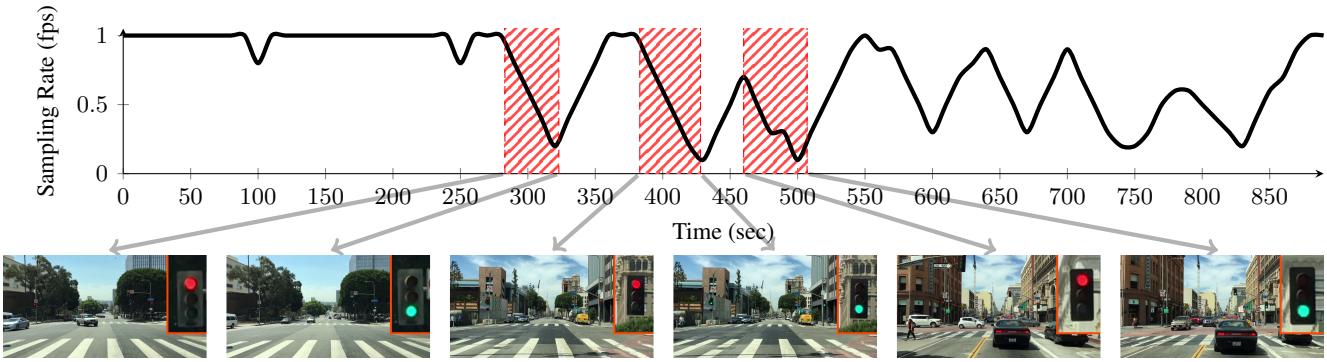

Figure 3: Adaptive frame sampling for a driving video. The sampling rate decreases every time the car slows down for the red traffic light and increases as soon as the light turns green.

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}:

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

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 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 [60] 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. Evaluation

4.1. Methodology

Datasets. We evaluate AMS on the task of semantic segmentation using four video datasets: Cityscapes [13] driving sequence in Frankfurt (1 video, 46 mins long)[1], LVS [46] (28 videos, 8 hours in total), A2D2 [23] (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 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 [7] model with Xception65 [11] backbone (2048×1024 input resolution) trained on the Cityscapes dataset [13] as the teacher model. For LVS, we follow Mullapudi *et al.* [46] in using Mask R-CNN [27] trained on the MS-COCO dataset [40] 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

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

as summarized in Table 4 in Appendix A.

Schemes. On the edge device, we use the DeeplabV3 with MobileNetV2 [54] 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 [17]. For the rest of the datasets we used a checkpoint pre-trained for Cityscapes [13].
- **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 [67, 1]. For tracking, we use the OpenCV implementation of Farneback optical flow estimation [18] 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 [44] 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 §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 [46] 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*. [46] 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 (§3.1.2) 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 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 [46], we use the Momentum Optimizer [50] with a momentum of 0.9.
- **AMS:** We use Algorithm 1 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 §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 [36] with a learning rate of 0.001 ($\beta_1 = 0.9$, $\beta_2 = 0.999$).

4.2. Results

Comparison to baselines. Table 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 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

[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.

Dataset	Metric	No Customization	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 [23]	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 [13]	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 [46]	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 1: Comparison of mIoU (in percent), Uplink and Downlink bandwidth (in Kbps) for different methods across 4 video datasets.

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

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 consistently 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 D.

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 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× more downlink bandwidth (15.7× on average), and 5.2–37.1× more uplink bandwidth (16.8× 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

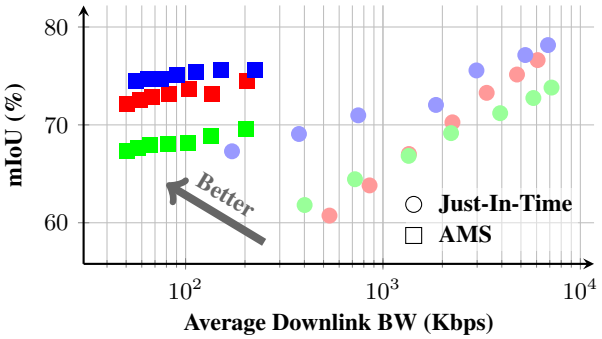

Figure 4: mIoU vs. downlink bandwidth for AMS and Just-In-Time with different parameters. Each color represents one dataset and each marker's shape/tint represents the scheme.

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 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 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 §3.1.1, 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 3 compares the gradient-guided method descibed in §3.1.2 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

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

Strategy	Fraction 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 3: Average difference in mIoU relative to full-model training (in percent) for different coordinate descent strategies on the Outdoor Scenes dataset.

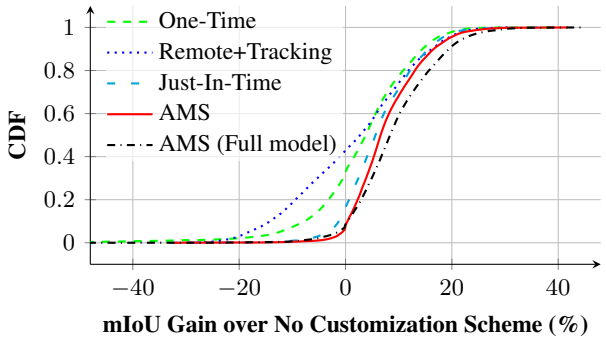

Figure 5: CDF of mIoU gain relative to No Customization across all frames for different schemes.

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

Overall, Table 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 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

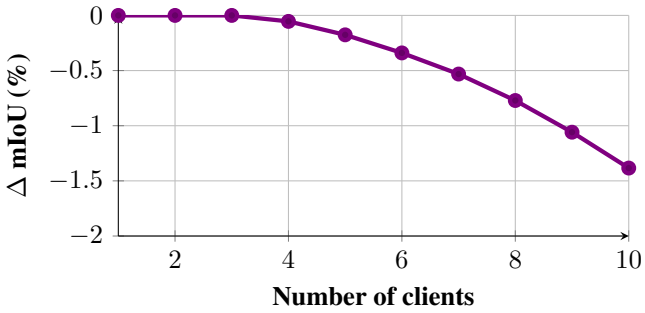

Figure 6: Average multiclient mIoU degradation compared to single-client performance on Outdoor Scenes dataset.

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 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 D for more details).

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× less bandwidth to achieve similar accuracy to recent online distillation methods.

6. Acknowledgments

We would like to thank our anonymous reviewers and meta-reviewer for their valuable feedback. This work was supported in part by NSF grants CNS-1751009, CNS-1955370, CNS-1910676, a Cisco Research Center Award, a Microsoft Faculty Fellowship, and awards from the Machine-LearningApplications@CSAIL and MIT.nano NCSOFT programs.

References

[1] Steven S. Beauchemin and John L. Barron. The computation of optical flow. *ACM computing surveys (CSUR)*, 27(3):433–466, 1995. 2, 6

[2] 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. 3, 12

[3] 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. 2

[4] 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. 12

[5] 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, page 13–16, New York, NY, USA, 2012. Association for Computing Machinery. 2

[6] 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. 1

[7] 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*, 40:834–848, 2018. 1, 3, 5

[8] 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. 2

[9] 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. 2

[10] 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. 2

[11] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017. 2, 3, 5

[12] Coral. Edge TPU performance benchmarks. https://coral.ai/docs/edgetpu/benchmarks/, 2020. 1

[13] 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. 5, 6, 7, 12, 13

[14] 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. 2

[15] P. Deutsch. Rfc1952: Gzip file format specification version 4.3, 1996. 4

[16] 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. 2

[17] 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. 6

[18] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003. 6

[19] 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. 3, 12

[20] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. *arXiv preprint arXiv:1902.08438*, 2019. 12

[21] 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. 2

[22] 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, Sept. 2015. 2

[23] 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. 5, 7, 12, 13

[24] 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. 2

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

[26] 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. 2

[27] 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. 5

[28] 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. 2

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

[30] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1, 3, 12

[31] 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. 2

[32] 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. 2

[33] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529*, 2017. 1

[34] 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. 3, 12

[35] 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. 2

[36] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015. 2, 4, 6

[37] 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. 12

[38] 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. 1

[39] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016. 2

[40] 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. 5

[41] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017. 3, 12

[42] 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. 12

[43] 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. 3, 12

[44] 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. 6

[45] 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. 2

[46] 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. 1, 2, 3, 5, 6, 7, 12, 13

[47] 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. 2, 4

[48] 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, page 1632–1641. JMLR.org, 2015. 4

[49] 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. 2

[50] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999. 6

[51] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *International Conference on Learning Representations (ICLR)*, 2017. 3, 12

[52] 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. 1

[53] 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. 12

[54] 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. 2, 3, 6

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

[56] 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. 2

[57] Tensorflow. Deelab semantic segmentation model zoo. https://github.com/tensorflow/models/blob/master/research/deeplab/g3doc/model_zoo.md, 2020. 1

[58] Tensorflow. Tensorflow object detection model zoo. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md, 2020. 1

[59] 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. 1

[60] 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. 5

[61] Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151:3–34, 2015. 2, 4

[62] 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. 2

[63] 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. 2

[64] Youtube. Choose live encoder settings, bitrates, and resolutions. https://support.google.com/youtube/answer/2853702?hl=en. Accessed: 2020-06-01. 2

[65] 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. 2

[66] 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. 2

[67] 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. 2, 6

[68] 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. 2

[69] 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. 2

[70] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 2