RECL: Responsive Resource-Efficient Continuous Learning for Video Analytics

Mehrdad Khani[1,2], Ganesh Ananthanarayanan[2], Kevin Hsieh[2], Junchen Jiang[3], Ravi Netravali[4], Yuanchao Shu[2], Mohammad Alizadeh[1], Victor Bahl[2]
[1]*MIT CSAIL,* [2]*Microsoft,* [3]*University of Chicago,* [4]*Princeton University*

Abstract

Continuous learning has recently shown promising results for video analytics by adapting a lightweight "expert" DNN model for each specific video scene to cope with the data drift in real time. However, current adaptation approaches either rely on periodic retraining and suffer its delay and significant compute costs or rely on selecting historical models and incur accuracy loss by not fully leveraging the potential of persistent retraining. Without dynamically optimizing the resource sharing among model selection and retraining, both approaches have a diminishing return at scale. RECL is a new video-analytics framework that carefully integrates model reusing and online model retraining, allowing it to quickly adapt the expert model given any video frame samples. To do this, RECL *(i)* shares across edge devices a (potentially growing) "model zoo" that comprises expert models previously trained for all edge devices, enabling history model reuse across video sessions, *(ii)* uses a fast procedure to online select a highly accurate expert model from this shared model zoo, and *(iii)* dynamically optimizes GPU allocation among model retraining, model selection, and timely updates of the model zoo. Our evaluation of RECL over 70 hours of real-world videos across two vision tasks (object detection and classification) shows substantial performance gains compared to prior work, further amplifying over the system lifetime.

1 Introduction

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 [1, 2]. Across these applications, it is often imperative to run analytics tasks directly on edge devices (e.g., using on-premises edge servers [3, 4]) to ensure that the system can deliver real-time results with low latency and in compliance with data privacy constraints [5–8]. However, the edge has limited compute resources, which cannot match the unrelenting growth of video analytics workloads, DNN models, and video streams [9, 10]. 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 [11–13]. For example, a high-end NVIDIA V100 GPU can only support two video streams running the state-of-the-art YOLOv5-L model [13] at 30 FPS, which translates to a steep cost of $1,100/month/stream on public clouds [14].

One common approach to reducing the resource requirements for video analytics is to use specialized and compressed DNNs [15–18]. 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 [19–21] 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) [20, 21], making model retraining the key bottleneck in scaling video analytics to more video streams with limited compute resources. Our study (Fig. 2) shows that accuracy drops sharply (by 40% in object detection) as 4× more cameras share the GPU cycles to retrain their models (§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. 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) [22]. 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 §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., [20]) 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 [23], 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 [23]) 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 (§3.1). Our model selector is inspired by sparse gating networks in the mixture of experts (MoEs) approach [24–26], 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) [23].
- 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 (§3.3).
- RECL shares GPU resources across the retraining jobs using an *iterative training scheduler* that dynamically prioritizes retraining jobs that progress faster (§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 [21], AMS [20], and ODIN [23]) 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 [27]. Moreover, the baseline systems need at least 3.2× 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.

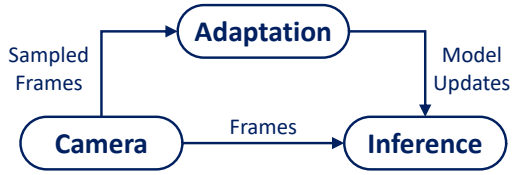


Figure 1: Overview of a video analytics system utilizing continuous learning. A typical adaptation module continuously retrains expert models or selects them from an existing collection of models trained in the past.

2 Background and Motivation

We first introduce the background of continuous retraining and deploying specialized DNNs for video analytics (§2.1). We then discuss the fundamental limitations of this approach and how reusing historical specialized DNNs can address these limitations effectively (§2.2).

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 [28]. 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 [29] 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 [15, 16, 30]. This approach has been widely adopted in modern systems such as Microsoft's Rocket [17] and Google's Learn2Compress [18].

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.) [21]. 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 [19–21, 31] 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) [20].

Fig. 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*) *in-*

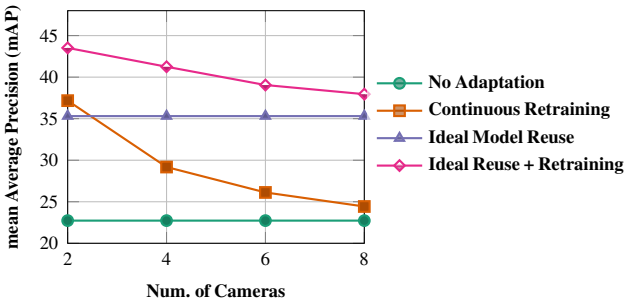


Figure 2: Object detection accuracy (mAP) of different designs under different numbers of cameras. Model reuse has the potential to significantly improve the accuracy in resource-constrained regimes (4, 6, and 8 cameras), and when combining model reuse and model retraining, performance *could* be greatly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

ference service: uses the received lightweight expert model for real-time inference on video frames from the camera service.

This paper focuses on the adaptation service. As retraining an expert model takes significant compute resources and time (§1), the adaptation service becomes a key bottleneck in resource efficiency and accuracy consistency. We observe that these systems [20, 21] 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.

2.2 The Case of Reusing Historical Expert Models

It is well known that a video deployment usually exhibits temporally and spatially recurrent patterns [22,23,32]. 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 §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) [13]. 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 [20] and Ekya [21].
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 [23], 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: Fig. 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 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. 3 shows a concrete example). We demonstrate this benefit with the CDF of mAP across all video segments for the 8 camera setting (Fig. 4). As the figure shows, Ideal Model Reuse has a much better

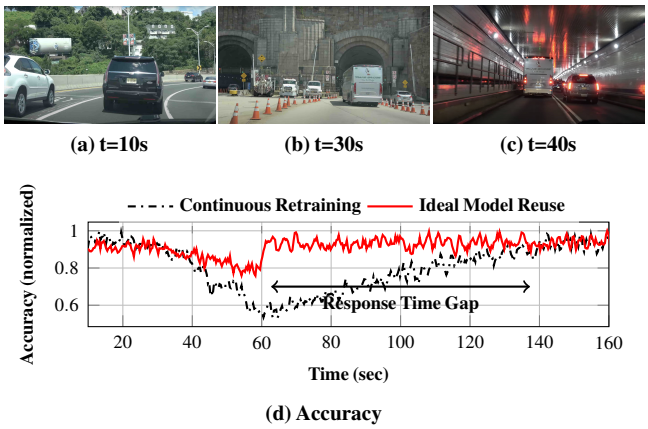


Figure 3: Example of a scene change when a car camera enters a tunnel and how fast Ideal Model Reuse and Continuous Retraining respond: Model updates arrive every 30 seconds. At t=60 sec, both schemes can access sample frames from the tunnel scene. Ideal Model Reuse switches to a good model for the new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

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. Fig. 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 [23]) 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.

3 Design of RECL

This paper 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

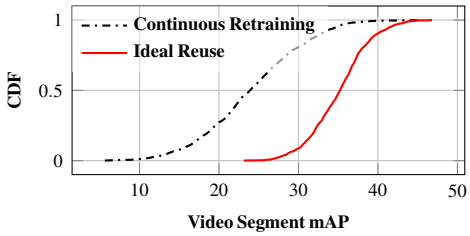


Figure 4: Ideal Reuse improves both average and tail accuracy (mAP) across video segments.

the deployed 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. 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 [20]). AMS takes the drift rate of the labels measured at the server as a signal for setting the frame sampling rate. As 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* (§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* (§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 (§3.3). The rest of the section will present their designs and rationales.

[1] We use fixed update windows, similar to Ekya [21]. 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.

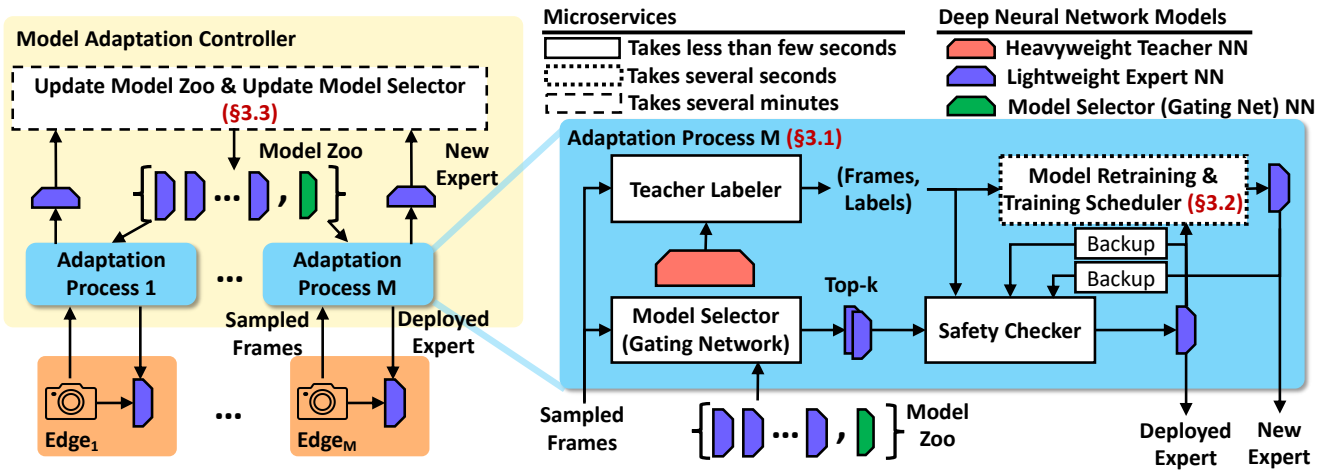


Figure 5: RECL system architecture. Edge devices (cameras) run real-time inference using lightweight models, and the model-adaptation controller manages a model zoo of expert models trained on history frames from edge devices (cameras) and, on receiving a model-update query, quickly selects a suitable 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.)

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/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 §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) [33]. 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 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* [26] 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. (§3.3 will explain how to update the gating network to handle the changing model zoo.)

An alternative approach [23,34] 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 (§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 [35] 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 §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 §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).

3.2 Model Retraining

So far, we discussed how to reuse the previously trained models. Like other continuous learning frameworks [20, 21, 23], 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 [21]), 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.

Algorithm 1 RECL GPU Sharing Algorithm

1: **Input:** training requests $\mathcal{R}$, micro-window number of seconds μ, 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 μ seconds
6: $acc_f \leftarrow r$.EVAL()
7: $budget \leftarrow budget - \mu$
8: **return** $(acc_f$ - $acc_i)/\mu$ ▷ Returns the accuracy gain
9: **end procedure**
10: **for** r in R **do** ▷ Initialize the gain estimates
11: $gain[r] \leftarrow$ PROCESSREQUEST(r)
12: **end for**
13: **while** $budget > \mu$ **do** ▷ Schedule the most promising
14: $r \leftarrow$ argmax$gain$ ▷ Find the request with max gain
15: $gain[r] \leftarrow$ PROCESSREQUEST(r)
16: **end while**

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:

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

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 1). 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 [36].

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 [37] 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.

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 (§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 §5.2.

4 Implementation

We have implemented RECL in Python and used Pytorch [38] for inference and training of ML tasks. For communication between the services, we use the gRPC [39] 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 subsampled video frames to the teacher labeler service running on the adaptation server. We use TensorRT [40] 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.

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× 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 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×) 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× lower overhead.
- Compute overhead of RECL decreases gracefully over time as more models are trained and added to the zoo.

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

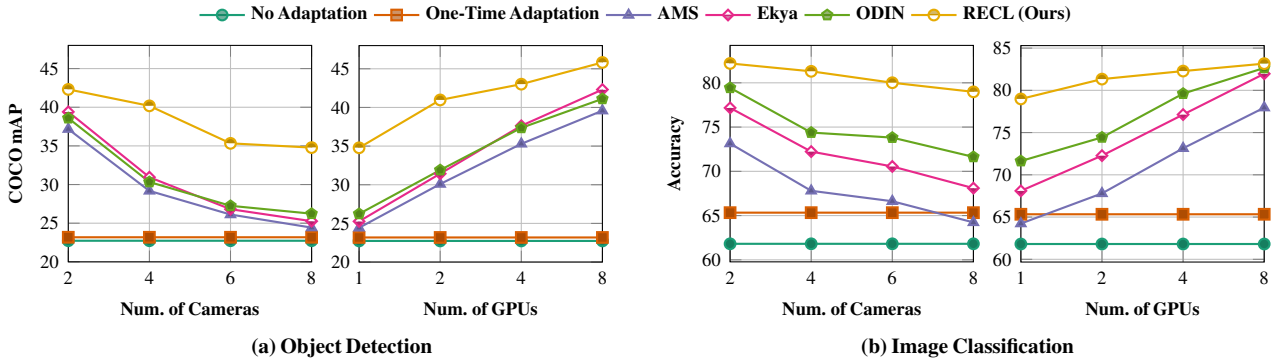
5.1 Methodology & Setup

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 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 [13] for the student and teacher models, respectively. For image classification, we use MobileNetV2 [42] and ResNet50 [43] for the student and teacher models. Details of these models are shown in Table 1. Our models are pre-trained on ImageNet [44] and COCO [45] 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 [20, 21]). 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

[2] Video sessions in other similar video datasets like Berkeley Driving Dataset (BDD) [41] were not long enough for our purpose. For example, each driving episode in BDD is only 40 seconds.



(a) Object Detection (b) Image Classification

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

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 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 [46] and Coral Edge TPU [47], 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 [20], 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 [19] and remote server inference in prior work [20], 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 [21]) 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) [21, §6.4], we do not compare RECL with these naive reuse heuristics.
- **ODIN:** ODIN [23] 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 returns the model with the least distance from the samples in each window as in the ODIN paper [23].

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 Fig. 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× more cameras on one GPU, and requires

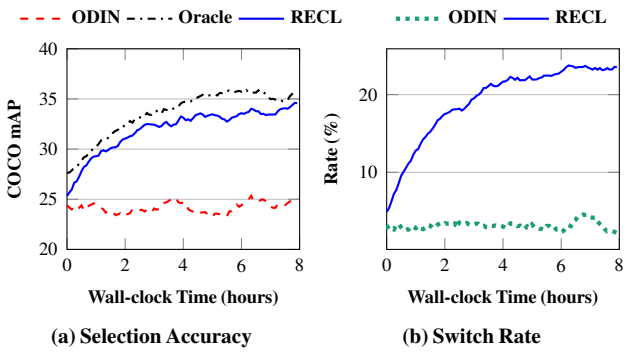


Figure 7: An example of RECL model selection performance over time. As the model zoo grows, (a) accuracy of the RECL-selected models gradually improves, and (b) the model selected by RECL's gating network has higher accuracy than models selected by ODIN (as evidenced by the fact that the safety checker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

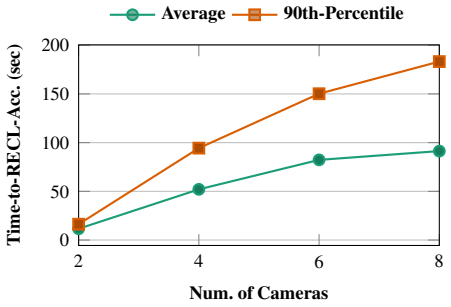


Figure 8: Model reuse impact on improving the response time.

3.2× 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. Without model reuse, RECL's scheduler provides about 1.5% mAP improvement compared to Ekya.
5. 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 Fig. 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

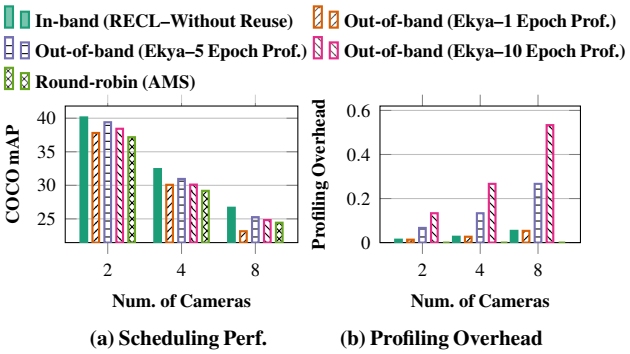


Figure 9: Impact of profiling on retraining performance: RECL's retraining scheduler (which uses a low-overhead in-band profiling) outperforms Ekya (which relies on out-of-band profiling on each job) and AMS (whic uses a round-robin scheduler).

is not surprising that the accuracy of the model selected by RECL improves over 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, 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 1).

We further notice that, in Fig. 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. Fig. 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 Fig. 7b, RECL's model hit ratio increases with a larger zoo, making our system both more accurate and efficient than ODIN. Moreover, notice that the safety checker picks the offered historical model 25% of the time in the case that the most recent trained model is also coming from RECL. To better understand the model reuse impact here, we design the following experiment.

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 the same window. We refer to this metric as Time-to-RECL-Accuracy. Figure 8 shows the

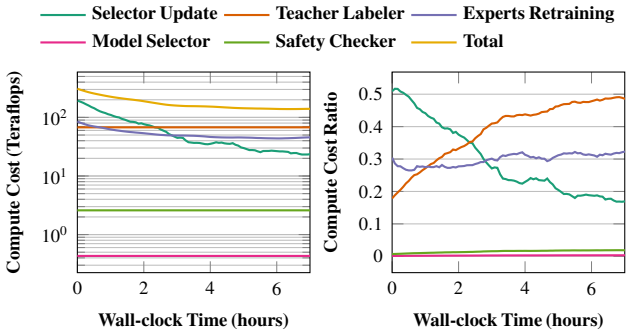


Figure 10: Breakdown of compute cost by the components of RECL controller. The total cost drops over time as the extra-cost of maintaining the model zoo significantly reduces, allowing RECL to enjoy the benefit of model reuse without much additional overhead.

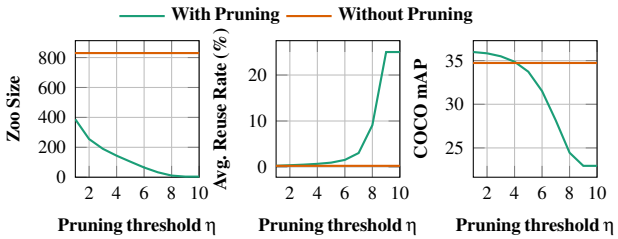


Figure 11: Pruning policy impact on RECL accuracy.

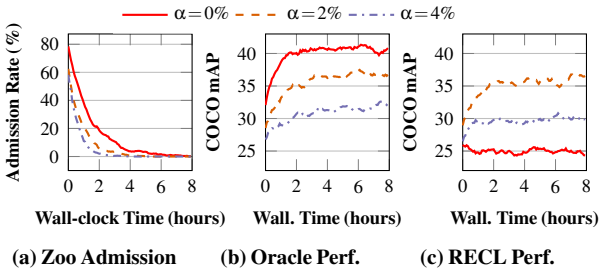


Figure 12: Impact of changing the admission rate through the α-promise threshold on RECL model selection performance.

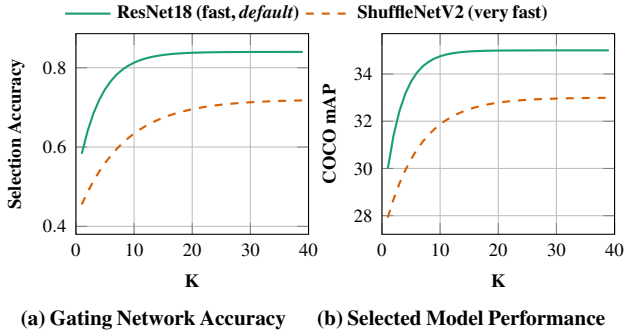


Figure 13: Impact of using top-k models suggested by the gating network for the default gating network and a faster gating network model.

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 (§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 turn off RECL's model reuse and 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. Fig. 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: Fig. 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 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.

5.3 Ablation Studies

Model zoo pruning: In Fig. 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.

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. Fig. 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. 12 and 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. The parameter γ controls the gating network update frequency (i.e., cost). As the Selector Update cost in Fig. 10 shows, this cost diminishes over time as the system collects a comprehensive set of experts. Therefore, RECL 's performance in steady state is not as sensitive to γ as it is to α and η.

Model selector top-k: As discussed in §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 Fig. 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 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 Fig. 10, our safety checker still has a negligible overhead compared to the other components in the system.

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 [48], 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.

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 [16, 20, 21], model architecture pruning [49, 50], configuration adaptation [32, 51], frame selection [52, 53], and DNN feature reusing [54, 55]. 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 [16, 56]. 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 [19–21] or on the most relevant images from the training set [31], and model selection techniques maintain, and then select a model from, a collection of history models [23] or a cascade of models with increasing capacities [31, 57].

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 [21] and AMS [20] 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. [26] 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 [58]. 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 [58] 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 [59]. 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 [60, 61]. 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 [23, 34].

RECL's model selection strategy (§3.1) builds on the literature on gating networks [26], but reduces the delay and compute overhead when adding new expert models. Instead of jointly training the new experts and the gating network [26], 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 [23], RECL's gating network enjoys better algorithmic intuition (see §3.1) and better empirical performance (§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., [37, 62]), inference tasks of video analytics (e.g., [51, 63]), and between inference and training jobs [21]. 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 [51], periodically [21], or by reusing compute data [37].

RECL is a custom design of GPU sharing for continuous learning across many edge devices. Compared to Ekya [21], 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 [37] proposed for large-scale DL clusters.

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.

9 Acknowledgements

We thank the NSDI reviewers and our shepherd, Dongsu Han, for their invaluable feedback. This work was supported in part by NSF grants CNS-1751009, CNS-1955370, CNS-2152313, CNS-2153449, CNS-2147909, and CNS-2140552, as well as gifts from Cisco and the sponsors of MachineLearningApplications@CSAIL program.

References

[1] 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.

[2] 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.

[3] Azure outposts. https://aws.amazon.com/outposts/.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

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

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

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

[15] 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.

[16] 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.

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

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

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

[28] 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.

[29] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

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

[37] 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.

[38] 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.

[39] gRPC. https://grpc.io/about/.

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

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] NVIDIA Jetson Nano. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[47] Coral Edge TPU. https://coral.ai/docs/edgetpu/benchmarks/.

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

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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-model-training/. Accessed: September 2022.

[60] 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.

[61] 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.

[62] 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.

[63] 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.