Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers

Romil Bhardwaj[1,2], Zhengxu Xia[3], Ganesh Ananthanarayanan[1], Junchen Jiang[3], Yuanchao Shu[1], Nikolaos Karianakis[1], Kevin Hsieh[1], Paramvir Bahl[1], and Ion Stoica[2]

[1]Microsoft, [2]UC Berkeley, [3]University of Chicago

Abstract

Video analytics applications use edge compute servers for processing videos. Compressed models that are deployed on the edge servers for inference suffer from *data drift* where the live video data diverges from the training data. Continuous learning handles data drift by periodically retraining the models on new data. Our work addresses the challenge of jointly supporting inference and retraining tasks on edge servers, which requires navigating the fundamental tradeoff between the retrained model's accuracy and the inference accuracy. Our solution Ekya balances this tradeoff across multiple models and uses a micro-profiler to identify the models most in need of retraining. Ekya's accuracy gain compared to a baseline scheduler is 29% higher, and the baseline requires 4× more GPU resources to achieve the same accuracy as Ekya.

1 Introduction

Video analytics applications, such as for urban mobility [2, 5] and smart cars [27], are being powered by deep neural network (DNN) models for object detection and classification, e.g., Yolo [36], ResNet [39] and EfficientNet [61]. Video analytics deployments stream the videos to *edge servers* [14, 15] placed on-premise [13, 38, 81, 84]. Edge computation is preferred for video analytics as it does not require expensive network links to stream videos to the cloud [81], while also ensuring privacy of the videos (e.g., many European cities mandate against streaming their videos to the cloud [11, 87]).

Edge compute is provisioned with limited resources (e.g., with weak GPUs [14, 15]). This limitation is worsened by the mismatch between the growth rate of the compute demands of models and the compute cycles of processors [12, 90]. As a result, edge deployments rely on *model compression* [67, 86, 94]. The compressed DNNs are initially trained on representative data from each video stream, but while in the field, they are affected by *data drift*, i.e., the live video data diverges significantly from the data that was used for training [23, 52, 77, 79]. Cameras in streets and smart cars encounter varying scenes over time, e.g., lighting, crowd densities, and changing object mixes. It is difficult to exhaustively cover all these variations in the training, especially since even subtle variations affect the accuracy. As a result, there is a sizable drop in the accuracy of edge DNNs due to data drift (by 22%; §2.3). In fact, the fewer weights and shallower architectures of compressed DNNs often make them unsuited to provide high accuracy when trained with large variations in the data.

Continuous model retraining. A promising approach to address data drift is continuous learning. The edge DNNs are incrementally *retrained* on new video samples even as some earlier knowledge is retained [28, 83]. Continuous learning techniques retrain the DNNs periodically [72, 93]; we refer to the period between two retrainings as the "retraining window" and use a sample of the data that is accumulated during each window for retraining. Such ongoing learning [42, 89, 96] helps the compressed models maintain high accuracy.

Edge servers use their GPUs [15] for DNN inference on many live video streams (e.g., traffic cameras in a city). Adding continuous training to edge servers presents a tradeoff between the live inference accuracy and drop in accuracy due to data drift. Allocating more resources to the retraining job allows it to finish faster and provide a more accurate model sooner. At the same time, during the retraining, taking away resources from the inference job lowers its accuracy (because it may have to sample the frames of the video to be analyzed).

Central to the resource demand and accuracy of the jobs are their *configurations*. For retraining jobs, configurations refer to the hyperparameters, e.g., number of training epochs, that substantially impact the resource demand and accuracies (§3.1). The improvement in accuracy due to retraining also depends on *how much* the characteristics of the live videos have changed. For inference jobs, configurations like frame sampling and resolution impact the accuracy and resources needed to keep up with analyzing the live video [22, 37].

Problem statement. We make the following decisions for retraining. (1) in each retraining window, decide which of the edge models to retrain; (2) allocate the edge server's GPU resources among the retraining and inference jobs, and (3) select the configurations of the retraining and inference jobs. We also constraint our decisions such that the inference accuracy *at any point in time* does not drop below a minimum value (so that the outputs continue to remain useful to the application). Our objective in making the above three decisions is to maximize the inference accuracy *averaged over the retraining window* (aggregating the accuracies during and after the retrainings). Maximizing inference accuracy over the retraining window creates new challenges as it is different from (i) video inference systems that optimize only the instantaneous accuracy [22, 32, 37], (ii) model training systems that optimize only the eventual accuracy [8, 17, 69, 85, 88, 95].

Addressing the fundamental tradeoff between the retrained model's accuracy and the inference accuracy is computationally complex. First, the decision space is multi-dimensional consisting of a diverse set of retraining and inference configurations, and choices of resource allocations over time. Second, it is difficult to know the performance of different configurations (in resource usage and accuracy) as it requires actually retraining using different configurations. Data drift exacerbates these challenges because a decision that works well in a retraining window may not do so in the future.

Solution components. Our solution Ekya has two main components: a resource scheduler and a performance estimator.

In each retraining window, the **resource scheduler** makes the three decisions listed above in our problem statement. In its decisions, Ekya's scheduler prioritizes retraining the models of those video streams whose characteristics have changed the most because these models have been most affected by data drift. The scheduler decides against retraining the models which do not improve our target metric. To prune the large decision space, the scheduler uses the following techniques. First, it simplifies the spatial complexity by considering GPU allocations only in coarse fractions (e.g., 10%) that are accurate enough for the scheduling decisions, while also being mindful of the granularity achievable in modern GPUs [4]. Second, it does not change allocations to jobs *during the retraining*, thus largely sidestepping the temporal complexity. Finally, our micro-profiler (described below) prunes the list of configurations to only the promising options.

To make efficient choices of configurations, the resource scheduler relies on estimates of accuracy after the retraining and the resource demands. We have designed a **micro-profiler** that observes the accuracy of the retraining configurations on a *small subset* of the training data in the retraining window with *just a few epochs*. It uses these observations to extrapolate the accuracies when retrained on a larger dataset for many more epochs. Further, we restrict the micro-profiling to only a small set of *promising* retraining configurations. These techniques result in Ekya's micro-profiler being $100\times$ more efficient than exhaustive profiling while still estimating accuracies with an error of 5.8%. To estimate the resource demands, the micro-profiler measures the retraining duration *per epoch* when 100% of the GPU is allocated, and scales for different allocations, epochs, and training data sizes.

Implementation and Evaluation. We have evaluated Ekya

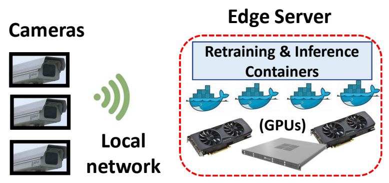


Figure 1: Cameras connect to the edge server, with consumer-grade GPUs for DNN inference and retraining containers.

using a system implementation and trace-driven simulation. We used video workloads from dashboard cameras of smart cars (Waymo [68] and Cityscapes [57]) as well as from traffic and building cameras over 24 hours. Ekya's accuracy compared to competing baselines is 29% higher. As a measure of Ekya's efficiency, attaining the same accuracy as Ekya will require $4\times$ more GPU resources on the edge for the baseline.

Contributions: Our work makes the following contributions.
1) We introduce the metric of *inference accuracy averaged over the retraining window* for continuous training systems.
2) We design an *efficient micro-profiler to estimate* the benefits and costs of retraining edge DNN models.
3) We design a scalable resource scheduler for *joint retraining and inference* on edge servers.
4) We release Ekya's source code and video datasets with 135 hours of videos and corresponding labels to spur future research in continuous learning at the edge. See aka.ms/ekya.

2 Continuous training on edge compute

2.1 Edge Computing for Video Analytics

Video analytics deployments commonly analyze videos on edge servers placed on-premise (e.g., from AWS [14] or Azure [15]). A typical edge server supports tens of video streams [19], e.g., on the cameras in a building, with customized models for each stream [59] (see Figure 1).Video analytics applications adopt edge computing for the following reasons [13, 38, 81].

1) Edge deployments are often in locations where the *uplink network to the cloud is expensive* for shipping continuous video streams, e.g., in oil rigs with expensive satellite network or smart cars with data-limited cellular network. [1]

2) Network links out of the edge locations experience *outages* [76, 81]. Edge compute provides robustness against disconnection to the cloud [26] and prevents disruptions [20].

3) Videos often contain *sensitive and private data* that users do not want sent to the cloud (e.g., many EU cities legally mandate that traffic videos be processed on-premise [11, 87]).

Thus, due to reasons of network cost and video privacy, it is preferred to run both inference and retraining on the edge compute device itself without relying on the cloud. In fact, with bandwidths typical in edge deployments, cloud-based solutions are slower and result in lower accuracies (§6.4).

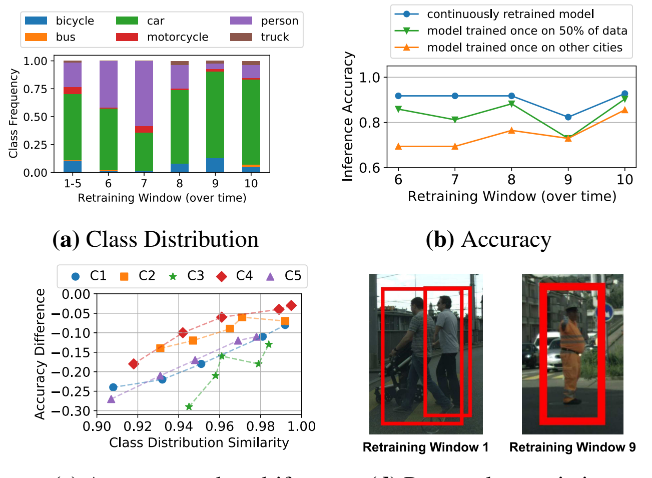
[1]The uplinks of LTE cellular or satellite links is $3-10$Mb/s [58, 65], which can only support a couple of 1080p 30 fps HD video streams whereas a typical deployment has many more cameras [81].

2.2 Compressed DNN Models and Data drift

Advances in computer vision research have led to high-accuracy DNN models that achieve high accuracy with a large number of weights, deep architectures, and copious training data. While highly accurate, using these heavy and general DNNs for video analytics is both expensive and slow [22, 34], which make them unfit for resource-constrained edge computing. The most common approach to addressing the resource constraints on the edge is to train and deploy *specialized and compressed* DNNs [53, 60, 64, 67, 86, 94], which consist of far fewer weights and shallower architectures. For instance, Microsoft's edge video analytics platform [5] uses a compressed DNN (TinyYOLO [75]) for efficiency. Similarly, Google released Learn2Compress[2] for edge devices to automate the generation of compressed models from proprietary models. These compressed DNNs are trained to only recognize the limited objects and scenes specific to each video stream. In other words, to maintain high accuracy, they forego generality for improved compute efficiency [22, 34, 72].

Data drift. As specialized edge DNNs have shallower architectures than general DNNs, they can only memorize limited amount of object appearances, object classes, and scenes. As a result, specialized edge DNNs are particularly vulnerable to *data drift* [23, 52, 77, 79], where live video data diverges significantly from the initial training data. For example, variations in the object pose, scene density (e.g. rush hours), and lighting (e.g., sunny vs. rainy days) over time make it difficult for traffic cameras to accurately identify the objects of interest (cars, bicycles, road signs). Cameras in modern cars observe vastly varying scenes (e.g., building types, crowd sizes) as they move through different neighborhoods and cities. Further, the *distribution* of the objects change over time, which reduces the edge model's accuracy [93, 99]. Due to their ability to memorize limited amount of object variations, edge DNNs have to be continuously updated with recent data and changing object distributions to maintain a high accuracy.

Continuous training. The preferred approach, that has gained significant attention, is for edge DNNs to *continuously learn* as they incrementally observe new samples over time [42, 89, 96]. The high temporal locality of videos allows the edge DNNs to focus their learning on the most recent object appearances and object classes [72, 82]. In `Ekya`, we use a modified version of iCaRL[89] learning algorithm to on-board new classes, as well as adapt to the changing characteristics of the existing classes. Since manual labeling is not feasible for continuous training systems on the edge, the labels for the retraining are obtained from a "golden model" - a highly accurate (87% and 84% accuracy on Cityscapes and Waymo datasets, respectively) but expensive model (deeper architecture with large number of weights). The golden model cannot keep up with inference on the live videos and we use it to label only a small fraction of the videos in the retraining window. Our approach is essentially that of supervising a low-cost "student" model with a high-cost "teacher" model (or knowledge distillation [33]), and this has been broadly applied in computer vision literature [42, 72, 93, 96].



(a) Class Distribution **(b)** Accuracy

(c) Accuracy vs data drift **(d)** Person class variations

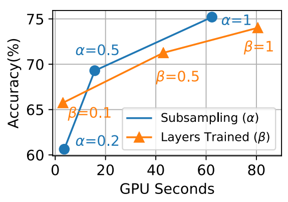
Figure 2: Continuous learning in the Cityscapes dataset. Shift in class distributions (a) across windows necessitates continuous learning (b). Model accuracy is not only affected by class distribution shifts (c), but also by changes in object appearances (d).

2.3 Accuracy benefits of continuous learning

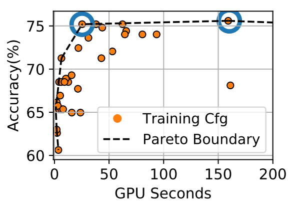
To show the benefits of continuous learning, we use the video stream from one example city in the Cityscapes dataset [57] that consists of videos from dashboard cameras in many cities. In our evaluation in §6, we use both moving dashboard cameras as well as static cameras over long time periods. We divide the video data in our example city into ten fixed *retraining windows* (200s in this example).

Understanding sources of data drift. Figure 2a shows the change of object class distributions across windows. The initial five windows see a fair amount of persons and bicycles, but bicycles rarely show up in windows 6 and 7, while the share of persons varies considerably across windows $6-10$. Figure 2c summarizes the effect of this data drift on model accuracy in five independent video streams, C1-C5. For each stream, we train a baseline model on the first five windows, and test it against five windows in the future and use cosine similarity to measure the class distribution shift for each window. Though accuracy generally improves when the model is used on windows with similar class distributions (high cosine similarity), the relationship is not guaranteed (C2, C3). This is because class distribution shift is not the only form of data drift. Illumination, pose and appearance differences also affect model performance (e.g. clothing and angles for objects in the person class vary significantly; Figure 2d).

Improving accuracy with continuous learning. Figure 2b plots inference accuracy of an edge DNN (a compressed ResNet18 classifier) in the last five windows using different training options. (1) Training a compressed ResNet18 with video data on all other cities of the Cityscapes dataset does not



(a) Effect of Hyperparameters



(b) Resource-accuracy

Figure 3: Measuring retraining configurations. GPU seconds refers to the duration taken for retraining with 100% GPU allocation. (a) varies two example hyperparameters, keeping others constant. Note the Pareto boundary of configurations in (b); for every non-Pareto configuration, there is at least one Pareto configuration that is better than it in *both* accuracy and GPU cost.

result in good performance. (2) Unsurprisingly, we observe that training the edge DNN once using data from the first five windows *of this example city* improves the accuracy. (3) *Continuous retraining* using the most recent data for training achieves the highest accuracy consistently. Its accuracy is higher than the other options by up to 22%.

Interestingly, using the data from the first five windows to train the larger ResNet101 DNN (not graphed) achieves better accuracy than the continuously retrained ResNet18. The substantially better accuracy of ResNet101 compared to ResNet18 when trained *on the same data* of the first five windows also shows that this training data was indeed fairly representative. But the lightweight ResNet18's weights and architecture limits its ability to learn and is a key contributor to its lower accuracy. Nonetheless, ResNet101 is 13× slower than the compressed ResNet18 [21]. This makes the efficient ResNet18 more suited for edge deployments and continuous learning enables it to maintain high accuracy even with data drift. Therefore, the need for continuous training of edge DNNs is ongoing and not just during a "ramp-up" phase.

3 Scheduling retraining and inference jointly

We propose *joint retraining and inference* on edge servers. The joint approach utilizes resources better than statically provisioning compute for retraining. Since retraining is periodic [72, 93] with far higher compute demands than inference, static provisioning causes idling. Compared to uploading videos to the cloud for retraining, our approach has advantages in privacy (§2.1), and network costs and accuracy (§6.4).

3.1 Configuration diversity of retraining and inference

Tradeoffs in retraining configurations. The hyperparameters for retraining, or "retraining configurations", influence the resource demands and accuracy. Retraining fewer layers of the DNN (or, "freezing" more layers) consumes lesser GPU resources, as does training on fewer data samples, but they also produce a model with lower accuracy; Figure 3a.

Figure 3b illustrates the resource-accuracy trade-offs for an edge DNN (ResNet18) with various hyperparameters: number of training epochs, batch sizes, number of neurons in the last

Configuration	Retraining Window 1		Retraining Window 2	
	End Accuracy	GPU seconds	End Accuracy	GPU seconds
Video A Cfg1A	75	85	95	90
Video A Cfg2A (*)	70	65	90	40
Video B Cfg1B	90	80	98	80
Video B Cfg2B (*)	85	50	90	70

Table 1: Hyperparameter configurations for retraining jobs in Figure 4's example. At the start of retraining window 1, camera A's inference model has an accuracy of 65% and camera B's inference model has an accuracy of 50%. Asterisk (*) denotes the configurations picked in Figures 4b and 4d.

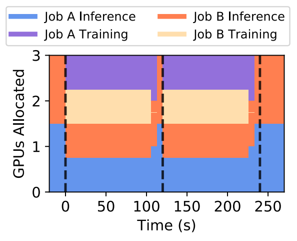
layer, number of frozen layers, and fraction of training data. We make two observations. First, there is a wide spread in the resource usage (measured in GPU seconds), by upto a factor of 200×. Second, higher resource usage does not always yield higher accuracy. For the two configurations circled in Figure 3b, their GPU demands vary by 6× even though their accuracies are the same (∼ 76%). Thus, careful selection of the configurations considerably impacts the resource efficiency. Moreover, the accuracy spread across configurations is dependent on the extent of data-drift. Retraining on visually similar data with little drift results in a narrower spread. With the changing characteristics of videos, it is challenging to efficiently obtain the resource-accuracy profiles for retraining.

Tradeoffs in inference configurations. Inference pipelines also allow for flexibility in their resource demands at the cost of accuracy through configurations to downsize and sample frames [59]. Reducing the resource allocation to inference pipelines increases the processing latency per frame, which then calls for sub-sampling the incoming frames to match the processing rate, that in turn reduces inference accuracy [32]. Prior work has made dramatic advancements in profilers that efficiently obtain the resource-accuracy relationship for *inference configurations* [37]. We use these efficient inference profilers in our solution, and also to ensure that the inference pipelines keep up with analyzing the live video streams.

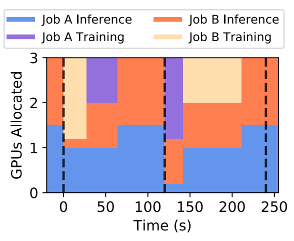
3.2 Illustrative scheduling example

We use an example with 3 GPUs and two video streams, A and B, to show the considerations in scheduling inference and retraining tasks jointly. Each retraining uses data samples accumulated since the *beginning of* the last retraining (referred to as the "retraining window").[2] To simplify the example, we assume the scheduler has knowledge of the resource-accuracy profiles, but these are expensive to get in practice (we describe our efficient solution for profiling in §4.3). Table 1 shows the retraining configurations (Cfg1A, Cfg2A, Cgf1B, and Cgf2B), their respective accuracies after the retraining, and GPU cost.

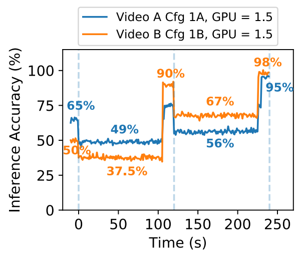
[2]Continuous learning targets retraining windows of tens of seconds to few minutes [72, 93]. We use 120 seconds in this example. Our solution is robust to and works with any given window duration for its decisions (See §6.2).



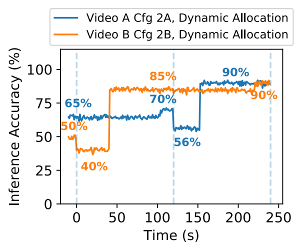
(a) Uniform scheduler



(b) Accuracy-optimal sched.



(c) Uniform scheduler



(d) Accuracy-optimal sched.

Figure 4: Resource allocations (top) and inference accuracies (bottom) over time for two retraining windows (each of 120s**). The left figures show a uniform scheduler which evenly splits the** 3 **GPUs, and picks configurations resulting in the most accurate models. The right figures show the accuracy-optimized scheduler that prioritizes resources and optimizes for inference accuracy averaged over the retraining window** (73% **compared to the uniform scheduler's** 56%**). The accuracy-optimized scheduler also ensures that inference accuracy never drops below a minimum (set to** 40% **in this example, denoted as** a_{MIN}**).**

The scheduler is responsible for selecting configurations and allocating resources for inference and retraining jobs.

Uniform scheduling: Building upon prior work in cluster schedulers [9, 80] and video analytics systems [32], a baseline solution for resource allocation evenly splits the GPUs between video streams, and each stream evenly partitions its allocated GPUs for retraining and inference tasks; see Figure 4a. Just like model training systems [29, 44, 45], the baseline always picks the configuration for retraining that results in the highest accuracy (Cfg1A, Cfg1B for both windows).

Figure 4c shows the *inference* accuracies of the two live streams. We see that when the retraining tasks take resources away from the inference tasks, the inference accuracy drops significantly (65% $\rightarrow$ 49% for video A and 50% $\rightarrow$ 37.5% for video B in Window 1). While the inference accuracy increases *after* retraining, it leaves too little time in the window to reap the benefit of retraining. Averaged across both retraining windows, the inference accuracy across the two video streams is only 56% because the gains due to the improved accuracy of the retrained model are undercut by the time taken for retraining (during which inference accuracy suffered).

Accuracy-optimized scheduling: Figures 4b and 4d illustrate an accuracy-optimized scheduler, which by taking a holistic view on the multi-dimensional tradeoffs, provides an an average inference accuracy of 73%. In fact, to match the accuracies, the above uniform scheduler would require nearly twice the GPUs (i.e., 6 GPUs instead of 3 GPUs).

This scheduler makes three key improvements. First, the scheduler selects the hyperparameter configurations based on their accuracy improvements *relative* to their GPU cost. It selects lower accuracy options (Cfg2A/Cfg2B) instead of the higher accuracy ones (Cfg1A/Cfg1B) because these configurations are substantially cheaper (Table 1). Second, the scheduler *prioritizes* retraining tasks that yield higher accuracy, so there is more time to reap the benefit from retraining. For example, the scheduler prioritizes B's retraining in Window 1 as its inference accuracy after retraining increases by 35% (compared to 5% for video A). Third, the scheduler controls the accuracy drops during retraining by balancing the retraining time and the resources taken away from inference.

4 Ekya: Solution Description

Continuous training on limited edge resources requires smartly deciding when to retrain each video stream's model, how much resources to allocate, and what configurations to use. Making these decisions presents two challenges.

First, the decision space of multi-dimensional configurations and resource allocations is computationally more complex than two fundamentally challenging problems of multi-dimensional knapsack and multi-armed bandits (§4.1). Hence, we design a **thief scheduler** (§4.2), a heuristic that makes the joint retraining-inference scheduling tractable in practice.

Second, the scheduler requires the model's exact performance (in resource usage and inference accuracy), but this requires retraining using all the configurations. We address this challenge with our **micro-profiler** (§4.3), which retrains only a few select configurations on a fraction of the data. Figure 5 presents an overview of `Ekya`'s components.

4.1 Formulation of joint inference and retraining

The problem of joint inference and retraining aims to maximize overall inference accuracy for all video streams $\mathcal{V}$ in a retraining window T with duration $\|T\|$. All work must be done in $\mathcal{G}$ GPUs. Thus, the total compute capability is $\mathcal{G}\|T\|$ GPU-time. Without loss of generality, let δ be the smallest granularity of GPU allocation. Each video $v \in V$ has a set of *retraining* configurations Γ and a set of *inference* configurations Λ (§3.1). Table 4 (§A) lists the notations.

Decisions. For each video $v \in \mathcal{V}$ in a window T, we decide: (1) the retraining configuration $\gamma \in \Gamma$ ($\gamma = \emptyset$ means no retraining); (2) the inference configuration $\lambda \in \Lambda$; and (3) how many GPUs (in multiples of δ) to allocate for retraining ($\mathcal{R}$) and inference (I). We use binary variables $\phi_{v\gamma\lambda\mathcal{R}I} \in \{0,1\}$ to denote these decisions (see Table 4 §A for the definition). These decisions require $C_T(v,\gamma,\lambda)$ GPU-time and yields overall accuracy of $A_T(v,\gamma,\lambda,\mathcal{R},I)$. $A_T(v,\gamma,\lambda,\mathcal{R},I)$ is averaged across the window T (§3.2), and the above decisions determine the inference accuracy at *each point in time*.

Optimization. Maximize the inference accuracy averaged

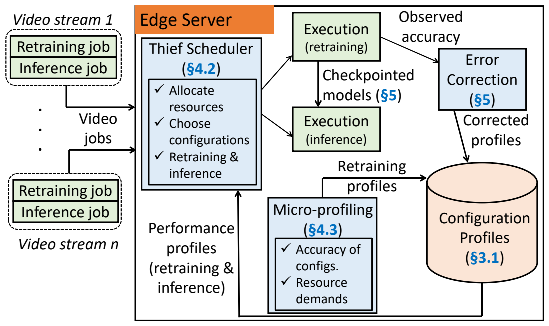


Figure 5: Ekya's components and their interactions.

across all videos in a retraining window within the GPU limit.

$$\arg\max_{\phi_{v\gamma\lambda\mathcal{R}I}} \frac{1}{\|\mathcal{V}\|} \sum_{\substack{\forall v\in\mathcal{V},\forall\gamma\in\Gamma,\forall\lambda\in\Lambda,\\ \forall\mathcal{R},\forall I\in\{0,1,\ldots,\frac{\mathcal{G}}{\delta}\}}} \phi_{v\gamma\lambda\mathcal{R}I}\cdot A_T(v,\gamma,\lambda,\mathcal{R},I)$$

subject to

$$\begin{aligned}
&1. \sum_{\substack{\forall v\in\mathcal{V},\forall\gamma\in\Gamma,\forall\lambda\in\Lambda,\\ \forall\mathcal{R},\forall I}} \phi_{v\gamma\lambda\mathcal{R}I}\cdot C_T(v,\gamma,\lambda) \le \mathcal{G}\|T\| \\
&2. \sum_{\substack{\forall v\in\mathcal{V},\forall\gamma\in\Gamma,\forall\lambda\in\Lambda,\\ \forall\mathcal{R},\forall I}} \phi_{v\gamma\lambda\mathcal{R}I}\cdot(\mathcal{R}+I) \le \frac{\mathcal{G}}{\delta} \\
&3. \sum_{\substack{\forall\gamma\in\Gamma,\forall\lambda\in\Lambda,\\ \forall\mathcal{R},\forall I}} \phi_{v\gamma\lambda\mathcal{R}I} \le 1, \forall v\in\mathcal{V}
\end{aligned} \qquad (1)$$

The first constraint ensures that the GPU allocation does not exceed the available GPU-time $\mathcal{G}\|T\|$ in the retraining window. The second constraint limits the *instantaneous* allocation (in multiples of δ) to never exceed the available GPUs. The third constraint ensures that at most one configuration is picked for retraining and inference each for a video v.

Our analysis in §A.1 shows that the above optimization problem is *harder* than the multi-dimensional binary knapsack problem and modeling the uncertainty of $A_T(v,\gamma,\lambda,\mathcal{R},I)$ is more challenging than the multi-armed bandit problem.

4.2 Thief Scheduler

Our scheduling heuristic makes the scheduling problem tractable by decoupling resource allocation (i.e., $\mathcal{R}$ and I) and configuration selection (i.e., γ and λ) (Algorithm 1). We refer to Ekya's scheduler as the "thief" scheduler and it iterates among all inference and retraining jobs as follows.

(1) It starts with a fair allocation for all video streams $v \in V$ (line 2 in Algorithm 1). In each step, it iterates over all the inference and retraining jobs of each video stream (lines 5-6), and *steals* a tiny quantum Δ of resources (in multiples of δ; see Table 4, §A) from each of the other jobs (lines 10-11).

(2) With the new resource allocations (temp_alloc[]), it then selects configurations for the jobs using the PickConfigs method (line 14 and Algorithm 2, §A) that iterates over all the configurations for inference and retraining for each video stream. For inference jobs, among all the configurations whose accuracy is $\ge a_{\text{MIN}}$, PickConfigs picks the configuration with the highest accuracy that can keep up with the inference of the live video stream given current allocation (line 3-4, Algorithm 2, §A).

For retraining jobs, PickConfigs picks the configuration that maximizes the accuracy $A_T(v,\gamma,\lambda,\mathcal{R},I)$ over the retraining window for each video v (lines 6-12, Algorithm 2, §A). EstimateAccuracy (line 7, Algorithm 2, §A) aggregates the instantaneous accuracies over the retraining window for a given pair of inference configuration (chosen above) and retraining configuration. Ekya's micro-profiler (§4.3) provides the estimate of the accuracy and the time to retrain for a retraining configuration when 100% of GPU is allocated, and EstimateAccuracy proportionately scales the GPU-time for the current allocation (in temp_alloc[]) and training data size. In doing so, it avoids configurations whose retraining durations exceed $\|T\|$ with the current allocation (first constraint in Eq. 1).

(3) After reassigning the configurations, Ekya uses the estimated average inference accuracy (accuracy_avg) over the retraining window (line 14 in Algorithm 1) and keeps the new allocations only if it improves up on the accuracy from prior to stealing the resources (line 15 in Algorithm 1).

The thief scheduler repeats the process till the accuracy stops increasing (lines 15-20 in Algorithm 1) and until all the jobs have played the "thief". Algorithm 1 is invoked at the beginning of each retraining window, as well as on the completion of every training job during the window.

Design rationale: We call out the key aspects that makes the scheduler's decision efficient by pruning the search space.

- *Coarse allocations:* The thief scheduler allocates GPU resources in quantums of Δ. Intuitively, Δ is the step size for allocation used by the scheduler. Thus, the final resource allocation from the thief scheduler is within Δ of the optimal allocation. We empirically pick a Δ that is coarse yet accurate enough in practice, while being mindful of modern GPUs[4]; see §6.2. Algorithm 1 ensures that the total allocation is within the limit (second constraint in Eq 1).
- *Reallocating resources only when a retraining job completes:* Although one can reallocate GPU resource among jobs at finer temporal granularity (*e.g.,* whenever a retraining job has reached a high accuracy), we empirically find that the gains from such complexity is marginal.
- *Pruned configuration list:* Our micro-profiler (described next) speeds up the thief scheduler by giving it only the more promising configurations. Thus, the list Γ used in Algorithm 1 is significantly smaller than the exhaustive set.

4.3 Performance estimation with micro-profiling

Ekya's scheduling decisions in §4.2 rely on estimations of post-retraining accuracy and resource demand of the retraining configurations. Specifically, at the beginning of each retraining window T, we need to *profile* for each video v and each configuration $\gamma \in \Gamma$, the accuracy after retraining using γ and the corresponding time taken to retrain.

Profiling in Ekya vs. hyperparameter tuning: While Ekya's profiling may look similar to hyperparameter tuning

Algorithm 1: Thief Scheduler.

```
Data: Training ($\Gamma$) and inference ($\Lambda$) configurations
Result: GPU allocations $\mathcal{R}$ and $I$, chosen configurations
        ($\gamma \in \Gamma$, $\lambda \in \Lambda$) $\forall v \in V$
1  all_jobs[] = Union of inference and training jobs of videos $V$;
   /* Initialize with fair allocation */
2  best_alloc[] = fair_allocation(all_jobs);
3  best_configs[], best_accuracy_avg = PickConfigs(best_alloc);
   /* Thief resource stealing */
4  for thief_job in all_jobs[] do
5      for victim_job in all_jobs[] do
6          if thief_job == victim_job then continue;
7          temp_alloc[] $\leftarrow$ best_alloc[];
8          while true do
               /* $\Delta$ is the increment of stealing */
9              temp_alloc[victim_job] $-= \Delta$;
10             temp_alloc[thief_job] $+= \Delta$;
11             if temp_alloc[victim_job] < 0 then break ;
               /* Calculate accuracy over retraining
                  window and pick configurations. */
12             temp_configs[], accuracy_avg =
                 PickConfigs(temp_alloc[]);
13             if accuracy_avg > best_accuracy_avg then
14                 best_alloc[] = temp_alloc[];
15                 best_accuracy_avg = accuracy_avg;
16                 best_configs[] = temp_configs[];
17             else
18                 break;
19 return best_alloc[], best_configs[];
```

(e.g., [46, 48, 62, 85]) at first blush, there are two key differences. First, Ekya needs the performance estimates of a broad set of candidate configurations for the thief scheduler, not just of the single best configuration, because the best configuration is jointly decided across the many retraining and inference jobs. Second, in contrast to hyperparameter tuning which runs separately of the eventual inference/training, Ekya's profiling must share compute resource with all retraining and inference.

Opportunities: Ekya leverages three empirical observations for efficient profiling of the retraining configurations. (*i*) Resource demands of the configurations are deterministic. Hence, we measure the GPU-time taken to retrain for *each epoch* in the current retraining window when 100% of the GPU is allocated to the retraining. This GPU-time must then be re-scaled for varying number of epochs, GPU allocations, and training data sizes in Algorithm 1. For re-scaling number of epochs and training data sizes, we linearly scale the GPU-time. For re-scaling GPU allocations, we use an offline computed profile of the model throughput for different resource allocations to account for sub-linear scaling. Our real testbed-based evaluation shows that these rescaling functions works well in practice. (*ii*) Post-retraining accuracy can be roughly estimated by training on a small subset of training data for a handful of epochs. (*iii*) The thief scheduler's decisions are not impacted by small errors in the estimations.

Micro-profiling design: The above insights inspired our approach, called *micro-profiling*, where for each video, we test the retraining configurations on a *small subset* of the retraining data and only for a *small number* of epochs (well before models converge). Our micro-profiler is 100$\times$ more efficient than exhaustive profiling (of all configurations on the entire training data), while predicting accuracies with an error of 5.8%, which is low enough in practice to *mostly* ensure that the thief scheduler makes the same decisions as it would with a fully accurate prediction. We use these insights to now explain the techniques that make Ekya's micro-profiling efficient.

1) Training data sampling: Ekya's micro-profiling works on only a small fraction (say, 5% – 10%) of the training data in the retraining window (which is already a subset of all the videos accumulated in the retraining window). While we considered weighted sampling techniques for the micro-profiling, we find that uniform random sampling is the most indicative of the configuration's performance on the full training data, since it preserves all the data distributions and variations.

2) Early termination: Similar to data sampling, Ekya's micro-profiling only tests each configuration for a small number (say, 5) of training epochs. Compared to a full fledged profiling that needs few tens of epochs to converge, such early termination greatly speeds up the micro-profiling process.

After early termination on the sampled training data, we obtain the (validation) accuracy of each configuration at each epoch it was trained. We then fit the accuracy-epoch points to the a non-linear curve model from [70] using a non-negative least squares solver [6]. This model is then used to extrapolate the accuracy that would be obtained by retraining with all the data for larger number of epochs. The use of this extrapolation is consistent with similar work in this space [55, 70].

3) Pruning bad configurations: Finally, Ekya's micro-profiling also prunes out those configurations for micro-profiling (and hence, for retraining) that have historically not been useful. These are configurations that are significantly distant from the configurations on the Pareto curve of the resource-accuracy profile (see Figure 3b), and thus unlikely to be picked by the thief scheduler. To bootstrap pruning, all configurations are evaluated in the first window. After every 2 windows, a fixed fraction of the worst performing configurations are dropped. While first few retraining windows must explore a big space of configurations, the search space size drops exponentially over time. Avoiding these configurations improves the efficiency of the micro-profiling.

Annotating training data: For both the micro-profiling as well as the retraining, Ekya acquires labels using a "golden model" (§2.2). This is a high-cost but high-accuracy model trained on a large dataset. As explained in §2, the golden model cannot keep up with inference on the live videos and we use it to label only a small subset of the videos for retraining. The delay of annotating training data with the golden model is accounted by the scheduler as follows: we subtract the data annotation delay from the retraining window and only pass the remaining time of the window to Algorithm 2 (§A).

5 Implementation and Experimental Setup

Implementation: Ekya uses PyTorch [66] for running and training ML models, and each component is implemented as a collection of long-running processes with the Ray[63] actor model. The micro-profiler and training/inference jobs run as independent actors which are controlled by the thief scheduler actor. Ekya achieves fine-grained and dynamic reallocation of GPU between training and inference processes using Nvidia MPS [4], which provides resource isolation within a GPU by intercepting CUDA calls and rescheduling them. Our implementation also adapts to errors in profiling by reactively adjusting its allocations if the actual model performance diverges from the predictions of the micro-profiler. Ekya's code and datasets are available at the project page: aka.ms/ekya

Datasets: We use both on-road videos captured by dashboard cameras as well as urban videos captured by mounted cameras. The dashboard camera videos are from cars driving through cities in the US and Europe, Waymo Open [68] (1000 video segments with in total 200K frames) and Cityscapes [57] (5K frames captured by 27 cameras) videos. The urban videos are from stationary cameras mounted in a building ("Urban Building") as well as from five traffic intersections ("Urban Traffic"), both collected over 24-hour durations. We use a retraining window of 200 seconds in our experiments, and split each of the videos into 200 second segments. Since the Waymo and Cityscapes dataset do not contain continuous timestamps, we create retraining windows by concatenating images from the same camera in chronological order to form a long video stream and split it into 200 second segments.

DNNs: We demonstrate Ekya's effectiveness on two machine learning tasks – object classification and object detection – using multiple compressed edge DNNs for each task: (*i*) object classification using ResNet18[39], MobileNetV2[53] and ShuffleNet[98], and (*ii*) object detection using TinyYOLOv3[75] and SSD[49]. As explained in §2.2, we use an expensive golden model (ResNeXt 101 [91] for object classification and YOLOv3 [75] for object detection) to get ground truth labels for training and testing.

Testbed and trace-driven simulator: We run Ekya's implementation on AWS EC2 p3.2xlarge instances for 1 GPU experiments and p3.8xlarge for 2 GPU experiments. Each instance has Nvidia V100 GPUs with NVLink interconnects.

We also built a simulator to test Ekya under a wide range of resource constraints, workloads, and longer durations. The simulator takes as input the accuracy and resource usage (in GPU time) of training/inference configurations logged from our testbed. For each training job, we log the accuracy over GPU-time. We also log the inference accuracy on the real videos. This exhaustive trace allows us to mimic the jobs with high fidelity under different scheduling policies.

Retraining configurations: Our retraining configurations combine the number of epochs to train, batch size, number of neurons in the last layer, number of layers to retrain, and the fraction of data between retraining windows to use for retraining (§3.1). For the object detection models (TinyYOLO and SSDLite), we set the batch size to 8 and the fraction of layers frozen between 0.7 and 0.9. The resource requirements of the configurations for the detection models vary by 153×.

Baselines: Our baseline, called *uniform scheduler*, uses (*a*) a fixed retraining configuration, and (*b*) a static retraining/inference resource allocation (these are adopted by prior schedulers [9, 32, 80]). For each dataset, we test all retraining configurations on a hold-out dataset [3] (*i.e.,* two video streams that were never used in later tests) to produce the Pareto frontier of the accuracy-resource tradeoffs (*e.g.,* Figure 3). The uniform scheduler then picks two points on the Pareto frontier as the fixed retraining configurations to represent "high" (Config 1) and "low" (Config 2) resource usage, and uses one of them for all retraining windows in a test.

We also consider two alternatives in §6.4. (1) *offloading retraining to the cloud*, and (2) *caching and re-using a retrained model* from history based on various similarity metrics.

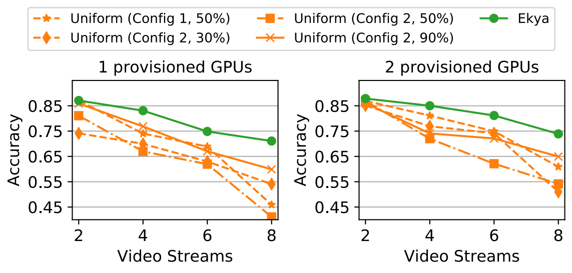
6 Evaluation

We evaluate Ekya's performance, and the key findings are:
1) Compared to static retraining baselines, Ekya achieves upto 29% higher accuracy for compressed vision models in both classification and detection. For the baseline to match Ekya's accuracy, it would need 4× additional GPU resources. (§6.1)
2) Both micro-profiling and thief scheduler contribute sizably to Ekya's gains. (§6.2) In particular, the micro-profiler estimates accuracy with low median errors of 5.8%. (§6.3)
3) The thief scheduler efficiently makes its decisions in 9.4s when deciding for 10 video streams across 8 GPUs with 18 configurations per model for a 200s retraining window. (§6.2)
4) Compared to alternate designs, including reusing cached history models trained on similar data/scenarios as well as retraining the models in the cloud, Ekya achieves significantly higher accuracy without the network costs (§6.4).

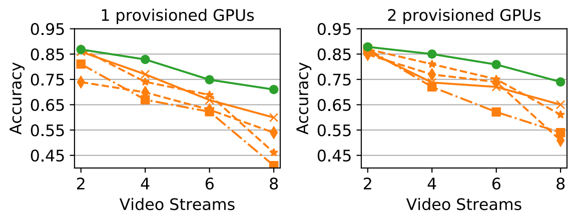
6.1 Overall improvements

We evaluate Ekya and the baselines along three dimensions—*inference accuracy* (% of images correctly classified for object classification, F1 score (measured at a 0.3 threshold for the Intersection-over-Union of the bounding box) for detection), *resource consumption* (in GPU time), and *capacity* (the number of concurrently processed video streams). Note that the evaluation always keeps up with the video frame rate (*i.e.,* no indefinite frame queueing). By default we evaluate the performance of Ekya on ResNet18 models, but we also show that it generalizes to other model types and vision tasks.

[3]The same hold-out dataset is used to customize the off-the-shelf DNN inference model. This is a common strategy in prior work (*e.g.,* [22]).



(a) Cityscapes



(b) Waymo

Figure 6: Effect of adding video streams on accuracy with different schedulers. When more video streams share resources, Ekya's accuracy gracefully degrades while the baselines' accuracy drops faster. ("Uniform (Cfg 1, 90%)" means the uniform scheduler allocates 90% GPU to inference, 10% to retraining)

Accuracy vs. Number of concurrent video streams: Figure 6 shows the ResNet18 model's accuracy with Ekya and the baselines when analyzing a growing number of concurrent video streams under a fixed number of provisioned GPUs for Waymo and Cityscapes datasets. The uniform baselines use different combinations of pre-determined retraining configurations and resource partitionings. For consistency, the video streams are shuffled and assigned an id (0-10), and are then introduced in the same increasing order of id in all experiments. This ensures that different schedulers tested for k parallel streams use the same k streams, and these k streams are always a part of any k' streams ($k' > k$) used for testing.

As the number of video streams increases, Ekya enjoys a growing advantage (upto 29% under 1 GPU and 23% under 2 GPU) in accuracy over the uniform baselines. This is because Ekya gradually shifts more resource from retraining to inference and uses cheaper retraining configurations. In contrast, increasing the number of streams forces the uniform baseline to allocate less GPU cycles to each inference job, while retraining jobs, which use fixed configurations, slow down and take the bulk of each window.

Generalizing to other ML models: Ekya's thief scheduler can be readily applied to any ML model and task (e.g., classification or detection) that needs to be fine-tuned continuously on newer data. To demonstrate this, we evaluate Ekya with:

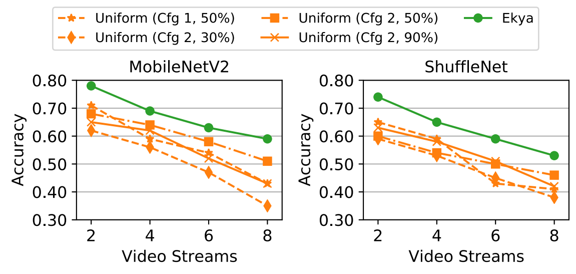
- *Other object classifiers:* Figure 7a shows the performance of Ekya when running MobileNetV2 and ShuffleNet as the edge models in two independent setups for object classification at the edge. Continuing the trend that we observed for ResNet18 (in Figure 6), Figure 7a shows that Ekya leads to up to 22% better accuracy than uniform baselines.
- *Object detection models:* In addition to object classification, we also evaluate using object detection tasks which detect the bounding boxes of objects in the video stream. Figure 7b shows Ekya outperforms the uniform baseline's F1 score by 19% when processing same number of concurrent video streams. Importantly, Ekya's design broadly applies to new tasks without any systemic changes.

These gains stem from Ekya's ability to navigate the rich resource-accuracy space of models by carefully selecting training and inference hyperparameters (e.g., the width multiplier in MobileNetV2, convolution sparsity in ShuffleNet). For the rest of our evaluation, we only present results with ResNet18 though the observations hold for other models.

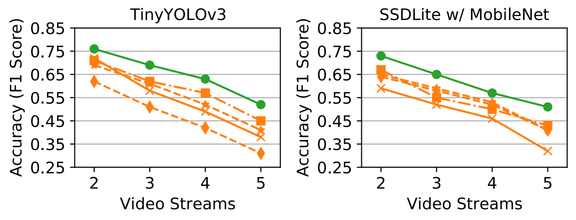
Number of video streams vs. provisioned resource: We compare Ekya's *capacity* (defined by the maximum number of concurrent video streams subject to an accuracy threshold) with that of uniform baseline, as more GPUs are available. Setting an accuracy threshold is common in practice, since applications usually require accuracy to be above a threshold for the inference to be usable. Table 2 uses the Cityscapes results (Figure 6) to derive the scaling factor of capacity vs.

Scheduler	Capacity		Scaling factor
	1 GPU	2 GPUs	
Ekya	**2**	**8**	**4x**
Uniform (Config 1, 50%)	2	2	1x
Uniform (Config 2, 90%)	2	4	2x
Uniform (Config 2, 50%)	2	4	2x
Uniform (Config 2, 30%)	0	2	-

Table 2: Capacity (number of video streams that can be concurrently supported subject to accuracy target 0.75) vs. number of provisioned GPUs. Ekya scales better than the uniform baselines with more available compute resource.

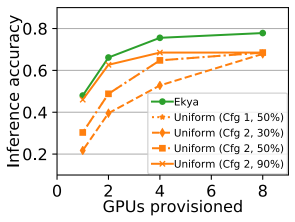


(a) Generalize across object classification models

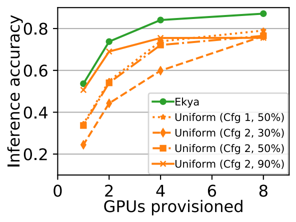


(b) Object Detection Models

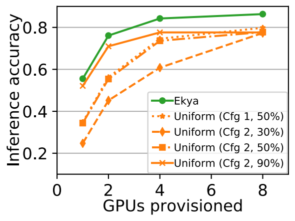
Figure 7: Improvement of Ekya extends to two more compressed DNN classifiers and two popular object detectors.



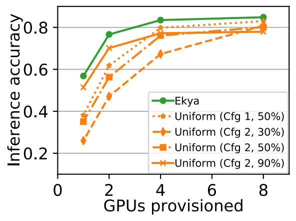
(a) Cityscapes



(b) Waymo

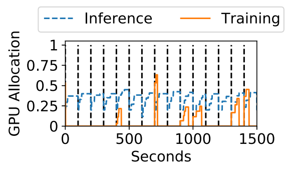


(c) Urban Building

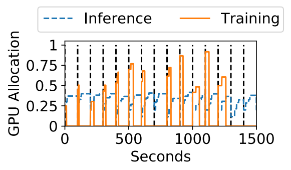


(d) Urban Traffic

Figure 8: Inference accuracy of different schedulers when processing 10 video streams under varying GPU provisionings.



(a) Video stream #1
(Inference accuracy = 0.82)



(b) Video stream #2
(Inference accuracy = 0.83)

Figure 9: `Ekya`'s resource allocation to two video streams over time. `Ekya` adapts when to retrain each stream's model and allocates resource based on the retraining benefit to each stream.

the number of provisioned GPUs and shows that with more provisioned GPUs, `Ekya` scales faster than uniform baselines.

Accuracy vs. provisioned resource: Finally, Figure 8 stress-tests `Ekya` and the uniform baselines to process 10 concurrent video streams and shows their average inference accuracy under different number of GPUs. To scale to more GPUs, we use the simulator (§5), which uses profiles recorded from real tests and we verified that it produced similar results as the implementation at small-scale. As we increase the number of provisioned GPUs, we see that `Ekya` consistently outperforms the best of the two baselines by a considerable margin and more importantly, with 4 GPUs `Ekya` achieves higher accuracy (marked with the dotted horizontal line) than the baselines at 16 GPUs (*i.e.,* 4× resource saving).

The above results show that `Ekya` is more beneficial when there is high contention for the GPU on the edge. Under low contention, the room for improvement shrinks. Contention is, however, common in the edge since the resources are tightly provisioned to minimize their idling.

6.2 Understanding Ekya's improvements

Resource allocation across streams: Figure 9 shows `Ekya`'s resource allocation across two example video streams over several retraining windows. In contrast to the uniform baselines that use the same retraining configuration and allocate equal resource to retraining and inference (when retraining takes place), `Ekya` retrains the model only when it benefits and allocates different amounts of GPUs to the retraining jobs of video streams, depending on how much accuracy gain is expected from retraining on each stream. In this case, more resource is diverted to video stream #1 (#1 can benefit more from retraining than #2) and both video streams achieve much higher accuracies (0.82 and 0.83) than the uniform baseline.

Component-wise contribution: Figure 10a understands the contributions of resource allocation and configuration selection (on 10 video streams with 4 GPUs provisioned). We construct two variants from `Ekya`: *Ekya-FixedRes*, which removes the smart resource allocation in `Ekya` (*i.e.,* using the inference/training resource partition of the uniform baseline), and *Ekya-FixedConfig* removes the microprofiling-based configuration selection in `Ekya` (*i.e.,* using the fixed configuration of the uniform baseline). Figure 10a shows that both adaptive resource allocation and configuration selection has a substantial contribution to `Ekya`'s gains in accuracy, especially when constrained (i.e., fewer resources are provisioned).

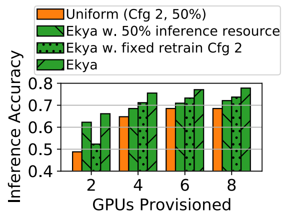
Retraining window sensitivity analysis: Figure 10b evaluates the sensitivity of `Ekya` to the retraining window size. `Ekya` is robust to different retraining window sizes. When the retraining window size is too small (10 seconds), the accuracy of `Ekya` is equivalent to no retraining accuracy due to insufficient time and resources for retraining. As the window increases, `Ekya`'s performance quickly ramps up because the thief scheduler is able to allocate resources to retraining. As the retraining window size further increases Ekya's performance slowly starts moderately degrading because of the inherent limitation in capacity of compressed models (§2.3).

Impact of scheduling granularity: A key parameter in `Ekya`'s scheduling algorithm (§4.2) is the allocation quantum Δ: it controls the runtime of the scheduling algorithm and the granularity of resource allocation. In our sensitivity analysis with 10 video streams, we see that increasing Δ from 1.0 (coarse-grained; one full GPU) to 0.1 (fine-grained; fraction of a GPU), increases the accuracy substantially by $\sim 8\%$. Though the runtime also increases to 9.5 seconds, it is still a tiny fraction (4.7%) of the retraining window (200s).

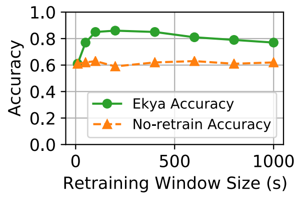
6.3 Effectiveness of micro-profiling

The absolute cost of micro-profiling is small; for our experiments, micro-profiling takes 4.4 seconds for a 200s window.

Errors of microprofiled accuracy estimates: `Ekya`'s micro-profiler estimates the accuracy of each configuration (§4.3) by training it on a subset of the data for a small number of epochs. To evaluate the micro-profiler's estimates, we run it on all configurations for 5 epochs and on 10% of the retraining data from all streams of the Cityscapes dataset, and calculate the estimation error against the retrained accuracies when trained

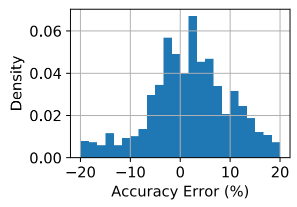


(a) Factor analysis

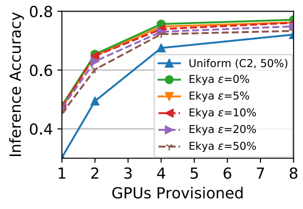


(b) Sensitivity to retraining window size.

Figure 10: (a) Component-wise impact of removing dynamic resource allocation (50% allocation) or removing retraining configuration adaptation (fixed Cfg 2). (b) Robustness of Ekya to a wide range of retraining window values.



(a) Distribution of accuracy estimation errors.



(b) Impact of an controlled error ε to accuracy estimates.

Figure 11: Evaluation of microprofiling performance. (a) shows the distribution of microprofiling's actual estimation errors, and (b) shows the robustness of Ekya's performance against microprofiling's estimation errors.

on 100% of the data for 5, 15 and 30 epochs. Figure 11a plots the distribution of the errors in accuracy estimation and and show that the micro-profiled estimates are largely unbiased with an median absolute error of 5.8%.

Sensitivity to microprofiling estimation errors: Finally, we test the impact of accuracy estimation errors (§4.3) on Ekya. We add gaussian noise on top of the predicted retraining accuracy when the microprofiler is queried. Figure 11b shows that Ekya is robust to accuracy estimate errors: with upto 20% error (which covers all errors in Figure 11a) in the profiler prediction, the maximum accuracy drop is 3%.

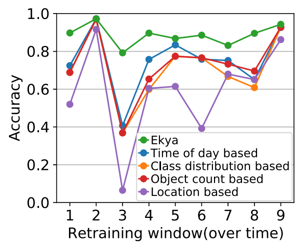
6.4 Comparison with alternative designs

Ekya vs. Cloud-based retraining: One may upload a subsampled video stream to the cloud, retrain the model, and download the model back to the edge [40]. While this solution is not an option for many deployments due to legal and privacy stipulations [11, 87], we still evaluate this option as it lets the edge servers focus on inference. Cloud-based solutions, however, results in lower accuracy due to significant network delays on the constrained networks typical of edges [81].

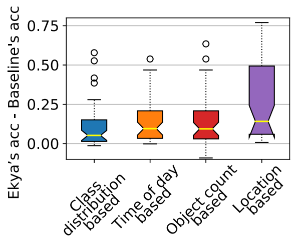
For example, consider 8 video streams running ResNet18 and a retraining window of 400 seconds. A HD (720p) video stream at 4Mbps and 10% data sub-sampling (typical in our experiments) amounts to 160Mb of training data per camera per window. Uploading 160Mb for each of the 8 cameras over

	Bandwidth (Mbps)		Acc.	Bandwidth Gap	
	Uplink	Downlink		Uplink	Downlink
Cellular	5.1	17.5	68.5%	10.2×	3.8×
Satellite	8.5	15	69.2%	5.9×	4.4×
Cellular (2×)	10.2	35	71.2%	5.1×	1.9×
Ekya	-	-	**77.8%**	-	-

Table 3: Retraining in the cloud under different networks [58, 65, 81] versus using Ekya at the edge. Ekya achieves better accuracy without using expensive satellite and cellular links.



(a) Ekya vs. re-using cached models over time



(b) Average gains in accuracy across video streams

Figure 12: Ekya vs. re-using cached models. Compared to cached-model selection techniques, models retrained with Ekya maintain a consistently high accuracy, since it fully leverages the latest training data and is thus more robust to data-drift.

a 4G uplink (5.1 Mbps [65]) and downloading the trained ResNet18 models (398 Mb each [7]) over the 17.5 Mbps downlink [65] takes 432 seconds (even excluding the model retraining time), which already exceeds the retraining window.

To test on the Cityscapes dataset, we extend our simulator (§5) to account for network delays during retraining, and test with 8 videos and 4 GPUs. We use the conservative assumption that retraining in the cloud is "instantaneous" (cloud GPUs are powerful than edge GPUs). Table 3 lists the accuracies with cellular 4G links (both one and two subscriptions to meet the 400s retraining window) and a satellite link, which are both indicative of edge deployments [81].

For the cloud alternatives to match Ekya's accuracy, we will need to provision additional uplink capacity of 5×-10× and downlink capacity of 2×-4× (of the already expensive links). In summary, Ekya's edge-based solution is better than a cloud alternate for retraining in *both* accuracy and network usage (Ekya sends no data out of the edge), all while providing privacy for the videos. However, when the edge-cloud network has sufficient bandwidth, e.g., in an enterprise that is provisioned with a private leased connection, then using the cloud to retrain the models can be a viable design choice.

Ekya vs. Re-using pretrained models: An alternative to continuous retraining is to cache *pretrained* models and reuse them. We pre-train and cache a few tens of DNNs from earlier windows of the Waymo dataset and test four heuristics for selecting cached models. *Class-distribution*-based selection picks the cached DNN whose training data class distribution has the closest Euclidean distance with the current window's data. *Time-of-day*-based selection picks the cached

DNN whose training data time matches the current window. *Object-count*-based selection picks the cached DNN based on similar count of objects. *Location*-based selection picks the cached DNNs trained on the same city as the current window.

Figure 12a highlights the advantages of Ekya over different model selection schemes. We find that since time-of-day-based, object-count-based, and location-based model selection techniques are agnostic to the class distributions of training data of cached models, the selected cached models sometimes do not cover all classes in the current window. Even if we take class distribution into account when picking cached models, there are still substantial discrepancies in the appearances of objects between the current window and the history training data. For instance, object appearance can vary due to pose variations, occlusion or different lighting conditions. In Window 3 (Figure 12a), not only are certain classes underrepresented in the training data, but the lighting conditions are also adverse. Figure 12b presents a box plot of the accuracy difference between Ekya and model selection schemes, where the edges of the box represent the first and third quartiles, the waist is the median, the whiskers represent the maximum and minimum values and the circles represent any outliers. Ekya's continuous retraining of models is robust to scene specific data-drifts and achieves upto 26% higher mean accuracy.

7 Limitations and Discussion

Edge hierarchy with heterogeneous hardware. While Ekya's allocates GPU resources on a single edge, in practice, deployments typically consist of a *hierarchy* of edge devices [19]. For instance, 5G settings include an on-premise edge cluster, followed by edge compute at cellular towers, and then in the core network of the operator. The compute resources, hardware (e.g., GPUs, Intel VPUs [1], and CPUs) and network bandwidths change along the hierarchy. Thus, Ekya will have to be extended along two aspects: (a) multi-resource allocation to include both compute and the network in the edge hierarchy; and (b) heterogeneity in edge hardware.

Privacy of video data. As explained in §2.1, privacy of videos is important in real-world deployments, and Ekya's decision to retrain only on the edge device is well-suited to achieving privacy. However, when we extend Ekya to a hierarchy of edge clusters, care has to be taken to decide the portions of the retraining that can happen on edge devices that are *not* owned by the enterprise. Balancing the need for privacy with resource efficiency is a subject for future work.

Generality beyond vision workloads. Ekya's thief scheduler is generally applicable to DNN models since it only requires that the resource-accuracy function be strictly increasing wherein allocation of more resources to training results in increasing accuracy. This property holds true for *most* workloads (vision and language DNNs). However, when this property does *not* hold, further work is needed to prevent Ekya's microprofiler from making erroneous estimations and its thief scheduler from making sub-optimal allocations.

8 Related Work

1) ML training systems. For large scale scheduling of training in the cloud, model and data parallel frameworks [3, 10, 24, 50] and various resource schedulers [30, 31, 56, 69, 95, 97] have been developed. These systems, however, target different objectives than Ekya, like maximizing parallelism, fairness, or minimizing average job completion. Collaborative training systems [18, 51] work on decentralized data on mobile phones. They focus on coordinating the training between edge and the cloud, and not on training alongside inference.

2) Video processing systems. Prior work has built low-cost, high-accuracy and scalable video processing systems for the edge and cloud [22, 32, 37]. VideoStorm investigates quality-lag requirements in video queries [32]. NoScope exploits difference detectors and cascaded models to speedup queries [22]. Focus uses low-cost models to index videos [34]. Chameleon exploits correlations in camera content to amortize *profiling costs* [37]. Reducto [47] and DDS [25] seek to reduce edge-to-cloud traffic by intelligent frame sampling and video encoding. All of these works optimize only the inference accuracy or the system/network costs of DNN inference, unlike Ekya's focus on retraining. More recently, LiveNAS[41] deploys continuous retraining to update video upscaling models, but focuses on efficiently allocating client-server bandwidth to different subsamples of a single video stream. Instead, Ekya focuses on GPU allocation for maximizing retrained accuracy across multiple video streams.

3) Hyper-parameter optimization. Efficient exploration of hyper-parameters is crucial in training systems to find the model with the best accuracy. Techniques range from simple grid or random search [17], to more sophisticated approaches using random forests [35], Bayesian optimization [85, 88], probabilistic modelling [71], or non-stochastic infinite-armed bandits [46]. Unlike the focus of these techniques on finding the hyper-parameters with the highest accuracy, our focus is on resource allocation. Further, we are focused on the inference accuracy over the retrained window, where producing the best retrained model often turns out to be sub-optimal.

4) Continuous learning. Machine learning literature on continuous learning adapts models as new data comes in. A common approach used is transfer learning [33, 51, 72, 74]. Research has also been conducted on handling catastrophic forgetting [43, 79], using limited amount of training data [73, 89], and dealing with class imbalance [16, 92]. Ekya builds atop continuous learning techniques for its scheduling and implementation, to enable them in edge deployments.

9 Acknowledgements

We thank the NSDI reviewers and our shepherd, Minlan Yu, for their invaluable feedback. This research is partly supported by NSF (CCF-1730628, CNS-1901466), UChicago CERES Center, a Google Faculty Research Award and gifts from Amazon, Ant Group, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Nvidia, Scotiabank, Splunk and VMware.

References

[1] Azure percept. https://azure.microsoft.com/en-us/services/azure-percept/.

[2] Google ai blog: Custom on-device ml models with learn2compress. https://ai.googleblog.com/2018/05/custom-on-device-ml-models.html. (Accessed on 03/09/2021).

[3] MxNet: a flexible and efficient library for deep learning. https://mxnet.apache.org/.

[4] Nvidia multi-process service. https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf. (Accessed on 09/16/2020).

[5] Reducing edge compute cost for live video analytics. https://techcommunity.microsoft.com/t5/internet-of-things/live-video-analytics-with-microsoft-rocket-for-reducing-edge/ba-p/1522305. (Accessed on 03/09/2021).

[6] scipy.optimize.nnls — scipy v1.5.2 reference guide. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.nnls.html. (Accessed on 09/17/2020).

[7] torchvision.models — pytorch 1.6.0 documentation. https://pytorch.org/docs/stable/torchvision/models.html. (Accessed on 09/16/2020).

[8] A Comprehensive List of Hyperparameter Optimization & Tuning Solutions. https://medium.com/@mikkokotila/a-comprehensive-list-of-hyperparameter-optimization-tuning-solutions-88e067f19d9, 2018.

[9] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Fair allocation of multiple resource types. In *USENIX NSDI*, 2011.

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A System for Large-Scale Machine Learning. In *USENIX OSDI*, 2016.

[11] Achieving Compliant Data Residency and Security with Azure.

[12] AI and Compute. https://openai.com/blog/ai-and-compute/, 2018.

[13] G. Ananthanarayanan, V. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. R. Sivalingam, and S. Sinha. Real-time Video Analytics – the killer app for edge computing. *IEEE Computer*, 2017.

[14] AWS Outposts. https://aws.amazon.com/outposts/.

[15] Azure Stack Edge. https://azure.microsoft.com/en-us/services/databox/edge/.

[16] E. Belouadah and A. Popescu. IL2M: Class Incremental Learning With Dual Memory. In *IEEE ICCV*, 2019.

[17] J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

[18] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander. Towards Federated Learning at Scale: System Design. In *SysML*, 2019.

[19] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, Matthai Philipose. Videoedge: Processing camera streams using hierarchical clusters. In *ACM/IEEE SEC*, 2018.

[20] CLIFFORD, M. J., PERRONS, R. K., ALI, S. H.,ANDGRICE, T. A. Extracting Innovations: Mining, Energy, and Technological Changein the Digital Age. In *CRC Press*, 2018.

[21] cnn-benchmarks. https://github.com/jcjohnson/cnn-benchmarks#resnet-101, 2017.

[22] D. Kang, J. Emmons, F. Abuzaid, P. Bailis and M. Zaharia. Noscope: Optimizing neural network queries over video at scale. In *VLDB*, 2017.

[23] D Maltoni, V Lomonaco. Continuous learning in single-incremental-task scenarios. In *Neural Networks*, 2019.

[24] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng. Large Scale Distributed Deep Networks. In *NeurIPS*, 2012.

[25] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang. Server-driven video streaming for deep learning inference. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 557–570, 2020.

[26] Edge Computing at Chick-fil-A. https://medium.com/@cfatechblog/edge-computing-at-chick-fil-a-7d67242675e2. 2019.

[27] Ganesh Ananthanarayanan, Victor Bahl, Yuanchao Shu, Franz Loewenherz, Daniel Lai, Darcy Akers, Peiwei Cao, Fan Xia, Jiangbo Zhang, Ashley Song. Traffic Video Analytics – Case Study Report. 2019.

[28] GI Parisi, R Kemker, JL Part, C Kanan, S Wermter . Continual lifelong learning with neural networks: A review. In *Neural Networks*, 2019.

[29] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 1487–1495, New York, NY, USA, 2017. Association for Computing Machinery.

[30] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *ACM SIGCOMM*, 2014.

[31] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. H. Liu, and C. Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *USENIX NSDI*, 2019.

[32] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodík, Matthai Philipose, Victor Bahl, Michael Freedman. Live video analytics at scale with approximation and delay-tolerance. In *USENIX NSDI*, 2017.

[33] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. In *NeurIPS Deep Learning and Representation Learning Workshop*, 2015.

[34] K. Hsieh, G. Ananthanarayanan, P. Bodík, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu. Focus: Querying Large Video Datasets with Low Latency and Low Cost. In *USENIX OSDI*, 2018.

[35] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization*, 2011.

[36] Joseph Redmon, Ali Farhadi . Yolo9000: Better, faster, stronger. In *CVPR*, 2017.

[37] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodík, Siddhartha Sen, Ion Stoica. Chameleon: Scalable adaptation of video analytics. In *ACM SIGCOMM*, 2018.

[38] Junjue Wang, Ziqiang Feng, Shilpa George, Roger Iyengar, Pillai Padmanabhan, Mahadev Satyanarayanan. Towards scalable edge-native applications. In *ACM/IEEE Symposium on Edge Computing*, 2019.

[39] K He, X Zhang, S Ren, J Sun . Deep residual learning for image recognition. In *CVPR*, 2016.

[40] M. Khani, P. Hamadanian, A. Nasr-Esfahany, and M. Alizadeh. Real-time video inference on edge devices via adaptive model streaming. *arXiv preprint arXiv:2006.06628*, 2020.

[41] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 107–125, New York, NY, USA, 2020. Association for Computing Machinery.

[42] Konstantin Shmelkov, Cordelia Schmid, Karteek Alahari . Incremental learning of object detectors without catastrophic forgetting. In *ICCV*, 2017.

[43] J. Lee, J. Yoon, E. Yang, and S. J. Hwang. Lifelong Learning with Dynamically Expandable Networks. In *ICLR*, 2018.

[44] A. Li, O. Spyra, S. Perel, V. Dalibard, M. Jaderberg, C. Gu, D. Budden, T. Harley, and P. Gupta. A generalized framework for population based training. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '19, page 1791–1799, New York, NY, USA, 2019. Association for Computing Machinery.

[45] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18(1):6765–6816, Jan. 2017.

[46] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52, 2017.

[47] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. 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*, pages 359–376, 2020.

[48] R. Liaw, R. Bhardwaj, L. Dunlap, Y. Zou, J. E. Gonzalez, I. Stoica, and A. Tumanov. Hypersched: Dynamic resource reallocation for model development on a deadline. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, page 61–73, New York, NY, USA, 2019. Association for Computing Machinery.

[49] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016.

[50] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning in the cloud. *PVLDB*, 5(8):716–727, 2012.

[51] Y. Lu, Y. Shu, X. Tan, Y. Liu, M. Zhou, Q. Chen, and D. Pei. Collaborative learning between cloud and end devices: an empirical study on location prediction. In *ACM/IEEE SEC*, 2019.

[52] M McCloskey, NJ Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, 1989.

[53] M Sandler, A Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen . Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[54] M. J. Magazine and M. Chern. A note on approximation schemes for multidimensional knapsack problems. *Math. Oper. Res.*, 9(2), 1984.

[55] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla. Themis: Fair and efficient GPU cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 289–304, Santa Clara, CA, Feb. 2020. USENIX Association.

[56] K. Mahajan, A. Singhvi, A. Balasubramanian, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla. Themis: Fair and efficient GPU cluster scheduling for machine learning workloads. In *USENIX NSDI*, 2020.

[57] 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. In *CVPR*, 2016.

[58] Measuring Fixed Broadband - Eighth Report, FEDERAL COMMUNICATIONS COMMISSION OFFICE OF ENGINEERING AND TECHNOLOGY. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-fixed-broadband-eighth-report. 2018.

[59] Microsoft-Rocket-Video-Analytics-Platform. https://github.com/microsoft/Microsoft-Rocket-Video-Analytics-Platform.

[60] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

[61] Mingxing Tan, Quoc V. Le . Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[62] U. Misra, R. Liaw, L. Dunlap, R. Bhardwaj, K. Kandasamy, J. E. Gonzalez, I. Stoica, and A. Tumanov. *RubberBand: Cloud-Based Hyperparameter Tuning*, page 327–342. Association for Computing Machinery, New York, NY, USA, 2021.

[63] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica. Ray: A distributed framework for emerging ai applications. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 561–577, USA, 2018. USENIX Association.

[64] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun . Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018.

[65] OPENSIGNAL. Mobile Network Experience Report . https://www.opensignal.com/reports/2019/01/usa/mobile-network-experience. 2019.

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. 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. Curran Associates, Inc., 2019.

[67] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *ICLR*, 2017.

[68] Pei Sun and Henrik Kretzschmar and Xerxes Dotiwalla and Aurelien Chouard and Vijaysai Patnaik and Paul Tsui and James Guo and Yin Zhou and Yuning Chai and Benjamin Caine and Vijay Vasudevan and Wei Han and Jiquan Ngiam and Hang Zhao and Aleksei Timofeev and Scott Ettinger and Maxim Krivokon and Amy Gao and Aditya Joshi and Yu Zhang and Jonathon Shlens and Zhifeng Chen and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset, 2019.

[69] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *ACM EuroSys*, 2018.

[70] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery.

[71] J. Rasley, Y. He, F. Yan, O. Ruwase, and R. Fonseca. HyperDrive: exploring hyperparameters with POP scheduling. In *ACM/IFIP/USENIX Middleware*, 2017.

[72] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, Kayvon Fatahalian. Online model distillation for efficient video inference. In *ICCV*, 2019.

[73] S. V. Ravuri and O. Vinyals. Classification accuracy score for conditional generative models. 2019.

[74] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. In *IEEE CVPR Workshop*, 2014.

[75] J. Redmon and A. Farhadi. Yolov3: An incremental improvement, 2018.

[76] Residential landline and fixed broadband services . https://www.ofcom.org.uk/__data/assets/pdf_file/0015/113640/landline-broadband.pdf. 2019.

[77] RM French. Catastrophic forgetting in connectionist networks. In *Trends in cognitive sciences*, 1999.

[78] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5), 1952.

[79] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L. Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *AAAI*, 2018.

[80] H. F. Scheduler. https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/FairScheduler.html.

[81] Shadi Noghabi, Landon Cox, Sharad Agarwal, Ganesh Ananthanarayanan. The emerging landscape of edge-computing. In *ACM SIGMOBILE GetMobile*, 2020.

[82] H. Shen, S. Han, M. Philipose, and A. Krishnamurthy. Fast video classification via adaptive cascading of deep models. In *CVPR*, 2017.

[83] Shivangi Srivastava, Maxim Berman, Matthew B. Blaschko, Devis Tuia . Adaptive compression-based lifelong learning. In *BMVC*, 2019.

[84] Si Young Jang, Yoonhyung Lee, Byoungheon Shin, Dongman Lee, Dionisio Vendrell Jacinto . Application-aware iot camera virtualization for video analytics edge computing. In *ACM/IEEE SEC*, 2018.

[85] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, 2012.

[86] Song Han, Huizi Mao, William J. Dally . Accelerating very deep convolutional networks for classification and detection. In *ICLR*, 2017.

[87] Sweden Data Collection & Processing.

[88] K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, and R. P. Adams. Scalable Bayesian Optimization Using Deep Neural Networks. In *ICML*, 2015.

[89] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, 2017.

[90] The Future of Computing is Distributed. https://www.datanami.com/2020/02/26/the-future-of-computing-is-distributed/, 2020.

[91] H. Wang, A. Kembhavi, A. Farhadi, A. L. Yuille, and M. Rastegari. Elastic: Improving cnns with dynamic scaling policies. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2258–2267, 2019.

[92] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu. Large scale incremental learning. In *IEEE CVPR*, 2019.

[93] Xi Yin, Xiang Yu, Kihyuk Sohn, Xiaoming Liu and Manmohan Chandraker. Feature transfer learning for face recognition with under-represented data. In *IEEE CVPR*, 2019.

[94] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *IEEE PAMI*, 2016.

[95] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *USENIX OSDI*, 2018.

[96] Z. Li and D. Hoiem . Learning without forgetting. In *ECCV*, 2016.

[97] H. Zhang, L. Stafman, A. Or, and M. J. Freedman. SLAQ: quality-driven scheduling for distributed machine learning. In *SoCC*, 2017.

[98] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

Notation	Description
$\mathcal{V}$	Set of video streams
v	A video stream ($v \in \mathcal{V}$)
T	A retraining window with duration $\|T\|$
Γ	Set of all retraining configurations
γ	A retraining configuration ($\gamma \in \Gamma$)
Λ	Set of all inference configurations
λ	An inference configuration ($\lambda \in \Lambda$)
$\mathcal{G}$	Total number of GPUs
δ	The unit for GPU resource allocation
$A_T(v,\gamma,\lambda,\mathcal{R},I)$	Inference accuracy for video v for given configurations and allocations
$C_T(v,\gamma,\lambda)$	Compute cost in GPU-time for video v for given configurations and allocations
$\phi_{v\gamma\lambda\mathcal{R}I}$	A set of binary variables ($\phi_{v\gamma\lambda\mathcal{R}I} \in \{0,1\}$). $\phi_{v\gamma\lambda\mathcal{R}I} = 1$ iff we use retraining config γ, inference config λ, $\mathcal{R}\delta$ GPUs for retraining, $I\delta$ GPUs for inference for video v

Table 4: Notations used in Ekya's description.

[99] Ziwei Liu, Zhongqi Miao, Xiaohang Zhan, Jiayun Wang, Boqing Gong, Stella X. Yu . Large-scale long-tailed recognition in an open world. In *CVPR*, 2019.

A Thief Scheduler

A.1 Complexity Analysis.

Assuming all the $A_T(v,\gamma,\lambda,\mathcal{R},I)$ values are known, the above optimization problem can be reduced to a multi-dimensional binary knapsack problem, a NP-hard problem [54]. Specifically, the optimization problem is to pick binary options ($\phi_{v\gamma\lambda\mathcal{R}I}$) to maximize overall accuracy while satisfying two capacity constraints (the first and second constraints in Eq 1). In practice, however, getting all the $A_T(v,\gamma,\lambda,\mathcal{R},I)$ is *infeasible* because this requires training the edge DNN using all retraining configurations and running inference using all the retrained DNNs with all possible GPU allocations and inference configurations.

The uncertainty of $A_T(v,\gamma,\lambda,\mathcal{R},I)$ resembles the multi-armed bandits (MAB) problem [78] to maximize the expected rewards given a limited number of trials for a set of options. Our optimization problem is more challenging than MAB for two reasons. First, unlike the MAB problem, the cost of trials ($C_T(v,\gamma,\lambda)$) varies significantly, and the optimal solution may need to choose cheaper yet less rewarding options to maximize the overall accuracy. Second, getting the reward $A_T(v,\gamma,\lambda,\mathcal{R},I)$ after each trial requires "ground truth" labels that are obtained using the large golden model, which can only be used judiciously on resource-scarce edges (§2.2).

In summary, our optimization problem is computationally more complex than two fundamentally challenging problems (multi-dimensional knapsack and multi-armed bandits).

Algorithm 2: PickConfigs

```
Data: Resource allocations in temp_alloc[], configurations
      ($\Gamma$ and $\Lambda$), retraining window $T$, videos $V$
Result: Chosen configs $\forall v \in V$, average accuracy over $T$
chosen_accuracies[] ← {}; chosen_configs[] ← {};
for v in V[] do
    infer_config_pool[] = Λ.where(resource_cost <
      temp_alloc[v.inference_job] && accuracy $\geq a_{MIN}$ );
    infer_config = max(infer_config_pool, key=accuracy);
    best_accuracy = 0;
    for train_config in Γ do
        /* Estimate accuracy of inference/training
           config pair over retraining window     */
        accuracy = EstimateAccuracy(train_config,
          infer_config, temp_alloc[v.training_job], T);
        if accuracy > best_accuracy then
            best_accuracy = accuracy;
            best_train_config = train_config;
    chosen_accuracies[v] = best_accuracy;
    chosen_configs[v] = {infer_config, best_train_config};
return chosen_configs[], mean(chosen_accuracies[]);
```