Large-scale Video Analytics with Cloud–Edge Collaborative Continuous Learning

YA NAN, Nanyang Technological University, Singapore
SHIQI JIANG, Microsoft Research Asia, China
MO LI, Nanyang Technological University, Singapore

Deep learning–based video analytics demands high network bandwidth to ferry the large volume of data when deployed on the cloud. When incorporated at the edge side, only lightweight deep neural network (DNN) models are affordable due to computational constraint. In this article, a cloud–edge collaborative architecture is proposed combining edge-based inference with cloud-assisted continuous learning. Lightweight DNN models are maintained at the edge servers and continuously retrained with a more comprehensive model on the cloud to achieve high video analytics performance while reducing the amount of data transmitted between edge servers and the cloud. The proposed design faces the challenge of constraints of both computation resources at the edge servers and network bandwidth of the edge–cloud links. An accuracy gradient-based resource allocation algorithm is proposed to allocate the limited computation and network resources across different video streams to achieve the maximum overall performance. A prototype system is implemented and experiment results demonstrate the effectiveness of our system with up to 28.6% absolute mean average precision gain compared with alternative designs.

CCS Concepts: • **Computer systems organization** → **Real-time systems**; • **Information systems** → **Multimedia streaming**; • **Computing methodologies** → **Computer vision;**

Additional Key Words and Phrases: Edge computing,video analytics,distributed system,continuous learning

ACM Reference format:
Ya Nan, Shiqi Jiang, and Mo Li. 2023. Large-scale Video Analytics with Cloud–Edge Collaborative Continuous Learning. *ACM Trans. Sensor Netw.* 20, 1, Article 14 (October 2023), 23 pages.
https://doi.org/10.1145/3624478

1 INTRODUCTION

Empowered by the advances in deep learning technologies, video analytics based on **deep neural network (DNN)** models are applied across a wide range of tasks including image classification, object detection, semantic segmentation, and so on, which are essential for growing applications such as intelligent video surveillance [45, 49, 64], smart transportation [31, 48, 59], and autonomous vehicles [32, 55]. With the increasing presence of networked cameras, the volume of video data to be processed is growing rapidly, which in turn challenges the scalability of existing design of video analytics systems.

Authors' addresses: Y. Nan and M. Li, Nanyang Technological University, 50 Nanyang Avenue, Singapore, 639798; e-mails: ya001@e.ntu.edu.sg, limo@ntu.edu.sg; S. Jiang, 5 Dan Ling Street, Beijing, 100080, Microsoft Research Asia, China; e-mail: shijiang@microsoft.com.

1550-4859/2023/10-ART14
https://doi.org/10.1145/3624478

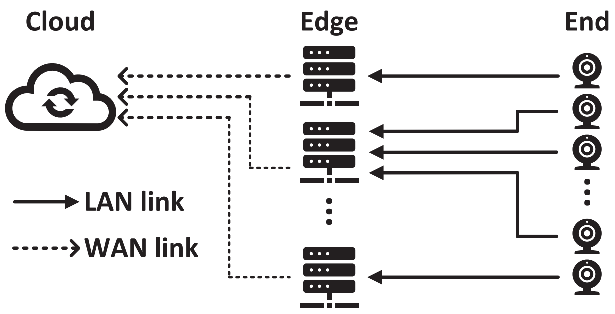


Fig. 1. The three-layer cloud–edge collaborative architecture for video analytics.

Deep learning–based video analytics demands high network bandwidth if the large volume of data is streamed through network and requires rich computation resources for processing at inference servers for reasonable performance. Generally, video analytics systems are deployed on the cloud, e.g., Microsoft Azure [8]. Cloud-based DNN inference enables the usage of complex but resource-consuming DNN models, e.g., Faster-RCNN [39] for object detection. All video sources, however, have to stream their data to the centralized cloud server, which demands excessive data transmissions via **wide-area networks (WAN)** due to the geographically large scale. When the network bandwidth is limited, data compression techniques such as frame filtering [24] and resolution downscaling [11] are applied that heavily degrade the performance of DNN models.

In contrast, edge-based DNN inference executes inference tasks on regional edge servers geocolocated with nearby video sources and more richly interconnected with high bandwidth **local area networks (LAN)**. While the network condition allows high-quality low-latency video data to be streamed to the edge servers, the generally throttled computation resources at edge servers limit the scale and complexity of DNN models that can be adopted. In such a case, only lightweight models can be used, e.g., SSD [29]. As a result, edge-based inference is less resilient to concept drift [51] and may suffer from low inference accuracy during runtime.

To leverage the advantages of both cloud- and edge-based inference, this article proposes a cloud–edge collaborative architecture. Figure 1 depicts a typical setting of the proposed design. The video sources, edge servers, and the cloud form a three-layer architecture. Instead of deploying stand-alone DNN models at edge servers, the architecture adopts a continuous learning approach—lightweight DNN models are deployed at edge servers for the inference task of each video stream and are periodically retrained on the cloud and sent back to edge servers. In such a design, frames uploaded to the cloud are used as training samples instead of for cloud inference. With periodical model retraining, the cloud–edge collaborative architecture helps the lightweight models learn the up-to-date data distributions with real-time ingested video content and alleviates the performance degradation caused by concept drift.

A major challenge in adopting the proposed cloud–edge collaborative architecture stays with the fact that both the computation resources at edge servers and the network bandwidth between the cloud and edge servers are constrained, and it is non-trivial to judiciously allocate the limited resources across video streams to yield a higher inference accuracy. Due to constrained computation and network resources, edge servers may not be able to process video streams with full frame rates or send all video frames to the cloud for retraining in real time. Meanwhile, some video content are more sensitive to resource dynamics while others are more resilient, and such diversity varies both over time and across streams. Consequently, appropriate inference and retraining configurations are desired to decide the amount of video data for inference and retraining, which corresponds to a comprehensive resource allocation problem to allocate available computation and network resources across video streams.

A general solution to such a resource allocation problem is of high complexity mainly due to two points. First, because video content varies dynamically, the expected accuracy of different retraining and inference configurations needs to be probed online regularly and over each video stream, which involves computationally intensive operations, including model training and inference. Second, the size of the search space for the optimal combination of configurations that yields the maximum overall accuracy grows with the number of available configurations as well as video streams. In this article, we propose a heuristic named *accuracy gradient* to quantify the sensitivity of different video content toward resource variation. Leveraging the accuracy gradient, we devise a **depth-first search (DFS)** algorithm that greatly reduces the probing overhead by pruning the search space and reducing probing count. Besides, the proposed scheme embeds a non-trivial system solution to probe the inference accuracy of each stream with regard to the amount of allocated computation and network resources, which reduces the amortized cost of a single probing operation. Combining the two designs, the computational overhead caused by resource allocation can be well constrained without detrimenting the accuracy gain from continuous learning.

Model retraining, while essential in our system, imposes computational demands on cloud resources and introduces delays due to the training time, which subsequently affects the timeliness of model updates. Though the cloud is often perceived to possess abundant computational capacities for such retraining, it is impractical to anticipate infinite scalability, especially as the system expands to accommodate an increasing number of video streams. Furthermore, prolonged training periods can make retrained models less up-to-date, undermining the very purpose of retraining. To solve the problem, we devise an aggregated model training technique to curtailing the training overhead on the cloud. Instead of retraining the full model of each stream independently, models are divided into backbone and head. Video streams are grouped and a common model backbone is shared within each group. The shared backbone is retrained at a lower frequency than head using data aggregated over time and across streams. Consequently, the overall training cost on the cloud is reduced, and model update timeliness can be further improved.

In summary, this article makes the following contributions:

— We propose a cloud–edge collaborative architecture to support scalable DNN-based video analytics by leveraging continuous learning.
— An accuracy gradient-based solution is proposed to address the major challenge in computation and network resource allocation during runtime that achieves high resource allocation efficiency with low system overhead.
— An aggregated model training technique is devised to reduce the training overhead on the cloud and improve model update timeliness.

The proposed system is implemented and fully evaluated with real-world video traces, including the Bellevue Traffic [3] and UA-DETRAC datasets [50]. **Mean average precision (mAP)** is used as the evaluation metric, and the experiment results suggest up to 28.6% absolute mAP gain over alternative solutions on object detection tasks, proving the effectiveness of the proposed cloud–edge collaborative continuous learning approach on real-world scenarios. A comprehensive ablation study is conducted to showcase the performance improvement achieved by each design component separately.

The rest of the article is organized as follows. Section 2 provides preliminary experiment results that motivate the study. Section 3 details the system design and implementation. Section 4 presents the evaluation results. Section 5 presents related works, and Section 7 concludes the article.

2 MOTIVATION

2.1 Benefit of Cloud–Edge Collaborative Continuous Learning

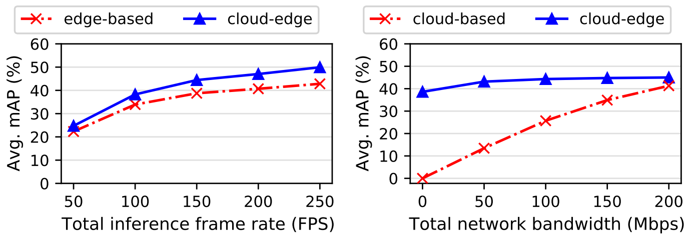
As Figure 1 illustrates, the three-layer cloud–edge collaborative architecture contains a cloud center connected with many edge servers, each connected to multiple video sources. Each edge server and its corresponding video sources are geo-colocated and connected with bandwidth-sufficient LAN, e.g., Gigabit Ethernet, while between edge servers and the cloud are WAN links with limited bandwidth, e.g., 4G or 5G cellular networks. During inference, an edge server maintains one lightweight DNN model for each connected video source and at the same time uploads video frames as training data for retraining its DNN models on the cloud. The retraining process for each video stream takes place periodically in *retraining windows* on the cloud, where the frames collected from the previous retraining window are labeled by a *golden model* maintained at the cloud and used to retrain the corresponding lightweight model. Each retrained model is thereafter downloaded to the edge server and used for future inference tasks.

With the computation and network resource constraints, video frames are downsampled, i.e., filtered, from the stream both for supplying to inference models at edge servers as well as uploading to the cloud for model retraining. In our design, two frame rates are used as the knobs for frame filtering, referred to as *inference frame rate* and *retraining frame rate*, according to which frames are uniformly sampled before inference and uploading. The inference frame rate of one stream corresponds to its computation cost incurred at the edge, and the retraining frame rate of one stream corresponds to the network resources on the cloud–edge link it consumes.

To demonstrate the benefit of combining edge inference and cloud-assisted continuous learning, we compare the approach with two straightforward alternatives, namely edge-based inference and cloud-based inference. For edge-based inference, video frames are processed by a lightweight object detection DNN, SSDLite with a MobileNetV2 backbone. For the cloud inference, frames are streamed to the cloud and inferred by a complex DNN, Faster-RCNN with a ResNet101 backbone. Both models are pre-trained on the COCO dataset [25]. For the continuous learning approach, video frames are processed at edge servers using SSDLite models, which are retrained every 20 seconds using pseudo-labels generated by the Faster-RCNN model on the cloud. To conduct the measurement, 20 video streams are selected from UA-DETRAC dataset and are allocated across two servers. The same inference and retraining frame rates are assigned to all streams, and hence both computation and network resources are equally shared. The inference accuracy of each stream is represented by mAP, and the average mAP across all streams is used to represent the overall performance of the system.

Figure 2 compares the achieved mAP of the cloud–edge collaborative approach with the edge-based as well as cloud-based approaches, respectively. Figure 2(a) gives the average mAP with solely edge-based inference and that of cloud–edge collaboration with an average bandwidth of 5 Mbps per stream. So 20 video streams consume 100 Mbps of uplink bandwidth in total. Computation resources on each edge server are denoted as the total inference frame rate ranging from 50 to 250 FPS, meaning how many frames the edge can process in each second with the lightweight model (SSLite in this case). Experiment results show that the continuous learning approach achieves a significant mAP gain even under relatively constrained bandwidth. With the total inference frame rate capped at 150 FPS on edge and a typical 100 Mbps 4G LTE cellular link[9], cloud–edge collaborative continuous learning can achieve up to 5.7% mAP gain compared with edge-based inference. The results indicate that periodical retraining and update can well alleviate the model degradation caused by concept drift compared with using a fixed model.

Figure 2(b) illustrates the average mAP with solely cloud-based inference and edge inference with cloud-assisted continuous learning with the total inference frame rate of each edge at 150



(a) Comparison with edge-based. (b) Comparison with cloud-based.

Fig. 2. Performance comparison with (a) the edge-based approach and (b) the cloud-based approach.

FPS. Network resources are denoted as the total bandwidth on the cloud–edge links ranging from 0 to 200 Mbps, which is typical bandwidth of nowadays 5G uplinks [36]. With a low bandwidth, e.g., 100 Mbps, few frames could be uploaded for cloud-based inference, resulting in mAP as low as 25.7%. However, the cloud–edge collaborative design achieves a much better accuracy with 18.6% mAP gain. The rationale behind is that network resources have a higher utility when uploaded frames are used to enhance edge models instead of for inference when the network bandwidth is limited.

Compared with purely edge-based solution, one concern about cloud–edge collaboration is the potential privacy violation, since it requires training data to be uploaded to cloud. However, some existing solutions [5, 30, 53] have proposed privacy-preserving video streaming and analytics techniques that adapt well to our cloud–edge collaborative architecture. Therefore, privacy issues are not considered in this article but the system is compatible with such privacy-preserving designs.

2.2 Content Diversity

In the measurement above, the computation and network resources are equally allocated to all video streams. However, video streams with varied content may have different sensitivities of inference accuracy to resource variation, i.e., some video content may gain high accuracy when more resources are allocated for the inference or model retraining while others may not, which we call *content diversity*. For example, if a video stream only contains static scenes or objects of slow motion, then the variation of inference frame rate may not greatly affect its inference accuracy. In other words, such a stream has a low sensitivity toward computation resource variation. Oppositely, a video stream containing high motions may be more sensitive to the inference frame rate, and thus it has high sensitivity toward computation resource variation. Such diversity also applies to network resources, since some streams experience larger concept drift and are more likely to achieve accuracy gain from retraining online than others.

We perform trace-driven emulation with the same UA-DETRAC dataset. Figure 3(a) and (b) demonstrate the spatial content diversity by selecting three different video streams and applying different inference and retraining frame rates, respectively, on them. Figure 3(c) and (d) demonstrate the temporal content diversity by selecting three different segments from the same stream and applying different inference and retraining frame rates. Inconsistent slopes of polylines corresponding to different video streams or segments in the figures indicate different sensitivities toward resource variation originated from content diversity, which exists not only spatially across video streams but also temporally over time on the same stream. The observations motivate us to explore such diversity to best allocate the limited computation and network resources to video steams. We later use a term to quantitatively express the content diversity and use the quantified diversity to guide resource allocation (Section 3.2).

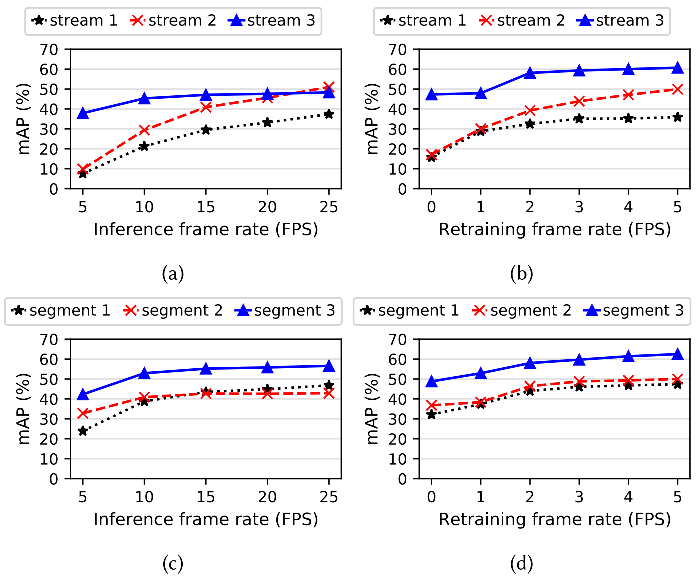


Fig. 3. The achieved mAP spatially across different video streams with varied (a) computation and (b) network resources and temporally across different segments from the same stream with varied (c) computation and (d) network resources.

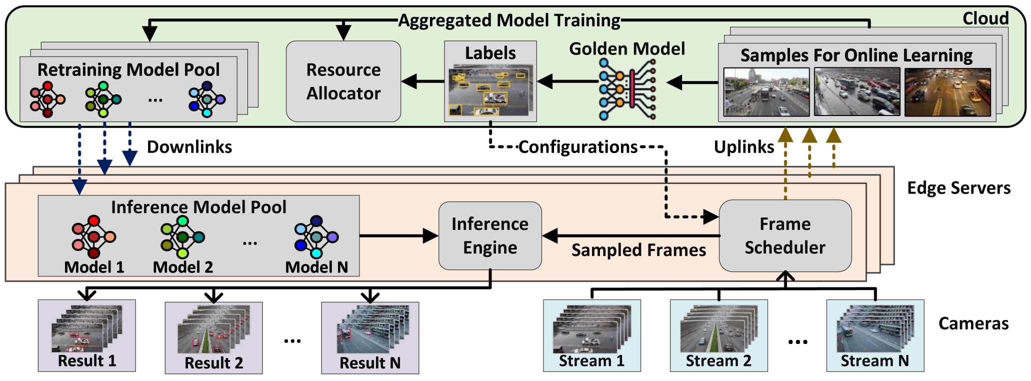


Fig. 4. Overall system architecture.

3 SYSTEM DESIGN

3.1 Overview

In the proposed cloud–edge collaborative architecture shown in Figure 4, all edge servers connected with the cloud form a set $\mathcal{E}$. Each edge server $e \in \mathcal{E}$ is connected with a set of video sources and processes their video streams S_e. All video frames captured at end devices can be fully streamed to the edge.

Key components on each edge server include a frame scheduler, an inference engine, and an inference model pool. Each edge server e maintains an inference model pool, containing lightweight models paired with every connected video stream $s \in S_e$. Besides, the edge server maintains a frame scheduler that schedules all video frames streamed from each source either to feed into the inference engine for inference or upload to the cloud for retraining the edge model according to its *configuration*. As stated previously in Section 2.1, the configuration of a stream consists of two knobs, namely *inference frame rate* and *retraining frame rate*. Frames are uniformly sampled, i.e., filtered, for inference and uploading according to inference and retraining frame rate, respectively. The frame scheduler uses a **weighted round robin (WRR)** algorithm [6] to schedule the frames of each stream to control network and GPU utilization so that the resource consumption of each stream follows the allocated amount (Section 3.4).

Key components on the cloud include a golden model, a retraining model pool, and a resource allocator. In the retraining model pool, a copy of the edge DNN model for each video stream $s \in \mathcal{S}$ is stored. Upon arrival, frames uploaded to the cloud are first supplied to a golden model to generate training labels. Retraining is triggered periodically every T-second-long retraining window. At the beginning of each retraining window, the edge model of each stream is fetched from the pool and retrained with most recently uploaded frames from the previous window. During retraining, the resource allocator monitors the validation accuracy, based on which it generates the resource allocation plan and corresponding frame rate configurations for the edge frame scheduler (Section 3.2). Besides, an aggregated model training approach is proposed to reduce the retraining overhead (Section 3.3). Finally, retrained edge models and configurations are sent to the edge server. Retrained edge models are stored in the inference model pool, and configured frame rates are updated for the frame scheduler.

3.2 Resource Allocation

The proposed system aims at allocating the available computation and network resources across multiple video streams. A set of discrete frame rates is used as a bridge between the inference accuracy and resource consumption—each video stream is assigned a proper inference frame rate and retraining frame rate that correspond to the GPU resources consumed at the edge server and the network bandwidth used for sending data to the cloud, respectively. The aim is to achieve a maximum average accuracy across all video streams under the constraint of the total computation resources at the edge and network bandwidth on the cloud–edge links.

Problem formulation. In each retraining window i, the resource allocator needs to decide on a combination of retraining frame rate ϕ and inference frame rate ω for each stream $s \in \mathcal{S}$ as an *allocation plan*. The allocation target is to maximize the overall inference accuracy with the resource usage not exceeding the computational constraint C_e^i and network constraint U_i, which can be formulated as a discrete optimization problem defined in Equation (1). A complete set of relevant notations is presented in Table 1.

$$\begin{aligned} \underset{x_\phi^{s,i},\, y_\omega^{s,i}}{argmax} \quad & \sum_{s\in\mathcal{S}} \sum_{\phi\in\Phi,\,\omega\in\Omega} x_\phi^{s,i-1} y_\omega^{s,i} A_i^s(\phi,\omega) \\ \text{s.t.} \quad & \sum_{e\in\mathcal{E}} \sum_{s\in\mathcal{S}_e} x^{s,i}(\phi) R_n(\phi) \le U_i \\ & \forall e \in \mathcal{E}, \sum_{s\in\mathcal{S}_e} y_\omega^{s,i} R_c(\omega) \le C_i^e. \end{aligned} \tag{1}$$

Note that in retraining window i, edge models are retrained with training samples collected in retraining window $i-1$. So $x_\phi^{s,i-1}$ is used instead of $x_\phi^{s,i}$ when deriving the inference accuracy in the ith retraining window in Equation (1).

An optimal solution for such a resource allocation problem can be obtained if we have perfect knowledge about how each possible configuration (ϕ, ω) corresponds to the inference accuracy, i.e., function A_i^s for every stream s and retraining window i. Then we can apply optimization

Table 1. Mathematical Notations

Notation	Description
T	Duration of retraining window
$\mathcal{E}$	Set of edge servers
$\mathcal{S}$	Set of streams
$\mathcal{S}_e$	Set of streams connected to edge server e
C_i^e	Computation resource on edge e in window i
U_i	Uplink network bandwidth in window i
Φ	Set of available retraining frame rates
Ω	Set of available inference frame rates
$x_{\phi}^{s,i}$	Binary. $x_{\phi s i} = 1$ indicates stream s uses retraining frame rate ϕ in window i
$y_{\omega}^{s,i}$	Binary. $y_{\omega s i} = 1$ indicates stream s uses inference frame rate ω in window i
$R_n(\phi)$	Network resource consumption by a stream with retraining frame rate ϕ
$R_c(\omega)$	Computation resource consumption by a stream with inference frame rate ϕ
$A_i^s(\phi, \omega)$	Accuracy of stream s in window i with frame rates ϕ and ω

techniques for solving the multi-dimensional multi-choice knapsack problem such as **dynamic programming (DP)**. In practice, however, we may only obtain an estimation function $\hat{A}_i^s$ based on probing the validation results in the previous retraining window online.

Besides, compared with solving traditional optimization problems where the computational complexity measures the overall overhead, in our problem context the dominating overhead comes from the number of times to trigger probing the function $\hat{A}_i^s$, because each time when probing $\hat{A}_i^s$ a complete DNN inference and result evaluation is invoked that is much more time-consuming than other computational operations involved in solving conventional knapsack problems. The probing cost incurred by DP grows linearly with the number of possible configurations (ϕ, ω) as well as video streams $\mathcal{S}$. Suppose the number of retraining frame rates $\|\Phi\|$, inference frame rates $\|\Omega\|$, and video streams $\|\mathcal{S}\|$ are m, n, p, respectively, then the time dominating complexity of DP-based online resource allocation is $O(mnp)$, which indicates a large allocation overhead when there is a large number of possible configurations and streams. Such overhead may lead to excessive computation resource consumption and large delay to execute the allocation algorithm, which negatively affects the timeliness of configuration update.

Accuracy Gradient-based Resource Allocation (AGRA). AGRA is a pruning-based DFS algorithm. Each unique path from the root to a leaf node in the search tree represents a possible allocation plan. Different from applying general DP-based method, we explore a practical observation, i.e., the resource-accuracy curves at both computation and network dimensions are concave, as shown previously in Figure 2. We define the *accuracy gradient* as the amount of accuracy variation with per unit of computation or network resource allocated to a video stream. The accuracy gradient of computation (network) resources can be viewed as the first derivative of accuracy function A_i^s on the inference (retraining) dimension. DFS is then performed to search for an optimal allocation plan. Leveraging the concave observation, at each branch (non-leaf) node of the search tree, we may use accuracy gradient and linear programming relaxation [42] to derive an upper bound accuracy for the pending allocation plan corresponding to the node. A pending plan is found not optimal if its upper bound accuracy is lower than that of the best allocation plan identified so far. With such a guarantee, the allocation plan can be early pruned without reaching any leaf node of the search tree, reducing the total number of pruning.

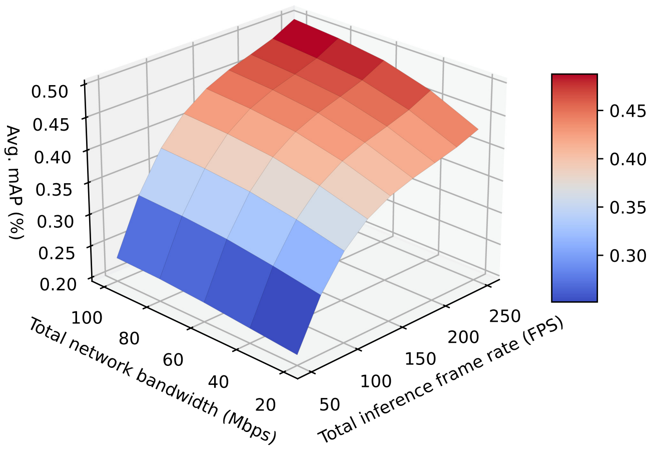


Fig. 5. The achieved mAP with varied computation and network resources.

In addition, we observe that the computation and network resource affections to the inference accuracy are not highly entangled. Figure 5 demonstrates the inference accuracy after retraining with varied computation and network resources in a three-dimensional (3D) format using the same setting as Section 2.1. It shows that the inference accuracy increases monotonically with both computation resources and networks. This observed relationship facilitates the decomposition of the two-dimensional allocation problem into two separate one-dimensional problems without encountering local optima. Such a decomposition avoids jointly allocating network and computation resources and reduces complexity. We first determine the retraining frame rate for each stream assuming a fixed inference frame rate and then determine the inference frame rate after the retraining frame rate is fixed. With the above two heuristics, we may reduce the probing overhead to $O((m+n)p)$ with a small constant on average and $O(p)$ at the best case, which greatly reduces online allocation overhead and improves the timeliness for configuration update.

Next, we first use examples to illustrate how the estimation function $\hat{A}_i^s$ can be probed online for both retraining and inference frame rate. We then present the complete AGRA algorithm with pseudo codes and a detailed explanation.

Probing $\hat{A}_i^s$ for retraining frame rate ϕ. Figure 6 illustrates how $\hat{A}_i^s$ is probed with different retraining frame rates, which correspond to the network resources allocated across video streams. Suppose a total of m candidate retraining frame rates form a finite set $\{\phi_1, \cdots, \phi_m\}$ and are ordered by their network bandwidth consumption, i.e., $\forall i, j, i > j \Rightarrow R_n(\phi_i) > R_n\left(\phi_j\right)$. At the beginning of retraining window i, for one specific video stream s, the cloud receives the frames sampled at the retraining frame rate during the previous window. Suppose the previously used retraining frame rate is ϕ_k. The training frames are split into a downsampled set and an incremental set, where the downsampled set contains frames sampled at a lower frame rate ϕ_{k-1}, and the rest of the frames form the incremental set. The downsampled set is first used to retrain the lightweight model, which is thereafter tested on a validation set and yields a validation accuracy $\hat{A}_i^s(*, \phi_{k-1})$, which gives the estimation of test accuracy using retraining frame rate ϕ_{k-1}. Here the asterisk represents a certain inference frame rate used for the model. The incremental set is then fed to the retrained model to generate a new model retrained using the full training dataset, from which $\hat{A}_i^s(*, \phi_k)$ is estimated. Instead of training the model every time from scratch, by recursively splitting the original training set into downsampled sets and incremental sets multiple rounds and training the edge model incrementally, the estimated test accuracy accuracies with different retraining frame rates can be probed with amortized overhead.

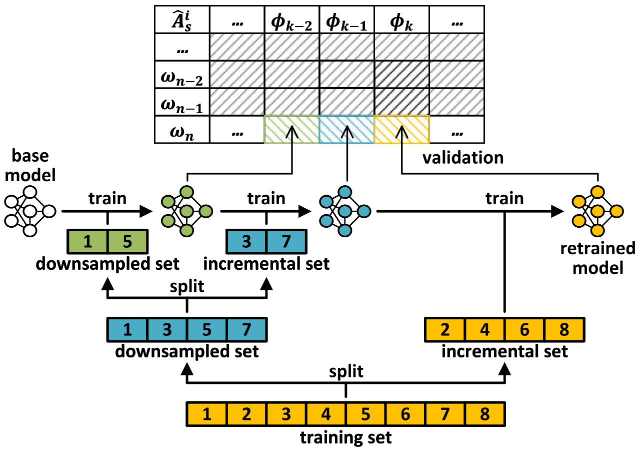


Fig. 6. Probing $\hat{A}_i^s$ for retraining frame rate ϕ.

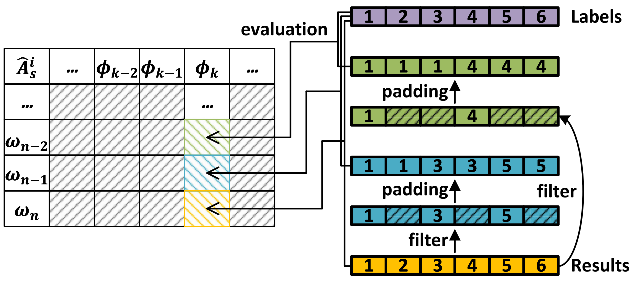


Fig. 7. Probing $\hat{A}_i^s$ for inference frame rate ω.

For validation, an extra set with a fixed number of consecutive frames at the beginning of the retraining window with the original frame rate is conveyed to the cloud as the validation dataset. In practice, we find that a validation set lasting for ~2 seconds is sufficient to produce accurate validation results, and the associated network bandwidth consumption is negligible.

Probing $\hat{A}_i^s$ for inference frame rate ω. Figure 7 illustrates how $\hat{A}_i^s$ is probed with different inference frame rates, which correspond to the computation resources allocated across video streams. A total number of n candidate inference frame rates form a finite set $\{\omega_1, \cdots, \omega_n\}$ and are ordered by their resource demand. At the beginning of retraining window i, for a certain stream s, after the retraining frame rate ϕ is decided and the retrained model is obtained, the validation dataset is fed to the retrained model to obtain the inference results. We can derive the inference accuracy obtained with the highest inference frame rate $\hat{A}_i^s(\omega_n, \phi)$ by comparing inference results with the pseudo-labels. The inference results are then uniformly downsampled to a lower frame rate, i.e., a proportion of inference results are dropped and padded with results from previous frames. The downsampled results are compared with the pseudo-labels to yield an inference accuracy corresponding to $\hat{A}_i^s(\omega_{n-1}, \phi)$. By gradually downsampling the inference results at multiple levels, the inference accuracies with different inference frame rates are estimated.

AGRA algorithm. Algorithm 1 describes the detailed AGRA procedure. The resource monitor on each edge server proactively measures the available GPU and uplink bandwidth resources and periodically synchronizes its measurements with the cloud. On the cloud, the overall uplink network bandwidth U is allocated across multiple video streams (line 2). After the retraining frame rates (i.e., network bandwidth allocation) are decided, for each edge server e, its computation resources C_e are allocated across video streams associated with it (line 4).

DFS is adopted to allocate the resources. A path from the root to a leaf node in the DFS tree represents a complete allocation plan where the frame rates of all streams are decided. Meanwhile, a path from the root to a branch node represents a pending allocation plan where the frame rates of part of streams are not decided. Starting from the root, when reaching a node on the ith layer of the search tree, the frame rate of the ith stream participating in the allocation is to be decided. So the DFS tree for the resource allocation problem of m streams and n candidate frame rates has m layers, each branch node with n children.

On searching each node (lines 6–19), S represents a set of streams whose frame rates are not decided, *resCurr* represents the amount of remaining resources, *accCurr* represents the overall accuracy of settled video streams, and *accBest* records the best overall accuracy achieved so far. When visiting a leaf node (line 8), a complete allocation plan is obtained and the current overall accuracy is directly returned. Otherwise, when visiting a branch node and at least one complete allocation plan has been found whose overall accuracy stored in *accBest* (line 9), an estimation function is used to estimate the upper bound of the overall accuracy of all unsettled streams at the current branch node. The branch can be confidently pruned without visiting its children if the estimated upper bound is still lower than previously recorded *accBest* accuracy (line 10). Otherwise, one video stream is selected and tried with all candidate frame rates, which generates multiple downsized problems with one less unsettled streams. The downsized problems are then solved recursively (lines 11–19), going one layer deeper in the search tree. Note that function to get resource consumption from configuration (line 13) is R_n and R_c when taking in retraining and inference frame rates, respectively.

The bound function takes in a set of streams and the amount of resource and produces the upper bound of the overall accuracy of those streams (lines 20–30). Accuracy gradient is derived by GRAD function (line 23) using the probing results described above. Mathematically, the gradient of a stream s in windows i for retraining frame rate ϕ_k can be estimated as follows:

$$GRAD_{s,i}^{net}(\phi_k) = \frac{\hat{A}_i^s(\phi_k, \omega_n) - \hat{A}_i^s(\phi_{k-1}, \omega_n)}{R_n(\phi_k) - R_n(\phi_{k-1})}. \tag{2}$$

The accuracy gradient of a stream s in windows i for inference frame rate ω_k can be estimated as follows:

$$GRAD_{s,i}^{com}(\omega_k) = \frac{\hat{A}_i^s(\phi, \omega_k) - \hat{A}_i^s(\phi, \omega_{k-1})}{R_c(\omega_k) - R_c(\omega_{k-1})}. \tag{3}$$

When deriving the accuracy upper bound, all streams start with the lowest frame rate and calculate corresponding accuracy gradients (line 23). The resources are thereafter repeatedly allocated to the stream with the highest gradient with linear relaxation until all resources are allocated (lines 24–29).

Theoretically, if the resource-accuracy curve is concave, then the accuracy gradient always decreases monotonically with the amount of allocated resources, then the bound function is guaranteed to provide an upper bound of a pending allocation plan. The upper bound can be used for pruning branches to avoid unnecessary probing over the entire search tree, leading to a much smaller constant factor in its complexity. In addition to that, AGRA further avoids joint allocation of network and computation resources by separately determining the retraining and inference frame rates. Such decomposition reduces the search space from $O(mnp)$ to $O((m+n)p)$.

ALGORITHM 1: Accuracy gradient-based resource allocation.

```
input  : network bandwidth $U$, computation resources $\{C_e | e \in \mathcal{E}\}$
         a set of streams $\mathcal{S}$, retraining configurations $\Phi$, inference configurations $\Omega$
output : retraining frame rates for each stream retCgfs, inference frame rates for each stream infCfgs
1  retCgfs, infCfgs ← { }, { }
2  dfs($\mathcal{S}$, $U$, 0, 0, retCgfs, $\Phi$) // allocate network bandwidth
3  for $e \in \mathcal{E}$ do
4      dfs($\mathcal{S}_e$, $C_e$, 0, 0, infCfgs, $\Omega$) // allocate computation resources on each edge
5  return retCgfs, infCfgs

6  Function dfs($S$, $resCurr$, $accCurr$, $accBest$, $cfgResults$, $cfgSet$)
7      if $S = \emptyset$ then
8          return $accCurr$ // all streams settled, return current accuracy
9      if $accBest > 0$ && $accCurr$ + get_bound($S$, $resCurr$, $cfgSet$) $< accBest$ then
10         return $accBest$ // current upper bound worse than historical best, prune
11     forall the $cfg \in cfgCandidates$ do
12         if cfg_to_res($cfg$) $< resCurr$ then
               // pick one configuration for the current stream
13             $accNext \leftarrow accCurr$ + probe_accuracy($S[1]$, $cfg$)
14             $resNext \leftarrow res\_curr$ − cfg_to_res($cfg$)
15             $SNext \leftarrow S \setminus \{S[1]\}$
               // recursively solve with one less stream
16             $accOverall$ ←dfs($SNext$, $resNext$, $accNext$, $accBest$, $cfgResults$, $cfgSet$)
17             if $accOverall > accBest$ then
18                 $accBest \leftarrow accOverall$ // update historical best if necessary
19                 $cfgResults[S[1]] \leftarrow cfg$

20 Function get_bound($S$, $res$, $cfgSet$)
21     $accOverall$, $indices$, $grads \leftarrow 0, \{\}, \{\}$
22     for $s \in S$ do
           // all streams start with the lowest configuration
23         $indices[s]$, $grads[s] \leftarrow 1$, get_grad($s$, $indices[s]$)
24     while $res > 0$ do
25         sort($S$, $key = grads$, $order = descend$) // sort by accuracy gradient
26         $s, \Delta \leftarrow S[1]$, cfg_to_res($cfgSet[indices[s] + 1]$) − cfg_to_res($cfgSet[indices[s]]$)
           // get accuracy upper bound with linear relaxation
27         $accOverall \leftarrow accOverall$ + min($res$, $\Delta$) $\times\ grads[s]$
28         $res$, $indices[s] \leftarrow res - \Delta$, $indices[s] + 1$
29         $grads[s]$ ←get_grad($s$, $indices[s]$)
30     return $accOverall$
```

3.3 Aggregated Model Training

Though the computation resources on the cloud are assumed ample for retraining all edge models, model retraining can still be time-consuming and computationally demanding when the system scales in practice (i.e., tens of hundreds of edge servers each with many video streams). The retraining cost may impair the timeliness of retrained models to be deployed at the edge. The overall

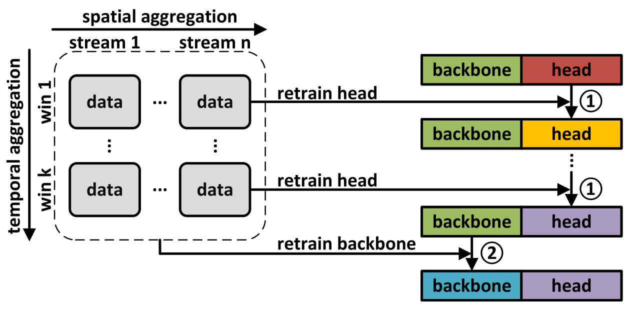


Fig. 8. Aggregated model training for a DNN model composed of backbone and head.

retraining cost on the cloud would grow linearly with the total number of video streams if each model is individually retrained.

To tackle such a problem, we take advantage of the fact that most modern DNN models can be split into a feature extraction backbone and a task-specific head [29, 44], where the former is generally much more complicated than the latter and training it contributes to the major part of training cost. Meanwhile, the feature distribution of a video stream shows less data drift compared with task-specific content [4, 14, 57], suggesting that the backbone tends to remain relatively stable over time. With such an observation, we choose to retrain the head and backbone of an edge model separately to reduce training cost. In such an approach, the model head is retrained and updated in every retraining window, while model backbone is retrained every k windows ($k > 1$). We use *aggregation length* to refer to the parameter k. Such an approach reduces training cost by training model backbone at a lower frequency.

In addition, we leverage the spatial correlation of video content distributions across geographically colocated video streams. In our setting, each edge server is regional and multiple video sources connected to the same edge servers are geo-colocated so their video content share similar feature distributions, e.g., lighting condition and motion. Such an assumption has been adopted and proven valid in existing works [19]. Meanwhile, the difference of data distribution across individual video streams is generally reflected by task-specific heads. Accordingly, our design retrains a common backbone shared across multiple geo-colocated streams, while each stream individually retrains its personalized head. The design ensures that only one model backbone is maintained at each edge server, and the overall retraining cost is constrained by the number of edge servers, instead of growing indefinitely with the number of video streams. The aggregated training set is downsampled to a fixed size to ensure unified resource consumption across different edge servers and retraining windows.

Figure 8 illustrates the detailed process of aggregated model training. Suppose n streams are connected to one edge server. The edge models of each stream share a common backbone and have individual heads. Only one of such models is illustrated in the figure. Generally, in each retraining window, the model backbone is frozen and only the task-specific head of each edge model is retrained individually using frames sampled from its corresponding stream, shown as ①. Every k retraining windows, aggregated retraining is triggered to update the model backbone of all edge models. When training a shared model backbone, the training datasets are aggregated not only from the past k windows (temporal aggregation) but also from the n streams of the same edge server (spatial aggregation). The aggregated training set is denoted as the dashed rectangle in the figure. Note that such aggregated training, shown as ②, only takes place once across n streams, after which the updated backbone is shared to all streams. Finally, the system resumes general training with the backbone frozen and each stream uses its individual training set from the previous window to retrain its personalized head.

3.4 Implementation Details

We provide other implementation details in this section.

Network and GPU utilization. The frame scheduler on an edge server maintains one individual inference queue and upload queue for each stream. When a frame arrives at the edge server, it is duplicated and stored in both queues. Frames are taken from the two queues in sequence and supplied to the edge model for inference or to the network interface for uploading to the cloud, respectively. The frame scheduler uses WRR algorithm to iteratively select frames from each stream's queue, where the weight assigned to each upload (inference) queue in WRR is proportional to the retraining (inference) frame rate of the corresponding stream. The WRR scheduler ensures that the computation and network resource consumption of each video stream is exactly the allocated amount so there is no need for explicit low-level resource management, e.g., hardware virtualization.

Pipelined model download. Transmitting multiple models from the cloud to the edge servers incurs high downlink bandwidth usage, which may cause network congestion and also reduce the timeliness of retrained model update. A pipelined model download technique is adopted to determine a proper retraining window duration and arrange model retraining and delivery of individual streams in a pipelined manner, so the downlink bandwidth is best utilized to avoid congestion. When pipelining the model download, it is ensured that only one edge model takes the full downlink bandwidth at a time, because sequential transmission provides better timeliness—only completely downloaded model can take effect for later inference. The desired duration of the retraining window T is equal to the product of the number of streams and the delay of model download. The downlink bandwidth can be measured during runtime to adjust T dynamically.

Adaptive strategy. Due to the complexity of online scenarios and the diversity of video content, neither continuous learning nor model aggregation is guaranteed to achieve better model performance. Inference accuracy after retraining may decrease if data from consecutive retraining windows or geo-colocated streams do not have a strong correlation. We apply an adaptive strategy to avoid possible performance degradation caused by model training. Instead of straightforward replacing models with retrained ones at the edge server, our system constantly monitors the validation accuracy of retrained models during aggregated model training. When the system observes decreasing post-retraining accuracy, it will drop the retrained model instead of updating it to the inference model pool, which ensures the stability and robustness.

Video compression. One option when streaming the video frames to the cloud is to compress frames into video, e.g., with H.264 [40] or VP8 [2]. Our observation in practice, however, suggests that the network traffic reduction does not benefit much from temporal compression, mainly due to the fact that the retraining frame rate is already very low. However, video compression may introduce additional content distortion as well as incur extra computation cost at the edge. After careful consideration, we choose to upload original frames without compression, but leave the system open for such alternatives.

Compatibility with other models, metrics, and tasks. Our system allows the use of other models, metrics, and tasks in addition to the ones used in motivation and evaluation as long as they follow the input and output of the system, i.e., the DNN model takes frames as input and produces inference results, and the evaluation metric takes in inference results and yields a value representing inference accuracy. For example, we can use more state-of-the-art models such as the ones with deep layer aggregation [58] and mean intersection over union as the evaluation metric. We make these components modifiable as user-defined plugins.

4 EVALUATION

In this section, the overall performance improvement of the proposed system is demonstrated with end-to-end experiments. Breakdown experiments are then performed to examine how design components contribute to such improvement.

4.1 Experiment Setting

Testbed. The proposed framework is implemented using Golang. Roles including video sources, edge servers, and the cloud are implemented as stand-alone processes and communicate using the Google Remote Procedure Call protocol. MMDetection [7], an object detection framework built upon PyTorch [37], is used for DNN inference and model retraining. Hyperparameters used for retraining and inference are specified by default configuration files in MMDetection, except that the number of training epochs is fixed to 40.

Edge servers are equipped with Nvidia Tesla T4 GPUs for inference, and the cloud uses Nvidia GeForce RTX3090 GPUs for retraining and generating training labels. The network connection between the cloud and edge is based on a 100 Mbps WAN link. Besides, we also implement a software-based resource emulator that can limit the computation resource and network bandwidth to a user-specified amount, allowing us to test the system performance with finer resource granularity.

Datasets. Bellevue Traffic and UA-DETRAC datasets are used to emulate the video input. During preprocessing, video traces from Bellevue Traffic dataset are truncated into multiple 1-hour-long sequences. For the UA-DETRAC dataset, since its video streams are already truncated, we assemble them into multiple 4-minute-long sequences.

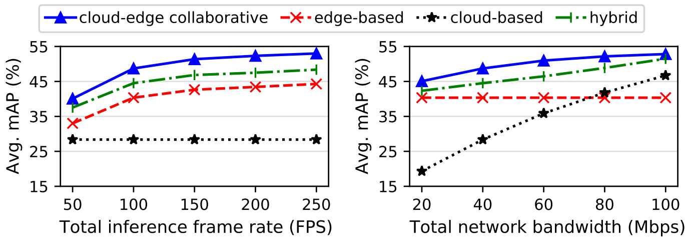
Task and model selection. Object detection is used as a vehicle DNN task to study the system performance. Faster-RCNN with a ResNet101 backbone is used as the golden model on the cloud, and SSDLite with a MobileNetV2 backbone is used as lightweight models deployed at edge servers. We use hard labels generated by the golden model with a threshold of 0.5. Both models are pre-trained on the COCO dataset. Average mAP over all streams is used as the accuracy metric.

4.2 End-to-end Study

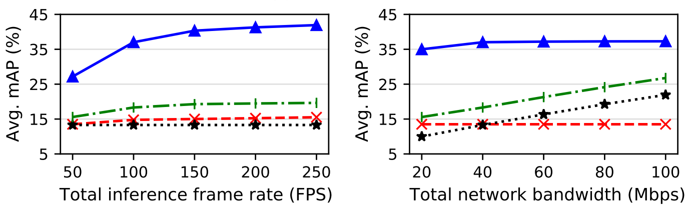
We compare the proposed system with three baseline approaches, namely (1) *edge-based* inference, where video streams are solely processed at edge servers [46], (2) *cloud-based* inference, where video streams are streamed to and processed on the cloud [11], and (3) *hybrid* inference, which combines the above two, sending frames that are hard to infer to cloud while keeping the rest processed at edge servers [16]. To give hybrid inference the benefit, an oracle obtained offline is used for frame selection, i.e., the system always selects a set of frames to be uploaded to the cloud that yields the maximum accuracy improvement, representing the accuracy upper bound for hybrid inference.

Apart from the three baselines described above, we also consider Ekya [3], an alternative that also leverages continuous learning but conducts both model retraining and inference solely at the edge. However, we find that resource contention among inference, retraining, and label generation on computation-constrained edge servers leads to excessive training delay that is even longer than the retraining window, making it hard to accommodate such a solution with typical edge server hardwares.

We start with experimenting edge servers each connected with 10 concurrent streams and vary the available computation resource on each edge server as well as network bandwidth on edge–cloud links. Figure 9 presents the results. The computation resource at edge servers is indicated by the total inference frame rate. Figure 9(a) gives the achieved mAP with the UA-DETRAC dataset.



(a) UA-DETRAC.



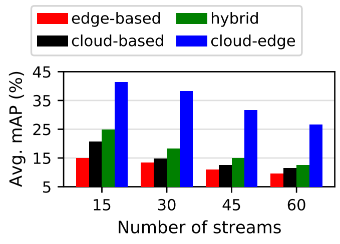
(b) Bellevue Traffic.

Fig. 9. Inference accuracy of different approaches with varied computation or network resource.

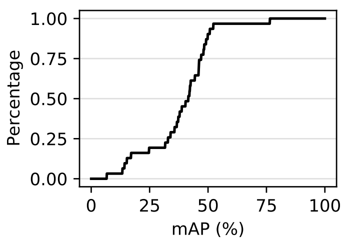
The edge server's total inference frame rate is varied from 50 to 250 FPS with fixed edge–cloud bandwidth at 50 Mbps (left figure), and then the edge–cloud bandwidth is varied from 20 to 100 Mbps with fixed edge computation resource at 100 FPS (right figure). Figure 9(b) shows the same set of results when the Bellevue Traffic dataset is used. The experimental results show that the performance of the proposed system grows monotonically with computation and network resources, which consistently outperforms all baselines under different resource bottlenecks. On the UA-DETRAC dataset, the cloud–edge collaborative design achieves up to 12.5%, 25.7%, and 4.8% mAP gain compared with edge-based, cloud-based, and hybrid inference, respectively, while on the Bellevue Traffic dataset these gains are 26.4%, 28.6%, and 22.3%.

Such results quantify how the proposed system benefits from cloud–edge collaborative continuous learning. On the one hand, when the network bandwidth is constrained, our approach gains higher network utility when using the uploaded frames as training samples rather than directly using them for cloud-side inference. On the other hand, the system can consistently improve the performance of lightweight models at the edge without introducing additional computation overhead. Note that each video sequence lasts 4 minutes in the UA-DETRAC dataset and 1 hour in Bellevue Traffic. The improved performance of our system on both datasets demonstrates the effectiveness of the continuous learning approach. This approach enhances model accuracy across various timescales, from short intervals that encompass several retraining windows to prolonged video streams.

We test the system scalability by increasing the number of video streams with constrained computation and network resources, i.e., with two edge servers capped with 100 FPS processing power and the network link capped with 50 Mbps bandwidth. Figure 10(a) presents the inference accuracy of the proposed approach in comparison with baselines. With the increased number of concurrent streams, we see all approaches experience throttled performance while the proposed system exhibits a consistent advantage over baselines with up to 26.4% mAP gain, which indicates high scalability of our system in servicing large-scale applications. Figure 10(b) presents the CDF of achieved mAPs across the 60 individual video streams with the proposed cloud–edge collaborative approach. We see balanced performance over all streams—over 95% of the streams have an

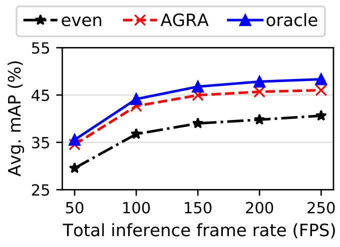


(a)

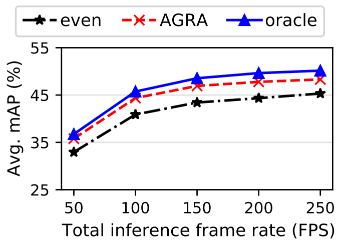


(b)

Fig. 10. (a) Inference accuracy of different approaches with a varied number of concurrent streams. (b) The CDF of achieved mAPs across 60 video streams.



(a) Avg. bandwidth of 1 Mbps.



(b) Avg. bandwidth of 1.5 Mbps.

Fig. 11. Comparison with even and oracle schemes with different computation and network bottlenecks.

mAP within the range of two standard deviations around the average, and all streams are within three standard deviations. No particular stream suffers performance degradation due to persistent resource starvation.

4.3 Breakdown Study

Resource allocation. We evaluate the effectiveness of our resource allocation approach (AGRA), compared with two alternatives: (1) *even*, where the system allocates computation and network resource evenly across all streams, and (2) *oracle*, where the system is assumed having the complete knowledge about how the allocated resources correspond to inference accuracies and allocates the resources with DP algorithm. The oracle scheme gives the theoretical upper bound gain of resource allocation and is practically impossible.

Figure 11 summarizes the achieved average mAP over all streams when different resource allocation schemes are used. In the experiment, 10 video streams are selected from UA-DETRAC dataset and placed at one edge server with varied total inference frame rate ranging from 50 to 250 FPS. When the network resource is more constrained (e.g., in Figure 11(a) where the average bandwidth per stream is 1 Mbps), oracle allocation achieves up to 8.0% mAP gain on average compared with even allocation. Meanwhile, AGRA achieves up to 6.0% mAP gain and consistently outperforms even allocation with varied computation resource bottleneck, which indicates AGRA, though not optimal, can be very close to oracle allocation and achieves non-trivial accuracy gain. When the network resource increases to 1.5 Mbps per stream in Figure 11(b), the performance gain of both oracle and AGRA over even allocation becomes slimmer, i.e., 4.2% mAP gain for oracle and 3.5% for AGRA. The reason is that the marginal gain from model retraining gradually diminishes with retraining frame rate, which makes both oracle and AGRA eventually converge to even allocation.

Table 2. The Cost of Resource Allocation When Different Schemes Are Used

Number of streams	Average cost per stream per window (s)		
	DP	DP-split	AGRA
5	27.53	12.52	9.01
10	27.58	12.51	7.23
50	27.57	12.53	5.12
100	27.59	12.52	4.82

We also evaluate the cost of our resource allocation scheme. We compare AGRA with two alternatives: (1) *DP*, the dynamic programming approach on 2D profile matrix without pruning, and (2) *DP-split*, which also uses dynamic programming but splits the 2D allocation problem to two 1D problems and allocates network and computation resources separately.

Table 2 presents the cost of online allocation measured as the average operation time per video stream per retraining window. The computation resource is fixed to a total inference frame rate of 100 FPS, and the network bandwidth is fixed to 100 Mbps. The number of concurrent video streams for resource allocation is varied from 5 to 100. The measured results show that the average online allocation cost of both *DP* and *DP-split* is constant with an increasing number of streams, suggesting a linear growth of the total cost when the system scales up with more streams. However, with the proposed pruning-based DFS approach, the amortized cost of AGRA gradually decreases with up to 5.7× speed-up. This is because when there are more streams, the DFS tree becomes shallower and an optimal allocation plan can be found with less probing, leading to more branches pruned, which makes AGRA scalable with the growth of the number of streams.

Aggregated model training. We perform experiments to demonstrate how aggregated model training can reduce online training cost at a small cost of inference accuracy. We pick 16 video streams from UA-DETRAC dataset and allocate them across 4 edge servers based on camera locations. The size of spatial aggregation is fixed to 4 (1 for each edge server). The temporal aggregation length k is varied from 1 to 4. We compare the achieved average mAP and training cost with common retraining technique that does not use aggregation.

Table 3 summarizes the average accuracy and the cost of model retraining amortized across the streams and retraining windows. The experiment is performed with two network bandwidth settings, i.e., an average uplink bandwidth of 1 and 2 Mbps per stream, respectively. The results show that aggregated model training significantly reduces the online retraining cost. The gain gets larger with temporal aggregation applied across more retraining windows (when k is increased from 1 to 4). When the average uplink bandwidth is set to 1 Mbps, aggregated and non-aggregated training achieve similar mAP. With a larger uplink bandwidth (2 Mbps per stream), the aggregated training approach may lead to slight drop of mAP (by 4% at most) but reduce up to 68.8% training cost. In our prototype implementation, we choose $k = 2$ to achieve a balanced tradeoff between accuracy and training cost.

5 RELATED WORK

Distributed video analytics. Distributed video analytics is a common way to improve system scalability. Existing studies aim at splitting a task either into multiple stages of a processing pipeline [13, 16, 27, 61] or parallel subtasks [20, 47, 63] and distributing them across machines. Among those studies, EdgeDuet [47] specifically adopts an edge-end collaborative design where hard data samples are offloaded to a powerful edge server while easy ones are kept at the end device for local processing. In such a parallel offloading setting, the cloud or edge servers are treated

Table 3. Accuracy and Training Cost with and without Aggregated Model Training

	Avg. bandwidth 1 Mbps		Avg. bandwidth 2 Mbps	
	Cost (s)	mAP (%)	Cost (s)	mAP (%)
w/o agg.	23.80	30.9	47.77	37.0
w/ spatial agg. & temporal agg. (of length k)				
$k = 1$	12.99	29.5	26.10	36.3
$k = 2$	9.26	30.1	18.37	35.5
$k = 3$	8.12	30.2	16.05	34.6
$k = 4$	7.59	29.7	14.90	33.0

as simple extension of computation capability. In our design, the cloud plays the role of retraining edge models to ensure the inference accuracy.

Continuous learning. Continuous learning aims at tackling the problem of concept drift. Continuous learning approaches have been adopted in existing video analytics designs to refine DNN models continuously online [10, 12, 26, 34, 41]. Most existing designs, however, mainly focus on training techniques without considering systematic design issues such as multi-stream resource contention. Ekya [3] proposes a continuous learning framework, which, however, builds both model inference and retraining at the edge server. Without the cloud–edge collaboration, the design is essentially limited by available computation resources at the edge. This article integrates existing continuous learning rationale into the cloud–edge collaborative architecture with system implementation and addresses specific challenges involved in resource allocation and management.

Cloud–edge learning systems. Some learning systems adopt a cloud–edge collaborative architecture [15, 18]. In such configurations, a learning task is divided into multiple steps, with a portion being offloaded to the edge. Edge servers typically manage steps that are less computationally demanding, such as data retrieval and preprocessing. This design leads to decreased uplink data transmission and enhanced resource efficiency, especially when compared to cloud-only learning systems. Moreover, cloud–edge learning systems can integrate with federated learning [28, 54, 56]. In this scenario, edge servers preprocess data to bolster privacy and anonymity prior to its transmission to the cloud. While much of the existing research on cloud–edge systems explores the use of edge computing for efficient or privacy-focused training task execution, our system focuses on video analysis, where the emphasis is on model inference with training primarily serving to improve the accuracy of lightweight models deployed at edge servers.

Dynamic configurations. Many video analytics systems support adapting configurations dynamically in an online manner [1, 11, 19, 21, 22, 24, 46, 62]. Such adaptation aims at achieving a desired tradeoff between inference accuracy and resource consumption by choosing a proper configuration, which can be spatial (resolution), temporal (frame rate), or model related (specific model settings). Most existing works, however, only consider computation resources on inference servers and focus on a single-stream scenario, where resource competition and multi-stream diversity are largely neglected. This article considers dynamic configurations within a cloud–edge collaborative system and adapts both the inference and retraining configurations for resource allocation.

6 DISCUSSION

Quantization and model pruning. Edge servers face computational resource limitations, making it challenging to run complex DNN models. To address this, two model compression techniques are introduced, i.e., quantization [17, 38] and model pruning[23, 33]. In the quantization method, model parameters are represented using lower precision formats. Meanwhile, model pruning discards parameters that have minimal impact on inference. Both methods aim to decrease the model's size and computation cost, allowing the model to better fit edge hardware constraints. In our system, the method for deriving the lightweight model at edge servers is not specified. In experiments, SSDLite with a MobileNetV2 backbone without any quantization or model pruning is used and undergoes conventional training procedures. However, when a lightweight model is derived from a compressed version of a complex model, specific training techniques designed for such compressed models, e.g., quantization-aware training [43], can be incorporated in the retraining phase to optimize continuous learning.

Usage in wireless scenarios. In our setting, the uplink bandwidth between edge servers and the cloud is relatively consistent over time. However, when edge servers utilize wireless access networks to connect to the cloud, several issues emerge. The primary concern is the simultaneous connection of multiple edge servers, which, combined with the natural interference of wireless signals, can cause significant fluctuations in wireless bandwidth. This variability can compromise the accuracy of the resource monitor in our system, leading to potential misestimations of available network resources in subsequent retraining windows, thereby negatively affecting the decision of the resource allocator. This issue can be addressed at the physical layer by mitigating Wi-Fi or cellular interference [35, 65] or at the transport layer by implementing bandwidth prediction algorithms tailored for wireless networks [52, 60]. Further exploration of these solutions is reserved for future research.

7 CONCLUSION

In this article, we present a systematic design of a video analytics framework with cloud–edge collaborative continuous learning. The resource allocator of the system manages frame rate selection for multiple streams to utilize limited computation and network resources. The aggregated model training technique saves computation resource consumption for retraining. Evaluation results show the system can achieve up to 28.6% absolute mAP gain on object detection tasks. We conclude the system is effective for real-world applications.

REFERENCES

[1] Michael R. Anderson, Michael J. Cafarella, Germán Ros, and Thomas F. Wenisch. 2019. Physical representation-based predicate optimization for a visual analytics database. In *ICDE*. IEEE, 1466–1477.

[2] James Bankoski, John Koleszar, Lou Quillio, Janne Salonen, Paul Wilkins, and Yaowu Xu. 2011. RFC 6386: VP8 Data Format and Decoding Guide.

[3] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. 2022. Ekya: Continuous learning of video analytics models on edge compute servers. In *NSDI*. USENIX Association, 119–135.

[4] Xingyuan Bu, Junran Peng, Junjie Yan, Tieniu Tan, and Zhaoxiang Zhang. 2021. GAIA: A transfer learning system of object detection that fits your needs. In *CVPR*. Computer Vision Foundation/IEEE, 274–283.

[5] Frank Cangialosi, Neil Agarwal, Venkat Arun, Srinivas Narayana, Anand D. Sarwate, and Ravi Netravali. 2022. Privid: Practical, privacy-preserving video analytics queries. In *NSDI*. USENIX Association, 209–228.

[6] Hemant M. Chaskar and Upamanyu Madhow. 2003. Fair scheduling with tunable latency: A round-robin approach. *IEEE/ACM Trans. Netw.* 11, 4 (2003), 592–601.

[7] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. 2019. MMDetection: Open MMLab detection toolbox and benchmark. *CoRR* abs/1906.07155 (2019).

[8] Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, and David Gollob. 2015. *Microsoft Azure: Planning, Deploying, and Managing Your Data Center in the Cloud* (1st ed.). Apress, USA.

[9] Erik Dahlman, Stefan Parkvall, and Johan Skold. 2013. *4G: LTE/LTE-advanced for Mobile Broadband*. Academic Press.

[10] Chuntao Ding, Ao Zhou, Yunxin Liu, Rong N. Chang, Ching-Hsien Hsu, and Shangguang Wang. 2022. A cloud-edge collaboration framework for cognitive service. *IEEE Trans. Cloud Comput.* 10, 3 (2022), 1489–1499.

[11] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. 2020. Server-driven video streaming for deep learning inference. In *SIGCOMM*. ACM, 557–570.

[12] Mohammad Farhadi, Mehdi Ghasemi, Sarma B. K. Vrudhula, and Yezhou Yang. 2020. Enabling incremental knowledge transfer for object detection at the edge. In *CVPR Workshops*. Computer Vision Foundation/IEEE, 1591–1599.

[13] Mengxi Hanyao, Yibo Jin, Zhuzhong Qian, Sheng Zhang, and Sanglu Lu. 2021. Edge-assisted online on-device object detection for real-time video analytics. In *INFOCOM*. IEEE, 1–10.

[14] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. 2017. Speed/accuracy trade-offs for modern convolutional object detectors. In *CVPR*. IEEE Computer Society, 3296–3297.

[15] Yutao Huang, Yifei Zhu, Xiaoyi Fan, Xiaoqiang Ma, Fangxin Wang, Jiangchuan Liu, Ziyi Wang, and Yong Cui. 2018. Task scheduling with optimized transmission time in collaborative cloud-edge learning. In *ICCCN*. IEEE, 1–9.

[16] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodík, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. VideoEdge: Processing camera streams using hierarchical clusters. In *SEC*. IEEE, 115–131.

[17] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*. Computer Vision Foundation/IEEE Computer Society, 2704–2713.

[18] Lin Jia, Zhi Zhou, Fei Xu, and Hai Jin. 2022. Cost-efficient continuous edge learning for artificial intelligence of things. *IEEE IoT J.* 9, 10 (2022), 7325–7337.

[19] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodík, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: Scalable adaptation of video analytics. In *SIGCOMM*. ACM, 253–266.

[20] Caihong Kai, Hao Zhou, Yibo Yi, and Wei Huang. 2021. Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability. *IEEE Trans. Cogn. Commun. Netw.* 7, 2 (2021), 624–634.

[21] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing deep CNN-based queries over video streams at scale. *Proc. VLDB Endow.* 10, 11 (2017), 1586–1597.

[22] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2020. Jointly optimizing preprocessing and inference for DNN-based visual analytics. *Proc. VLDB Endow.* 14, 2 (Oct. 2020), 87–100.

[23] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient ConvNets. In *ICLR (Poster)*. OpenReview.net.

[24] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *SIGCOMM*. ACM, 359–376.

[25] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *ECCV (5), Lecture Notes in Computer Science, Vol. 8693*. Springer, 740–755.

[26] Heting Liu and Guohong Cao. 2022. Deep learning video analytics through online learning based edge computing. *IEEE Trans. Wirel. Commun.* 21, 10 (2022), 8193–8204.

[27] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge assisted real-time object detection for mobile augmented reality. In *MobiCom*. ACM, 25:1–25:16.

[28] Tianyu Liu, Boya Di, Peng An, and Lingyang Song. 2021. Privacy-preserving incentive mechanism design for federated cloud-edge learning. *IEEE Trans. Netw. Sci. Eng.* 8, 3 (2021), 2588–2600.

[29] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single shot multibox detector. In *ECCV (1), Lecture Notes in Computer Science, Vol. 9905*. Springer, 21–37.

[30] Rui Lu, Siping Shi, Dan Wang, Chuang Hu, and Bihai Zhang. 2022. Preva: Protecting inference privacy through policy-based video-frame transformation. In *SEC*. IEEE, 175–188.

[31] Mehdi Malboubi, Liyuan Wang, Chen-Nee Chuah, and Puneet Sharma. 2014. Intelligent SDN based traffic (de)aggregation and measurement paradigm (iSTAMP). In *INFOCOM*. IEEE, 934–942.

[32] Ana I. Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso García, and Davide Scaramuzza. 2018. Event-based vision meets deep learning on steering prediction for self-driving cars. In *CVPR*. Computer Vision Foundation/IEEE Computer Society, 5419–5427.

[33] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning convolutional neural networks for resource efficient inference. In *ICLR (Poster)*. OpenReview.net.

[34] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. 2019. Online model distillation for efficient video inference. In *ICCV*. IEEE, 3572–3581.

[35] Woongsoo Na, Nhu-Ngoc Dao, and Sungrae Cho. 2016. Mitigating wifi interference to improve throughput for in-vehicle infotainment networks. *IEEE Wirel. Commun.* 23, 1 (2016), 22–28.

[36] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, Feng Qian, and Zhi-Li Zhang. 2021. A variegated look at 5G in the wild: Performance, power, and QoE implications. In *SIGCOMM*. ACM, 610–625.

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 8024–8035.

[38] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. In *ICLR (Poster)*. OpenReview.net.

[39] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. 2017. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 6 (2017), 1137–1149.

[40] Iain E. Garden Richardson. 2003. *H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia*. Wiley.

[41] Mehrdad Khani Shirkoohi, Pouya Hamadanian, Arash Nasr-Esfahany, and Mohammad Alizadeh. 2021. Real-time video inference on edge devices via adaptive model streaming. In *ICCV*. IEEE, 4552–4562.

[42] Prabhakant Sinha and Andris A. Zoltners. 1979. The multiple-choice knapsack problem. *Operat. Res.* 27, 3 (1979), 503–515.

[43] Shyam Anil Tailor, Javier Fernández-Marqués, and Nicholas Donald Lane. 2021. Degree-quant: Quantization-aware training for graph neural networks. In *ICLR*. OpenReview.net.

[44] Mingxing Tan, Ruoming Pang, and Quoc V. Le. 2020. EfficientDet: Scalable and efficient object detection. In *CVPR*. Computer Vision Foundation/IEEE, 10778–10787.

[45] Chuan Wang, Hua Zhang, Liang Yang, Si Liu, and Xiaochun Cao. 2015. Deep people counting in extremely dense crowds. In *ACM Multimedia*. ACM, 1299–1302.

[46] Can Wang, Sheng Zhang, Yu Chen, Zhuzhong Qian, Jie Wu, and Mingjun Xiao. 2020. Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics. In *INFOCOM*. IEEE, 257–266.

[47] Xu Wang, Zheng Yang, Jiahang Wu, Yi Zhao, and Zimu Zhou. 2021. EdgeDuet: Tiling small object detection for edge assisted autonomous mobile vision. In *INFOCOM*. IEEE, 1–10.

[48] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. 2018. IntelliLight: A reinforcement learning approach for intelligent traffic light control. In *KDD*. ACM, 2496–2505.

[49] Longhui Wei, Shiliang Zhang, Hantao Yao, Wen Gao, and Qi Tian. 2017. GLAD: Global-local-alignment descriptor for pedestrian retrieval. In *ACM Multimedia*. ACM, 420–428.

[50] Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. 2020. UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking. *Comput. Vis. Image Underst.* 193 (2020), 102907.

[51] Gerhard Widmer and Miroslav Kubat. 1996. Learning in the presence of concept drift and hidden contexts. *Mach. Learn.* 23, 1 (1996), 69–101.

[52] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *NSDI*. USENIX Association, 459–471.

[53] Hao Wu, Xuejin Tian, Minghao Li, Yunxin Liu, Ganesh Ananthanarayanan, Fengyuan Xu, and Sheng Zhong. 2021. PECAM: Privacy-enhanced video streaming and analytics via securely-reversible transformation. In *MobiCom*. ACM, 229–241.

[54] Qiong Wu, Kaiwen He, and Xu Chen. 2020. Personalized federated learning for intelligent IoT applications: A cloud-edge based framework. *IEEE Open J. Comput. Soc.* 1 (2020), 35–44.

[55] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. 2017. End-to-end learning of driving models from large-scale video datasets. In *CVPR*. IEEE Computer Society, 3530–3538.

[56] Xiaolong Xu, Wentao Liu, Yulan Zhang, Xuyun Zhang, Wanchun Dou, Lianyong Qi, and Md. Zakirul Alam Bhuiyan. 2022. PSDF: Privacy-aware IoV service deployment with federated learning in cloud-edge computing. *ACM Trans. Intell. Syst. Technol.* 13, 5 (2022), 70:1–70:22.

[57] Keren Ye, Adriana Kovashka, Mark Sandler, Menglong Zhu, Andrew G. Howard, and Marco Fornoni. 2020. SpotPatch: Parameter-efficient transfer learning for mobile object detection. In *ACCV (6), Lecture Notes in Computer Science, Vol. 12627*. Springer, 239–256.

[58] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. 2018. Deep layer aggregation. In *CVPR*. Computer Vision Foundation / IEEE Computer Society, 2403–2412.

[59] Zhengxu Yu, Shuxian Liang, Long Wei, Zhongming Jin, Jianqiang Huang, Deng Cai, Xiaofei He, and Xian-Sheng Hua. 2020. MaCAR: Urban traffic light control via active multi-agent communication and action rectification. In *IJCAI*. ijcai.org, 2491–2497.

[60] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive congestion control for unpredictable cellular networks. In *SIGCOMM*. ACM, 509–522.

[61] Xiao Zeng, Biyi Fang, Haichen Shen, and Mi Zhang. 2020. Distream: Scaling live video analytics with workload-adaptive distributed edge intelligence. In *SenSys*. ACM, 409–421.

[62] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodík, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live video analytics at scale with approximation and delay-tolerance. In *NSDI*. USENIX Association, 377–392.

[63] Wuyang Zhang, Zhezhi He, Luyang Liu, Zhenhua Jia, Yunxin Liu, Marco Gruteser, Dipankar Raychaudhuri, and Yanyong Zhang. 2021. Elf: Accelerate high-resolution mobile deep vision with content-aware parallel offloading. In *MobiCom*. ACM, 201–214.

[64] Yiru Zhao, Bing Deng, Chen Shen, Yao Liu, Hongtao Lu, and Xian-Sheng Hua. 2017. Spatio-temporal autoencoder for video anomaly detection. In *ACM Multimedia*. ACM, 1933–1941.

[65] Xiaolong Zheng, Zhichao Cao, Jiliang Wang, Yuan He, and Yunhao Liu. 2014. ZiSense: Towards interference resilient duty cycling in wireless sensor networks. In *SenSys*. ACM, 119–133.

Received 18 April 2023; revised 1 September 2023; accepted 11 September 2023