

ECCO: Leveraging Cross-Camera Correlations for Efficient Live Video Continuous Learning

ABSTRACT

Recent advances in video analytics address real-time data drift by continuously retraining specialized, lightweight DNN models for individual cameras. However, the current practice of retraining a separate model for each camera suffers from high compute and communication costs, making it unscalable. We present *ECCO*, a new video analytics framework designed for *resource-efficient* continuous learning. The key insight is that the data drift, which necessitates model retraining, often shows temporal and spatial correlations across nearby cameras. By identifying cameras that experience similar drift and retraining a shared model for them, *ECCO* can substantially reduce the associated compute and communication costs. Specifically, *ECCO* introduces: (i) a lightweight grouping algorithm that dynamically forms and updates camera groups; (ii) a GPU allocator that dynamically assigns GPU resources across different groups to improve retraining accuracy and ensure fairness; and (iii) a transmission controller at each camera that configures frame sampling and coordinates bandwidth sharing with other cameras based on its assigned GPU resources. We conducted extensive evaluations on three distinctive datasets for two vision tasks. Compared to leading baselines, *ECCO* improves retraining accuracy by 6.7%-18.1% using the same compute and communication resources, or supports 3.3 \times more concurrent cameras at the same accuracy.

1 INTRODUCTION

The rapid expansion of camera deployments [1, 12, 18] is driving a growing demand for live video analytics, with the market expected to reach 22.6 billion USD by 2028 [34]. Live video analytics uses deep neural network (DNN) models to perform vision tasks such as object detection and classification. These analytics are at the core of applications in diverse fields like enterprise security [30], traffic monitoring [44], and autonomous driving [58]. To process live video streams in real time and ensure low-latency results, it is often crucial to deploy DNNs and run inference directly on edge devices [5, 20]. However, since edge devices often have limited resources (with less powerful GPUs [2, 3]), these devices typically use lightweight, specialized models instead of complex, generic models [17, 36].

These lightweight models are initially trained using representative data from each camera. Once deployed, these models face challenges from *data drift*, where the live video

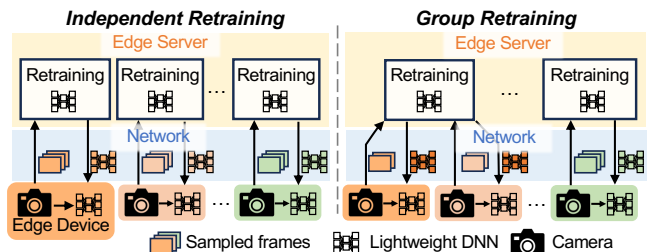


Figure 1: Frameworks of existing continuous retraining systems (left) and our proposed system (right).

data changes significantly from the initial training data. For example, cameras deployed in urban areas or mounted on moving vehicles may record changes in object types, activities, lighting conditions, or crowd densities over time. As a result, the accuracy of the DNN’s predictions can substantially decline. A promising solution to tackle data drift is *continuous learning*. This approach retrains the lightweight models on an edge server using more recent video frames transmitted from the cameras, helping the models to adapt to new data patterns. Recent studies [25, 37] have shown the benefits of continuous learning in enhancing the robustness and accuracy of video analytics systems.

A key challenge in continuous learning is resource efficiency, as practical deployments are often constrained by compute and communication resources. Two critical resources in this setting are the GPUs at the edge server and the bandwidth between the distributed cameras and the server. Prior works have only focused on improving GPU efficiency (i.e., model accuracy per GPU unit). For example, Ekya [7] and AdaInf [45] optimize GPU scheduling across retraining (and inference) tasks from multiple cameras. RECL [24] enhances GPU efficiency by reusing historical models as starting points for retraining. However, these methods have two significant limitations. First, they largely overlook bandwidth efficiency, which is related to GPU usage and should be optimized jointly. Second, all these systems assume training a separate model for each camera, a strategy we refer to as “*independent retraining*” (see Fig. 1 (left)). This design can result in redundant GPU computation, especially when cameras exhibit correlated data patterns.

Core idea: We introduce *group retraining*, a new approach that groups cameras experiencing similar data drift and re-trains a shared model using their collective data (see Fig. 1 (right)). The rationale for group retraining is two-fold. First, data drift often exhibits temporal and spatial correlation

among some cameras, e.g., traffic cameras at the same intersection or cameras mounted on vehicles traveling together may encounter similar environmental changes. Second, lightweight models, despite their compact architecture, can generalize across similar data distributions and may even benefit from subtle variations observed by different cameras [28, 32]. By retraining one model for a group of cameras instead of individual models for each, we reduce the number of models that need retraining. This reduces compute costs and lowers the data transmission requirements for each camera to the edge server.

Technical challenges: Realizing the potential of group retraining involves addressing several system-level challenges. First, efficiently identify cameras with similar data drift is challenging because video streams are high-dimensional, making direct cross-camera comparisons computationally expensive. Moreover, as live video content changes continuously, camera grouping cannot be a one-time operation and must be updated over time. Second, allocating GPU resources efficiently across camera groups is non-trivial. As we will show in §3.1, naive extensions of existing GPU allocation algorithms are ill-suited for group retraining. They tend to favor larger groups while under-provisioning smaller ones, resulting in unfairness among cameras. Third, in addition to GPU resources, network bandwidth is a critical yet often overlooked constraint. Since different groups’ models are retrained using live data transmitted from their distributed cameras, bandwidth allocation across groups must be coordinated with GPU allocation to maximize retraining efficiency. This coordination is challenging because it often requires assigning unequal bandwidth shares to different groups, which may conflict with the equal-share behavior enforced by standard congestion control mechanisms.

Our solution: We introduce *ECCO*, a new continuous retraining framework for live video analytics that leverages cross-camera correlations to improve resource efficiency, scalability, and responsiveness. *ECCO* addresses the above challenges through three key strategies: (i) A lightweight grouping algorithm that first narrows down candidate cameras using metadata (e.g., drift time and location), and then makes grouping decisions by evaluating the accuracy gain. (ii) A new formulation of the GPU allocation problem tailored for group retraining, along with a GPU allocator that jointly optimizes for overall performance and fairness across groups. (iii) A transmission controller at each camera that adapts frame sampling (data volume) to match its assigned GPU resource and adjusts its transmission rate using a customized congestion control algorithm, that enables bandwidth allocation across groups in proportion to their GPU shares in a best-effort manner.

We implemented and evaluated *ECCO* on two computer vision tasks: object detection and instance segmentation, and compared it with state-of-the-art video analytics systems for three datasets. Using the same compute and communication resources, *ECCO* improves the mean Average Precision (mAP) by 6.7%-16.6% for object detection and 9.3%-18.1% for instance segmentation over strong baselines. While this improvement may appear to be a small change in accuracy, in practice this improvement translates to supporting 3.3× more cameras at the same accuracy level. We also show that *ECCO*’s GPU allocator optimizes both performance and fairness, which significantly reduces the accuracy gap across groups while maintains comparable overall accuracy to the baseline allocator. The transmission controller improves bandwidth efficiency, requiring only 25%-33% of the bandwidth used by baselines to reach similar accuracy with the same GPU budget. Another finding is that *ECCO*’s advantage in responsiveness becomes more pronounced under low-bandwidth conditions, reducing retraining latency by more than 5× due to group retraining’s effective data aggregation and natural model reuse within the group.

2 MOTIVATION AND OBSERVATIONS

2.1 Limitations of Independent Retraining

To highlight the inefficiencies of current retraining approaches, we first revisit the standard retraining pipeline shown in Fig. 1 (left). Edge devices perform real-time inference on live video using local lightweight models (“students”). When data drift is detected¹, the edge device sends a retraining request to the edge server and begins continuously sampling and transmitting video frames as retraining data. The server uses a high-accuracy but resource-intensive model (“teacher”) to annotate these frames, retrains a separate student model for each camera using its own data, and returns the updated model to the corresponding device.

This design faces an obvious scalability challenge: as the number of cameras increases, the retraining workload grows linearly, placing significant strain on both GPU and bandwidth resources, which increases retraining latency. Moreover, when multiple cameras experience similar data drift, independently retraining separate models becomes redundant and inefficient.

2.2 Why Group Retraining?

Cross-camera correlations in data drift. The main insight motivating our work is that data drift among cameras can exhibit spatial and temporal correlations. Cameras that are geographically close often experience similar environmental changes at similar times. Fig. 2(a) and (b) illustrate examples

¹Several prior works [4, 21, 40] have been proposed to detect data drift or scene changes in video streams and can serve as retraining triggers.

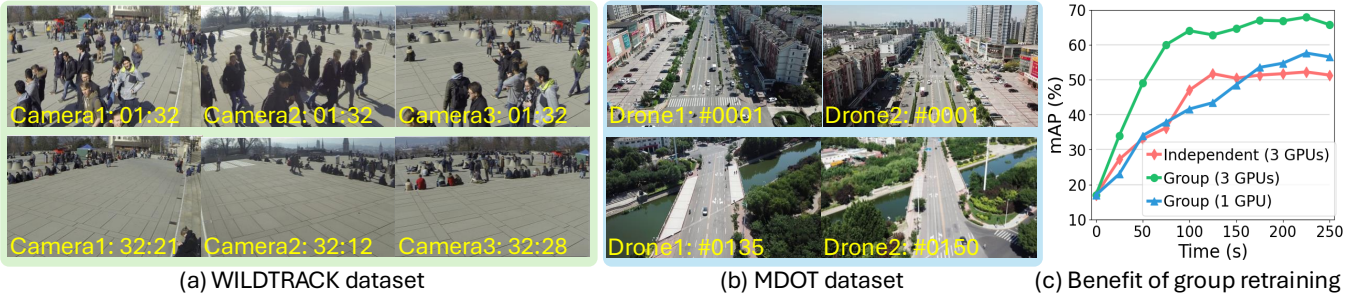


Figure 2: A motivation study. (a) & (b): Example frames from two public datasets showing that data drift exhibits significant temporal and spatial correlations among cameras. (c) Model accuracy of three retraining settings.

of such drift using two representative camera types: static cameras from the WILDTRACK [8] dataset and mobile cameras from the MDOT [35] dataset. In WILDTRACK, static cameras monitoring a plaza show similar changes in foreground content (e.g., pedestrian density and behavior). In MDOT, mobile cameras mounted on drones flying in formation capture similar shifts in both foreground (e.g., vehicle densities) and background (e.g., buildings and vegetation) as they move through urban and suburban areas. This phenomenon is also observed in other public datasets such as Cityflow [47], Bellevue Traffic Video [10], VERI-Wild [33], and DukeMTMC [43]. These examples suggest that independent retraining can be inefficient when data distributions across cameras are closely aligned.

We propose a new approach, *group retraining*, which aggregates retraining requests from cameras with correlated data drift and uses their collective data to retrain a single shared model. While prior work has used inter-camera correlations for tasks such as object re-identification [19] or video inference configuration [22], our approach is, to our knowledge, the first to exploit these correlations to reduce the cost of model retraining.

Benefits of group retraining. To demonstrate the potential benefits of group retraining, we conduct a case study using three drone videos selected from the MDOT dataset, identified as correlated based on manual inspection. These videos exhibit similar scene changes as the drones fly in formation, resembling the scenario illustrated in Fig. 2(b). We use a high-performance object detection model, YOLO11x (194.9 BFLOPs), as the teacher and a lightweight version, YOLO11n (6.5 BFLOPs), as the student model. We compare three retraining settings: (i) Independent retraining: Each camera retrains its own model using 1 GPU (3 GPUs in total). (ii) Group retraining (3 GPU): A shared model is trained using data from all three cameras with 3 GPUs. (iii) Group retraining (1 GPU): The shared model is retrained using the same data with only 1 GPU.

Fig. 2(c) reports object detection accuracy over time, measured by mean Average Precision (mAP) averaged across

the three cameras. We observe that: (i) With the same GPU resources, group retraining (green) achieves higher accuracy and lower retraining latency than independent retraining (red). (ii) Even with only 1 GPU, group retraining performs comparably to independent retraining using 3 GPUs. These results suggest that group retraining can significantly reduce retraining cost in GPU use, while improving responsiveness.

3 DESIGN OF ECCO

This paper proposes *ECCO*, a continuous learning system for live video analytics that improves compute and communication efficiency by leveraging cross-camera correlations through group retraining.

Overall architecture (Fig.3 and Fig.4): *ECCO* comprises a server, multiple edge devices (cameras), and the network connecting them. We make no assumptions about the underlying network connectivity.

Fig.3 illustrates steady-state operations, where cameras have already been grouped based on data drift similarity. Each group’s retraining is handled by a single job. Retraining is managed in discrete retraining windows, which serve as the basic unit for coordination and resource management. Within each retraining window, *ECCO* improves resource efficiency and retraining accuracy through two coordinated modules.

The GPU allocator at the server dynamically distributes GPU resources across groups, guided by an objective that balances overall retraining performance with fairness among groups (§3.1). The GPU allocation information is transmitted to cameras to guide their transmission strategies.

To maximize retraining accuracy, the transmission controller at each camera configures training data and regulates transmission to match the allocated GPU budget. First, it selects a sampling configuration (frame rate and resolution) based on both the GPU budget and the observed scene characteristics (§3.2.1). Second, it aligns its bandwidth usage with the assigned GPU share. This is achieved using a customized GAIMD congestion control algorithm [55],

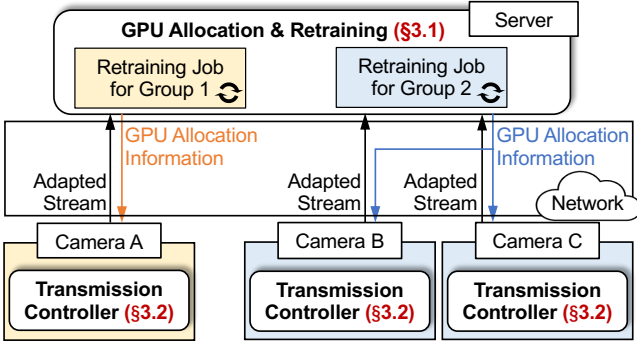


Figure 3: Steady state of ECCO. The server-side GPU allocator and camera-side transmission controllers coordinate to support group retraining. Colors indicate different groups.

which enforces flows to compete with controlled aggressiveness proportional to their GPU share, thereby approximating GPU-proportional bandwidth allocation under network constraints (§3.2.2). While non-AIMD approaches are possible, they are not the focus of our efforts.

Fig. 4 illustrates how camera groups are created and updated in response to data drift. Camera grouping involves two stages. The first is initial grouping, triggered when a camera detects drift and initiates a retraining request. At this point, the server determines whether the camera should join an existing group that targets similar data drift or form a new group. The second is regrouping, performed periodically on existing groups during retraining. This process adapts group membership to evolving data distributions in live video streams, as some cameras may gradually drift and no longer remain similar to others in their current group.

3.1 GPU Allocation for Group Retraining

To scale and serve more cameras, ECCO must efficiently allocate GPU resources across camera groups. Unlike existing allocation methods designed for independent retraining [7, 24, 45], group retraining introduces new challenges that those methods do not address. We thus formulate a new optimization objective that balances overall retraining accuracy with fairness across groups. To achieve this, we design a GPU allocation algorithm that tracks each group’s current accuracy and its accuracy improvement with added compute resources. It then dynamically allocates more GPU resources to groups that either have low accuracy or show larger improvement when given more GPU resources.

Limitations of existing approaches: Prior work typically allocates GPU resources to maximize total accuracy improvement (or average accuracy) across all cameras over a retraining window. However, directly applying this strategy to group retraining introduces bias: it implicitly favors larger

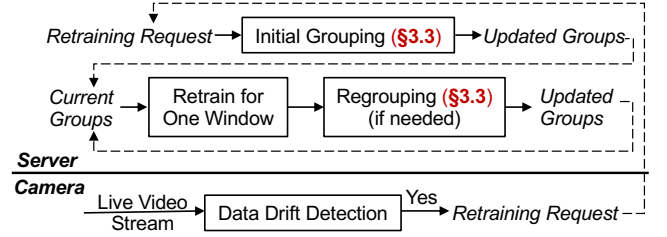


Figure 4: Dynamic camera grouping in ECCO. Initial grouping is triggered by retraining requests, while periodic regrouping updates groups during retraining.

groups, since improvements from a group with more cameras contribute more to the global accuracy sum.

To illustrate this issue, consider two groups: G1 with four cameras and G2 with one. If given equal GPU time, G1’s model accuracy improves by 10%, and G2’s by 15%. Since G1’s improvement benefits four cameras, its total contribution is $4 \times 10\% = 40\%$, while G2’s is only 15%. Thus, existing algorithms [7, 24] tend to allocate most GPU resources to G1, leading to prolonged starvation for G2—until the training efficiency of G1 (i.e., accuracy gain per GPU second) naturally declines as training converges. The root cause is that existing objectives implicitly weight groups by size, which is suitable for independent retraining but introduces a systematic allocation bias against smaller groups in the group retraining setting.

Objective reformulation: To address this, we define a new objective that balances system-wide accuracy and fairness across groups. Consider a set of retraining jobs \mathcal{J} , one per group, running on G GPUs over a retraining window T of duration $\|T\|$, with total compute capacity $G\|T\|$ GPU-time. Let $A_j(g_j)$ be the accuracy of group j ’s model after receiving g_j GPU-time, averaged over its n_j camera members. We optimize:

$$\max_{\{g_j\}_{j \in \mathcal{J}}} \left[\alpha \frac{\sum_{j \in \mathcal{J}} n_j^\beta A_j(g_j)}{\sum_{j \in \mathcal{J}} n_j^\beta} + \min_{j \in \mathcal{J}} A_j(g_j) \right] \text{ s.t. } \sum_{j \in \mathcal{J}} g_j \leq G\|T\| \quad (1)$$

The first term captures the overall average accuracy across all groups, weighted by n_j^β , where $\beta \leq 1$ controls the influence of group size, i.e., how much distinction is made between groups of different sizes. The second term promotes fairness by maximizing the worst-performing group’s accuracy. The parameter α adjusts the balance between the two goals.

GPU allocation algorithm: We build on the resource allocation algorithm in [24] and propose a modified version tailored to our optimization objective. We time-share the GPU across multiple retraining jobs by dividing a retraining window T into W micro-windows. During each micro-window, one retraining job exclusively uses all GPUs. The key idea

Algorithm 1 GPU Allocation for Group Retraining

```
1: Input: retraining jobs  $\mathcal{J}$ , retraining window size of  $W$ 
   micro-windows, constants  $\alpha, \beta$ 
2: budget  $\leftarrow W$ , initialize  $Acc[], AccGain[], ObjGain[]$ 
3: procedure MICRORETRAINING( $j$ )
4:    $acc_i \leftarrow j.EVAL()$ 
5:   Train job  $j$  for one micro-window
6:    $acc_f \leftarrow j.EVAL()$ 
7:   budget  $\leftarrow$  budget  $- 1$ 
8:    $Acc[j] \leftarrow acc_f$ ;  $AccGain[j] \leftarrow acc_f - acc_i$ 
9: procedure CALOBJECTIVEGAIN
10:  for  $j$  in  $\mathcal{J}$  do
11:     $ObjGain[j] \leftarrow \frac{an_j^\beta}{\sum_{j \in \mathcal{J}} n_j^\beta} AccGain[j]$ 
12:     $ObjGain[\text{argmin}(Acc)] += AccGain[\text{argmin}(Acc)]$ 
13:  for  $j$  in  $\mathcal{J}$  do ▷ Initial training pass
14:    MicroRetraining( $j$ ); CalObjectiveGain()
15: Estimate per-group GPU resource (§3.2)
16: while budget  $> 0$  do
17:    $j \leftarrow \text{argmax}(ObjGain)$ 
18:   MicroRetraining( $j$ ); CalObjectiveGain()
```

is to greedily assign GPU time to the group that yields the highest marginal improvement in the objective Eq. 1.

Algorithm 1 outlines the procedure. It starts with an initial training pass (Lines 13–14), where each job trains for one micro-window to establish its short-term accuracy trajectory. After this, we measure each job’s “accuracy gain”, defined as the improvement in accuracy before and after training (Lines 3–8). This is then converted into a job-specific “objective gain” (Lines 9–12), which estimates its contribution to the overall objective (Eq.1). For most jobs, the objective gain corresponds to the first term of Eq.1, $\frac{an_j^\beta}{\sum_{j \in \mathcal{J}} n_j^\beta} AccGain$, capturing their marginal contribution to the weighted average accuracy. For the lowest-accuracy job, we also include the second term of Eq. 1, giving $(\frac{an_j^\beta}{\sum_{j \in \mathcal{J}} n_j^\beta} + 1)AccGain$, where the additional $AccGain$ serves as a fairness bonus to promote balance and prevent starvation. After this initialization, the algorithm repeatedly selects the job with the highest objective gain and assigns it the next micro-window (Lines 16–18). Objective gains are updated after each micro-window to reflect the latest performance trends. This greedy allocation continues until the GPU time budget is exhausted, approximating maximization of Eq. 1 under the GPU constraint.

GPU allocation estimation for transmission control:

While the actual GPU allocation is performed dynamically as detailed above, the transmission controller at each camera (§3.2) requires an upfront estimate of its group’s expected

compute budget to guide transmission. To provide this signal, the server uses the objective gains from the initial training pass to estimate each group’s GPU share for the current window (Line 15). Specifically, the estimated GPU resource c_j for group j is proportional to its objective gain relative to all groups: $c_j = \frac{ObjGain[j]}{\sum_{i \in \mathcal{J}} ObjGain[i]} G\|T\|$. We also define $p_j = \frac{ObjGain[j]}{\sum_{i \in \mathcal{J}} ObjGain[i]}$ as the normalized GPU share weight for group j . The pair (c_j, p_j) is then communicated to the group’s cameras as GPU allocation information.

3.2 Resource-Aware Transmission Control

At the camera side, video transmission is governed by two external resources: GPU allocation and bandwidth availability. GPU allocation is decided by the server, which determines both (i) the group’s capacity to consume data and (ii) its relative priority for receiving more data. In practice, this capacity can be expressed as the maximum number of pixels per second that the GPU can process. Bandwidth availability, in contrast, is not explicitly allocated; it is constrained by network conditions and realized through congestion control.

Given these constraints, the camera controls three parameters of transmission: frame rate, resolution, and compression level. We refer to frame rate and resolution together as the *sampling configuration*. Whenever the server provides updated GPU allocation information, the camera selects a sampling configuration whose frame rate–resolution product (pixels per second) stays within the GPU budget; multiple valid choices exist, and we discuss selection strategies in §3.2.1. During streaming, the camera then adjusts the compression level continuously to ensure the selected configuration can be delivered within the bandwidth actually achieved on the network. To approximate GPU-proportional weighted bandwidth allocation while respecting network constraints, we employ a customized GAIMD congestion control algorithm.

3.2.1 Adaptation of Sampling Configuration. The sampling configuration impacts retraining quality because a limited GPU budget caps training throughput (e.g., total number of pixels processed per second [46]). As a result, increasing frame rate often requires lowering resolution, and vice versa. Retraining performance is therefore tightly coupled with how this tradeoff is managed under GPU constraints.

To study this tradeoff, we conduct a case study using two representative camera types: a static, high-mounted traffic camera (A) and a mobile, vehicle-mounted camera (B), both simulated in the CARLA [13] autonomous driving simulator (see Fig. 5). Using the model setup in §2.2, we retrain a model for each camera using various sampling configurations while keeping the GPU budget fixed. To ensure a fair comparison, we fix the transmission bitrate to 1 Mbps across all settings. This isolates the impact of sampling choices under consistent

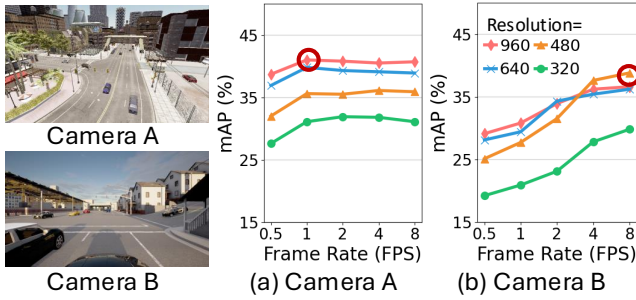


Figure 5: Impact of sampling configurations (frame rate and resolution) on retraining accuracy under a fixed GPU budget for two distinct camera types.

GPU and bandwidth conditions. Fig. 5 shows the resulting retraining accuracy across configurations.

Two observations emerge. First, with the same GPU budget, retraining accuracy varies significantly across sampling configurations—by up to 2 \times . Second, the optimal configuration (circled in the figure) varies across camera types. This variation stems from intrinsic differences in camera characteristics, such as placement and mobility. The static camera benefits more from higher resolution to capture small, distant objects, while the mobile camera benefits more from higher frame rates to adapt to rapid scene changes. These results highlight the importance of camera-specific configuration based on GPU budgets.

We leverage this observation to guide the design of our sampling configuration selection method. First, each camera conducts offline profiling to evaluate the accuracy of different frame rate and resolution combinations (f, q) across different GPU budget levels. Given that retraining occurs within fixed-length retraining windows, which are further discretized into microwindows, the number of distinct GPU resource levels is limited. This results in a lookup table that maps GPU budgets to their corresponding optimal sampling configuration.

At runtime, upon receiving its group’s GPU allocation c_j , each camera queries its table to select the optimal sampling configuration (f^*, q^*). To balance data contributions across the n_j group members, it scales the frame rate to f^*/n_j while keeping q^* unchanged. This ensures that the total volume of sampled frames aligns with the group’s compute capacity.

3.2.2 Adaptive Weighted Bandwidth Allocation. After sampling, each camera encodes frames based on available bandwidth and transmits them to the server. Since they all send to the same destination, cameras naturally share one or more network links, leading to contention and bandwidth bottlenecks. A naïve equal-allocation strategy can result in inefficient bandwidth usage, especially considering GPU resources can be allocated unevenly across cameras. We argue that “utility-based” bandwidth allocation, where each camera’s

Table 1: Retraining accuracy under equal vs. GPU-proportional bandwidth allocation.

BW allocation schemes	Camera A mAP (%)	Camera B mAP (%)	Overall mAP (%)
Equal (1.5 Mbps each)	34.7	26.1	30.4
Proportional (0.9/2.1)	33.4	30.8	32.1

bandwidth share is proportional to its GPU allocation, better aligns data delivery with compute capacity and leads to improved retraining quality.

To illustrate this, we conduct a simple case study using cameras A and B in Fig. 5. Camera A starts retraining with a model accuracy of 28% mAP, and camera B with 16%. To help B catch up and improve overall performance, we allocate 30% of the GPU to A and 70% to B. Each camera uses its optimal sampling configuration under the assigned GPU budget. We fix the total uplink bandwidth to 3 Mbps and compare two bandwidth allocation strategies: equal allocation vs. GPU-proportional allocation (i.e., 0.9 Mbps for A and 2.1 Mbps for B).

Table 1 shows that equal bandwidth allocation leads to lower retraining accuracy for the high-GPU camera B, likely due to delayed, dropped, or degraded frames. This results in under-utilization of compute resources and reduced overall retraining accuracy. In contrast, proportional allocation allows each camera to deliver training frames in time to match its GPU share, thus improving retraining accuracy.

Achieving such GPU-proportional weighted bandwidth allocation is however challenging because of the distributed and heterogeneous nature of the network. Cameras connect to the server through diverse paths with varying conditions, and two types of constraints can occur together: (i) multiple cameras may share an uplink bottleneck with unknown capacity; and (ii) individual cameras, especially mobile ones, may also be constrained by their own weak local links.

To address this, we design a distributed rate control mechanism based on the GAIMD (Generalized Additive Increase Multiplicative Decrease) congestion control algorithm [55]. The core idea is to scale each camera’s aggressiveness in bandwidth competition according to its GPU share. By customizing the GAIMD parameters, each camera converges to a steady-state sending rate that approximates GPU-proportional bandwidth sharing.

Specifically, each camera tunes its GAIMD parameters, the additive increase factor α and multiplicative decrease factor β , to control its transmission rate. This design leverages a known result: the steady-state throughput of a GAIMD flow is roughly proportional to $\alpha/(1 - \beta)$ [56]. Based on this, upon receiving the GPU share weight p_j from the server, each camera fixes $\beta = 0.5$ and sets $\alpha = p_j/n_j$, where n_j is the number of cameras in its group j . The resulting GAIMD rate serves as the target sending rate for the video stream.

Algorithm 2 Dynamic Camera Grouping Algorithm

```
1: Input: retraining jobs  $\mathcal{J}$ , new retraining request  $\hat{r}$ 
2: procedure GROUPREQUEST( $\mathcal{J}, \hat{r}$ )
3:   for  $j$  in  $\mathcal{J}$  do
4:     if  $\forall r$  in  $j, |r.t - \hat{r}.t| \leq \varepsilon$  and  $|r.loc - \hat{r}.loc| \leq \delta$ 
       then ▷ Correlation filtering
5:        $acc_j \leftarrow j.EVAL(\hat{r}.subsamples)$ 
6:       if  $acc_j \geq \hat{r}.acc$  then ▷ Performance check
7:          $JobCandidate[j] \leftarrow acc_j$ 
8:   if  $JobCandidate \neq \emptyset$  then
9:      $\hat{j} \leftarrow \text{argmax}(JobCandidate); \hat{j} \leftarrow \hat{j} \cup \{\hat{r}\}$ 
10:  else
11:     $\hat{j} \leftarrow \text{InitializeNewJob}(\hat{r}); \mathcal{J} \leftarrow \mathcal{J} \cup \{\hat{j}\}$ 
12: procedure UPDATEGROUPING( $\mathcal{J}$ )
13:   while each retraining window  $n$  ends do
14:     for  $j$  in  $\mathcal{J}$  do
15:       for  $r$  in  $j$  do
16:          $r.acc_n \leftarrow j.EVAL(r.subsamples)$ 
17:         if  $\frac{r.acc_n - r.acc_{n-1}}{r.acc_{n-1}} < -p$  ( $p > 0$ ) then
18:            $j.Remove(r); r.Update(t, loc);$ 
19:           GroupRequest( $\mathcal{J}, r$ )
```

The video encoder tracks this target and adjusts compression (quantization) in real time to stay close to it, while keeping the selected frame rate and resolution fixed.

A key advantage of this approach is that it requires neither explicit coordination between cameras nor knowledge of the network topology. It automatically adapts to both shared bottlenecks and individual camera constraints. While some cameras may fall short of their target bandwidth due to local network conditions, the system still approximates GPU-proportional bandwidth allocation in a best-effort manner.

3.3 Dynamic Camera Grouping

While the previous two subsections focus on steady-state operations for group retraining, *ECCO* must also handle the creation and update of camera groups to make retraining effective. This includes initial grouping to accommodate new retraining requests and periodic checks to determine whether existing groups remain valid or require regrouping. To keep the process lightweight, *ECCO* exploits metadata for fast pre-filtering, which narrows down candidate groups and avoids unnecessary computation. Final grouping and regrouping decisions are then based on actual accuracy improvements that directly reflect the goal of retraining. Algorithm 2 outlines this process.

Grouping initialization for new retraining requests: At runtime, each camera detects data drift locally and issues a retraining request when drift is observed. Several existing techniques [4, 21, 40] can be used for drift detection. The

request includes metadata (request time and location), sampled frames, and a copy of the device’s lightweight model. Upon receiving the request, *ECCO* checks whether any ongoing retraining jobs (groups) show temporal and spatial correlations with the new request. It compares the new request’s metadata against those in ongoing jobs to determine if they fall within a predefined time window and geographical range (Line 4). The rationale is that cameras operating close in space often experience similar data drifts at similar times. If no correlated group is found, *ECCO* initiates a new retraining job for the request, starting with the device’s sent model and sampled frames (Line 11).

If correlated groups are found, *ECCO* evaluates these jobs’ model performance using a small subset of the sample frames from the new request (Line 5). The model that shows the most significant improvement on the new request is selected, and the new request is then integrated into the corresponding job; this involves adding its metadata to the job’s metadata and aggregating its sample frames into the job’s training data (Line 9). This performance check ensures that the grouping decision is grounded in actual model accuracy rather than relying solely on metadata similarity, thereby preventing incorrect group assignments that could harm retraining accuracy. Since this grouping process is efficient, the delay it introduces is negligible compared to the duration of a retraining window.

Periodic reevaluation of existing groups: Since video content evolves over time, retraining groups must be updated dynamically. For example, mobile cameras in a fleet may initially share similar scenes and be grouped together, but changes in routes can cause their data distributions to diverge. To account for such dynamics, *ECCO* periodically reassesses existing groups at the end of each retraining window. It iteratively evaluates the performance of every group model on each camera member and compares it to the performance in the previous window (Line 16-17). If a camera’s accuracy drops beyond a threshold, it signals that the camera has undergone a second drift during the window and no longer aligns with the group’s data distribution. The camera is then removed from the group. There can be situations where only some cameras are removed from a group, and others where all are removed due to group-wide new drift. In either case, each removed camera is treated as a “new” retraining request with updated metadata and reprocessed through the initial grouping logic. This allows removed cameras to form new groups if they now share similar data distributions.

4 IMPLEMENTATION AND EXPERIMENTAL SETUP

We implement *ECCO* in Python, using the Ultralytics framework [49] (built on PyTorch [39]) for both model training and

inference. The system runs on a server equipped with five NVIDIA GeForce RTX 4090 GPUs. To emulate cameras and their transmission to the server, we use a hybrid setup that combines Docker-based container emulation and NS-3 [42] simulation.

For each retraining window, we run an NS-3 simulation to generate per-camera bandwidth traces over time, based on GAIMD-driven, GPU-aware bandwidth allocation (§3.2.2). Each camera is emulated as an independent Docker container on the server. At the start of each window, each container encodes its video at a selected frame rate and resolution (§3.2.1) using FFmpeg. During streaming, video is split into 1-second segments. For each segment, FFmpeg’s target bitrate is set to the average bandwidth of the corresponding NS-3 trace segment, so that compression adapts automatically. Linux traffic control (tc) enforces the NS-3 trace by shaping outgoing packets to match the simulated bandwidth.

Datasets: We evaluate *ECCO* using three diverse datasets. (i) *CityFlow dataset* [47] includes over 3 hours of synchronized videos captured by 40 traffic cameras at 10 city intersections. (ii) *MDOT dataset* [35] consists of a total of 155 groups of video clips (totaling more than 259k frames) captured by five drones. We arranged video clips from the same camera chronologically to create continuous video streams. (iii) *CARLA simulated datasets*: The core concept behind *ECCO* involves leveraging the correlations between multiple cameras. To assess how these correlations affect *ECCO*’s performance, it is essential to have datasets that feature videos with varying degrees of similarity. However, such datasets are rare in public collections. To address this, we employ the CARLA simulator [13] to create multiple datasets. CARLA is an autonomous driving simulator built on the Unreal Engine [16], which has been used in developing industrial autonomous vehicle systems like Apollo [6] and Autoware [15]. We generate traffic flows, place traffic cameras at varying distances from each other, and record the videos. By altering the positions of the cameras—from close proximity to widely spaced—we generate datasets that exhibit different levels of camera similarity.

Models: We demonstrate *ECCO*’s performance on two machine learning tasks – object detection and instance segmentation. Instance segmentation is a more complex computer vision task compared to detection. For object detection, we use YOLO11-Nano and YOLO11-X [48] as student and teacher models, respectively. Instance segmentation used YOLO11n-Seg and YOLO11x-Seg for the student and teacher models. All models are pre-trained on COCO [11] datasets.

Baselines: We compare *ECCO* with the following continuous learning frameworks:

- **Naive baseline:** This baseline retrains a separate model for each camera (no grouping) and allocates GPU resources

uniformly across all concurrent retraining jobs (without optimizing GPU usage). Each camera uses a fixed sampling configuration, and bandwidth is evenly shared among cameras (without optimizing bandwidth efficiency).

- **Ekya:** Ekya [7] supports both inference and retraining tasks on edge devices by carefully scheduling GPU resources between these jobs. Unlike Ekya, *ECCO* allocates the server’s GPU resources solely for retraining jobs, as inference is handled on edge devices. For a fair comparison, we evaluate Ekya in a retraining-only setting. Although Ekya utilizes more advanced GPU allocation mechanisms than the naive baseline, it retrains separate models for each camera (independent retraining) and does not exploit potential similarities in data drift across cameras for grouped retraining, as *ECCO* does.
- **RECL:** RECL [24] selects a historical model from a shared “model zoo” to initialize retraining, aiming to accelerate convergence by starting from a previously trained model. While this strategy improves over Ekya, RECL still performs retraining independently for each camera. As the official implementation is unavailable, we re-implement RECL’s GPU allocation algorithm following the descriptions in the paper. To build the model zoo, we fine-tune a student model on the first two minutes of video from each camera and store each resulting model separately. RECL also adopts an adaptive frame uploading mechanism from AMS [26], which adjusts each camera’s sampling frame rate based on scene dynamics. However, this adaptation is driven purely by video content and does not align the sampling configuration with GPU allocation, nor does it coordinate bandwidth allocation. Since RECL demonstrates substantial improvement over AMS in its evaluation, we do not compare *ECCO* against AMS.

Metrics: We evaluate *ECCO* and the baselines along two dimensions: (i) Inference accuracy: We use mean Average Precision (mAP), a standard metric for both object detection and instance segmentation tasks [11, 14]. mAP quantifies both precision and recall across various Intersection over Union (IoU) thresholds, reflecting the model’s accuracy in predicting bounding boxes (detection output) and segmentation masks (segmentation output). (ii) Response time: The duration required to retrain models to achieve a predefined accuracy after retraining is triggered.

5 EVALUATION

We evaluate *ECCO* on two video analytics tasks using three datasets. Our evaluation focuses on the following aspects:

§5.1 What is the end-to-end performance of *ECCO* compared to leading baselines?

§5.2 How well does *ECCO* scale with increasing workloads?

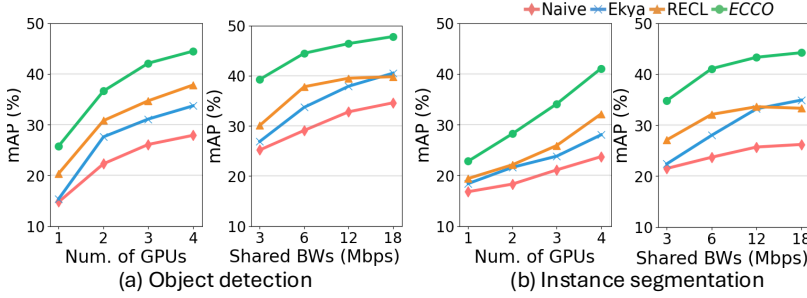


Figure 6: Average accuracy across different GPU resources and shared bandwidth resources.

§5.3 How does camera similarity impact *ECCO*’s performance advantage over baseline methods?

§5.4 How effective are the three modules of *ECCO*?

§5.5 How does *ECCO* improve responsiveness to retraining requests, especially under low-bandwidth conditions?

Our key findings include:

- With the same compute and communication resources, *ECCO* improves mAP by 6.7%-16.6% for object detection and 9.3%-18.1% for instance segmentation, consistently outperforming all baselines.
- *ECCO* can scale to support $3.3\times$ more cameras than baselines while maintaining the same accuracy.
- *ECCO*’s three modules jointly enable dynamic and accurate camera grouping, improve GPU and bandwidth efficiency, and enhance retraining quality.
- *ECCO* reduces response times by more than $5\times$ compared to baselines under low-bandwidth conditions. These benefits stem from group retraining’s data aggregation and natural model reuse within a group.

5.1 End-to-End Evaluation

We first evaluate the end-to-end performance of *ECCO* against three baselines under varying GPU and bandwidth constraints. The experiments use a fixed workload consisting of all 6 cameras from Scene 03 of the CityFlow dataset. For the Naive baseline and Ekya, each camera samples frames at 5 FPS with a vertical resolution of 960, reflecting a default high-rate, high-resolution setting. RECL adjusts only the frame rate. As none of the baselines consider shared or local bandwidth constraints, we assume only a shared bandwidth constraint in this experiment and equal bandwidth sharing across cameras for all baselines. We consider two experimental conditions: (i) varying the number of GPUs while fixing total shared bandwidth at 6 Mbps (i.e., corresponding to 1 Mbps per camera on average, used as a representative constrained bandwidth setting), and (ii) varying the total shared bandwidth while fixing the number of GPUs at 4. We use 4 GPUs due to testbed limits, emulating a small slice of a larger deployment. We report the average accuracy across all cameras in Fig. 6.

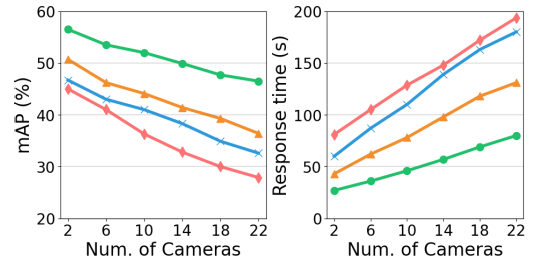


Figure 7: Average retraining accuracy and response time scaling across multiple cameras.

Overall, *ECCO* consistently outperforms all baselines. Under the same compute and communication budgets, it improves mAP by 6.7%-16.6% for object detection and 9.3%-18.1% for instance segmentation over the baselines. For object detection, *ECCO* requires only $1.8\times - 2.4\times$ fewer GPUs to maintain an mAP of 35%, due to its group retraining strategy. By merging retraining across correlated cameras, *ECCO* reduces the number of concurrent models, enabling more GPU resources per model on average. Such efficiency is further improved by optimizing GPU resource allocation across groups. *ECCO* also achieves the same detection accuracy (around 40% mAP) while using only 25%-33% of the bandwidth required by the baselines. This benefit stems from its transmission control module, which jointly adjusts sampling configurations and bandwidth sharing to match GPU allocation. This design ensures that available bandwidth is used fully and efficiently.

Among the baselines, RECL performs best due to its model reuse and GPU allocation strategy. However, as it reduces the sampling rate based on scene dynamics and does not adapt to bandwidth availability, it misses the opportunity to benefit from more training data when more bandwidth is available. In addition, RECL introduces overhead from continuously updating a large model zoo and retraining a model selector, which is not reported here. Finally, there is no guarantee that a historical model will perfectly match the current data drifts. In contrast, *ECCO* directly identifies and leverages current correlations among retraining requests without incurring much extra costs.

5.2 Scalability of *ECCO*

We evaluate the scalability of *ECCO* by testing it on workloads with an increasing number of cameras. Specifically, we use video streams from up to 22 cameras in Town 3 of the CARLA simulator, and the detailed scene and camera placements are visualized in the appendix. The experiment is conducted on the object detection task using a fixed compute budget of 4 GPUs and a simulated shared bandwidth of 50 Mbps. Fig. 7 reports the average retraining accuracy

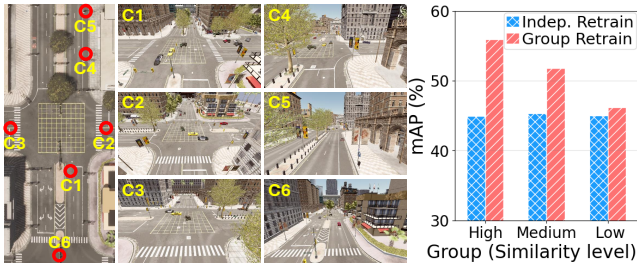


Figure 8: Impact of camera similarity. Left: Camera placement and example frames from six cameras in a CARLA town scene. Cameras are manually grouped into high (C1–C2–C3), medium (C1–C4–C5), and low (C1–C5–C6) similarity groups. Right: Retraining accuracy under independent vs. group retraining across similarity levels.

and response time (using a mAP threshold of 0.4) across all cameras.

As workload increases, the retraining accuracy of all three baselines drops significantly, and their response times grow quickly. This degradation stems from their independent retraining approach, which causes the compute demand to grow linearly with the number of cameras. In contrast, *ECCO* exhibits a more moderate degradation due to its group retraining strategy. Compared to the best-performing baseline, RECL, *ECCO* supports 3.3× more cameras using the same resource budget to achieve a similar mAP of 46%. In terms of responsiveness, under the 22-camera workload, *ECCO* reduces the response time to 41.3%–61.1% of the baselines.

5.3 Impact of Camera Similarity

This section provides an intuitive understanding of when *ECCO*’s group retraining is beneficial compared to independent retraining, and when it is not. We evaluate how the effectiveness of group retraining depends on the similarity among cameras within a group. To visualize and control similarity, we simulate six cameras in a region of Town 10 in the CARLA simulator and show their positions and fields of view in Fig. 8. We disable *ECCO*’s grouping module and manually construct three groups of three cameras each: a high-similarity group (C1–C2–C3), where cameras capture highly overlapping scenes with similar content; a medium-similarity group (C1–C4–C5), where cameras are in nearby locations with partially correlated views; and a low-similarity group (C1–C5–C6), where cameras observe distinct and non-overlapping scenes. We simulate a sudden rain event (weather-induced data drift) and apply group retraining to each group using a fixed 3-GPU and 3 Mbps shared bandwidth budget. We compare the results to independent retraining using Ekya, configured with the same resources.

As shown in Fig. 8, group retraining outperforms independent retraining when camera similarity is high, improving mAP by up to 11.9%. However, this advantage diminishes with decreasing similarity, and in the low-similarity group,



Figure 9: An example of *ECCO*’s dynamic camera grouping. Lines show the retraining accuracy over time for each camera, differentiated by marker types. Bars below represent camera groupings, with identical colors indicating the same group. *ECCO* dynamically regroups camera 3 when it no longer benefit from the current group due to scene divergence.

group retraining provides little improvement due to limited shared feature across cameras.

5.4 Performance of Modules in *ECCO*

In this subsection, we evaluate the effectiveness of the three modules in *ECCO*, respectively.

5.4.1 Dynamic camera grouping. To evaluate the effectiveness of *ECCO*’s dynamic camera grouping, we collect driving videos from three vehicles navigating a town map in CARLA. Mobile cameras introduce a more challenging scenario due to rapid and frequent scene changes, which require dynamic assessment and regrouping. Fig. 9 shows an example of dynamic grouping in *ECCO*, including each camera’s grouping status and retraining accuracy over time. We also present video frames at selected timestamps to qualitatively verify the grouping behavior.

As the vehicles move sequentially from suburban to urban areas, their cameras experience similar data drift caused by background transitions, leading to a drop in model accuracy to below 15% mAP for all three cameras. *ECCO* first receives a retraining request from camera 1 and starts a new retraining job. Later, it adds cameras 2 and 3 to the same job upon receiving their requests, as they share similar metadata with camera 1, and the ongoing training model from camera 1 performs better on their data than their current local models.

As the shared model is retrained, all three cameras benefit from improved accuracy. However, during retraining window 6, camera 3 takes a different route and enters a tunnel, while cameras 1 and 2 continue along the city road. This causes camera 3’s data distribution to diverge significantly. When *ECCO* reassesses the group model’s performance on camera 3 at the end of the window, it detects a significant accuracy drop, indicating camera 3 no longer aligns with the current group. *ECCO* then removes camera 3 from the group and treats its case as a new retraining request. Since

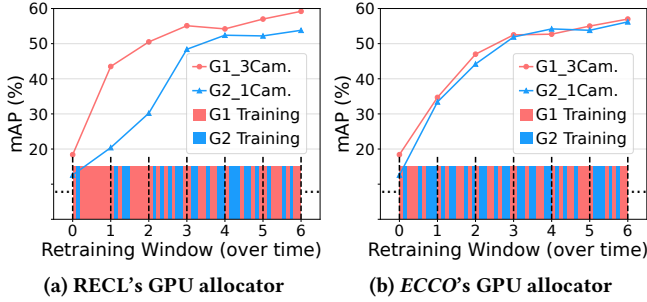


Figure 10: *ECCO* with RECL’s GPU allocator vs. its native allocator. The lines above show average accuracy for two distinct groups (3 cameras vs. 1 camera). The bars below represent GPU allocation over time. RECL’s allocator focuses on overall accuracy, neglecting smaller groups, while *ECCO* balances overall accuracy and fairness among cameras.

no existing group matches its updated metadata, a separate retraining job is initiated for camera 3.

5.4.2 GPU allocator. We evaluate the performance of *ECCO*’s GPU allocator by replacing it with RECL’s allocator. The experiment used four drone videos from the MDOT dataset, featuring three drones operating in an adjacent area and one drone in a distinct area. This setup ultimately divided the drones into two groups: one with three cameras and another with a single camera. Fig. 10 illustrates the GPU resource allocation for both allocators and the average retraining accuracy over time for each group. As both allocators share GPU resources on a time-shared basis, we used a “one-hot bar” to display the GPU allocation results.

When applying the RECL allocator to our system, it allocated most of the GPU resources to group 1 in the first two retraining windows. As a result, group 2 experienced resource starvation, leading to a significant accuracy gap of up to 23% mAP between the two groups. This is because the RECL allocator is designed to maximize the total system accuracy, favoring the group with more cameras due to their larger impact on overall accuracy improvement. In contrast, *ECCO* offers a more balanced approach. It achieves a near-synchronous accuracy increase among different groups. This is because *ECCO*’s allocator not only seeks to improve overall accuracy, but it also considers fairness by boosting the low-performance groups.

5.4.3 Resource-Aware Transmission Controller. We evaluate our transmission controller through an ablation study. In the ablated baseline, the controller is disabled at each camera: all cameras sample frames at a fixed rate of 5 *fps* and resolution of 960. Bandwidth sharing follows the traditional AIMD rule ($\alpha = 1$, $\beta = 0.5$) across cameras, meaning each camera competes equally for shared bandwidth, subject to its own local uplink cap. All other system components remain unchanged. We use 6 cameras from the CARLA dataset, evenly grouped into three groups (A, B, C), and fix

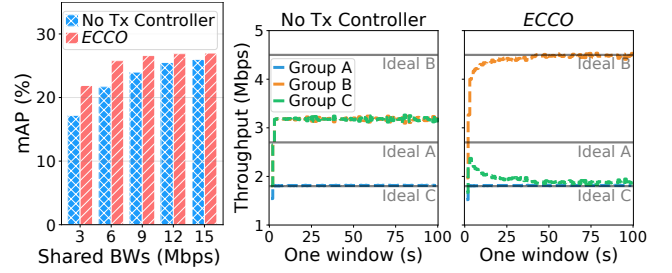


Figure 11: Ablation study of the transmission controller. Left: Retraining accuracy under varying shared bandwidths, showing that the controller improves accuracy, especially under limited bandwidth. Right: Per-group bandwidth traces at 9 *Mbps* shared bandwidth. The controller approximates GPU-proportional bandwidth allocation, whereas the baseline deviates significantly from the ideal allocation target.

the GPU budget to 1 GPU. The total shared bandwidth is varied from 3 *Mbps* to 15 *Mbps*. To emulate heterogeneous network conditions, we cap the uplink of the two cameras in Group A to 1 *Mbps*.

Fig. 11 (Left) shows the average retraining accuracy across all cameras. As the shared bandwidth increases, the performance bottleneck shifts from communication to computation, leading to an increase in accuracy for both methods that eventually plateaus. *ECCO* reaches its peak accuracy using only one-third of the bandwidth required by the baseline and achieves up to 4.7% higher accuracy under limited bandwidth (3 *Mbps*). Fig. 11 (Right) zooms in on a retraining window at 9 *Mbps* shared bandwidth, where GPU allocation across Groups A, B, and C is approximately in a 3:5:2 ratio. It compares the per-group bandwidth traces of the two methods against the ideal GPU-proportional target. *ECCO* closely approximates the target allocation, with Groups B and C proportionally sharing the remaining bandwidth after Group A’s local constraint is saturated. In contrast, the baseline deviates significantly due to the lack of rate differentiation. These results demonstrate that aligning communication with compute resources improves retraining accuracy, and that our transmission controller achieves compute-aware, adaptive bandwidth sharing under heterogeneous network conditions.

5.5 *ECCO*’s Benefits in Responsiveness

In §5.2, we showed that *ECCO* improves responsiveness to data drift through optimized compute and communication resource usage. Beyond resource efficiency, here we highlight two additional factors that contribute to its responsiveness: natural model reuse and data aggregation within a group. To isolate and evaluate these effects, we conduct intra-group experiments using Group 1 in Fig. 10, which includes three drone video streams. We compare *ECCO* with two methods: (i) RECL, which selects a historical model as the retraining starting point; (ii) *ECCO* + RECL, which combines group retraining with historical model reuse.

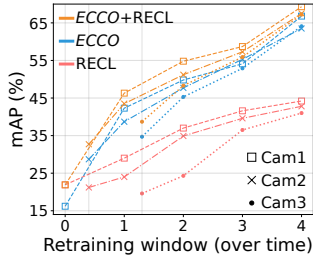


Figure 12: Retraining accuracy for each camera within a group over time. Group retraining enhances the initial accuracy of later cameras naturally.

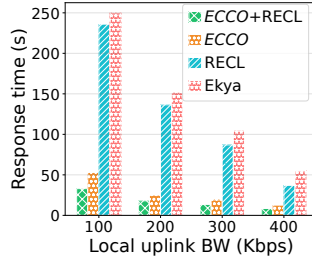


Figure 13: Average response time across cameras under low-bandwidth conditions. Group retraining enhances responsiveness through data aggregation.

Natural model reuse. Fig. 12 shows each camera’s retraining accuracy over time. For cameras 2 and 3, *ECCO* and *ECCO* + *RECL* achieve up to 15% higher initial mAP than *RECL*. This is because group retraining allows later retraining requests to start from a model that has already been partially updated using data from earlier cameras in the same group. In contrast, *RECL* relies on static historical models, which may not perfectly match the current, drifted data distribution. For camera 1, *RECL* achieves higher initial accuracy because it reuses an appropriate historical model, whereas *ECCO* starts retraining from scratch. *ECCO* + *RECL* inherits the strengths of both approaches and consistently yields the highest initial accuracy across all cameras.

Data aggregation. Group retraining improves responsiveness under poor network conditions, which are common in mobile scenarios such as drones or vehicles. Fig. 13 shows the average time required to reach 35% mAP under various low-bandwidth constraints on each camera’s local uplink. *RECL* and *Ekya* exhibit up to 5× longer response times compared to group retraining methods. This is because individual retraining must wait for sufficient data from a single camera, whereas group retraining aggregates data streams from multiple cameras, effectively increasing the available training data and speeding up training. Incorporating *RECL* into *ECCO* further reduces response time by initializing retraining from a stronger starting point.

6 RELATED WORK

Live video continuous learning: Prior studies have focused on building video analytics systems to provide high accuracy, low cost, and fast responses. These systems employ techniques such as model merging [38], model architecture pruning [52, 54], model distillation [23, 27], configuration adaptation [22, 29, 51, 57], and frame selection [9, 31]. However, all these efforts have aimed to optimize only the *inference* accuracy or the compute/network costs of DNN *inference*. In contrast, our work focuses on serving *continuous*

learning for live video analytics, a relatively unexplored focus area until recently. One pioneering effort in this domain is *Ekya* [7], which introduces a scheduler that optimally allocates GPU resources between retraining and inference tasks on edge servers. Building on this, *RECL* [24] further integrates model reuse with continuous retraining to enhance resource efficiency and responsiveness to data drift in live videos. The most recent study, *AdaInf* [45], manages GPU resource allocation to ensure service level objectives (SLOs) are guaranteed across multiple retraining models.

However, these systems share a common limitation: they handle retraining requests from different cameras independently, neglecting the similarity and potential for synergy between them, which can result in redundant retraining costs. *ECCO* addresses this issue by recognizing and exploiting the potential correlations between different camera feeds. By merging similar retraining requests, we improve the resource efficiency of the system.

Leveraging cross-camera correlations: Cross-camera correlations have been well recognized and utilized in previous work. In the computer vision community, these correlations are extensively studied in two main tasks: person re-identification (re-id) and multi-target, multi-camera (MTMC) tracking. Many studies have proposed new neural network architectures that use multi-camera correlations to address these tasks [41, 50, 53, 59]. In the systems literature, *Chameleon* [22] and *Spatula* [19] are two notable examples. *Chameleon* uses the temporal and spatial correlations among different videos to reduce the cost of neural network configuration profiling. Similarly, *Spatula* exploits cross-camera correlations to lower the inference costs in applications such as re-id and MTMC tracking.

ECCO stands apart from these approaches, as it aims to reduce the continuous learning costs of video analytics systems through cross-camera correlations. Guided by this objective, we present a new concept—group retraining. We have also developed an end-to-end framework that optimizes the use of compute and communication resources.

7 CONCLUSION

In this paper, we introduced *ECCO*, a novel video analytics framework that significantly enhances the efficiency of continuous learning by leveraging cross-camera correlations. *ECCO* smartly groups cameras experiencing similar data drifts to retrain a shared model, thereby reducing redundancy and optimizing resource utilization in terms of both computing power and data transmission. Extensive evaluations on multiple datasets demonstrated that *ECCO* markedly outperforms existing systems in accuracy, efficiency, and scalability.

REFERENCES

- [1] 2018. Paris Hospitals to Get 1,500 CCTV Cameras to Combat Violence Against Staff. <https://www.thelocal.fr/20180517/paris-hospitals-to-get-1500-cctv-cameras-to-combat-violence-against-staff>. (2018).
- [2] 2019. AWS Outposts. <https://aws.amazon.com/outposts/>. (2019). Accessed: 2025-01-26.
- [3] 2019. Azure Stack Edge. <https://azure.microsoft.com/en-us/services/databox/edge/>. (2019). Accessed: 2025-01-26.
- [4] Pablo F Alcantarilla, Simon Stent, German Ros, Roberto Arroyo, and Riccardo Gherardi. 2018. Street-view change detection with deconvolutional networks. *Autonomous Robots* 42, 7 (2018), 1301–1322.
- [5] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodik, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-time video analytics: The killer app for edge computing. *Computer* 50, 10 (2017), 58–67.
- [6] Apollo. 2025. Apollo Open Platform. <https://apollo.auto/>. (2025). Accessed: 2025-01-27.
- [7] 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 *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 119–135.
- [8] Tatjana Chavdarova, Florian Chollet, Alexey Ivanov, Ricard Marxer, Gerhard Rigoll, Julien Letessier, and Jérôme François. 2018. WILD-TRACK: A Multi-Camera HD Dataset for Dense Unscripted Pedestrian Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 1820–1828. <http://www.multisens.eu/>
- [9] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM conference on embedded networked sensor systems*. 155–168.
- [10] City of Bellevue. 2025. Bellevue Traffic Video Dataset. <https://github.com/City-of-Bellevue/TrafficVideoDataset>. (2025). Accessed: 2025-01-27.
- [11] COCO Consortium. 2025. COCO Dataset. <https://cocodataset.org/>. (2025). Accessed: 2025-01-27.
- [12] Monica Davey. 2018. Can 30,000 Cameras Help Solve Chicago’s Crime Problem? <https://www.nytimes.com/2018/05/26/us/chicago-police-surveillance.html>. (2018).
- [13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. 1–16. <https://arxiv.org/abs/1711.03938>
- [14] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. 2010. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision* 88 (2010), 303–338.
- [15] Autoware Foundation. 2025. Autoware. <https://www.autoware.org/>. (2025). Accessed: 2025-01-27.
- [16] Epic Games. 2025. Unreal Engine. <https://www.unrealengine.com/>. (2025). Accessed: 2025-01-27.
- [17] Song Han, Huizi Mao, and William J. Dally. 2017. Accelerating very deep convolutional networks for classification and detection. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- [18] Zheping Huang. 2015. Absolutely everywhere in Beijing is now covered by police video surveillance. <https://qz.com/518874/>. (2015).
- [19] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Paramvir Bahl, and Joseph Gonzalez. 2020. Spatula: Efficient cross-camera video analytics on large camera networks. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 110–124.
- [20] Si Young Jang, Yoonhyung Lee, Byoungheon Shin, and Dongman Lee. 2018. Application-aware IoT camera virtualization for video analytics edge computing. In *Proceedings of the Symposium on Edge Computing (SEC)*. ACM.
- [21] Haitao Jiang, Abdelsalam Helal, Ahmed K Elmagarmid, and Anupam Joshi. 1998. Scene change detection techniques for video database systems. *Multimedia systems* 6, 3 (1998), 186–195.
- [22] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*. 253–266.
- [23] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. Noscope: optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529* (2017).
- [24] Mehrdad Khani, Ganesh Ananthanarayanan, Kevin Hsieh, Junchen Jiang, Ravi Netravali, Yuanchao Shu, Mohammad Alizadeh, and Victor Bahl. 2023. {RECL}: Responsive {Resource-Efficient} continuous learning for video analytics. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 917–932.
- [25] Mehrdad Khani, Pouya Hamadani, Arash Nasr-Esfahany, and Mohammad Alizadeh. 2021. Real-time video inference on edge devices via adaptive model streaming. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.
- [26] Mehrdad Khani, Pouya Hamadani, Arash Nasr-Esfahany, and Mohammad Alizadeh. 2021. Real-Time Video Inference on Edge Devices via Adaptive Model Streaming. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 1420–1429. https://openaccess.thecvf.com/content/ICCV2021/html/Khani_Real-Time_Video_Inference_on_Edge_Devices_via_Adaptive_Model_Streaming_ICCV_2021_paper.html
- [27] Mehrdad Khani, Pouya Hamadani, Arash Nasr-Esfahany, and Mohammad Alizadeh. 2021. Real-time video inference on edge devices via adaptive model streaming. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4572–4582.
- [28] Hari Kishan Kondaveeti, Valli Kumari Vatsavayi, Srileakhana Mangapathi, and Reddy M. Yasaswini. 2023. Lightweight Deep Learning: Introduction, Advancements, and Applications. In *Advancement in Business Analytics Tools for Higher Financial Performance*. IGI Global, 242–259. <https://doi.org/10.4018/978-1-6684-8386-2.ch012>
- [29] Ferdi Kossmann, Ziniu Wu, Eugenie Lai, Nesime Tatbul, Lei Cao, Tim Kraska, and Sam Madden. 2023. Extract-Transform-Load for Video Streams. *Proc. VLDB Endow.* 16, 9 (May 2023), 2302–2315. <https://doi.org/10.14778/3598581.3598600>
- [30] Marek Kulbacki, Jakub Segen, Zenon Chaczko, Jerzy W. Rozenblit, Michał Kulbacki, Ryszard Klempous, and Konrad Wojciechowski. 2023. Intelligent Video Analytics for Human Action Recognition: The State of Knowledge. *Sensors* 23, 9 (2023), 4258. <https://doi.org/10.3390/s23094258>
- [31] 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 *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 359–376.
- [32] Hou-I Liu, Marco Galindo, Hongxia Xie, Lai-Kuan Wong, Hong-Han Shuai, Yung-Hui Li, and Wen-Huang Cheng. 2024. Lightweight Deep Learning for Resource-Constrained Environments: A Survey. *arXiv preprint arXiv:2404.07236* (2024).

- [33] Yihang Lou, Yan Bai, Jun Liu, Shiqi Wang, Ling-Yu Duan, and Wen Gao. 2019. VERI-Wild: A Large Dataset and a New Method for Vehicle Re-Identification in the Wild. *IEEE Transactions on Image Processing* 28, 8 (2019), 4001–4015. <https://doi.org/10.1109/TIP.2019.2906401>
- [34] MarketsandMarkets. 2024. Video Analytics Market worth \$22.6 billion by 2028. (2024). <https://www.marketsandmarkets.com/PressReleases/iva.asp> Accessed: 2025-01-26.
- [35] Michigan Department of Transportation. 2025. MDOT Traffic Dataset. <https://mdotnetpublic.state.mi.us/drive/>. (2025). Accessed: 2025-01-27.
- [36] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning convolutional neural networks for resource-efficient inference. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- [37] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. 2019. Online model distillation for efficient video inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.
- [38] Arthi Padmanabhan, Neil Agarwal, Anand Iyer, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Guoqing Harry Xu, and Ravi Netravali. 2023. Gemel: Model Merging for {Memory-Efficient},{Real-Time} Video Analytics at the Edge. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 973–994.
- [39] PyTorch. 2025. PyTorch. <https://pytorch.org/>. (2025). Accessed: 2025-01-27.
- [40] R.J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam. 2005. Image change detection algorithms: a systematic survey. *IEEE Transactions on Image Processing* 14, 3 (2005), 294–307. <https://doi.org/10.1109/TIP.2004.838698>
- [41] Kumar S Ray and Soma Chakraborty. 2019. Object detection by spatio-temporal analysis and tracking of the detected objects in a video with variable background. *Journal of Visual Communication and Image Representation* 58 (2019), 662–674.
- [42] George F Riley and Thomas R Henderson. 2010. The ns-3 network simulator. In *Modeling and tools for network simulation*. Springer, 15–34.
- [43] Ergys Ristani, Francesco Solera, Roger Zou, Rita Cucchiara, and Carlo Tomasi. 2016. Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking. *arXiv preprint arXiv:1609.01775* (2016). <https://arxiv.org/abs/1609.01775>
- [44] Md Nahid Sadik, Tahmim Hossain, and Faisal Sayeed. 2024. Real-Time Detection and Analysis of Vehicles and Pedestrians using Deep Learning. *arXiv preprint arXiv:2404.08081* (2024). <https://arxiv.org/abs/2404.08081>
- [45] Sudipta Saha Shubha and Haiying Shen. 2023. AdaInf: Data Drift Adaptive Scheduling for Accurate and SLO-guaranteed Multiple-Model Inference Serving at Edge Servers. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 473–485.
- [46] Bharat Singh, Mahyar Najibi, and Larry S Davis. 2018. Sniper: Efficient multi-scale training. *Advances in neural information processing systems* 31 (2018).
- [47] Zheng Tang, Dongfang Wu, Yifan Jiang, Wei Li, Hao Luo, Xiaoyang Wang, Lu Sheng, Jing Shao, and Chen Change Loy. 2019. CityFlow: A City-Scale Benchmark for Multi-Target Multi-Camera Vehicle Tracking and Re-Identification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8797–8806. <https://doi.org/10.1109/CVPR.2019.00901>
- [48] Ultralytics. 2025. YOLO11 Models. <https://docs.ultralytics.com/models/yolo11/>. (2025). Accessed: 2025-01-27.
- [49] Ultralytics. 2025. YOLOv5. <https://github.com/ultralytics/yolov5>. (2025). Accessed: 2025-01-27.
- [50] Longhui Wei, Shiliang Zhang, Wen Gao, and Qi Tian. 2018. Person transfer gan to bridge domain gap for person re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 79–88.
- [51] Mike Wong, Murali Ramanujam, Guha Balakrishnan, and Ravi Netravali. 2024. {MadEye}: Boosting Live Video Analytics Accuracy with Adaptive Camera Configurations. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 549–568.
- [52] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10734–10742.
- [53] Tong Xiao, Shuang Li, Bochao Wang, Liang Lin, and Xiaogang Wang. 2017. Joint detection and identification feature learning for person search. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3415–3424.
- [54] Ran Xu, Rakesh Kumar, Pengcheng Wang, Peter Bai, Ganga Meghanath, Somali Chaterji, Subrata Mitra, and Saurabh Bagchi. 2021. Approxnet: Content and contention-aware video object classification system for embedded clients. *ACM Transactions on Sensor Networks (TOSN)* 18, 1 (2021), 1–27.
- [55] Y.R. Yang and S.S. Lam. 2000. General AIMD congestion control. In *Proceedings 2000 International Conference on Network Protocols*. 187–198. <https://doi.org/10.1109/ICNP.2000.896303>
- [56] Y. R. Yang and S. S. Lam. 2000. *General AIMD Congestion Control*. Technical Report TR-2000-09. Department of Computer Sciences, The University of Texas at Austin. https://cs-www.cs.yale.edu/homes/yry/research/TechReports/gaimd_TR.pdf Equation (21) derives the steady-state expected window size under GAIMD: $E[W] = \frac{\alpha}{(1-\beta)b} E[X]$.
- [57] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live video analytics at scale with approximation and {Delay-Tolerance}. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 377–392.
- [58] Shuo Zhang, Zhaojian Li, Jianqiang Wang, and Keqiang Li. 2024. Online Legal Driving Behavior Monitoring for Self-Driving Vehicles. *Nature Communications* 15, 1 (2024), 44694. <https://doi.org/10.1038/s41467-024-44694-5>
- [59] Liang Zheng, Hengheng Zhang, Shaoyan Sun, Manmohan Chandraker, Yi Yang, and Qi Tian. 2017. Person re-identification in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1367–1376.

A APPENDIX

Here we provide more details on the dataset generation using the CARLA simulator. We set up fixed traffic cameras at various locations in Town 3 within the CARLA ecosystem. These cameras are used for assessing the scalability of *ECCO* (Section 5.2). The locations and orientations of these traffic cameras are illustrated in the figure below.

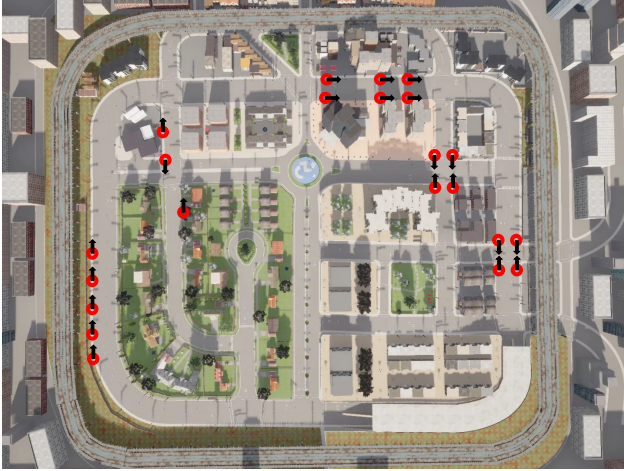


Figure 14: Overview of camera placement in Town 3, CARLA simulator. Red markers with black arrows indicate the location and direction of traffic cameras.