Gecko: Resource-Efficient and Accurate Queries in Real-Time Video Streams at the Edge

Liang Wang[1], Xiaoyang Qu[2], Jianzong Wang[2§], Guokuan Li[1§], Jiguang Wan[1], Nan Zhang[2], Song Guo[3], Jing Xiao[2]
[1]*Huazhong University of Science and Technology, Wuhan, China*
[2]*Ping An Technology (Shenzhen) Co., Ltd., Shenzhen, China*
[3]*Hong Kong University of Science and Technology, Hong Kong*
[1]{*iggiewang, liguokuan, jgwan*}*@hust.edu.cn*
[2]*quxiaoy@gmail.com, jzwang@188.com, nzhang889@gmail.com, xiaojing661@pingan.com.cn*
[3]*songguo@cse.ust.hk*

***Abstract*—Surveillance cameras are ubiquitous nowadays and users' increasing needs for accessing real-world information (e.g., finding abandoned luggage) have urged object queries in real-time videos. While recent real-time video query processing systems exhibit excellent performance, they lack utility in deployment in practice as they overlook some crucial aspects, including multi-camera exploration, resource contention, and content awareness. Motivated by these issues, we propose a framework Gecko, to provide resource-efficient and accurate real-time object queries of massive videos on edge devices. Gecko (i) obtains optimal models from the model zoo and assigns them to edge devices for executing current queries, (ii) optimizes resource usage of the edge cluster at runtime by dynamically adjusting the frame query interval of each video stream and forking/joining running models on edge devices, and (iii) improves accuracy in changing video scenes by fine-grained stream transfer and continuous learning of models. Our evaluation with real-world video streams and queries shows that Gecko achieves up to 2x more resource efficiency gains and increases overall query accuracy by at least 12% compared with prior work, further delivering excellent scalability for practical deployment.**

I. Introduction

Recent years have witnessed substantial growth in the deployment of surveillance cameras. According to research [1], the worldwide market for video surveillance is expected to grow from $53.7 billion in 2023 to $83.3 billion by 2028. To fully unleash these deployed cameras' potential, video analytics [2]–[5] based on Deep Neural Networks (DNNs) has been developed to assist various public and commercial institutions in acquiring useful information from video streams.

Querying objects is one of the important video analytics jobs, which is adopted across a broad spectrum of applications, such as public safety and traffic management [6]. Specifically, the goal of video query is to respond to users' queries for diverse needs that involve video streams from a group of cameras. As shown in Figure 1(a), a query task may need to query in numerous video streams and provide the set of cameras and video frames containing query objects.

Within these applications, it is generally ideal to execute query tasks directly at the edge. By deploying computation near the video data sources, edge computing for video analytics [7]–[9] can successfully meet *real-time* performance while reducing the high overhead of transmitting video streams. However, most edge devices only support specialized lightweight models due to their limited resources (e.g., with weak GPUs [10]). Compared to standard DNN models, these lightweight DNN models have reduced weights and less complex architectures. They can only memorize a limited number of object appearances and only perform well in a few scenarios.

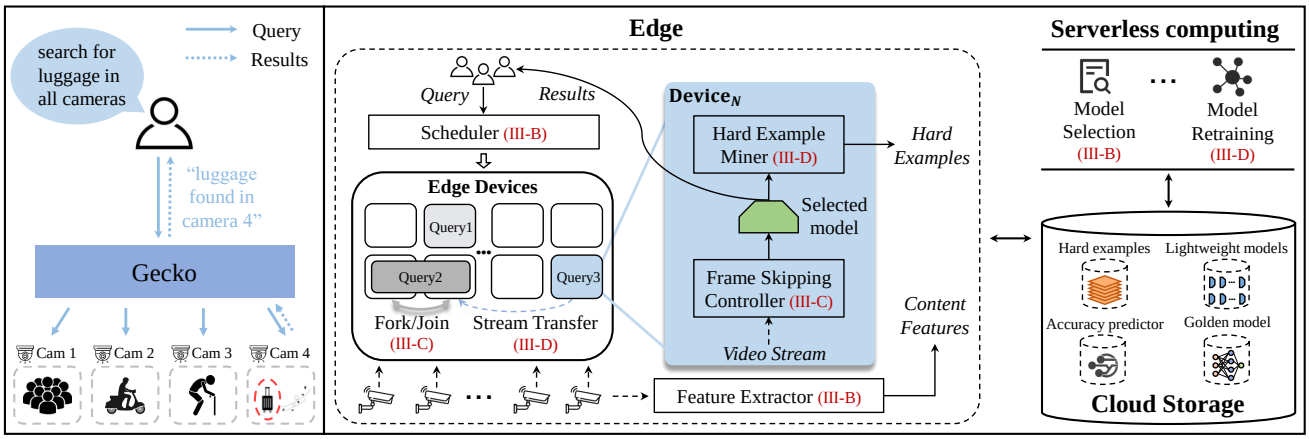
Recently, several edge-based solutions in the network community (e.g., SurveilEdge [11]) and the database community (e.g., Video-zilla [12]) have initially shown the effectiveness of processing object queries over real-time video data via edge-cloud collaboration. However, these solutions have not solved several performance challenges for achieving resource efficiency and high accuracy in practice:

• **A1:** ***Multi-camera exploration and model sharing*.** Previous real-time video query approaches [3], [11], [13] only focus on optimizations of querying within a single video stream. In fact, many practical real-time video query applications go beyond individual cameras [14]. On the one hand, when receiving a query that involves a large number of cameras, the query may want to apply the same model to different video streams. On the other hand, further queries may also use the same model while querying the same object. The framework should eliminate redundancies by sharing models among multiple camera streams at the start of query processing, for resource efficiency and scalability to large-scale cameras.

• **A2:** ***Runtime resource contention*.** Real-time video queries need to guarantee high accuracy and low latency under *computational resource contention* [15] of edge devices. To strive for optimal resource efficiency, it is imperative to adaptively allocate or schedule available resources at runtime based on the contention effects from query processing and model execution.

• **A3:** ***Video content awareness*.** The query should adapt to the content of the video streams. The rationale behind content adaptation is threefold. Firstly, lightweight models often exhibit fluctuating levels of accuracy across various video contents (e.g., due to different distributions of the objects). Thus, the system should identify and employ the most accurate model from among those functionally equivalent

§Corresponding authors.



(a) Real-time video query **(b) Architecture overview of Gecko**

Fig. 1. Overview of Gecko. (a) illustrates an example of real-time video queries with Gecko. The user input includes query *objects* (luggage in this example) and query *areas* (cameras), and the output results are video frames with detected query objects. (b) shows Gecko's components and their interactions.

for the current scene [13]. Secondly, considering available computational resources, the video frame query interval needs to be configured based on the changing video content. Thirdly, since real-world video data changes over time, specialized edge models are especially susceptible to *data drift* [8]. For example, a lightweight model untrained on images containing night will perform poorly when it encounters such conditions in practice [16]. Adapting to these dynamic factors can improve overall accuracy.

To address these challenges, we propose Gecko, a *resource-efficient* framework for executing highly *accurate* real-time video queries on edge devices. It leverages serverless computing for model selection and model retraining, while considering resource utilization of the edge cluster. First, for each online video query, Gecko promptly selects suitable models from the shared model zoo, schedules them to edge devices, and attaches camera streams to the corresponding model to query. Second, during query execution, Gecko reallocates computational resources to meet the latency and accuracy objective when subjected to resource contention. Finally, Gecko can both carefully transfer a stream to another optimal model for inference and continually train models running on devices to improve accuracy under data drift.

We implement Gecko on NVIDIA Jetson modules and evaluate its performance using real-world surveillance videos. Compared with SurveilEdge [11] and RECL [13], two real-time video query solutions at the edge, Gecko achieves up to 2x more resource efficiency gains and increases query accuracy by at least 12%. Moreover, Gecko exhibits excellent scalability for deployment in practice.

Overall, we make the following contributions in this paper:

- We design Gecko, a framework for resource-efficient and accurate real-time video queries at the edge (§III).
- Gecko extracts content-aware features, uses an accuracy predictor for model selection based on the extracted features, and schedules selected models and camera streams to enable multiple-stream model sharing (§III-B).
- Gecko allows runtime resource reallocation under resource contention, which results from varying video content and time-varying online queries (§III-C).
- Gecko further improves query accuracy by performing fine-grained stream transfer and continuous learning of edge models on video content (§III-D).
- We implement and evaluate Gecko to demonstrate the efficiency and accuracy of our design (§IV).

II. Related Work

Video Query Processing Frameworks. Recent works on video query processing can be generally summarized into three broad optimizations. The first uses specialized lightweight models to answer queries directly [17]. The second focuses on filtering out unimportant frames and processing the filtered critical frames [18]. Several systems combine the two approaches [19], [20]. And the third makes the optimal configuration choice considering the resource-quality tradeoff [3], [21]. Gecko falls into the first and second categories—using lightweight, small, fast, yet accurate models for specific video scenes and adjusting the real-time processing frame rate for every stream. Different from most of these works, Gecko applies an edge computing paradigm. The closest framework to Gecko is SurveilEdge [11], which implements the object queries in real-time video streams on edge devices. Particularly, when receiving a new query request, Gecko can respond faster by selecting one or more models which may have identified a similar scene and object distribution from historical models in cloud storage, instead of training specific DNNs in SurveilEdge.

Adaptive Vision Systems. Vision algorithms suffer from accuracy and latency issues when running on edge devices, especially under changing video scenes and varying resource contention. ROMA [22] analyzes the impact of dynamically changing video contents on real-time accuracy and latency variation. As videos contain similar content within a series of consecutive frames, current video analytics algorithms or

applications have utilized the content information to improve accuracy and reduce latency. AdaScale [23] improves speed and accuracy for video object detection by selecting the input image scale based on content. ODIN [16] can detect data drift and recover from it based on the similarity of video scenes. ApproxDet [15] uses lightweight features of past video content to choose the optimal execution branch. LiteReconfig [24] uses both lightweight and heavy features for selecting the configuration tailored to the video content. In contrast, Gecko applies a content-aware accuracy predictor to precisely deduce which models in cloud storage are accurate for given video streams. Gecko also adaptively adjusts the frame processing interval to the content of each video stream. On the other hand, managing resources for video analytics applications is an active area of work. VideoStorm [3] designs resource configuration adaptation to maximize performance. VideoEdge [21] considers the positioning of video analytics components throughout a hierarchy of clusters under different resource restraints. Mainstream [25] adapts concurrent applications sharing fixed edge resources to reduce aggregate per-frame computation time and optimize aggregate result quality. MIRSA [26] provides multi-device interaction to share both edge and cloud computing resources. Inspired by these works, Gecko supports model scheduling and model sharing for online queries to improve GPU utilization on edge devices.

Continuous Learning in Video Analytics. Video analytics systems are increasingly embracing continuous learning through model distillation [8], [13], [27] to retrain lightweight models to adapt to changing video scenes and enhance inference accuracy. Ekya [8] offers resource-sharing strategies for cost-effective model retraining. AMS [27] dynamically adjusts the frame sampling rate at edge devices based on scene variations to minimize the need for frequent retraining. RECL [13] integrates Ekya and AMS, and responds faster than them by selecting a model from the model zoo when requesting a model. Gecko further considers scalability and retrains models using serverless computing and hard examples [28] in videos.

Serverless Service. Serverless computing as an emerging cloud-computing paradigm has experienced massive growth and benefits many tasks, including video processing [29]. Recent efforts have integrated serverless computing into the edge-based systems for video analytics [30], [31], which are part of the main motivation of Gecko to utilize serverless computing for on-demand model selection and online continuous learning.

III. Gecko Design

To provide accurate real-time video queries in practice, we design Gecko, an accuracy-augmented resource-efficient video query framework. Below, we introduce its architecture and delve into the details of each technique.

A. Overview

Goals. For practical deployment, Gecko is designed with two primary goals: (a) *resource-efficient*. Gecko considers resource contention among massive video streams to improve resource utilization of edge devices (solving **A1** and **A2**). (b) *accurate*. Gecko can achieve high accuracy of object queries in various and changing video scenes (solving **A1** and **A3**).

TABLE I
List of content features extracted by Gecko. The extraction cost is evaluated on the NVIDIA Jetson TX2 board.

Name	Dimension	Cost	Definition
Light	5	0.42 ms	Composed of width, height, number of objects, average size of objects divided by the frame size, fraction of pixels covered by all objects
HoC	768	17.23 ms	Histogram of Color on red, green, blue channels
HOG	5400	30.15 ms	Histogram of Oriented Gradients
MobileNet	1280	179.89 ms	Effective and efficient feature extractor, average pooled from the feature map before the fully-connected layer

Architecture & Workflow. Figure 1(b) shows the architecture of Gecko. To illustrate the query workflow, we assume that multiple cameras are connected to the system and continuously stream their captured videos to the edge. Notably, edge devices will not process these video streams unless responding to a query. When the user submits a query request to Gecko, Gecko first performs model selection and scheduling (§III-B). The *feature extractor* extracts current content features from the video streams over which the query is to be performed. The system invokes the serverless function of *model selection* that takes these features as parameters to determine the most accurate model for each video. The *scheduler* decides which devices to use for query execution, assigns the selected models to the devices, and attaches the video streams as inputs to the corresponding optimal models. At runtime, Gecko employs two strategies to optimize the reallocation of resources (§III-C). The *frame skipping controller* dynamically adjusts the query frame interval of each video stream. Meanwhile, computing resources of edge devices can be adaptively reallocated by the *fork/join* operations of model execution. To cope with data drift and improve accuracy, Gecko incorporates two additional approaches (§III-D). One practical approach is to perform *stream transfer* at a fine-grained level. When a video stream's scene undergoes obvious changes, transferring it to another model which performs inference best for current content can avoid query performance degradation. Secondly, continuous learning is utilized to adapt models to dynamic scenes. The *hard example miner* automatically mines hard examples from videos, which refer to video frames where the query object has significant visual changes within the same class, making it challenging for the model to detect the object correctly. When the number of these frames reaches a threshold, serverless *model retraining* is triggered to maintain the desired accuracy.

B. Model Selection and Scheduling

On receiving a query, Gecko should quickly select one or more high-accuracy lightweight models from a collection of

history models and schedule them to devices for execution. Gecko achieves it by: (i) extracting the content features from current video frames; (ii) using a reliable and fast selection procedure to navigate the model zoo in cloud storage; (iii) assigning models to devices based on resource usage statistics.

Extracting Content Features. Gecko contains a feature extractor, which is designed to map from the frame representation to its content-aware feature representation *f*, as the former has excessive redundancy. The extractor possesses discriminative qualities, allowing the features to predict the accuracy of each model relative to video content characteristics. Table I summarizes Gecko's content features, dimensions, extraction time (cost), and definitions. These features are inspired by the recent work in object detection [24]. Gecko can extract some lightweight features at no cost, such as the video frame's width, height, number of objects, and average object size. To characterize color and gradient information, Gecko utilizes two conventional vision features: Histograms of Color (HoC) and Histograms of Oriented Gradients (HOG). In addition, the feature extractor incorporates MobileNetV2 [32], a commonly used DNN that is computationally efficient for heavy content feature extraction. Although feature extraction incurs latency costs, such costs can be acceptable.

Model Selection. We propose the accuracy predictor for model selection based on given extracted content features. The predictor is a gating network (similar to the method in RECL [13]) to infer which models accurately fit current video content. It is designed as a serverless function *A(m, f)* called to estimate the accuracy of model *m*, relying on the input features *f*. *A(m, f)* is achieved using a 6-layer neural network (striking an optimal tradeoff among latency, selection accuracy, and update cost) with ReLU activation, 256 neurons incorporated in all hidden layers, and residual connections [33]. Due to the significant variation in feature dimensions, spanning 1 to 3 orders of magnitude, the first layer utilizes fully-connected projections to project both low- and high-dimensional content features into fixed 256-dimensional vectors, which are then concatenated. The output layer has *M* neurons, where *M* is the number of models. The network finally assigns an accuracy score to each model. As the model zoo size increases, the output size of the gating network must be correspondingly updated. Since the logic for accuracy prediction remains constant in the zoo, we simply add neurons for the new models to the final layer and train the accuracy predictor without altering the connectivity weights for the existing expert models.

Model Scheduling. Then, the model selection results are sent back to the scheduler component. Each selected model is associated with one or more video streams in which it achieves optimal accuracy. Gecko needs to schedule the execution of these models to the edge devices. Each model has binding metadata, including GPU compute consumption for model running and processing each additional video stream. The metadata can either be user-provided or obtained from the previous execution. As resources can be reallocated at runtime (§III-C) and the framework cannot perceive the duration of query execution, we can focus on space-scale scheduling without considering time-scale optimization. The scheduler of Gecko is fairly simple. Models are sorted based on their GPU demands from least to greatest, and then each model gets assigned to the currently most idle device in order. The head-of-line (HOL) blocking [34] issue that is commonly encountered in small-scale clusters can be effectively addressed by prioritizing jobs that demand fewer resources.

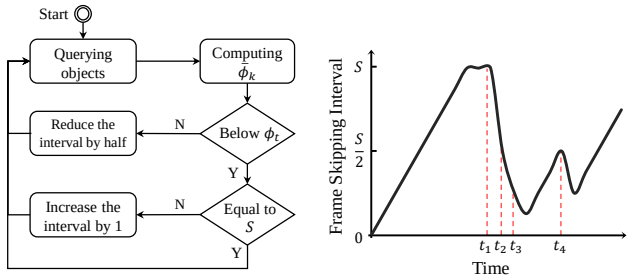


Fig. 2. Illustration of Dynamic Frame Skipping Control. *Left*: Control flow diagram. *Right*: AIMD-based frame skipping interval change (marked timestamps correspond to the video scenes exhibiting dramatic changes).

C. Runtime Resource Reallocation

Different from previous real-time video query solutions [11], [13] that optimize the performance within a single video stream, Gecko considers resource contention among massive video streams to improve GPU utilization. To this end, Gecko designs two strategies of resource reallocation at runtime to adapt to variable query frequency across streams and different inference demands across models, respectively.

Dynamic Frame Skipping Control. The frame query interval in different video streams should adapt video content. The short frame processing interval enhances the likelihood of detecting query objects in more frames, providing more opportunities for accurate queries. However, it is sufficient to query at 2$\sim$3 FPS when querying objects moving slowly. When there is resource contention from multiple tasks on the resource-constrained edge device, selecting an appropriate frame processing rate that suits each video scene for video query is better. Hence, we propose the dynamic frame skipping control, adjusting the frame interval adaptively based on the extent and speed of video scene changes.

First, we adopt the following approach to obtain a robust signal [27] for scene change. Compared to raw pixels, labels typically take values in a considerably smaller space (i.e., query object classes), making them a more reliable signal for detecting change. The edge device calculates a metric ϕ for each stream during querying to monitor the rate at which labels change over time for video frames. Consider a series of frames $\{\mathbf{I}_k\}_{k=0}^{n}$, with $\{\mathcal{T}(\mathbf{I}_k)\}_{k=0}^{n}$ denoting the model's output labels on these frames. For each frame $\mathbf{I}_k$, define ϕ_k using the same loss function (e.g., CIoU [35]) that defines the object query task, with $\mathcal{T}(\mathbf{I}_k)$ and $\mathcal{T}(\mathbf{I}_{k-1})$ serving as the prediction and ground-truth labels, respectively. In other words, ϕ_k is the loss (error) of the model's prediction on $\mathbf{I}_k$ relative to the label $\mathcal{T}(\mathbf{I}_{k-1})$. Therefore, the smaller ϕ_k is, the more similar the

labels for $\mathbf{I}_k$ and $\mathbf{I}_{k-1}$ are, i.e., slowly-changing or stationary scenes tend to obtain lower scores.

Second, when dynamically adjusting the frame skipping interval, we take the approach inspired by the Additive Increase/-Multiplicative Decrease (AIMD) algorithm [36], best known for its use in Internet congestion control. For slowly-changing or stationary scenes, or scenes without any query objects, the frame skipping controller gradually increases the frame skipping interval, while quickly reducing the interval when the query object is detected and the scene changes dramatically. Our insight pertains to the spatiotemporal characteristics of videos in practical applications, i.e., cameras observing similar scenes tend to capture videos containing similar objects and demonstrate the periodicity of busy times [11]. Figure 2 shows the workflow of adjusting the frame skipping interval. In detail, we set a threshold ϕ_t, and the controller computes the average ϕ_k over recent frames and periodically checks whether $\bar{\phi}_k$ is below the threshold. If it is, the frame interval is increased by 1; otherwise, the interval is reduced by half. In addition, we use the variable S to constrain the maximum interval not to exceed this value.

FORK/JOIN *of Model Execution.* To maintain optimal resource efficiency, Gecko continually monitors resource availability. When a change in resource contention status is detected, such as insufficient GPU memory, Gecko needs to redo scheduling decisions. This involves adjusting model execution among edge devices to ensure efficient resource allocation, which is implemented with two primitives: FORK and JOIN.

The FORK mechanism allows model execution creates a copy on another device. Here we assume that the network latency associated with the transmission of frames to the edge device is negligible. If the device's query processing service rate (T_s) is faster than the total cache frame arrival rate (T_a) coming from all connected cameras, each frame can be processed without delay. However, several subsequent frames can not be processed in time while serving one frame if the frame input rate is high. On the other hand, the maximum supported video stream capacity is limited and varies among different devices, which can be counted by performance profiling. Thus, model execution may be required to fork for more computing resources to process current video streams. Specifically, FORK is triggered to ensure: (i) service rate T_s on a single device is greater than T_a, and (ii) at runtime, the device keeps video streams below the maximum supported number so that query application performance remains stable. In contrast, if the number of streams drops (e.g., a query completes) or the frame input rate is consistently low (e.g., videos are stationary or slowly-changing scenes), keeping running multiple of the same models on different devices can lead to computing redundancy. The JOIN primitive can merge queries into as few devices as possible for processing, releasing redundant models to provide more available computing resources.

Transferring video streams is a critical aspect that Gecko must address to ensure the correct process of the two primitives. If frames arriving during a transfer aren't processed, it can impact the accuracy of the queries. Our method is that while the destination device prepares for the transferred stream, the frame is still input normally into the source device. When the destination device becomes responsible for analyzing the frames after the stream attaches, it sends a *transfer finish ACK* to the source device. The source detaches the stream after receiving the *transfer finish ACK* to accomplish the requirement of the transferred stream.

Algorithm 1: Hard Example Mining

Input : $\mathcal{L}$ - Prediction bbox list.
θ_L, θ_M, θ_H - Low, middle, high bbox thresholds.
λ - Image threshold.

Function `GeckoMine(`$\mathcal{L}$, θ_L, θ_M, θ_H, λ`)`
  $m \leftarrow 0$, $n \leftarrow$ `Length(`$\mathcal{L}$`)`
  for `all` *bbox* $\in \mathcal{L}$ **do**
    if $\theta_L \leq$ *bbox.score* $< \theta_M$ **then**
      return `true`
    else if $\theta_M \leq$ *bbox.score* $< \theta_H$ **then**
      $m \leftarrow m + 1$
  `// calculate hard example coefficient.`
  `factor` $\leftarrow \frac{m}{n}$
  if `factor` $\geq (1 - \lambda)$ **then**
    return `true`
  return `false`

D. Query Accuracy Improvement

To enable continuous improvement in accuracy even with data drift during the query process, Gecko employs *Fine-grained Stream Transfer* to maintain the accuracy for each single stream and *Continuous Learning* to improve the overall accuracy of the single model.

Fine-grained Stream Transfer. Beyond transferring video streams mentioned in Section III-C, Gecko also performs a fine-grained level of stream transfer. Our insight is that real-time video query suffers from a significant limitation in their ability to accurately process massive streams, which assumes that all frames in a video come from a static distribution [16]. In practical scenarios, visual data exhibits temporal drifts as it stems from a time-evolving and dynamic distribution. We propose a distance metric using the content features to detect shifts tailored for visual data in Gecko. Once the shift is detected in a video stream, Gecko performs model selection again to determine if the stream could be transferred as input to another model that is best for the current video content for higher accuracy. Specifically, we employ the Hellinger distance to calculate and quantify the similarity between two frames. Gecko periodically (e.g., every 30 seconds) extracts the content features from the current video frame at query runtime (as shown in Table I, costs are negligible). We define the Hellinger distance between the initial frame X for model selection and the current frame Y as

$$d_H(X,Y) = \frac{1}{|F|} \sum_{f \in F} \sqrt{\sum_{i=1}^{k_f} \left(\sqrt{\frac{X_{fi}}{\sum_{j=1}^{k_f} X_{fj}}} - \sqrt{\frac{Y_{fi}}{\sum_{j=1}^{k_f} Y_{fj}}} \right)^2} \quad (1)$$

where F denotes the set of all features and k_f is the number of dimensions in f. By taking the average over all features,

we can calculate the distance between two frames. If the distance between the video stream's initial and current frame is considerable, the existing model may no longer be suitable for the current distribution. Notably, for robustness reasons, only multiple consecutive large deviations in distance will trigger the migration decision.

Continuous Learning. Another effective strategy to cope with data drift is continuous learning of lightweight models. The single model can be retrained to improve the overall query accuracy across all connecting video streams. The model performance continuously boosts by learning knowledge from new scenarios. Gecko employs serverless computing to execute a set of operations for continuous learning. Serverless offers ample computational power of the cloud, with the benefit of high scalability (i.e., enabling continuous learning of as many models as needed) due to its elasticity feature.

Collecting the appropriate frames for retraining is a crucial part of continuous learning. The hard example miner automatically selects hard examples for model retraining. Algorithm 1 shows the hard example mining procedure. We define three thresholds, θ_L, θ_M, and θ_H, representing the low, medium, and high detection bounding box thresholds of inference confidence score, respectively. They divide the confidence score of inference boxes into two intervals. For the $[\theta_L, \theta_M)$ range, as long as an output box confidence score is within the range, the frame is a hard example. For the $[\theta_M, \theta_H)$ range, it requires calculating the image hard example coefficient, which is equal to the number of inference boxes within this range divided by the total number of inference boxes in the frame. We establish a factor threshold, denoted as λ, whereby frames with a coefficient higher than (1-λ) are considered as hard examples. Through extensive experimentation, we discover that optimal settings for θ_L, θ_M, and θ_H are 0.3, 0.5, and 0.7. In addition, our findings suggest that setting the λ parameter to 0.8 is best, and any value within the range of 0.6 to 0.9 proves effective. These parameters make the best tradeoff between capturing enough challenging cases for effective retraining and avoiding too many easy or irrelevant examples that could potentially dilute learning. However, these values might need to be adjusted based on specific application requirements.

When the number of uploaded hard examples corresponding to each model exceeds a given threshold, continuous learning will be triggered. First, since manual labeling is tedious and potentially a colossal waste of time for continuous training of edge models, we get the labels using a highly accurate "golden model" that employs a deep architecture with a large number of weights is used to label the hard example frames automatically. The golden model is pre-trained on extensive datasets and can cover all unseen domains. Notably, the golden model can run effectively with abundant computing resources offered by serverless in the cloud. Then, the retraining function starts, utilizing the frames with labels as the training dataset to optimize the lightweight model, boosting the accuracy of real-time video queries for the current scene. This retrained model is considered promising if its accuracy is higher than the previous model with the evaluation dataset. Finally, the promising model is transmitted back to the edge device for redeployment. Meanwhile, it is added to the model collection, and the accuracy predictor is updated. Furthermore, to handle catastrophic forgetting [8], [27], a random subset of hard examples used for the current session is replaced by a small-scale historical data set before each retraining session.

IV. Evaluation

In this section, we evaluate Gecko on a physical edge cluster and perform numerous queries with three different video query workloads.

A. Experimental Setup

Implementation: Gecko's core modules are written in 2.6k lines of Python code. We implement Gecko in Python3 (v3.7.3) using PyTorch (v1.9.0) for the feature extractor, the frame skipping controller, and the hard example miner. We also use `cv2` and `scikit-image` to extract content-aware features. Moreover, the implementation of Gecko utilizes the DeepStream SDK [37], which enables attaching and detaching query video streams to inference pipelines with lightweight models at runtime. The non-edge-side implementation is based on Alibaba Cloud Function Compute as the serverless computing platform for model selection and model retraining. We follow the same retraining algorithm as in Ekya [8] though our framework is generally applicable. We leverage Alibaba Cloud Object Storage Service as the cloud storage for object-like data (e.g., models and video frames). For experiment workloads, we construct three different workloads—*workloads 1/2/3* respectively represent the scenarios of querying one class of object on *each/multiple/all* camera(s). During querying, the results are saved at edge devices.

Testbed: We conduct experiments on a cluster of 3 embedded platforms: an NVIDIA Jetson TX2, an NVIDIA Jetson AGX Xavier, and a more powerful NVIDIA Jetson AGX Orin.

Dataset & Models: We evaluate Gecko using 62 surveillance videos as camera streams collected from YouTube using the keyword "live camera HD". To ensure that our video sessions encompass meaningful data drifts, we select videos lasting from a few minutes up to an hour with a cumulative length of 25 hours. Our dataset contains diverse cities and scenes, including varying weather conditions and times of day. Notably, in every experiment, we refrain from playing the same video segment on any edge devices twice to prevent artificially amplifying the gain from model reuse. Surveillance video is a particularly challenging situation for testing our framework due to the frequent changes in scenes and objects. For the lightweight and golden models, we employ YOLOX-Nano and YOLOX-X [38], respectively. Our models are pre-trained on the COCO [39] dataset.

Metrics: We employ the average GPU usage, accuracy, and response time as metrics to evaluate the performance of different schemes for real-time object queries in videos. To obtain the average GPU usage, we run `nvidia-smi` at each edge device at regular intervals throughout the execution of our workloads. For accuracy assessment, we compare the results

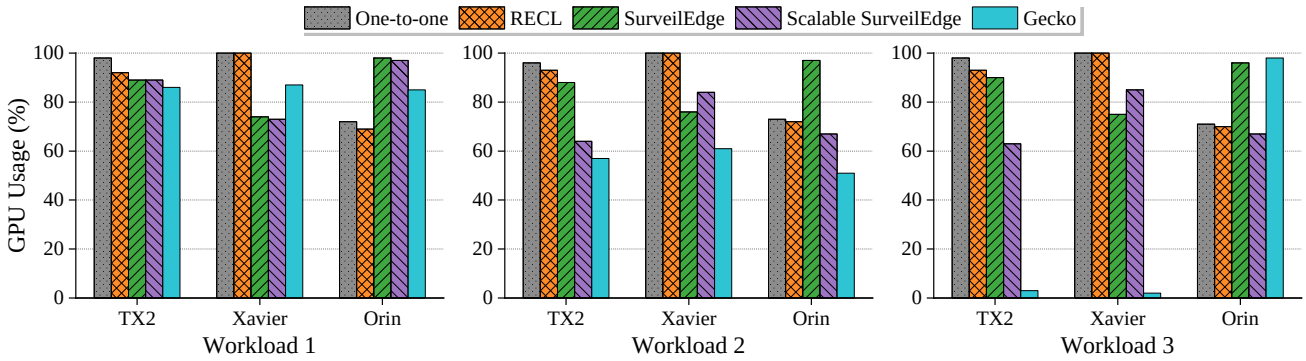


Fig. 3. End-to-End Evaluation: Average GPU usage comparison of different methods across three workloads.

on the edge device with the results for the same frames using the golden model (similar to prior work [11], [13]). Notably, we use the F-1 score, which accounts for both precision p and recall r of the queries, to measure the accuracy of the queries.

Baselines: We compare Gecko against the following schemes:

- **One-to-one**. We run one Docker [40] container at the edge device to query one class of object in a camera.
- **RECL**. RECL [13] is a novel video analytics system with model selection and model adaptation. We only use it to query objects in our test though it applies to multiple tasks. It has an online procedure for model selection from the model zoo and employs a server to adapt lightweight expert models running on edge devices continually. Comparing Gecko with this scheme will demonstrate the advantage of having a scalable system with model sharing.
- **SurveilEdge**. SurveilEdge [11] is a collaborative cloud-edge system designed for real-time querying of surveillance video streams using specific models. The ensuing experimental results will demonstrate comprehensive enhancements of Gecko compared to SurveilEdge.
- **Scalable SurveilEdge**. We additionally implement a scalable version of SurveilEdge that can query the same class of objects across multiple cameras using a single model.

B. Results

End-to-end Performance. Initially, we evaluate the end-to-end performance of Gecko in comparison to baselines over 12 cameras, which replay videos from our dataset. Upon the end of one video, we go on with another distinct video from the dataset, ensuring no repetition as it positively influences the accuracy gain attributed to model selection. For methods that require the cloud server, we employ an NVIDIA V100 GPU cloud server. And for methods without scheduling algorithms, we fix the allocation of two cameras to the TX2, five cameras to the Xavier, and five cameras to the Orin. In this case, we simplify workload 2 to query a specific class of object across all cameras attached to the single device. Figure 3 summarizes the GPU usage of each device across the three workloads. We also report the overall accuracy and average response time for each workload in Table II. Note that ensuring timeliness under resource contention may require skipped processing of many frames containing query objects, which significantly drops the overall average accuracy. The main takeaways are:

TABLE II
END-TO-END EVALUATION: ACCURACY AND RESPONSE TIME FOR DIFFERENT METHODS ACROSS THREE WORKLOADS.

Scheme	Query Workload	Overall Accuracy	Average Response Time
One-to-one	Workload 1	65.9%	10.6s
	Workload 2	68.5%	3.3s
	Workload 3	63.4%	3.4s
RECL	Workload 1	76.4%	11.8s
	Workload 2	75.9%	6.8s
	Workload 3	75.3%	6.9s
SurveilEdge	Workload 1	81.1%	63.3s
	Workload 2	78.8%	16.9s
	Workload 3	79.4%	16.9s
Scalable SurveilEdge	Workload 1	80.8%	64.2s
	Workload 2	74.7%	16.4s
	Workload 3	70.6%	16.2s
Gecko	Workload 1	**92.7%**	12.4s
	Workload 2	**92.1%**	6.2s
	Workload 3	**91.3%**	4.0s

1. Overall, Gecko outperforms all baselines by a large margin, executing query tasks with lower GPU usage while maintaining or enhancing performance. Gecko improves accuracy by up to 23.6%∼27.9% compared to the one-to-one method across all three workloads. With respect to resource consumption, Gecko obtains at least a 2x GPU utilization improvement for workloads 2 & 3. Notably, for workload 3, all video streams are allocated to Orin for query execution, allowing the remaining two idle devices to handle more upcoming queries.
2. One-to-one has the fastest response time as it only requires pulling and executing containers on the edge devices. Besides responsiveness, this method has the worst performance. Gecko delivers lower query processing time as each video query pipeline is executed entirely on the edge device.
3. RECL has similar or even faster response times than Gecko, as it also provides model selection but no scheduling.

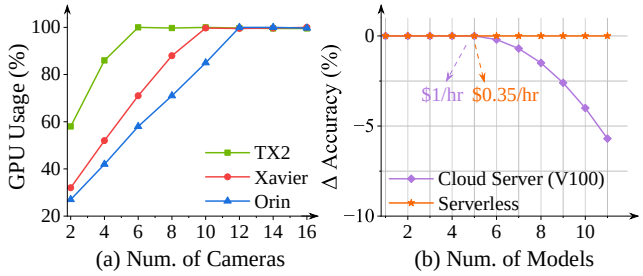


Fig. 4. Scalability analysis. (a) Number of cameras supported by 3 edge devices. (b) Average accuracy degradation comparison between the single cloud server and serverless when retraining multiple models.

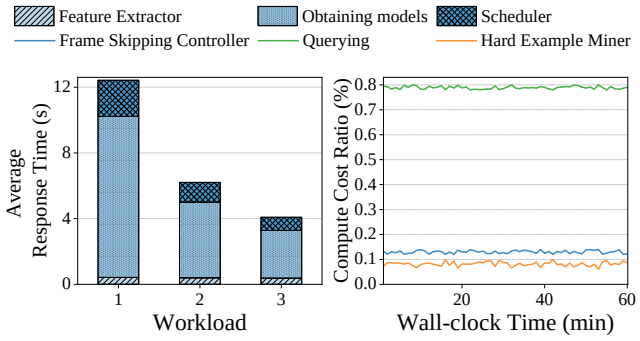


Fig. 5. Breakdown of cost by the components of Gecko.

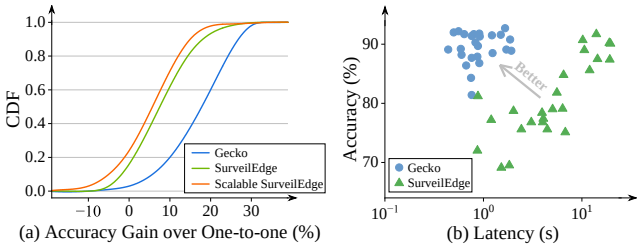


Fig. 6. Accuracy and latency analysis. (a) CDF of accuracy improvement over the one-to-one scheme across all video segments. (b) Latency vs. Accuracy for Gecko and SurveilEdge.

Nonetheless, it is unsuited for our scenario as it does not take into account resource contention among a number of video streams. The results demonstrate that RECL encounters significant GPU resource contention on the device Xavier across all workloads, which leads to RECL having the second-lowest overall query accuracy of all methods.

4. SurveilEdge has comparatively high accuracy in our experiments, yet approximately 12% lower compared to Gecko. This is mainly because it cannot cope with dynamic video content changes (data drifts) at runtime. Additionally, SurveilEdge does not offer model sharing, resulting in its GPU utilization still being less than half that of Gecko. Moreover, SurveilEdge's response time is the slowest, as it requires fine-tuning a specific model in the cloud based on selected samples before the query can begin.
5. Scalable SurveilEdge supports model sharing, achieving higher GPU utilization while compromising marginally on accuracy compared to naive SurveilEdge. However, due to a lack of runtime resource reallocation, the scalable SurveilEdge is also not as resource efficient as Gecko.

Scalability Analysis. For the practical deployment of Gecko, it is vital to consider the scalability to handle numerous cameras and queries. Figure 4(a) shows the maximum number of video streams attached to a single model running on edge devices TX2, Xavier, and Orin is 4, 9, and 12, respectively. Figure 4(b) shows the accuracy degradation (w.r.t. single model) when a single GPU server and serverless support multiple models. Up to 5 models can be retrained on an NVIDIA Tesla V100 GPU on the server without causing accuracy loss, while serverless can support unlimited model retraining due to its elasticity feature. Moreover, serverless costs only $0.35 per hour when supporting 5 models retraining, while renting a V100 GPU in the cloud expenses at least $1 per hour.

Breakdown of Cost. Figure 5 shows the cost of different components in Gecko. The average response time diminishes in the sequence of workloads 1, 2, and 3. The reduction can be attributed to a decrease in the number of pulled models, and increasing streams that can share models already running on edge devices. Furthermore, compared to querying, the frame skipping controller and the hard example miner only take up a minor computational overhead stably due to optimizing resource reallocation at runtime.

Accuracy Consistency. To demonstrate the benefit that Gecko consistently improves accuracy, Figure 6(a) plots the cumulative distribution of accuracy improvement of Gecko, SurveilEdge and the scalable SurveilEdge in comparison to the one-to-one scheme across all video segments. Gecko achieves better accuracy than One-to-one in 97% of video segments, while SurveilEdge and the scalable SurveilEdge are only better than One-to-one in 84% and 76% of the video data. In addition, Gecko achieved 20% higher accuracy than over one-to-one in 40% of videos, while SurveilEdge and the scalable SurveilEdge achieved it in only 7% and 3% of videos.

Latency/Accuracy Distribution. We further fully evaluate Gecko's latency and accuracy benefit compared to SurveilEdge using workload 1. We measure the average query latency per frame and the query averaged accuracy in each video segment. As shown in Figure 6(b), SurveilEdge cannot simultaneously achieve the same latency and accuracy as Gecko, as it relies on cloud-based inference to improve the query accuracy of specific frames, which brings great waiting latency.

C. Ablation Studies

Model Selection. To examine the effect of model selection introduced in §III-B, we measure the performance of Gecko when only using one continuously retrained model to query all objects in a class (all-in-one). As shown in Figure 7(a), Gecko with model selection consistently outperforms the all-in-one scheme over time, and is more robust. This is because edge models are designed with fewer weights and less complex architectures compared to general models. One edge model can memorize only a limited number of object appearances and scenes, and is vulnerable to a dramatic drop in accuracy

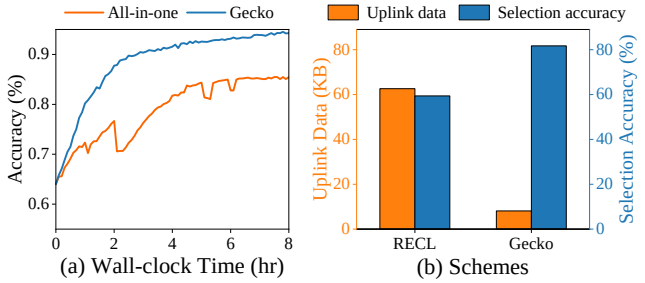


Fig. 7. Impact of Gecko's model selection. (a) shows query accuracy over time, and (b) shows uplink data and selection accuracy statistics compared with RECL's model selection.

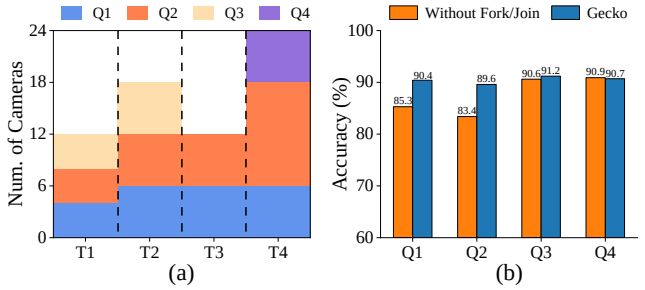


Fig. 8. Impact of FORK/JOIN primitives on the accuracy of different queries in workload 2.

when live video data diverges significantly from the recent training data. In addition, we compare the accuracy predictor of Gecko with RECL's gating network (using ResNet18 [33] that given an image, assigns a score to each model) within the same collection of models. Figure 7(b) shows the uplink data and selection accuracy of the two schemes. Owing to the lack of feature extraction, RECL uploads nearly 8x more data than Gecko. In contrast, Gecko exhibits a 20% higher probability of selecting the optimal models than RECL.

FORK/JOIN Primitives. We estimate the impacts of the FORK/JOIN of model execution on query accuracy at runtime. To this end, we utilize workload 2 and query 4 distinct objects on the cluster of the three edge devices. The four queries Q1∼Q4 are executed in a blended mode, with the number of involved cameras varying dynamically over time T1∼T4, as depicted in Figure 8(a). Figure 8(b) shows that certain queries suffer a considerable decline in accuracy due to resource contention when FORK/JOIN primitives are disabled.

Fine-grained Stream Transfer. Transferring streams to attach another specific model in response to data drift can affect query performance. To this end, we compare Gecko's accuracy when disabling the fine-grained transfer and explore the rate of reusing models presently running on edge devices, which obviates the need to pull a new model from the cloud. Table III illustrates that disabling transfer leads to a 1∼3% decrease in accuracy for workloads 2 and 3. Furthermore, it is worth noting that more than 70% of transfer streams can reuse models that are already running on devices, thereby incurring no computational costs of executing new models.

Hard Example Mining. Table IV compares the hard example mining method described in §III-D with other frame sampling methods for retraining. The fixed sampling rate method consistently uploads frames by periodically random sampling (5 FPS) on real-time video. Adaptive sampling [27] dynamically modifies the frame sampling rate at the edge device depending on the degree and velocity of changes in the video scene. Table IV shows that Gecko's hard example mining method is effective. Overall, it achieves a 5.2% and 3.8% improvement in accuracy compared to the fixed sampling rate method and adaptive sampling, while reducing the costly uplink bandwidth overhead.

TABLE III
EFFECT OF FINE-GRAINED TRANSFER WITH THREE WORKLOADS.

Workload	Accuracy (%)		Reuse rate (%)
	No transfer	Gecko	
Workload 1	92.6	**92.7**	-
Workload 2	90.3	**91.8**	89.5
Workload 3	88.6	**91.2**	73.7

TABLE IV
EFFECT OF VARIOUS TRAINING FRAME SAMPLING STRATEGIES.

Strategy	Up BW (Kbps)	Average accuracy (%)
Fixed Sampling Rate	2485	86.70
Adaptive Sampling	275	88.03
Hard Example	**169**	**91.86**

V. CONCLUSION

In this paper, we present Gecko to enable resource-efficient and accurate object queries in real-time video streams at the edge. Specifically, we design a content-aware feature-based accuracy predictor for model selection and a scheduler to assign selected models to edge devices. To handle resource contention at runtime, Gecko adaptively reallocates computing resources for different streams by dynamic frame skipping control and different models by FORK/JOIN primitives. Besides, Gecko improves query accuracy on changing video content through fine-grained stream transfer and continuous learning of lightweight models. Compared to prior work for real-time object queries in videos, our experiments show that Gecko successfully increases query accuracy by approximately 12% with 2× lower resource costs.

In the future, we plan to enhance our work along two paths. (1) Support additional execution targets to further improve query experience. (2) Add optimization (e.g., model merging [41]) by more fine-grained profiling and planning to fully exploit the affiliated resources of edge devices.

ACKNOWLEDGMENT

This research is supported by the Key Research and Development Program of Guangdong Province under Grant No.2021B0101400003, the National Natural Science Foundation of China under Grant No.62072196, and the Creative Research Group Project of NSFC No.61821003.

REFERENCES

[1] MarketsandMarkets, "Video Surveillance Market - Global Forecast to 2028," https://www.marketsandmarkets.com/Market-Reports/video-surveillance-market-645.html.

[2] M. Zhang, F. Wang, and J. Liu, "Casva: Configuration-adaptive streaming for live video analytics," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 2168–2177.

[3] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 377–392.

[4] L. Zhang, Y. Zhang, X. Wu, F. Wang, L. Cui, Z. Wang, and J. Liu, "Batch adaptative streaming for video analytics," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 2158–2167.

[5] C. Qu, R. Singh, A. Esquivel-Morel, and P. Calyam, "Learning-based multi-drone network edge orchestration for video analytics," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 1219–1228.

[6] L. Wang, K. Lu, N. Zhang, X. Qu, J. Wang, J. Wan, G. Li, and J. Xiao, "Shoggoth: Towards efficient edge-cloud collaborative real-time video inference via adaptive online learning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[7] M. Hanyao, Y. Jin, Z. Qian, S. Zhang, and S. Lu, "Edge-assisted online on-device object detection for real-time video analytics," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[8] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2022, pp. 119–135.

[9] K. Yang, J. Yi, K. Lee, and Y. Lee, "Flexpatch: Fast and accurate object detection for on-device high-resolution live video analytics," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 1898–1907.

[10] NVIDIA, "NVIDIA Jetson," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/.

[11] S. Wang, S. Yang, and C. Zhao, "Surveiledge: Real-time video query based on collaborative cloud-edge deep learning," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2519–2528.

[12] B. Hu, P. Guo, and W. Hu, "Video-zilla: An indexing layer for large-scale video analytics," in *Proceedings of the 2022 International Conference on Management of Data (SIGMOD)*, 2022, pp. 1905–1919.

[13] M. Khani, G. Ananthanarayanan, K. Hsieh, J. Jiang, R. Netravali, Y. Shu, M. Alizadeh, and V. Bahl, "Recl: Responsive resource-efficient continuous learning for video analytics," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023, pp. 917–932.

[14] G. H. Apostolo, P. Bauszat, V. Nigade, H. E. Bal, and L. Wang, "Live video analytics as a service," in *Proceedings of the 2nd European Workshop on Machine Learning and Systems*, 2022, pp. 37–44.

[15] R. Xu, C.-l. Zhang, P. Wang, J. Lee, S. Mitra, S. Chaterji, Y. Li, and S. Bagchi, "Approxdet: content and contention-aware approximate object detection for mobiles," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 449–462.

[16] A. Suprem, J. Arulraj, C. Pu, and J. Ferreira, "Odin: Automated drift detection and recovery in video analytics," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, p. 2453–2465, 2020.

[17] D. Kang, P. Bailis, and M. Zaharia, "Blazeit: optimizing declarative aggregation and limit queries for neural network-based video analytics," *Proceedings of the VLDB Endowment*, vol. 13, no. 4, pp. 533–546, 2019.

[18] F. Bastani, S. He, A. Balasingam, K. Gopalakrishnan, M. Alizadeh, H. Balakrishnan, M. Cafarella, T. Kraska, and S. Madden, "Miris: Fast object track queries in video," in *Proceedings of the 2020 International Conference on Management of Data (SIGMOD)*, 2020, pp. 1907–1921.

[19] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: optimizing neural network queries over video at scale," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1586–1597, 2017.

[20] J. Cao, K. Sarkar, R. Hadidi, J. Arulraj, and H. Kim, "Figo: Fine-grained query optimization in video analytics," in *Proceedings of the 2022 International Conference on Management of Data (SIGMOD)*, 2022, pp. 559–572.

[21] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 115–131.

[22] J. Lee, B. Varghese, and H. Vandierendonck, "Roma: Run-time object detection to maximize real-time accuracy," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2023, pp. 6405–6414.

[23] T.-W. Chin, R. Ding, and D. Marculescu, "Adascale: Towards real-time video object detection using adaptive scaling," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 431–441, 2019.

[24] R. Xu, J. Lee, P. Wang, S. Bagchi, Y. Li, and S. Chaterji, "Litereconfig: Cost and content aware reconfiguration of video object detection systems for mobile gpus," in *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys)*, 2022, pp. 334–351.

[25] A. H. Jiang, D. L.-K. Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M. A. Kozuch, P. Pillai, D. G. Andersen, and G. R. Ganger, "Mainstream: Dynamic stem-sharing for multi-tenant video processing," in *2018 USENIX Annual Technical Conference (ATC)*, 2018, pp. 29–42.

[26] Y. Huang, H. Zhao, X. Qiao, J. Tang, and L. Liu, "Towards video streaming analysis and sharing for multi-device interaction with lightweight dnns," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[27] M. Khani, P. Hamadanian, A. Nasr-Esfahany, and M. Alizadeh, "Real-time video inference on edge devices via adaptive model streaming," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 4572–4582.

[28] S. Jin, A. RoyChowdhury, H. Jiang, A. Singh, A. Prasad, D. Chakraborty, and E. Learned-Miller, "Unsupervised hard example mining from videos for improved object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 307–324.

[29] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter, "Sprocket: A serverless video processing framework," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2018, pp. 263–274.

[30] M. Zhang, F. Wang, Y. Zhu, J. Liu, and B. Li, "Serverless empowered video analytics for ubiquitous networked cameras," *IEEE Network*, vol. 35, no. 6, pp. 186–193, 2021.

[31] B. Hou, S. Yang, F. Kuipers, L. Jiao, and X. Fu, "Eavs: Edge-assisted adaptive video streaming with fine-grained serverless pipelines," in *INFOCOM 2023 - IEEE International Conference on Computer Communications*, 2023.

[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[34] Q. Hu, M. Zhang, P. Sun, Y. Wen, and T. Zhang, "Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023, pp. 457–472.

[35] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-iou loss: Faster and better learning for bounding box regression," in *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, vol. 34, no. 07, 2020, pp. 12 993–13 000.

[36] L. Cai, X. Shen, J. Pan, and J. W. Mark, "Performance analysis of tcp-friendly aimd algorithms for multimedia applications," *IEEE Transactions on Multimedia*, vol. 7, no. 2, pp. 339–355, 2005.

[37] NVIDIA, "DeepStream SDK," https://developer.nvidia.com/deepstream-sdk.

[38] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430*, 2021.

[39] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," *arXiv preprint arXiv:1405.0312*, 2014.

[40] D. Merkel *et al.*, "Docker: Lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 239, no. 2, p. 2, 2014.

[41] A. Padmanabhan, N. Agarwal, A. Iyer, G. Ananthanarayanan, Y. Shu, N. Karianakis, G. H. Xu, and R. Netravali, "Gemel: Model merging for memory-efficient,real-time video analytics at the edge," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023, pp. 973–994.