2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM) | 978-1-6654-0876-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/WoWMoM54355.2022.00034

`SmartDet`: Context-Aware Dynamic Control of Edge Task Offloading for Mobile Object Detection

Davide Callegaro*, Marco Levorato* and Francesco Restuccia†
* Computer Science Dept., University of California, Irvine, United States
† Department of Electrical and Computer Engineering, Northeastern University, United States

***Abstract*—Mobile devices such as drones and autonomous vehicles increasingly rely on object detection (OD) through deep neural networks (DNNs) to perform critical tasks such as navigation, target-tracking and surveillance, just to name a few. Due to their high complexity, the execution of these DNNs requires excessive time and energy. Low-complexity object tracking (OT) is thus used along with OD, where the latter is periodically applied to generate "fresh" references for tracking. However, the frames processed with OD incur large delays, which does not comply with real-time applications requirements. Offloading OD to edge servers can mitigate this issue, but existing work focuses on the optimization of the offloading process in systems where the wireless channel has a very large capacity. Herein, we consider systems with constrained and erratic channel capacity, and establish *parallel* OT (at the mobile device) and OD (at the edge server) processes that are resilient to large OD latency. We propose `Katch-Up`, a novel tracking mechanism that improves the system resilience to excessive OD delay. We show that this technique greatly improves the quality of the reference available to tracking, and boosts performance up to 33%. However, while `Katch-Up` significantly improves performance, it also increases the computing load of the mobile device. Hence, we design `SmartDet`, a low-complexity controller based on deep reinforcement learning (DRL) that learns to achieve the right trade-off between resource utilization and OD performance. `SmartDet` takes as input highly-heterogeneous context-related information related to the current video content and the current network conditions to optimize frequency and type of OD offloading, as well as `Katch-Up` utilization. We extensively evaluate `SmartDet` on a real-world testbed composed by a JetSon Nano as mobile device and a GTX 980 Ti as edge server, connected through a Wi-Fi link, to collect several network-related traces, as well as energy measurements. We consider a state-of-the-art video dataset (ILSVRC 2015 - VID) and state-of-the-art OD models (EfficientDet 0, 2 and 4). Experimental results show that `SmartDet` achieves an optimal balance between tracking performance – mean Average Recall (mAR) and resource usage. With respect to a baseline with full `Katch-Up` usage and maximum channel usage, we still increase mAR by 4% while using 50% less of the channel and 30% power resources associated with `Katch-Up`. With respect to a fixed strategy using minimal resources, we increase mAR by 20% while using `Katch-Up` on 1/3 of the frames.**

I. Introduction

Real-time object detection (OD) is a critical component of a wide array of current and future applications and systems, including autonomous vehicles [1] and city-monitoring [2]. In a nutshell, OD aims at the precise identification and positioning of objects contained in an image or a sequence of images. The outcome is a set of bounding boxes (see Fig. 2 for a graphical example) and associated labels describing the objects.

This work was supported by the Intel Corporation and the NSF grant MLWiNS-2003237 and CNS-2134567.

The majority of existing frameworks leverages deep neural networks (DNNs) to perform OD [3, 4]. However, state-of-the-art DNNs have very large complexity and cannot be entirely executed on mobile devices [5]. While lower-complexity algorithms exist [6, 7], they achieve poor performance – *e.g.*, measured as accuracy, recall, or precision (see Section III-A for a definition of the metrics) – compared to state-of-the-art models. For instance, Yolo-Lite [8] achieves a frame rate of 22 frames per second on embedded devices, but has a mean average precision (mAP) of 12.36% on the COCO dataset [9]. EfficientDet 0-7 [10] is a family of state-of-the-art OD models that offer increasing performance at the price of an increasing complexity. EfficientDet 7 achieves mean Average Precision (mAP) of 55.1%, but leverages 52M parameters. Even EfficientDet 0, the simplest model in the family, which achieves 33% on the COCO dataset, is 2x times more complex than SSD-MobileNet v2: a lower-performance DNN specifically designed for mobile platforms, which achieves mAP of 20%, and in our experiments can provide up 6 frames per second (fps), while significantly increasing power consumption. We remark that pruning and quantization, two techniques widely used to make DNN simpler, greatly degrade OD performance.

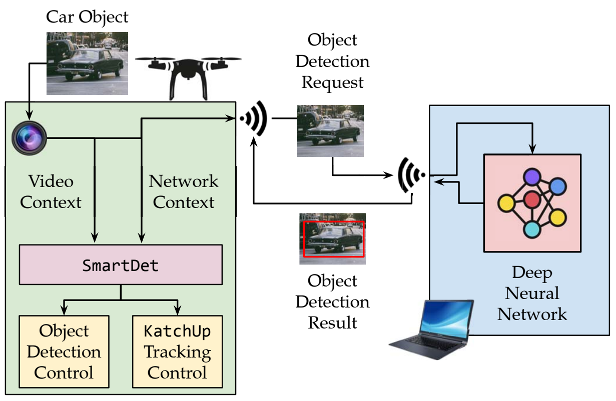


Fig. 1: Overview of `SmartDet` main components.

There are two main strategies to address this issue: (1) *Edge Computing:* [11] the mobile device offloads the OD stream to edge servers - compute capable devices located at the wireless network's edge; (2) *Object Tracking:* using a lower complexity object tracker in conjunction with a high(er) complexity object detector. Prior work considers these two strategies in isolation, and we contend that such approach fails to provide acceptable performance in many settings of great relevance (see discussion below). Our paper presents `SmartDet`: the first framework to propose the use of object detection and tracking in an edge computing setting. Notably, the composition of the two strategies presents both unique challenges and opportunities, which we are the first to explore in this paper.

Before introducing `SmartDet`, we first discuss the two strategies mentioned above.

(1) *Joint Detection and Tracking:* To address the excessive computational overhead associated with OD, object tracking (OT) is often used in mobile computing contexts [12]. Trackers assume temporal correlation in the sequence of images, and use a previously computed reference to analyze a new image [13]. The idea behind OT is fairly simple; given a video, OD is performed periodically, and its outcome serves as reference for OT on the remaining frames. Since OT is less computationally expensive than OD, energy consumption and computing load are reduced [14]. However, due to constraints in the computing power of mobile devices, the execution of OD may take a large amount of time. We show in Section V-B that an outdated OD reference can degrade mean Average Recall (mAR) performance by up to 25% on the average due to these effects, which are exacerbated in videos with highly dynamic objects.

(2) *Edge Computing:* prior work considered approaches where all the frames are processed using OD, and mobile devices offload the streams of OD tasks to edge servers. This partially addresses the issue of high computational complexity, as edge servers has considerably more computing power and energy resources compared to mobile devices. However, an approach purely based on offloading OD has the following drawbacks: (*a*) wireless channels usually have a constrained and erratic capacity, especially in applications such as autonomous vehicles where mobile devices are often moving. This leads to high communication latency and large latency variations [5, 15]; (*b*) frequent transfer of images consumes a large amount of channel capacity – *e.g.*, up to 20% of available Wi-Fi bandwidth in our experiments – possibly resulting in channel congestion [16]; (*c*) as all frames are transferred to the edge server for analysis, each mobile device imposes a considerable processing load to the edge server. Existing work focuses on scenarios where the wireless link capacity is extremely large and substantially steady (*e.g.*, 350Mbps, and the edge servers have – individually or collectively – high computing power (*e.g.*, see [17]).

In contrast with existing work, in this paper we address the challenging scenarios where *the capacity of the wireless channel is limited and erratic, and the edge servers have limited computing power*. We propose to establish two parallel processes: the mobile device executes OT on all frames, and only some of the frames are sent to the edge server for OD. This approach assigns to the mobile device a lightweight analysis process, thus reducing the requirements on available resources and takes advantage of the greater computing power of edge servers, while imposing a moderate communication and computing load. However, in order to maximize the performance of such system there are several challenges that need to be addressed: (*1*) variations in the capacity of the channel may still result in some of the OD references to refer to outdated frames, which may harm tracking performance; (*2*) the tracking performance is greatly influenced not only by reference delay, but by other parameters such as OD period and accuracy – which in turn determine channel and server load. To address the above key issues, this paper makes the following novel contributions:

• We introduce a new tracking strategy, which we refer to as `Katch-Up` (Section IV-A), to make the edge-mobile system resilient to OD delay. In `Katch-Up`, when an object detection outcome is received, we re-track the sequence of images starting from the time at which the frame was generated. This technique greatly improves the quality of the reference available to tracking against OD delay, thus boosting performance. Fig. 2 shows two examples of pictures where `Katch-Up` was and was not applied to the tracking process - the better quality of the bounding boxes is apparent;

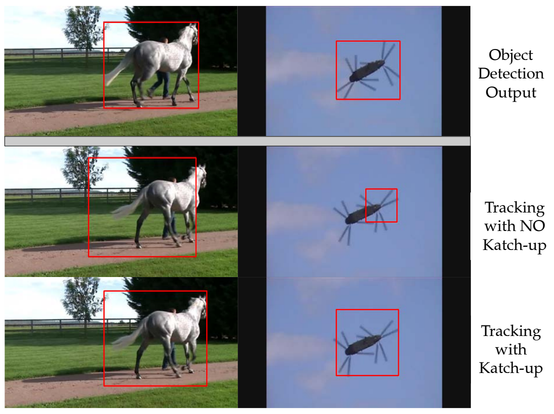


Fig. 2: Examples of bounding boxes produced by tracking with and without `Katch-Up` (400ms object detection delay).

• While `Katch-Up` increases performance, it also increases the computing load of the mobile device. Thus, we define a dynamic control problem based on Deep Reinforcement Learning (DRL) (Section IV-B), where the controller takes as input contextual and historical information and determines (i) `Katch-Up` activation, (ii) which frames are submitted for object detection and (iii) which object detection model is used. This formulation enables tight control of the delay/energy/accuracy trade-off based on contextual information. The use of DRL is motivated by (*i*) the fast temporal variations of the system do not allow for long-term static optimization and require tight dynamic control; (*ii*) the future statistics of the system state and performance depend on past controllable parameters (*i.e., actions of the DRL agent in our framework*), so that the optimization needs to be formulated as a correlated control sequence controlling the system's state trajectory, rather than a one-shot optimization of the system parameters. We demonstrate in Section V that performance is a function not only of "system" variables such as delay, but also of measurable "content" variables such as the dynamics of the objects, and "algorithm" parameters (*e.g.*, the object detection model).

• We refer to the resulting framework, illustrated in Fig. 1, as `SmartDet`. We train and evaluate `SmartDet` on a real-world experimental platform. Our results demonstrate that by adapting the strategy to the context, `SmartDet` achieves superior tracking performance (4% improvement) using considerably less power (60% reduction in `Katch-Up` activation) and channel and edge server resources (50% reduction) compared to any non-adaptive strategy. With respect to a fixed strategy without `Katch-Up`, `SmartDet` improves mAR by 20%. Importantly, the `SmartDet` DRL agent uses significantly different control strategies for different parameters of the system (link quality) and video (target mobility), thus confirming the need for context-aware control.

II. Related work

Thanks to its relevance in many critical real-world applications, real-time video analytics has recently attracted significant attention. Prior work has proposed techniques to reduce the computation burden and latency of image analysis algorithms to match the resources and constraints of mobile applications, including model pruning [7], advanced compression [6] and split DNNs [18]. For the same purpose, some recent contributions apply a joint OD and OT strategy on video streams [13, 19]. Among others, the recent ApproxDet framework [19] is one of the closest to our work. However, the latter focuses on a purely local computing scenario, where the mobile device executes both OD and OT. In this context, ApproxDet selects which frames are processed using OD and which using OT, as well as some computing parameters. However, this methodology suffers from a critical issue – frames analyzed using OD incur a large delay. As a result: (*a*) the bounding boxes for those frames would become available after an excessive amount of time to support real-time applications; (*b*) during OD processing, a non-negligible number of frames would be completely disregarded; and (*c*) in a real-world setting, where the captured scene evolves during OD analysis, the reference provided by OD would become obsolete, and tracking performance would significantly degrade unless slowly-changing videos were considered, as demonstrated by our results in Section V. Noticeably, ApproxDet only considers a subset of slowly varying videos from *ILSVRC 2015 - VID* with large subjects, and only shows 95 percentile latency. In this paper, we propose an edge computing-based solution where OD and OT are executed *in parallel* on different machines. Although this approach provides firm guarantees on bounding boxes delay, it makes control more challenging (*e.g.*, due to the erratic behavior of the wireless channel), which we address by developing a context-aware DRL controller. Furthermore, we introduce `Katch-Up` to increase resiliency to OD delay.

In the class of OD-only solutions based on edge computing, to decrease OD latency image segmentation has been explored by recent some frameworks, including ELF [17]. The core idea is to adapt computing based on previous OD outcomes to optimize analysis over multiple edge servers. Specifically, ELF produces a region proposal prediction, based on LSTM with attention networks, that predicts the new bounding boxes given the previous ones. Next, the frame is fragmented to distribute the load to the different edge servers based on their load. Thus, ELF focuses on remote OD only, while we propose the use of dual and parallel OD-OT, which provides firm latency guarantees. Moreover, the scenario considered in ELF centers on load distribution across multiple edge servers over a high-capacity channel. Conversely, the key innovation of `SmartDet` is to increase resiliency to erratic and limited channel capacity, as well as to latency variations, to support OT based on the current video content and networking contexts.

III. Real-Time Distributed Video Analysis

In this Section, we provide an overview of the distributed video analysis scenario considered in this paper.

A. Background and System Model

Figure 3 depicts the edge-based object detection process under investigation. Specifically, we consider a mobile device capturing a sequence of images[1] $f_1, \dots f_{N_i}$, at a fixed rate of r images per second. The general objective of the system is to analyze the images to detect objects. Specifically, each image f_i is associated with a vector of object descriptors $O_i{=}(b_1, l_1, \dots, b_{N_i}, l_{N_i})_i$, where b_j and l_j are respectively the bounding boxes enclosing the j-th object in the image and its label. The bounding box is defined as the minimum rectangle enclosing all the pixels of an object, and the label is an integer corresponding to a class describing the nature of the object in a finite set. We note that the number of objects N_i in the image f_i is a function of the image itself.

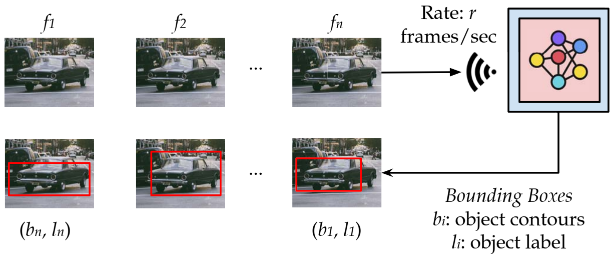


Fig. 3: Object detection (OD) in edge-based systems.

We denote as O_i the vector containing the reference ground truth. The system extracts an approximation $\hat{O}_i$ of O_i using the object detection function $\phi(\cdot)$, i.e., $\hat{O}_i{=}\phi(f_i)$. The quality of the approximation is defined by metrics such as mean Average Precision (mAP) or mean Average Recall (mAR). These metrics evaluate the quality of the bounding boxes generated by the algorithm, as well as their classification. Henceforth, we will use mAR to measure the ability of our approach to recognize targets. This metric is based on recall, that is, the normalized number of targets correctly labeled in a single frame with an

[1] In this paper, we will use the words image and frame interchangeably.

intersection over union (defined as the intersection area of the ground truth and predicted bounding boxes divided by their union area) larger than 0.5. To compute mAR, the recall is averaged over a whole video or a portion of it.

We consider deep neural networks (DNNs) for object detection (OD), which are the *de-facto* new standard to perform object detection in real-world applications. Many of these networks are categorized in families of networks: CenterNet Hourglass [20], SSD Resnet [21], Faster RCNN Resnet, Yolo [8], just to name a few. A significant number of these architectures are scalable, which creates a number of DNNs that have performance proportional to their size. However, the complexity of most of these models is beyond the capabilities of current mobile devices. Even relatively powerful embedded computers such as the NVIDIA Jetson Nano we use in this paper cannot execute even medium size DNN models for object detection due to memory constraints. Other models are supported, but their execution requires an excessive time and significantly increases power consumption. We report specific values in Section V Table 1.

We then take a joint OD and object tracking (OT) approach, and define two distinct functions for the estimation of $\hat{O}_i$. Formally, in addition to the OD function $\hat{O}_i{=}\sigma^{\rm od}(f_i)$, we define the OT function as $\hat{O}_i{=}\sigma^{\rm ot}(\hat{O}_{i-1}, f_i)$. Thus, $\sigma^{\rm ot}$ takes as input the current image as well as the estimated object descriptors associated with the previous image to leverage temporal correlation in the image stream. Different types of OT algorithms have been proposed. Some of these methods are based on features extraction algorithms [22] (such as Histogram of Oriented Gradients) or deep architectures (using for example siamese networks, resulting in algorithms such as GOTURN [23]). Other algorithms are based on optical flow, using classical techniques such as the Lucas-Kanade point tracking. Among these, MedianFlow [24] takes the median of flow vectors generated to predict where the new location of the bounding box. In this paper, we use MedianFlow due to its low-complexity, which satisfies the latency and resource constraints which characterize real-time applications.

Critically, OT algorithms rely not only on a good estimate of the object descriptors associated with the previous image, *but also on a limited change in the image*. Due to the nature of these algorithms, their performance is inversely proportional to the rate with which the video changes. Expanding the neighboring region where to look for a matching set of features (extracted with DNNs or HOGs or macro-blocks of pixels) to follow fast moving objects results in higher uncertainty and consequently poorer tracking performance. Furthermore, error accumulation and consequent target instability have been well documented [19, 25]. For these reasons, OD is periodically executed to "reset" the bounding boxes by providing a new and independent reference, which is then sequentially updated using OT as new images are acquired [14]. The OT algorithm we adopted is orders of magnitude less complex compared to OD [14]. On the other hand, OD-designated frames still incur a large latency. As a result, in traditional approaches such as ApproxDet [19], where the mobile device executes both OD and OT, tracking is halted while waiting for the outcome of object detection. Thus, either the incoming frames during this time are discarded, or they are buffered and processed with a larger accumulated delay. We note that in both cases the correlation between the OD reference and the images processed with OT decreases due to the time lag.

B. Edge Offloading of Object Detection

Our core idea is to divide the video analysis into two parallel yet intermingled processes: *object tracking* executed locally at the mobile device on all frames, and *object detection* executed remotely at the edge server on a subset of frames $\mathcal{E}{\subseteq}\{0, 1, \ldots\}$. The key advantages of this strategy are the following: (i) the edge server has a larger computing power compared to the mobile device, so that the execution time of object detection is reduced, (ii) the two processes can be fully executed in parallel without sharing resources, and (iii) the overall energy consumption at the mobile device is reduced. However, offloading the execution of object detection to the edge server requires the transportation of the image to be analyzed over a wireless link. In many real-world settings, the channel capacity is constrained and erratic (*e.g.*, autonomous vehicles, millimeter wave communications, *etc.*). Moreover, offloading may result in channel congestion, thus increasing delay and amplifying data rate instability. Thus, it becomes necessary to parsimoniously send frames to the edge server.

Let us denote with $\Delta_i^{\rm od}$, $i{\in}\mathcal{E}$, the total time from the capture of the image i to the reception of the vector $\hat{O}_i^{\rm od}$ when the frame is sent to the edge server. Δ_i^{od} is the sum of communication time and computing time. The former is a function of the perceived data rate and the number of bytes used to represent the image. The latter is a function of the OD DNN model used at the edge server and its computing power. Both delay components are time-varying as they depend on channel and system parameters. We denote as $\Delta_i^{\rm ot}$ the time from the generation of the frame i to the availability of the vector $\hat{O}_i^{\rm ot}$. Due to the low complexity of the OT algorithm we adopt, we assume that the time $\Delta_i^{\rm ot}$ is fixed and smaller than the inter-frame generation $1/r$. We note that as object tracking is applied to all the images, an estimate of the bounding boxes for all the frames is readily available to the mobile device.

Consider a frame $i \in \mathcal{E}$, which is both processed locally using OT and sent to the edge server for OD. In a short amount of time, $\hat{O}_i^{\rm ot}$ becomes available as OT is executed using one of the available vectors $\hat{O}_{i-1}$. Then, $\hat{O}_i^{\rm ot}$ can be used as reference for the successive frame and so on. When $\hat{O}_i^{\rm od}$ from OD is received, the bounding boxes and labels are used as reference for the OT function $\sigma^{\rm ot}(\cdot)$ applied to frame $i{+}\lceil\Delta_i^{\rm od}\times r\rceil$. Since the OT reference is outdated, the tracking performance may degrade. While it is possible to continue using the reference obtained from object tracking on the previous frame, due to error accumulation in object tracking, a periodic *refresh* is needed. In Section V, we characterize this degradation as a function of key video and system parameters.

IV. The SmartDet Framework

Let us now present the two key contributions of this paper: (*i*) `Katch-Up`: a methodology to make the distributed video analysis system less sensitive to object detection delay, and (*ii*) `SmartDet`, a real-time control engine to optimize the tradeoff between performance and resource usage. The schematics of `SmartDet` are depicted in Fig. 4. The core of `SmartDet` is a Deep Reinforcement Learning (DRL) agent that controls which images are sent to the edge server for OD and which model is used to analyze them, as well as whether `Katch-Up` is used or not. To support these decisions, images are internally routed to the main modules: object tracker, `Katch-Up` (and `Katch-Up` buffer) and the transmission interface. The edge server receives the frames, and analyze them using the OD model indicated by the DRL controller, then returning the estimated bounding boxes and labels to the mobile device. A critical module of `SmartDet` is the state extractor, that builds state features from variables, parameters and data received by the other modules.

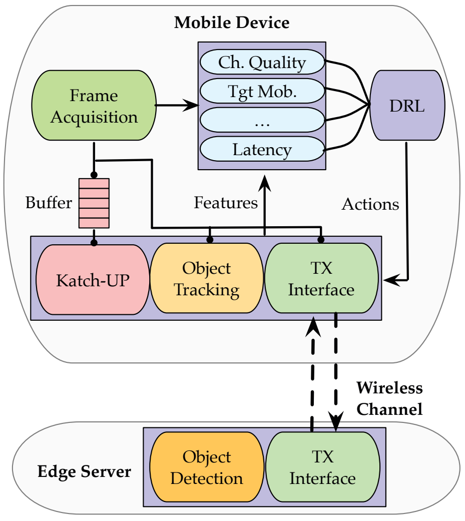


Fig. 4: Main components of `SmartDet`. At the mobile device, the DRL module (state extraction and controller) determines which frames are sent for OD and what model is used for their analysis, and the activation of `Katch-Up`. To support these actions, frames are internally routed to the different modules (object tracker, `Katch-Up` and `Katch-Up` buffer and TX interface) to support these functions. The edge server performs object detection on the received frames using the model indicated by the DRL controller.

A. *The `Katch-Up` Smart Tracking Algorithm*

As mentioned earlier, one of the main issues of OD edge offloading is the feeding of outdated references to the OT due to the communication and computing delay. As a result, applying OT algorithms to frames that might substantially differ will entail large errors due to the high uncertainty of the transposing vectors [24]. To mitigate the effect described above, we propose `Katch-Up`. Our intuition is simple yet effective: when a vector $\hat{O}_i^{\text{od}}$ from OD applied to frame i is received, the mobile device re-executes the tracker on the frames starting $i+1$ until the process "catches up" with the primary tracking process. To make an example (represented in Fig. 5), assume frame i is sent to the edge server and the corresponding reference $\hat{O}_i^{\text{od}}$ is received right before frame $i+n$ is acquired. During this time, OT is applied to frames from i to $i+n-1$ based on the reference available at the time (*i.e.*, the outcome of tracking applied to the previous frame based on a chain of tracking started from an older object detection reference). In `Katch-Up`, when $\hat{O}_i^{\text{od}}$ is received the tracking process is duplicated. *Process 1* continues to analyze incoming frames $i+n, i+n+1, i+n+2, \dots$ using the reference vector $\hat{O}_{j-1}^{\text{ot}}$ to perform tracking on frame j. *Process 2* restarts the tracking of frames $i+1, i+2, \dots$ taking $\hat{O}_i^{\text{od}}$ as a starting point to build the sequence $\hat{O}_{i+1}^{\text{ot}}, \hat{O}_{i+2}^{\text{ot}}, \dots$. Process 2 is executed as the maximum possible speed, meaning that tracking is continuous, rather than based on the frame arrival timing. Thus, Process 2 proceeds faster than Process 1, and eventually *catches up* with the latter one. Meaning, Process 2 and Process 1 generate a bounding box vector with the same index. At that point, Process 2 is terminated, and Process 1 continues using as reference the latest outcome of Process 2.

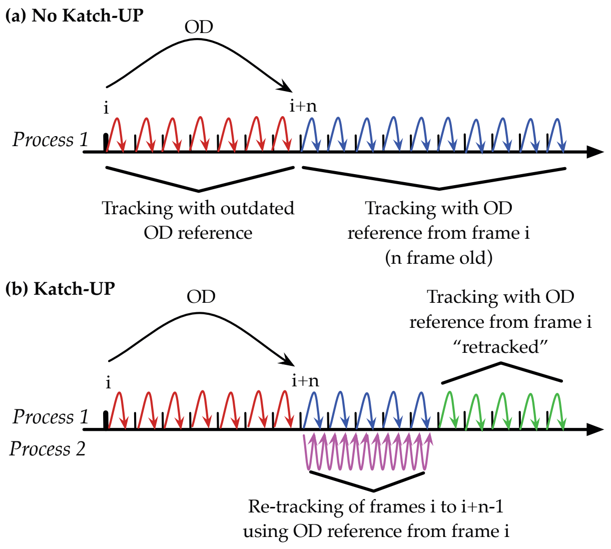

(a) No Katch-UP
OD
i
i+n
Process 1
Tracking with outdated OD reference
Tracking with OD reference from frame i (n frame old)
(b) Katch-UP
OD
Tracking with OD reference from frame i "retracked"
i
i+n
Process 1
Process 2
Re-tracking of frames i to i+n-1 using OD reference from frame i

Fig. 5: Tracking process with and without `Katch-Up` referring to the explanation in Section IV-A.

The key advantage of `Katch-Up` is that the sequence $\hat{O}_{i+1}^{\text{ot}}, \hat{O}_{i+2}^{\text{ot}}, \dots$ generated by Process 2 is more accurate compared to that generated by Process 1, as the former is based on a more recent reference from object detection. Thus, `Katch-Up` boosts tracking performance (as demonstrated in Section V), but also increases the computing load at the mobile device, as well as memory usage as some already processed frames need to be buffered.

B. Real-Time DRL-Based Control in `SmartDet`

Motivation. The system performance in terms of latency/accuracy/energy is determined by several factors and parameters, including: (*i*) whether or not to activate `Katch-Up`, (*ii*) how many and which frames are to be sent to the edge server for object detection, and (*iii*) which object detection model to use. For example, if `Katch-Up` is active, the quality of tracking improves, but energy consumption at the mobile device increases. If more frames are sent to the edge server, then tracking has more frequent references, but channel load – and thus possibly communication latency – increases as well as server load – and thus possibly computing latency. If a more complex model is used, then the reference quality for tracking improves, but so does the latency to receive the bounding boxes. Intuitively, the optimal point is determined by several variables. For instance, if the channel has a large capacity, then transmitting more frames will not significantly affect overall delay, while possibly improving tracking. Moreover, if the communication delay is small, then the use of a more complex model, with a larger execution time, may be advantageous. Conversely, if the channel capacity is small, then using a less complex model may result in a tolerable overall latency. Notice that different object detection models take as input images of different size, and may be more or less sensitive to compression. In other words, there might be a dependency between Δ_i^{od} and the model used and desired accuracy. Importantly, these tradeoffs are greatly influenced by the parameters of the video itself. Moreover, the mobility of the targets in the video influences the optimal parameter choice, where fast changes may require low-latency, more frequent, object detection These tradeoffs are detailed in Section V. The decisions in (*i*)–(*iii*) will become the knobs used by `SmartDet` to control the tradeoff between performance and resource usage.

Why DRL? We formulate our decision making process as a dynamic control problem, where a controller selects real-time actions at a fine temporal granularity based on the perceived state of the system and context. This approach is motivated by the time-varying nature of the system we consider, as well as by the correlation between current decisions and future states of the system. For instance, the period determines the sampling instants of the state, as well as how outdated the reference from OD is when applying OT to future frames. Thus, a reinforcement learning approach is the most suitable to solve our problem. Moreover, we observe that the state space is extensive, and the features are heterogeneous in nature, where some of them are continuous. A traditional "tabular" Q-Learning approach in practice would require the quantization of all features to generate a discrete space. The resulting state space would be either too large to handle using direct recursive estimation, or poorly representative. Therefore, a Deep Q-Learning approach is the natural choice for our problem.

DRL Algorithm. In DRL, the controller selects an action $u \in \mathcal{U}$ based on the current state $s \in \mathcal{S}$, where $\mathcal{U}$ and $\mathcal{S}$ are the action and state space, respectively. We synchronize the state update and action selection with the generation of images that are sent to the edge server for object detection. We index these instants with $t{=}1,2,\ldots$, and denote the state and action at time t as s_t and u_t, respectively. We adopt a Q-Learning formulation and define the function $\omega(\cdot)$ as $Q(u_{t+1}, s_{t+1}){=}\omega(s_t)$, where

$$Q(u_{t+1}, s_{t+1}) = E_{s_{t+1}|s_t,u_t}\left[E_{r_{t+1}|s_{t+1},u_t,s_t}\left[r_{t+1}|s_{t+1},u_t,s_t|\right]\right] + \gamma \max_{u'} E_{s_{t+1}|s_t,u_t}\left[Q(s_{t+1},u')\right], \quad (1)$$

and r_t is the reward accrued at time t. Thus, the Q-value $Q(s,u)$ captures the long-term – discounted – reward associated with taking action u in state s. We refer the reader to [26, 27] for a comprehensive discussion on DRL.

In the following, we define the action space, state space, cost function and network architecture of the DRL agent. We remark that the time granularity of state update and decision making is not the same as that of frame generation, as decisions are made only when a frame is sent for object detection. This also reduces the computation burden to the mobile device.

*1) **Action Space**:* We define the action $u(t)$ as the vector $(k(t), p(t), m(t))$, where (*a*) $k(t){\in}\{0,1\}$ determines whether `Katch-Up` will be used when the outcome $\hat{O}_{i_t}^{\mathrm{od}}$ is received; (*b*) $p(t) \in \{1,\ldots,P\}$ is a variable controlling the object detection period, that is, the number of frames until the next frame is sent to the edge server for object detection; (*c*) the variable $m(t){\in}\{0,\ldots,M\}$ determines which model is used for object detection of the next frame sent.

*2) **Reward Function**:* We define the reward as function of the state and action, and not of the temporal index t. Rewards refer to sequences of frames in between OD. Next, we define the reward function $R(s,u)$ as the composition of the following metrics: (*i*) mAR ($R_1(s,u)$): counts the percentage of the targets we correctly track with respect to the total number of targets; (*ii*) *KU usage* ($R_2(s,u)$): is the fraction of frames processed using `Katch-Up`, (*iii*) *Period* ($R_3(s,u)$): is the number of frames until the next scheduled OD (normalized to the maximum period). An exhaustive description is not provided due to space constraints. Note that we do not include OD latency directly in the reward function as it is not a direct application metric. Indeed, the latency perceived by the application is that of OT. However, OD latency influences tracking performance, and thus mAR. We then define $R(s,u)$ as the weighted sum

$$R(s,u) = \alpha_1 R_1(s,u) + \alpha_2 R_2(s,u) + \alpha_3 R_3(s,u). \quad (2)$$

*3) **State Space**:* The state $s{\in}\mathcal{S}$ is designed to provide information to the controller to make decisions on the action to be selected. At a high level, to estimate the Q function, the controller needs to predict some characteristics of the video and surrounding system, and connect them to a reward given the action. To this purpose, the DNN ω embedded in the controller implicitly builds a model for the temporal evolution of parameters such as video characteristics and channel. Next, we include in the state a set of *features* over a window of N past decision instants. That is, at a decision instant t, we include these features computed at $t{-}N{+}1, t{-}N{+}2, \ldots, t$. We can group these features as follows:

(*a*) *Contextual*: this includes image size and the center-to-center distance between targets as a proxy on how quickly objects are moving in the frame;
(*b*) *Self-awareness*: the latency incurred by the frames sent for object detection, KU usage (same as in the reward computation), and the selected action vector;
(*c*) *Self-evaluation*: Intersection over Union (IoU) between the running tracking and the received detection.

Examples of "*correlations*" that the agent will need to learn by experience include the temporal correlation of video change rate and OD latency. Note that the latter depends not only on the time varying channel capacity and server load, but also on the model used for OD, so that the agent *needs to implicitly learn model-to-model latency maps.* The agent also needs to learn the *non-trivial relationship between OD,* `Katch-Up`*, the (future) latency and video parameters and mAR.*

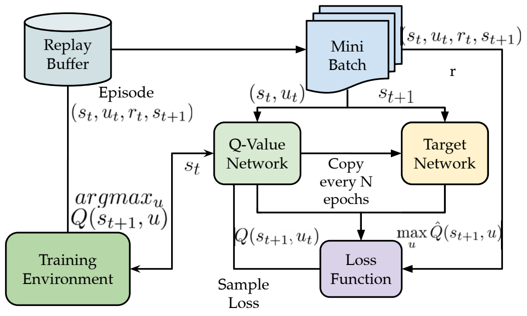


Fig. 6: Deep Q Neural network with replay buffer architecture.

4) Implementation details: We use a network composed of 5 layers of 64, fully connected, ReLu activated nodes. The low-complexity of the network easily fits the constraints of mobile devices. In the platform we consider, the network can be executed at up to $10Hz$ increasing the power consumption by only $< 1\%$. We adopt a double Deep-Q Learning structure (see Fig. 6) to train the network [28], where the Q-value network learns the relationship between the input state and the output Q-values (one for each action) by using the target network for the $Q(s_{t+1}, u)$ value, and Eq. 1 to combine it with the reward. To balance exploration-exploitation, we use the, effective and stable, ϵ-greedy scheme [26].

V. Experimental Results

We describe our experimental setup and dataset generation process in Section V-A. Then, we discuss our experimental trade-off analysis in Section V-B. Finally, we evaluate and compare `SmartDet` against baselines in Section V-C.

A. Experimental Setup and Data Collection

All the experiments were performed indoor in a campus setting. As mobile device, we use an NVIDIA Jetson Nano, quad-core ARM 1.9GHz CPU and mounting a 128-core GPU operating at 0.95GHz, with performance comparable to current generation mobile phones and small autonomous vehicles [29]. As edge server, we use a ThinkPad P72 with hexa-core CPU operating up to 4.3GHz, 32GB of memory and NVIDIA GPU Quadro P600 that has 384 cores operating at 1.45GHz, and a custom server, mounting 6 core CPU running up to 4.00 GHz, 32 GB of RAM, GPU GTX 980 Ti with 2816 cores at 1.4GHz. We set up the laptop in hotspot mode, using its Wireless-AC 9560 card, to which the mobile device connects using Realtek WiFi dongle supporting IEEE 802.11n.

Model/Server type	Laptop [s]	Server [s]	Avg. Image Size [kB]
D0	0.12	0.089	52.15
D1	0.215	0.11	69.8
D2	0.33	0.16	93.3
D3	0.59	0.255	116.3
D4	1.08	0.4	138.8

TABLE I: Execution time and image size for the various EfficientDet models. Processing units available at Laptop: Quadro P600; and at Server: GTX 980 Ti.

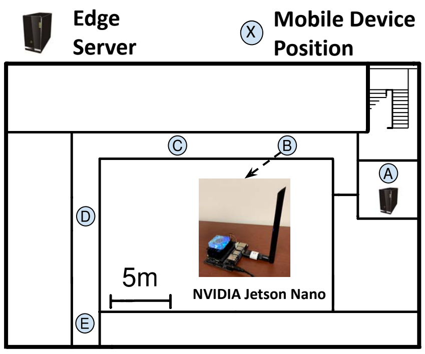


Fig. 7: Representation of the experimental environment. Mobile device: NVidia Jetson Nano; Server: ThinkPad P72 and Server with GTX 980Ti GPU.

We perform our evaluation on ILSVRC2015-VID dataset, which we send over the network in order to collect the network latency in different link quality conditions, and resizing images corresponding to the optimal input of each object detection. Fig. 7 shows the topologies used in the experiments. We load the EfficientDet 0, 2, 4 models at the server, and associate them with control actions. We adopt MedianFlow as the object tracking algorithm to allow the mobile device to execute tracking in real time. In our preliminary evaluation, the DNN-based tracker GOTURN and CSRT (a common alternative) would achieve a frame rate below 2fps and 3fps, respectively.

Table I reports the execution time of the various EfficientDet models on the laptop and server, and their input image size. We can see a progressive increase of execution time as the model's complexity increases. On the Laptop, EfficientDet 0 takes 0.12s, whereas EfficientDet 4 takes almost 10 times as much (1.08s). In the server, besides the lower execution times, we observe a less steep progression, where the execution of EfficientDet 4 takes about 4 times longer than that of EfficientDet 0.

We evaluated the instantaneous power consumption (on the Jetson Nano) of the OT algorithm when the `Katch-Up` is off and on. In the former case, the power consumption is $3512 \pm 242mW$, whereas in the latter increases to

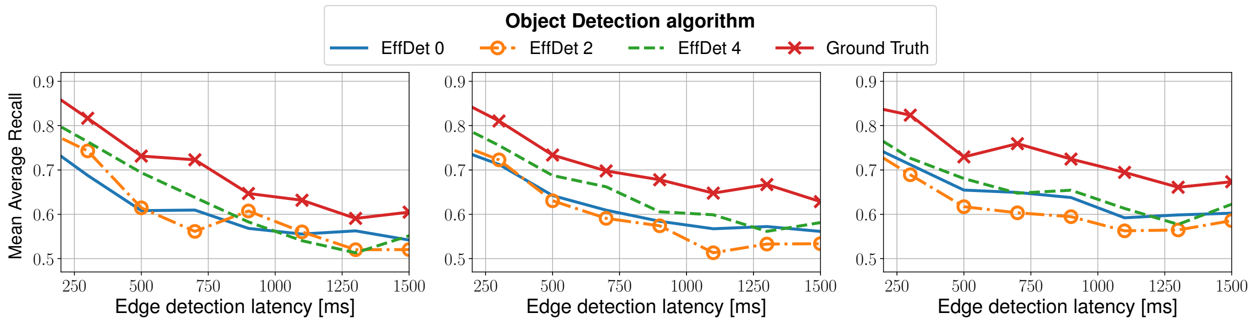


Fig. 8: mAR as a function of OD latency for different periods (0.5, 1, 1.5s) for different OD models. Katch-Up OFF.

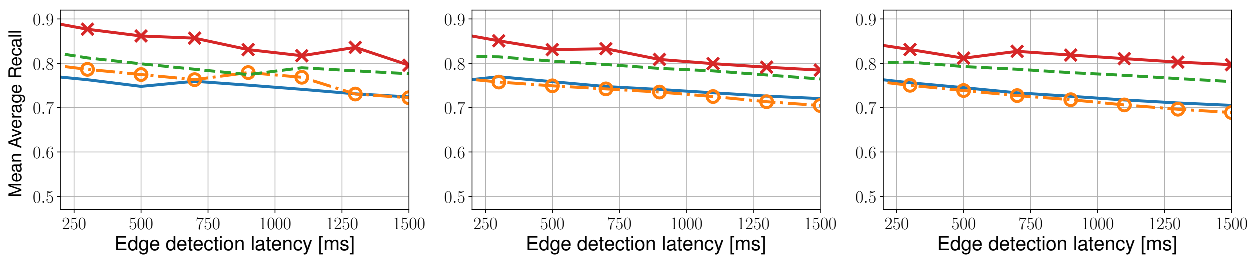
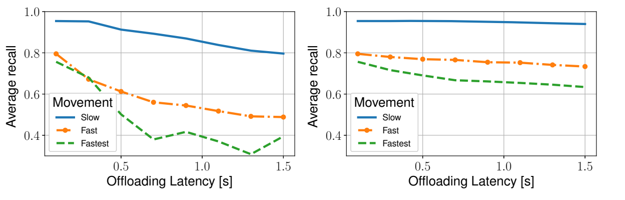


Fig. 9: mAR as a function of OD latency for different periods (0.5, 1, 1.5s) for different OD models. Katch-Up ON.

$3939 \pm 459mW$. Thus, the mAR improvement granted by the Katch-Up technique comes at the price of an increase of power consumption of about 11%. It is then critical to activate the Katch-Up only when necessary. We standardize the three components of the reward function so that all changes in the actions' outcome across the reward metrics are similarly reflected in the feedback signal (as defined in Eq. 2).

B. Tradeoff and Trends Analysis

We first analyze the major trends in the system. The objective is to illustrate the key tradeoffs that will drive the controller actions. Fig. 8 shows the mAR as a function of the OD latency when the Katch-Up is inactive. The different lines correspond to various EfficientDet models (0, 2 and 4) and the ground truth. Here, the latency is abstracted from channel and system parameters. The different plots correspond to different OD periods (0.5, 1 and 1.5 seconds). The degradation of mAR as the latency increases is apparent: in the system without Katch-Up, even when the ground truth is available for a large number of frames (1 every 5), the mAR rapidly goes from 8.5 to 6.5 (23% decrease) as the latency goes from 250ms to 1500ms. A similar decrease is observed when EfficientDet models are used, and for different periods. Note that executing EfficientDet 2, a medium complexity model in the considered set, takes 0.33 and 0.16s on the laptop and server, respectively, so that a very small latency is not expected even in ideal channel conditions. We notice that in general a larger period – that is, fewer frames are sent to the edge server for object detection – results in a less pronounced, performance degradation.

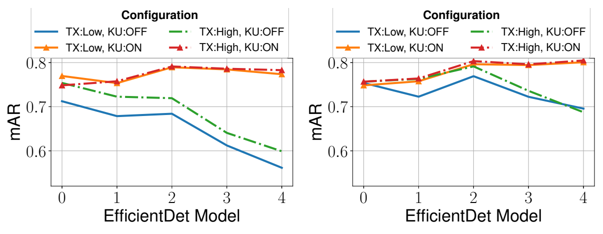


(a) Katch-Up OFF (b) Katch-Up ON

Fig. 10: mAR as a function of object detection latency for different video motion speeds. Period: $1500ms$, EffDet: 2, mAR is normalized to object detection only performance.

Fig. 9 shows the same trends when Katch-Up is ON. Notably, besides increasing mAR in general, the Katch-Up makes the mAR much less sensitive to latency. When the latency is above 0.5 seconds, the superior performance of Katch-Up is apparent. For a latency of 1500ms and period 0.5, the mAR goes from 0.5 to 0.7 when EfficientDet 0 or 2 is used, from 0.5 to 0.77 when EfficientDet 4 is used, and from 0.6 to 0.8 when using the ground truth. Thus, as the latency of object detection increases, the controller can resort to Katch-Up to maintain a high mAR. We note that while the difference between the various models is minimal when the Katch-Up is off, the activation of Katch-Up makes the difference between EfficientDet 0/2 and 4 perceivable, and the controller may leverage this difference when selecting the video analysis configuration.

We now analyze the effect of video parameters on mAR. Fig. 10 shows the mAR as a function of the OD latency for



(a) Laptop (b) Server

Fig. 11: mAR as a function of the model for different WiFi quality (Low=20, High=44), Katch-Up (ON, OFF), and Server (Laptop and Server).

different video change rate, which we measure as the change in position of the targets, when the `Katch-Up` is off (a) and on (b). In the figure, EfficientDet 2 is used, and the period is set to $1500ms$. Here, we classify videos as *slow*, *fast*, *fastest* corresponding to change of position of $< 5\% < 20\%$ or more than 20% of the frame width. The impact of motion parameters on mAR is perceivable. For `Katch-Up` off, at 0.5s latency the three classes have mAR of 0.92, 0.6 and 0.5, respectively. The difference increases with the latency. We can observe how `Katch-Up` increases the performance and makes it less sensitive to latency for all the classes. These results demonstrate how control needs to take into account video parameters to balance energy expense and performance.

We now analyze the trends using real-world hardware and channel capacity configurations. Fig. 11 depicts the mAR as a function of the EfficienDet Model for low and high channel index of the WiFi link connecting the mobile device to the edge server and `Katch-Up` on and off. In (a) and (b), we use the laptop and the server as edge server. We can see how that as the complexity of the model, and thus its execution time, increases, the configuration with `Katch-Up` off suffers a degradation of mAR despite the improved quality of detection. This is due to the sensitivity of the system to latency. Conversely, when `Katch-Up` is on, the performance increases when the transitioning from EfficienDet 0 to 2, and then slightly decreases from 2 to 4. Notably, we see that `Katch-Up` makes the performance less sensitive to channel quality. When the server is used instead of the laptop, the smaller execution time makes the use of larger models more advantageous in both cases. However, we see how very large models (EfficientDet 3 and 4) still suffer a performance loss when the `Katch-Up` is off, while they maximize performance when the `Katch-Up` is on. These results further demonstrate how this selection needs to be based on features of the system in addition to characteristics of the video itself.

C. *`SmartDet` Evaluation*

We now show the performance and policy structure of the DRL agent we designed and trained. In the plots, we set $\alpha = [\alpha_1, \alpha_2, \alpha_3] = [0.1, 0.2, 0.7]$. Fig. 12a depicts mAR, index of channel/edge utilization (where $0, 0.5$ and 1 correspond to period of $15, 10, 5$), `Katch-Up` usage (expressed as percentage of frames to which `Katch-Up` is applied) and normalized model used (EffDet 0, 2, 4 respectively mapped to $0, 0.5, 1$) of `SmartDet` against 2 fixed configurations. The first fixed policy (Policy 1) maximizes resource usage by applying `Katch-Up` on all the frames and setting the period to its minimum value. The fixed second policy (Policy 2) completely deactivates `Katch-Up`, and sets the period to its maximum value, thus minimizing resource usage. Both policies use EfficientDet 4 to maximize OD accuracy. In summary, by learning to dynamically adapt local analysis and offloading parameters to the current state, `SmartDet` achieves optimally controls resource usage while maximizing mAR. With respect to Policy 1, we still increase mAR by 4% while applying `Katch-Up` to only 1/3 of the frames and doubling the period. With respect to Policy 2, we increase mAR by 20%, while increasing `Katch-Up` usage to 1/3 and channel usage by 1/3.

We now analyze the decision making of the agent. In Fig. 12b, we show the mAR performance and channel usage, `Katch-Up` activation and used model for different Wi-Fi channel quality index. First, we observe that mAR increases with channel quality as expected, and is extremely stable from low to high channel quality. Notably, the `SmartDet` agent uses different strategies for different channel qualities, thus demonstrating how the optimal parameter configuration needs to take into account contextual variables. As expected, as the channel improves the agent activates `Katch-Up` less often, due to the decreased OD latency. We note that channel/edge usage and model is non-obvious. Indeed, for low and high channel index the agent selects simpler – and thus faster – EfficientDet models, while sending more frames to the edge server for OD. This behaviour exposes some of the interdependencies between the different metrics. Notice how using higher EfficientDet requires higher `Katch-Up` to be performed (more complex OD has higher transmission and computation delay), and forces the agent, who is rewarded with higher mAR, to offload less frequently.

Fig. 12c shows the metrics as a function of the target movement. Again, we see how this parameter influences the decision made by the agent, and how the agent is capable of making mAR uniform across different classes of videos. When the video has slow target movements, the `SmartDet` agent uses less channel/edge resources, activates `Katch-Up` less often and uses a more complex, and thus slow, EfficientDet model compared to portions of videos where targets move fast. This strategy privileges fast OD reference turnout accepting a reference quality degradation as tracking would not be able to otherwise follow the fast targets. The more frequent `Katch-Up` activation further improves the "freshness" of the reference. When the targets have medium mobility, the `SmartDet` agent uses a different strategy, using more complex EfficientDet models applied to fewer frames while compensating activating more often `Katch-Up`.

VI. Conclusions

This paper focused on edge-assisted real-time OT at mobile devices, where the edge server periodically performs OD to generate references for the tracker. In this context, the key issues

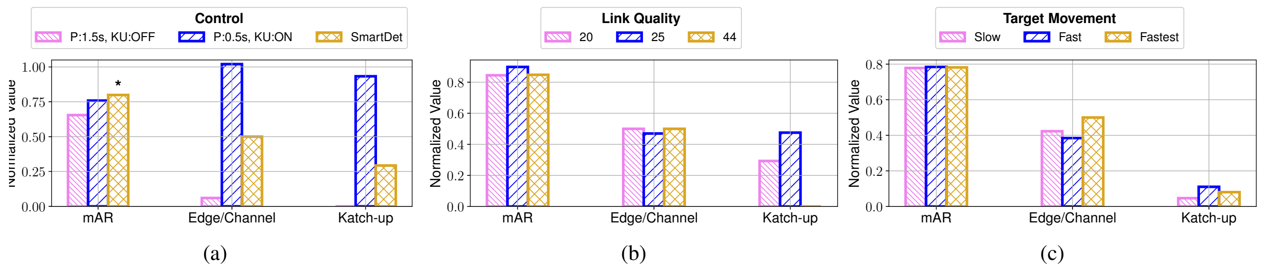


Fig. 12: (a) mAR performance and resource usage of `SmartDet` against fixed strategies. (b) mAR performance and resource usage of `SmartDet` in different link conditions. (c) mAR performance and resource usage of `SmartDet` for video sections with different target mobility.

are (*i*) remote OD may have a large and erratic latency due to channel capacity and server limitations, and (*i*) the system and characteristics of the video are time-varying. To address these issues, we made two main innovations: (*i*) we proposed `Katch-Up`, a tracking strategy that boosts performance while sacrificing computing load and energy at the mobile device; (*ii*) a DRL agent which dynamically controls tracking and offloading parameters to adapt image analysis to time-varying characteristics of the video and system variables. Results on a real-world experimental platform demonstrate the ability of the system to provide optimal tracking performance while parsimoniously using channel and energy resources.

References

[1] L. Liu and M. Gruteser, "EdgeSharing: Edge Assisted Real-time Localization and Object Sharing in Urban Streets," 2021.

[2] X. Zeng, B. Fang, H. Shen, and M. Zhang, "Distream: Scaling Live Video Analytics with Workload-Adaptive Distributed Edge Intelligence," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, SenSys '20, (New York, NY, USA), p. 409–421, Association for Computing Machinery, 2020.

[3] J. Xu, H. Cao, D. Li, K. Huang, C. Qian, L. Shangguan, and Z. Yang, "Edge Assisted Mobile Semantic Visual SLAM," in *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1828–1837, 2020.

[4] e. a. Abdelzaher, "Toward an Internet of Battlefield Things: A Resilience Perspective," *Computer*, vol. 51, no. 11, pp. 24–36, 2018.

[5] J. Lee, P. Wang, R. Xu, V. Dasari, N. Weston, Y. Li, S. Bagchi, and S. Chaterji, "Benchmarking Video Object Detection Systems on Embedded Devices under Resource Contention," in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, EMDL'21, (New York, NY, USA), p. 19–24, Association for Computing Machinery, 2021.

[6] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "Deepiot: Compressing Deep Neural Network Structures for Sensing Systems with a Compressor-Critic Framework," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pp. 1–14, 2017.

[7] Y. Huang, X. Qiao, J. Tang, P. Ren, L. Liu, C. Pu, and J. Chen, "DeepAdapter: A Collaborative Deep Learning Framework for the Mobile Web Using Context-Aware Network Pruning," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 834–843, 2020.

[8] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018.

[9] 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," 2015.

[10] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," 2020.

[11] L. Liu, H. Li, and M. Gruteser, "Edge Assisted Real-Time Object Detection for Mobile Augmented Reality," in *The 25th Annual International Conference on Mobile Computing and Networking*, pp. 1–16, 2019.

[12] B. Blanco-Filgueira, D. García-Lesta, M. Fernández-Sanjurjo, V. M. Brea, and P. López, "Deep Learning-Based Multiple Object Visual Tracking on Embedded System for IoT and Mobile Edge Computing Applications," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5423–5431, 2019.

[13] R. Xu, R. Kumar, P. Wang, P. Bai, G. Meghanath, S. Chaterji, S. Mitra, and S. Bagchi, "ApproxNet: Content and Contention-Aware Video Analytics System for Embedded Clients," 2021.

[14] R. Girdhar, G. Gkioxari, L. Torresani, M. Paluri, and D. Tran, "Detect-and-Track: Efficient Pose Estimation in Videos," in *Proceedings of the IEEE Conf. on Comp. Vision and Pattern Recognition*, pp. 350–359, 2018.

[15] D. Callegaro, M. Levorato, and F. Restuccia, "SeReMAS: Self-Resilient Mobile Autonomous Systems Through Predictive Edge Computing," *Proc. of IEEE International Conference on Sensing, Communication and Networking (SECON)*, 2021.

[16] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint Configuration Adaptation and Bandwidth Allocation for Edge-Based Real-Time Video Analytics," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 257–266, IEEE, 2020.

[17] W. Zhang, Z. He, L. Liu, Z. hua Jia, Y. Liu, M. Gruteser, D. Raychaudhuri, and Y. Zhang, "Elf: Accelerate High-resolution Mobile Deep Vision with Content-Aware Parallel Offloading," 2021.

[18] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, "Distilled Split Deep Neural Networks for Edge-Assisted Real-Time Systems," in *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pp. 21–26, 2019.

[19] 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*, SenSys '20, (New York, NY, USA), p. 449–462, Association for Computing Machinery, 2020.

[20] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "CenterNet: Keypoint Triplets for Object Detection," 2019.

[21] J.-S. Lim, M. Astrid, H.-J. Yoon, and S.-I. Lee, "Small Object Detection using Context and Attention," 2019.

[22] A. Lukezic, T. Vojir, L. ˇCehovin Zajc, J. Matas, and M. Kristan, "Discriminative Correlation Filter with Channel and Spatial Reliability," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6309–6318, 2017.

[23] D. Held, S. Thrun, and S. Savarese, "Learning to Track at 100 FPS with Deep Regression Networks," 2016.

[24] Z. Kalal, K. Mikolajczyk, and J. Matas, "Forward-Backward Error: Automatic Detection of Tracking Failures," in *2010 20th international conference on pattern recognition*, pp. 2756–2759, IEEE, 2010.

[25] G. Ciaparrone, F. Luque Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, and F. Herrera, "Deep Learning in Video Multi-Object Tracking: A Survey," *Neurocomputing*, vol. 381, pp. 61–88, 2020.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[27] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[28] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[29] Skydio, "Skydio 2 autonomous drone - skydio inc.."