Per-Object IoU Forecasting for Deadline-Aware Real-Time Embedded Detection Control

Erfan Foorginejad, Akshar Chavan, and Marco Brocanelli

Dept. of ECE, The Ohio State University, Columbus, Ohio, USA
{foorginejad.1, chavan.43, brocanelli.1}@osu.edu

Abstract. Real-time object detection on edge platforms is constrained by inference latency due to limited resources. Existing runtime controllers adapt detector configurations using coarse frame-level feedback, but fail to account for object-level accuracy degradation caused by detection temporal mismatch. This leads to suboptimal scheduling decisions under dynamic workloads. We propose a lightweight per-object IoU decay model that predicts accuracy loss and derives closed-form deadline estimates for each detected object. These deadline predictions provide fine-grained, deadline-aware feedback that can be integrated into runtime control policies. We integrate the model into a new closed-loop detector selection framework and evaluate it under streaming conditions on Argoverse-HD and MOT17 across heterogeneous hardware platforms. Experiments on real hardware testbeds show that our model and control algorithm improve deadline prediction accuracy by 56% and streaming AP by 39% on average across 71 video scenarios, while incurring acceptable overhead. The code is available at `https://github.com/3Eerfan/OLAP-ECCV2026`.

Keywords: Streaming Perception · Real-Time Object Detection

1 Introduction

Reliable streaming object detection is essential for perception-driven systems such as autonomous vehicles, robotics, augmented reality, and resource-constrained edge devices. While modern detectors achieve high accuracy in offline evaluation, Li et al. [17] introduced the problem of *streaming perception*, showing that real-time performance can degrade significantly due to temporal mismatch. This degradation is more pronounced in resource-constrained mobile and edge devices. When detections are delayed, the resulting bounding boxes correspond to earlier frames, creating a temporal misalignment between the estimated and true object locations. Figure 1 illustrates this phenomenon. Even when offline detection results (red) are accurate, real-time bounding boxes (blue) may

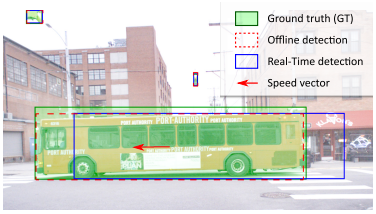


Fig. 1: Example of real-time (i.e., streaming) compared to offline object detection and ground-truth.

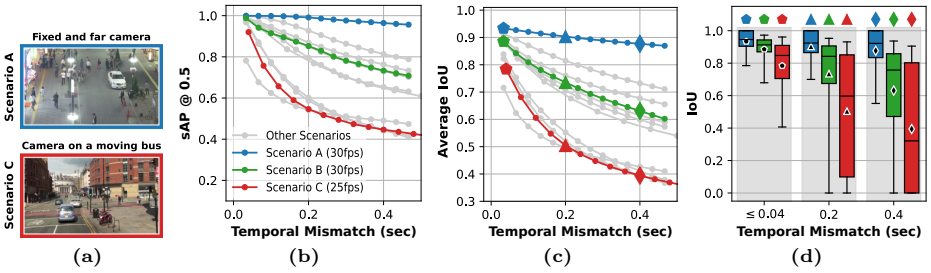


Fig. 2: Effect of temporal mismatch on accuracy metrics. (a) A representative frame of scenarios A and C, (b) sAP and (c) Average IoU versus temporal mismatch for three representative scenarios (A–C). (d) Per-object IoU distributions for three temporal mismatch values, showing large within-scenario variation that average metrics hide.

lag behind the moving objects. This degradation is minimal for static objects but substantial for fast-moving ones, leading to rapid decay in Intersection-over-Union (IoU) and unreliable perception. Thus, high offline accuracy alone does not guarantee reliable streaming perception under strict resource constraints.

To reason about latency-induced degradation, we adopt the notion of *temporal mismatch*, introduced by [17]. Temporal mismatch measures the elapsed time between the frame used for inference and the frame where the resulting detection output is applied. As temporal mismatch increases, bounding boxes become increasingly outdated. Existing works mitigate this issue through: (i) faster detector architectures [12, 32], (ii) tracking or forecasting methods that update boxes between detections [1, 17], and (iii) runtime control schemes that adapt model complexity or scheduling to balance accuracy and latency [13, 15, 27]. Control-based methods are particularly promising because, while they benefit from the first two approaches, they also explicitly manage the accuracy-latency tradeoff. However, current systems typically rely on frame-level averages, such as predicted AP [15] or average IoU [13], which obscure substantial per-object heterogeneity.

To illustrate this point, Figure 2 shows the effect of temporal mismatch across scenarios from MOT17 [24] and Argoverse-HD [2,17], using visual tracking to propagate detections across skipped frames. Even when frame-level streaming AP (sAP, following the terminology of [17]) and IoU remain relatively high (Figures 2b and 2c), individual objects may suffer severe accuracy drops. For instance, at a temporal mismatch of 0.2 seconds, Scenario B exhibits a 0.73 average IoU, yet some objects fall to 0 IoU (e.g., the green-triangle case in Figures 2c and 2d). In safety-critical contexts, understanding object-level accuracy dynamics is essential. For example, maintaining high IoU for a pedestrian crossing in front of a vehicle is far more critical than for distant or parked vehicles. This motivates lightweight *per-object* accuracy estimation models suitable for integration into *priority-aware* feedback control systems. This object-level perspective is also consistent with Tian et al. [30], who represent the surrounding scene as object-level tokens for large language model reasoning.

Building on the findings of Li et al. [17], our study shows that streaming accuracy should be proactively controlled at the object level rather than at the frame level. We model temporal IoU decay using object-specific properties to predict per-object deadlines with a lightweight model, and present a control algorithm capable of prioritizing the accuracy of critical objects based on user-defined weights. Specifically, our main contributions are:

- **Object-Level Accuracy Prediction (OLAP):** We present a novel lightweight model that predicts per-object temporal IoU decay and derives closed-form deadline estimates to enable deadline-aware detection control.
- **Closed-Loop Integration:** We incorporate our OLAP model into a proof-of-concept runtime controller (OLAP-C) that dynamically selects among multiple detection models to optimize real-time accuracy.
- **Experimental Results:** Across MOT17 and Argoverse-HD scenarios on three hardware platforms, our OLAP-based control system consistently outperforms state-of-the-art baselines [13,15], improving deadline prediction accuracy by 56% and streaming AP (sAP) by 39% on average across 71 video scenarios, particularly on the most resource-constrained device.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 presents our approach and proof-of-concept framework. Section 4 reports experimental results, and Section 5 concludes the paper.

2 Related Work

Existing methods for improving real-time detection accuracy can be categorized into three groups:

Increasing Throughput: These methods focus on reducing the inference time of detection models while maintaining offline accuracy [10, 21, 23, 29, 32, 34]. For example, Jocher et al. [11] proposed YOLO11n, which achieves a 4.8× speedup over EfficientDet-D1 [29] while maintaining similar offline detection accuracy of 0.4 AP on the COCO dataset. These methods are mainly effective when the available computing resources allow the inference throughput to match the camera frame rate, i.e., when the temporal mismatch is zero as in the offline setting. Other methods focus on throughput maximization by adjusting the model complexity based on the available resources [5,14,22,25], compromising the detection accuracy to maintain a high throughput.

Compensating for Temporal Mismatch: These methods aim to compensate for the detection accuracy decay caused by temporal mismatch. Some solutions [17] employ a lightweight detection-free tracker to update detected bounding boxes until the next detection output is available. Other solutions [6, 16] train a custom architecture on real-time detections to adjust the box location at the output frame based on the temporal mismatch. These strategies provide meaningful improvements in real-time accuracy and can be directly leveraged within practical systems. However, their effectiveness generally deteriorates at larger temporal mismatches [20], as also highlighted in Figure 2. Moreover, they

operate passively: once deployed, they do not adapt based on hardware capacity, runtime load, or scene complexity. As a result, they lack an explicit mechanism to manage accuracy/latency tradeoffs under dynamic conditions.

Context-Aware Control Mechanisms: These methods employ closed-loop control algorithms to regulate real-time detection accuracy. FlexPatch [33], for instance, identifies regions with potentially degraded tracking accuracy and selectively triggers detection on those patches. However, this strategy is largely reactive. To enable proactive feedback control, runtime decision-making requires an accuracy prediction model. Sela et al. [27] propose a tree-based predictive model that estimates detection accuracy from runtime features and exposes multiple configurable parameters. While their data-driven model offers high precision, its high memory usage (13GB) and overhead place it outside the scope of our work targeting constrained devices. To reduce overhead, Lee et al. [15] introduce a lightweight frame-level AP prediction model to guide detector switching for AP maximization. Similarly, Kong et al. [13] develop a lightweight frame-level IoU model to coordinate edge offloading decisions. Despite being efficient, these approaches rely on coarse frame-level accuracy estimates and do not account for large variations across objects within the same frame (see Figure 2).

To address these limitations, we propose a lightweight per-object IoU decay model that provides closed-form deadline estimates. This model is complementary to high-level controllers like Chanakya [7], e.g., in complex systems like UniAD [9], the analytical deadline estimates produced by OLAP can serve as high-fidelity input features for [7]. We then design a proof-of-concept closed-loop controller that incorporates these estimates as runtime feedback to improve priority-aware real-time accuracy under resource constraints.

3 Proposed Methodology

Building on the streaming detection timeline presented in [17], we revisit the sequence of events using the example shown in Figure 3 to help define our notation in the rest of the paper. In this scenario, the camera captures frames at a constant rate, producing a sequence of frames $\{F_i\}_{i\geq 1}$, while the detection algorithm processes the most recently captured frame as soon as computing resources become available. Since the detector's processing throughput is lower than the camera's frame rate, some frames are ignored during detection; we refer to these as *dropped frames*. For dropped frames, the system can (i) use a zero-order hold [17], i.e., simply reuse the most recent detection output without tracking, or (ii) use a lightweight tracking algorithm to adapt the bounding box location until a new detection result is generated [13,17]. For simplicity, Figure 3 illustrates case (i) using a single bounding box. This is solely for explanatory purposes. In practice, each detection inference D_k outputs bounding boxes for multiple objects, and the proposed method handles this by applying the same procedure independently to each object. In addition, the method is agnostic to whether bounding boxes on dropped frames are reused directly or updated through tracking, and therefore supports either approach without modification.

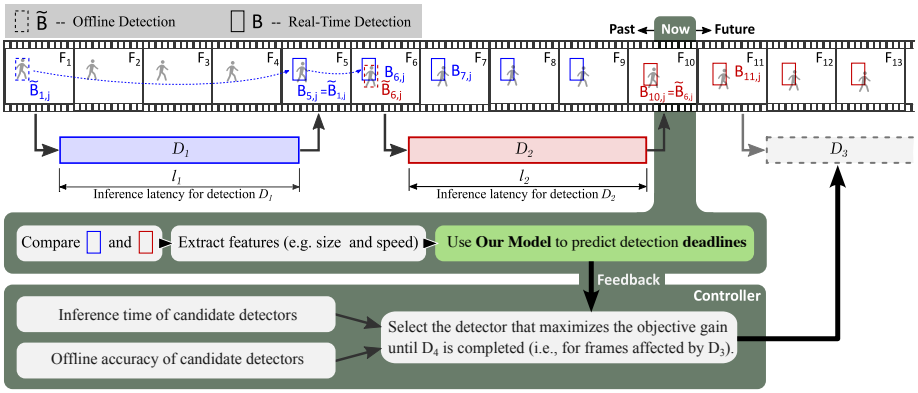


Fig. 3: Timeline of streaming detection, and the proposed closed-loop control.

Figure 3 shows that, while the detection D_2 is busy processing frame F_6, the most recent detection output (blue bounding box) from detection D_1 is being reused for the **dropped frames** $\{F_6, F_7, F_8, F_9\}$. As a result, at frame F_9, the *temporal mismatch* is $a_9 = 4$ frames, corresponding to the number of frames elapsed since the last processed frame. The updated bounding box (red) becomes available at frame F_{10}. At this point, we seek to predict how the accuracy of this output evolves over future frames as temporal mismatch increases. Based on this prediction, the control mechanism must ensure that a new detection output D_3 is available before the red bounding box becomes unreliable, meaning before the IoU between the streaming detection and the ground truth drops below the user-defined threshold. Conversely, if the control mechanism over-provisions computational resources or selects a faster detector with lower accuracy, it may trigger new detection outputs earlier than necessary, resulting in avoidable accuracy degradation or wasted resources.

Defining *deadline* as the maximum temporal mismatch at which the IoU between the streaming detection and the ground truth remains above the user-defined threshold, this situation raises a critical question: **how can we estimate a per-object deadline at runtime, before the next detection cycle begins?** Such an algorithm must be lightweight and generalizable across various scenarios. Furthermore, it must predict the deadline for each object in the scene, enabling the control mechanism to make informed decisions about system-level control knobs such as model selection or resource allocation. Next, we formalize the problem and present the proposed methodology.

3.1 Runtime IoU & Deadline Prediction Models

Let the current frame index i denote the moment immediately following the k^{th} detection cycle, at which the control mechanism must determine its next action (e.g., $i = 10$ in the example presented in Figure 3). Thus, every real-time detection D_k begins at frame F_{i-l_k} and completes at frame F_i, where $l_k \in \mathbb{N}$ is

the inference time of D_k in number of frames. Let s be the prediction offset (in number of frames). Then the temporal mismatch a at frame F_{i+s} is $a_{i+s} = s + l_k$, which is defined as the number of frames elapsed since the last processed frame. In the example, at frame F_{13} (three frames from "Now" in Figure 3), the temporal mismatch with prediction offset $s = 3$ is $a_{13} = 3 + 4 = 7$, since the last processed frame is F_6. Similarly, at the source frame F_6 in the example, the bounding box $B_{6,j}$ is used as the real-time detection and has accumulated a temporal mismatch of $a_6 = 5$ frames. Using this notation, we now describe the runtime bounding box features that enable reliable per-object accuracy prediction.

Feature Extraction. Referring to frame F_6, we denote $\tilde{B}_{6,j}$ as the *offline bounding box*, i.e., the bounding box that would be produced by running the same detector on frame F_6 in an offline setting (equivalently, when the temporal mismatch is zero). Note that $\tilde{B}_{6,j}$ is not available at frame F_6 in the real-time timeline due to inference latency. Instead, it becomes available only after the detector finishes processing, i.e., after l_2 time units in this example. In this case, the real-time detection D_2 completes at frame F_{10}, producing bounding box $B_{10,j}$, which corresponds to the offline bounding box $\tilde{B}_{6,j}$ for frame F_6. Thus, at frame F_{10}, we can compare $B_{6,j}$ (the object's estimated bounding box at F_6) with $B_{10,j}$ (its zero-mismatch/offline counterpart for F_6) to extract several features using the function $\mathbf{f}(\cdot)$. This function $\mathbf{f}(\cdot)$ takes as input the bounding box used for object j at frame F_{i-l_k} (e.g., $B_{6,j}$), its corresponding offline bounding box available at frame F_i (e.g., $B_{10,j}$), and the temporal mismatch l_{k-1} experienced by the real-time bounding box $B_{i-l_k,j}$ (e.g., l_1 in Figure 3). Each bounding box is defined by four parameters. Specifically, $B_{i-l_k,j} = [x_c, y_c, w, h]$ and $B_{i,j} = [x'_c, y'_c, w', h']$, where (x_c, y_c) and (x'_c, y'_c) denote the centers of the two boxes in image coordinates, while (w, h) and (w', h') denote their widths and heights in pixels, respectively. The output of $\mathbf{f}(\cdot)$ is a vector of ten features defined as follows:

$$\mathbf{f}(B_{i,j},\ B_{i-l_k,j},\ l_{k-1}) = [f_1, \ldots, f_{10}] \in \mathbb{R}^{10},$$

$$f_1 = \frac{w}{h}, \qquad f_6 = \frac{v_x}{w},$$

$$f_2 = \sqrt{wh}, \qquad f_7 = \frac{v_y}{h},$$

$$f_3 = \sqrt{(x_c - x_0)^2 + (y_c - y_0)^2}, \qquad f_8 = \sqrt{v_x^2 + v_y^2},$$

$$f_4 = \text{atan2}(y_c - y_0,\ x_c - x_0), \qquad f_9 = \text{atan2}(v_y,\ v_x),$$

$$f_5 = \text{GIoU}(B_{i,j}, B_{i-l_k,j}), \qquad f_{10} = l_{k-1},$$

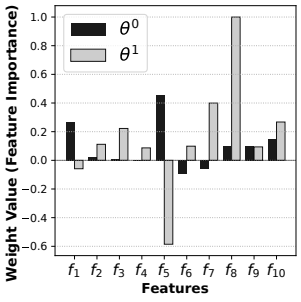


Fig. 4: Feature importance

where $v_x = (x_c - x'_c)/l_{k-1}$ and $v_y = (y_c - y'_c)/l_{k-1}$ are the horizontal and vertical velocities of the object, respectively. Following the feature ordering, f_1 and f_2

capture object geometry (aspect ratio and size), while f_3 and f_4 encode object location in polar coordinates relative to the center of the image (x_0, y_0). Feature f_5 measures the real-time spatial dislocation using Generalized-IoU (GIoU [26]), which is preferred over IoU because it captures distance even when boxes do not overlap. Features f_6 and f_7 represent normalized velocities along the horizontal and vertical axes, and f_8 and f_9 describe the velocity magnitude and direction. Finally, f_{10} corresponds to the inference time of the previous detection D_{k-1}. Figure 4 shows the impact of each feature on the model parameters θ^0 and θ^1, which will be introduced in the next section. As we can see, while speed (f_8) and previous GIoU (f_5) have the largest impacts on the model, all features contribute to achieving high deadline prediction accuracy, which will be analyzed in detail in Section 4.2.

OLAP Model. Given the features for each object, the goal is to predict the real-time IoU for bounding box j at frame F_{i+s} for $s \in \mathcal{B}$, where $\mathcal{B}$ is the set of prediction offsets required for control decisions. For a general index s, we seek a prediction model $\widehat{G}_{i+s,j}$ such that:

$$\text{IoU}^{RT}_{i+s,j}(\tilde{B}_{i+s,j}, B_{i+s,j}) = \widehat{G}_{i+s,j}\left(\mathbf{f}(B_{i,j}, B_{i-l_k,j}, l_{k-1}), a_{i+s}\right) + \epsilon_{i+s,j}, \quad (1)$$

where $\tilde{B}_{i,j}$ is the offline bounding box for object j at frame F_i, $B_{i,j}$ is the real-time estimated bounding box for object j at frame F_i, $\mathbf{f}(\cdot) : \mathbb{N}^9 \to \mathbb{R}^{10}$ is the function that extracts features from the bounding boxes, $\epsilon_{i+s,j}$ is the prediction error, and a_{i+s} is the temporal mismatch that includes the prediction offset s. In the remainder of this section, we propose a model to approximate $\text{IoU}^{RT}_{i+s,j}$ by minimizing $\epsilon_{i+s,j}$. Then we show how to use it to predict the deadline for a desired IoU threshold.

Our observations indicate that the IoU decays exponentially as the temporal mismatch increases (Figure 2(c)). Therefore, we propose a Generalized Linear Model (GLM) that takes the feature vector $\mathbf{f}(\cdot)$ as input and predicts the intercept β and decay factor γ of the following exponential function, which models the IoU for upcoming frames F_{i+s}:

$$\text{IoU}^{RT}_{i+s,j} = \widehat{G}^{exp}_{i+s,j} + \epsilon_{i+s,j} = \beta \cdot e^{-\gamma \cdot a_{i+s}} + \epsilon_{i+s,j}, \quad (2)$$

where β and γ are obtained through GLM link functions [4] defined as follows:

$$\begin{bmatrix}\beta \\ \gamma\end{bmatrix} = \begin{bmatrix}\text{Sigmoid}(\mathbf{f}^\top(B_{i,j}, B_{i-l_k,j}, l_{k-1})\,\boldsymbol{\theta}^0 + b^0) \\ \text{Softplus}(\mathbf{f}^\top(B_{i,j}, B_{i-l_k,j}, l_{k-1})\,\boldsymbol{\theta}^1 + b^1)\end{bmatrix}, \quad (3)$$

where $\boldsymbol{\theta}^0, \boldsymbol{\theta}^1 \in \mathbb{R}^{10}$ and $b^0, b^1 \in \mathbb{R}$ are the model parameters to be learned during training. Here, the link functions Sigmoid$(\cdot)$ and Softplus$(\cdot)$ ensure that the predictions remain within the valid ranges. Equation (2) provides per-object IoU prediction for a given time offset a_{i+s} and is suitable for control algorithms requiring explicit models as accuracy feedback [19].

Deadline Prediction Model. By inverting Equation (2), we can now predict the deadline for a desired IoU threshold τ as follows:

$$\widehat{C}_{i,j} = \widehat{G}_j^{\log}(\tau) = \left\lfloor \frac{\ln(\beta/\tau)}{\gamma} \right\rfloor, \tag{4}$$

where $\widehat{C}_{i,j}$ is the predicted deadline, expressed in number of frames, for object j in frame F_i. The proposed prediction models provide per-object estimates of real-time IoU decay and the corresponding detection deadlines, which can be integrated in real-time feedback control algorithms. We provide the necessary details about model training in Section 4.1.

3.2 OLAP-C: Control Algorithm

Algorithm 1 presents a closed-loop controller that integrates the proposed deadline prediction model in Equation 4 to select the detector for the next detection cycle (Appendix A provides an illustration of the algorithm's operation). Algorithm 1 *serves as a proof of concept. The proposed OLAP model can support a wide range of system-level decisions beyond model switching, including resource allocation, task scheduling* (e.g., [28]), and Dynamic Voltage and Frequency Scaling (DVFS). For example, DVFS can be added as a hierarchical outer loop to select more energy-efficient frequencies that maintain feasible options for OLAP-C.

At the end of detection cycle k, the controller considers a set of available detectors $\mathcal{D} = \{d_1, \ldots, d_M\}$. Each detector $d_m \in \mathcal{D}$ is characterized by a detectability profile $P_m \in \mathcal{P}$ (e.g., the probability of detecting objects of a given size) and an estimated inference time $\hat{l}_m \in \mathcal{I}$. Specifically, inspired by prior work [15], $P_m = [p_m^S, p_m^M, p_m^L]$, where p_m^S, p_m^M, and p_m^L denote the probabilities that model d_m detect small, medium, and large objects, respectively. To guide the next detector selection, the controller uses the runtime state R_k, defined as the tuple $\langle d_k, l_k, n^S, n^M, n^L \rangle$, where d_k is the detector used in cycle k, l_k is the measured inference time (in number of frames), and n^S, n^M, and n^L denote the number of small, medium, and large objects detected, respectively. The controller also uses the pretrained model parameters $\boldsymbol{\theta} = \langle \boldsymbol{\theta}^0, \boldsymbol{\theta}^1, b^0, b^1 \rangle$, and the bounding boxes from the previous and current detection cycles, denoted by $\mathbf{B}_{i-l_k}$ and $\mathbf{B}_i$, respectively, where i is the frame index at which the current detection result becomes available. In addition, the controller takes as input the user-defined IoU threshold τ and object priority weights $\boldsymbol{\omega}$ to predict per-object deadlines. The weights $\boldsymbol{\omega}$ encode relative object-level priority and remain user-defined because object criticality depends on the scenario and application.

The algorithm first updates the estimated inference times $\mathcal{I}^{\text{new}}$ of all candidate detectors using the measured inference time l_k of the current detector d_k. The function UPDATE$(\cdot)$ proportionally rescales the offline estimated inference times to reflect the observed runtime conditions (Line 1). This calibration ensures that the predicted drop window evaluations align with the observed system latency. The algorithm then applies a MATCH$(\cdot)$ function to match objects from the previous detection cycle $\mathbf{B}_{i-l_k}$ with those in the current cycle $\mathbf{B}_i$ using the

Algorithm 1 OLAP-based Real-Time Accuracy Control: OLAP-C

Input: $\mathcal{D}$, $\mathcal{P}$, $\mathcal{I}$: set of detectors and their properties
R_k: Parameters of detection cycle k, $\boldsymbol{\theta}$: Pretrained parameters $< \boldsymbol{\theta}^0, \boldsymbol{\theta}^1, b^0, b^1 >$
$\mathbf{B}_{i-l_k}$: Bounding box outputs from D_{k-1}, $\mathbf{B}_i$: Bounding box outputs from D_k
τ: User-defined IoU threshold, $\boldsymbol{\omega}$: User-defined object importance weights

1: $\mathcal{I}^{\text{new}} \leftarrow \textsc{Update}(\mathcal{I}, d_k, l_k)$
2: $\mathcal{M}_{k-1,k} \leftarrow \textsc{Match}(\mathbf{B}_{i-l_k}, \mathbf{B}_i)$
3: $\Gamma \leftarrow \emptyset$
4: **for** $(p, q) \in \mathcal{M}_{k-1,k}$ **do**
5: $\quad \mathbf{x} \leftarrow \mathrm{f}(B_{i-l_k,p}, B_{i,q}, l_k)$
6: $\quad \beta \leftarrow \mathrm{Sigmoid}(\boldsymbol{\theta}^0 \mathbf{x} + b^0)$
7: $\quad \gamma \leftarrow \mathrm{Softplus}(\boldsymbol{\theta}^1 \mathbf{x} + b^1)$
8: $\quad C_q \leftarrow \lfloor \ln(\beta/\tau) \,/\, \gamma \rfloor$
9: $\quad \Gamma \leftarrow \Gamma \cup \{C_q\}$
10: $\bar{l} \leftarrow \left\lceil \frac{1}{|\mathcal{D}|} \sum_{m=1}^{|\mathcal{D}|} l_m \right\rceil$
11: **for** $m \leftarrow 1$ **to** $|\mathcal{D}|$ **do**
12: $\quad$ **for** $a \leftarrow l_k$ **to** $l_k + \hat{l}_m$ **do**
13: $\quad\quad \mathrm{TP}_{m,a} \leftarrow 0$
14: $\quad\quad$ **for** $C_q \in \Gamma$ **do**
15: $\quad\quad\quad$ **if** $a < C_q$ **then**
16: $\quad\quad\quad\quad \mathrm{TP}_{m,a} \leftarrow \mathrm{TP}_{m,a} + \omega_q$
17: $\quad\quad \mathrm{Pr}_{m,a} \leftarrow \mathrm{TP}_{m,a} / \sum_{q=1}^{|\Gamma|} \omega_q$
18: $\quad \mathrm{AP}_m^{\text{drop}} \leftarrow \frac{1}{\hat{l}_m} \sum_{a=l_k}^{l_k+\hat{l}_m} \mathrm{Pr}_{m,a}$
19: $\quad$ **for** $a \leftarrow \hat{l}_m$ **to** $\hat{l}_m + \bar{l}$ **do**
20: $\quad\quad \overline{\mathrm{TP}}_{m,a} \leftarrow 0$
21: $\quad\quad$ **for** $C_q \in \Gamma$ **do**
22: $\quad\quad\quad$ **if** $a < C_q$ **then**
23: $\quad\quad\quad\quad \overline{\mathrm{TP}}_{m,a} \leftarrow \overline{\mathrm{TP}}_{m,a} + \omega_q$
24: $\quad\quad \overline{\mathrm{Pr}}_{m,a} \leftarrow \overline{\mathrm{TP}}_{m,a} / \sum_{q=1}^{|\Gamma|} \omega_q$
25: $\quad S_m \leftarrow (P_m \oslash P_{m_k})[n^S, n^M, n^L]^\top$
26: $\quad \mathrm{AP}_m^{\text{future}} \leftarrow \left(\frac{1}{\bar{l}} \sum_{a=\hat{l}_m}^{\hat{l}_m+\bar{l}} \overline{\mathrm{Pr}}_{m,a} \right) S_m$
27: $\quad \mathrm{RAP}_m \leftarrow \mathrm{AP}_m^{\text{drop}} + \mathrm{AP}_m^{\text{future}}$
28: $m^* \leftarrow \arg\max_{m \in \{1,\ldots,|\mathcal{D}|\}} \mathrm{RAP}_m$
29: **Return** d_{m^*}

Hungarian assignment algorithm based on the GIoU cost. This function yields a set of matched index pairs $\mathcal{M}_{k-1,k}$, where each pair (p, q) links corresponding objects across the two detection cycles (Line 2).

After initializing the deadline set Γ (Line 3), the algorithm processes each matched object pair $(p, q) \in \mathcal{M}_{k-1,k}$. For each pair, it extracts object-level features $\mathbf{x}$ (Line 5), then applies the pretrained parameters $\boldsymbol{\theta}$ to estimate the IoU decay parameters β and γ (Lines 6–7). Given the target IoU threshold τ, the corresponding deadline C_q is computed in closed form and added to Γ (Lines 8–9).

The average estimated inference time $\bar{l}$ (Line 10) defines a common future evaluation horizon that captures the expected impact of the next detector. The algorithm then computes the Relative Average Precision (RAP) for each detector $d_m \in \mathcal{D}$ (Lines 11–27) as the sum of two components: $\mathrm{AP}^{\text{drop}}$ and $\mathrm{AP}^{\text{future}}$.

Upcoming Drop Window. The *Drop* component, denoted as $\mathrm{AP}_m^{\text{drop}}$, estimates the short-term accuracy loss while detector d_m is executing, i.e., during the frames where no new detection output is available and previously generated bounding boxes are reused and evaluated from temporal mismatch l_k to $l_k + l_m$ (Lines 12–18). During this window, for each temporal mismatch a, objects whose predicted deadline satisfies $a < C_q$ contribute to a weighted true-positive score using the user-defined object importance weight ω_q. The resulting *importance-weighted precision* $Pr_{m,a}$ reduces to standard precision when $\omega_q = 1$ for all objects (Line 17). Averaging $Pr_{m,a}$ over the drop window yields $\mathrm{AP}_m^{\text{drop}}$.

Post-Inference Window. The *Future* component, denoted $\mathrm{AP}_m^{\mathrm{future}}$, estimates the expected detection quality after the output of d_m becomes available. The evaluation spans a future window of length $\bar{l}$, corresponding to the average inference time across candidate detectors (Lines 19–26). For each future temporal mismatch a, objects contribute to the weighted true-positive score when $a < C_q$, and the corresponding object importance-weighted precision is computed as above. Averaging over the future window and incorporating a detector-specific scaling factor S_m (Line 25) to reflect offline detection probabilities yields $\mathrm{AP}_m^{\mathrm{future}}$, which captures the intrinsic detection capability of d_m while preserving object-level importance (Line 26). Finally, the algorithm aggregates the Drop and Future window estimates to compute a single *Relative Average Precision* (RAP) score RAP_m for each detector $d_m \in \mathcal{D}$ (Line 27). The detector that maximizes the predicted real-time performance is then selected (Line 28). The algorithm returns the selected detector d_{m^*} for the next detection cycle D_{k+1} (Line 29).

4 Experimental Evaluation

In this section, we evaluate the effectiveness of our deadline-aware, priority-based closed-loop control framework under realistic real-time streaming conditions. Our goal is to compare (i) the *deadline prediction* accuracy based on the proposed OLAP model against the deadlines predicted with state-of-the-art frame-level baseline models [13, 15], and (ii) the *closed-loop detector selection* performance of OLAP-C against the state-of-the-art baseline controller [15]. We explain the baselines in more detail in each respective section.

4.1 Experimental Setup

Datasets. We evaluate our solutions on two standard benchmarks with diverse scene dynamics: **Argoverse-HD** [2, 17] and **MOT17** [24]. We report results for 71 video scenarios across the two datasets. For each dataset, we split each training video temporally: the first 60% of frames are used for training-data generation and the remaining 40% for testing.

Hardware platforms. We repeat all streaming experiments on three representative edge devices with different compute budgets: (i) **Intel Node** (Intel Core i7-10810U @ 1.10 GHz), (ii) **Apple M3** (MacBook Air), and (iii) NVIDIA **Jetson** Orin Nano.

Streaming protocol and metrics. We stream each video at its native frame rate. While a detector is running, incoming frames are treated as *dropped frames*. For each dropped frame we update the bounding boxes using a lightweight visual tracker, producing a real-time stream of boxes for every frame. During streaming, we save per-frame detections (YOLO format) and compute average precision against dataset ground truth to measure end-to-end streaming detection quality. We refer to streaming AP as sAP, following the terminology introduced by [17]. In addition, to assess example scenarios with objects of heterogeneous criticality, we report the **per-object IoU** to directly capture object-level accuracy.

Visual tracking on dropped frames. We use **MOSSE** [1] as the default tracker for all streaming experiments. For the *deadline prediction* analysis in Figure 5, we additionally test the sensitivity to the **KCF** [8] tracking backend.

Detector options. The controllers select among two YOLO11 detectors [12] (fine-tuned on the training portion of the datasets) and six input resolutions, yielding 12 options, as shown in Table 1. We also include the YOLO11**x** model in the candidate set; however, due to its substantially longer inference time on our devices, the controllers almost never select it during the experiments (see Appendix C for frequency of detector selection on each hardware). Thus, we omit it from the reported results, although it may become practical on faster hardware. On Jetson, we enable half precision (float16) and exclude input size 1920 due to memory constraints. For the detectability matrix $\mathcal{P}$, we define three object-size bins (small, medium, and large) by bounding-box area using a strategy similar to ROMA [15]: object size thresholds are chosen so that each bin contains no less than 20% and no more than 40% of the training objects.

Table 1: Detector Indices d_m

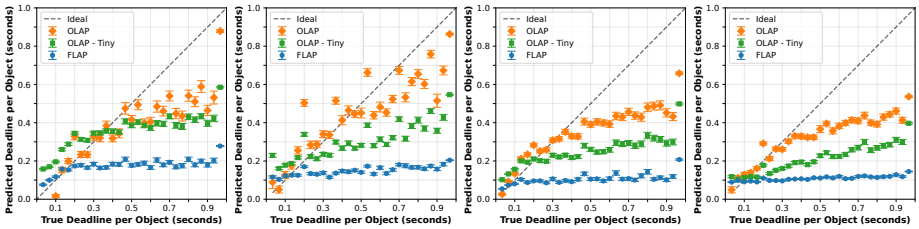
Input Size	yolo11**n**	yolo11**m**
288	0	6
416	1	7
640	2	8
1120	3	9
1440	4	10
1920	5	11

Training $\boldsymbol{\theta}$ for the deadline prediction model. We trained a single OLAP model on the first 60% of all videos in the dataset (≈6 hours including training-data synthesis on a desktop CPU) providing robustness to unseen videos. We train OLAP using ground truth labels, a tracker (see Figure 5), and the induced temporal mismatch to model the object-specific IoU decay by simulating real-time detections using synthetic inference times $l_k = l^{\text{sim}}$ and collecting feature-target pairs for objects in F_{i+s}. Thus, OLAP does not require re-training across detectors, as it relies on geometric object features (e.g., scale, velocity) that capture universal object dynamics independent of detector.

Specifically, for each object j detected at frame F_i, we extract features from $B_{i,j}$ and $B_{i-l^{\text{sim}},j}$ and compute targets $y_{i+s,j}$ for $s \in \mathcal{B}$ (Equation (1)). We set $\mathcal{B} = \{0, 1, 2, \ldots, 29\}$ to cover a wide range of temporal mismatch, and repeat this process for $l^{\text{sim}} \in \{1, 3, 5, \ldots, 19\}$ to span diverse inference-time regimes. To associate the same object j across frames, we use GIoU as the cost matrix in Hungarian assignment [3, 31]. After normalizing features to zero mean and unit variance, we train $\boldsymbol{\theta} = \langle \boldsymbol{\theta}^0, \boldsymbol{\theta}^1, b^0, b^1 \rangle$ (see Equation (2)) by minimizing the regularized MSE: $L = \frac{1}{n} \sum \epsilon_{i,j}^\top \epsilon_{i,j} + \lambda \|\boldsymbol{\theta}\|^2$ with $\lambda = 0.01$.

4.2 Experimental Results

Deadline prediction accuracy and tracker sensitivity. This experiment evaluates the accuracy of the proposed OLAP model in predicting per-object deadlines across different tracking backends and IoU thresholds. We test whether applying a state-of-the-art frame-level prediction strategy at the object level improves deadline estimation, and whether richer object-level features further improve accuracy. As a Frame-Level Accuracy Prediction (**FLAP**) baseline, we



(a) $\tau = 0.5$ | MOSSE **(b)** $\tau = 0.5$ | KCF **(c)** $\tau = 0.75$ | MOSSE **(d)** $\tau = 0.75$ | KCF

Fig. 5: Deadline prediction accuracy on the Argoverse-HD test splits across two IoU thresholds (τ) and two tracking backends (MOSSE vs. KCF).

adapt the runtime accuracy predictor from ARISE [13]. This predictor is conceptually aligned with the ROMA [15] runtime accuracy model, but instead of estimating the AP drop directly, it predicts the average per-frame IoU decay. Because our evaluation requires IoU estimates to derive deadlines, ARISE serves as a representative FLAP baseline. To isolate the effect of granularity, we also implement **OLAP-tiny**, an object-level variant using the same two core features as FLAP (l_{k-1} and $\mathrm{IoU}(B_{i,j}, B_{i-l_k,j})$). The proposed **OLAP** model further extends this formulation with additional object-specific features, including dynamic properties such as object speed.

Figure 5 compares the predicted versus the actual per-object deadlines on the test videos for two IoU thresholds ($\tau \in \{0.5, 0.75\}$) and two trackers (MOSSE and KCF). Across settings, FLAP performs competitively for short deadlines (up to six frames or 0.2 s), where average accuracy remains stable. However, it systematically underestimates longer deadlines, which are common in resource-constrained deployments, reflecting the limitations of frame-level aggregation. Using a similar approach at the object level, OLAP-tiny consistently improves predictions over FLAP, achieving 32.3% average RMSE reduction across all settings (ranging from 24.9% to 38.7% depending on the scenario). These gains are consistent across both IoU thresholds, confirming that modeling per-object variation alone yields substantial and robust improvements over frame-level aggregation. Nevertheless, OLAP-tiny still lags behind OLAP, which leverages all the features described in Section 3.1. Relative to OLAP-tiny, OLAP provides an additional 34.5% average reduction in RMSE, showing that incorporating richer object-dynamics features yields meaningful gains. By modeling these richer dynamics, OLAP maintains accurate predictions over substantially longer deadlines (up to 15 frames or 0.5 s). Beyond this time, our model still improves estimation accuracy over the state-of-the-art FLAP model (by up to 67% and 56% on average), but conservatively estimates deadlines because object motion is more likely to change over longer horizons. Overall, these experiments show that the proposed OLAP model substantially improves object-level deadline prediction.

Closed-loop streaming performance. We next evaluate the end-to-end detector selection under streaming on Argoverse-HD and MOT17 scenarios. We

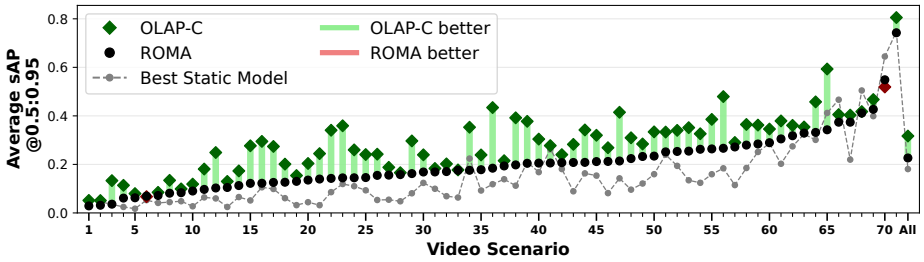


Fig. 6: Comparison of ROMA and our method on average sAP over MOT17 and Argoverse-HD validation video scenarios on the Intel Node.

compare our deadline-aware controller OLAP-C against **ROMA** [15], the state-of-the-art closed-loop model-selection controller for real-time detection. To ensure a fair comparison, because the source code is unavailable, we implement ROMA and integrate the same tracking backend used in our pipeline to update boxes across dropped frames. We also include a **Best Static** baseline that simply uses a single detector throughout each video (with the same tracker). Across scenarios and devices, the best static choice on average is YOLO11m_640 (i.e., d_8 in Table 1). After collecting real-time detections, we compute frame-level sAP@0.5:0.95 (COCO mAP standard [18]) to quantify real-time accuracy.

Figure 6 summarizes the results on the Intel Node across the evaluated scenarios. Relative to Best Static, ROMA improves average sAP by 25%, with gains of up to 0.157 sAP points. Our method improves average sAP by 75% over Best Static, reaching improvements of up to 0.316 sAP points in some scenarios. Compared to ROMA, our method yields an additional 39% on average, with gains of up to 0.25 sAP points. Overall, the gains indicate that explicitly regulating detector choice using object-level deadline feedback leads to a considerably stronger closed-loop performance.

Cross-device comparison and runtime overhead. Figure 7 compares (a) the inference latency of candidate detectors (as indexed in Table 1) on various devices and (b) the achieved sAP, as well as the control algorithm execution time overhead compared to the detection inference latency during streaming across all 71 videos. As shown in Figure 7a, inference times differ substantially across devices. The Intel Node is the slowest platform, with inference times ranging from 49 ms to 3985 ms (average 816 ms). On the Apple M3, inference times range from 21 ms to 562 ms (average 120 ms). The Jetson platform exhibits the tightest latency distribution, with inference times between 54 ms and 161 ms (average 74 ms). The latency spread is most pronounced on the Intel Node, increasing the importance of selecting an appropriate operating point in real time. Figure 7b shows that our method consistently achieves the highest average sAP across all devices. On the Intel Node, OLAP-C achieves 0.287 sAP, representing a 39% improvement over ROMA and an 86.4% improvement over Best Static. On the Apple M3, OLAP-C reaches 0.391 sAP, corresponding to a 24.6% gain over ROMA and a 29.8% gain over Best Static. On Jetson, OLAP-C attains

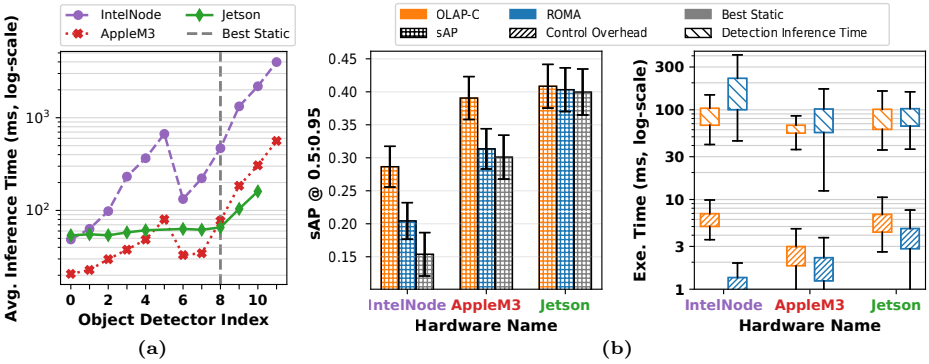


Fig. 7: (a) Inference time of candidate detectors across devices. (b) Average streaming sAP, execution time of real-time detection, and control overhead for Best Static, ROMA, and our OLAP-C method on various edge devices.

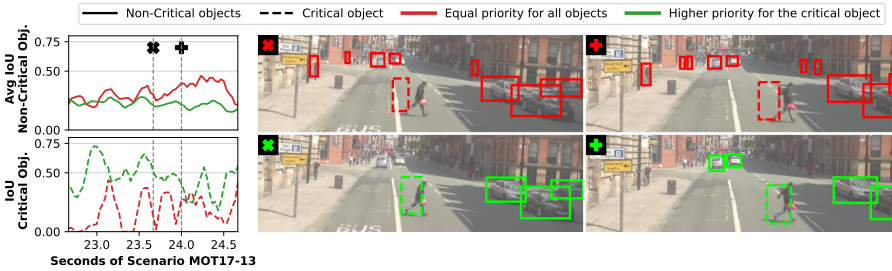


Fig. 8: Qualitative example: per-object priority in real-time detection. The controller can trade average accuracy for more frequent and accurate updates of a critical object.

0.408 sAP, yielding a more modest but consistent 1.3% improvement over ROMA and 2.2% over Best Static. The largest absolute gains appear on the Intel Node, supporting the claim that deadline-aware control is most beneficial under limited compute budgets, while remaining effective on faster hardware. Figure 7b also compares the control algorithms' execution time with detections inference time during streaming (log scale). Our controller incurs slightly higher overhead than ROMA, but remains acceptable for real-time use, remaining below 10% of the average detector inference time.

These experiments show that OLAP-C consistently improves streaming accuracy across heterogeneous hardware, with the largest gains on the most constrained device. The gains are largest on Intel because suboptimal high-latency detector choices increase temporal mismatch and sharply reduce streaming accuracy. On faster devices, the performance gap narrows because detector latencies are more balanced, but OLAP-C still maintains a consistent advantage by adapting to device-specific latency and scene dynamics. More details about control-signal and sAP timeline analysis are provided in Appendix B.2.

Priority-aware control. Finally, we illustrate how the object-level priority affects control decisions. Figure 8 focuses on an interval from the MOT17-13-DPM scenario, where a pedestrian crosses the street while the ego-vehicle (bus) approaches. We label the pedestrian as the *critical object* and treat other objects as *non-critical* because they are farther and pose no immediate collision risk.

We run our controller in two modes: (i) **Equal priority**, assigning uniform weights $\boldsymbol{\omega} = 1$ in Algorithm 1, and (ii) **Critical object priority**, assigning the pedestrian a higher weight $\omega_{\text{critical}} = 10$. With equal priority (red), the controller selects a higher-latency OLAP-C detector to improve the overall accuracy for many objects, which increases average IoU across non-critical objects (red solid line), but delays the important updates for the critical pedestrian, thus reducing its IoU (red dashed line). With critical-object priority (green), the controller selects lower-latency detectors to update the pedestrian more frequently. This improves the pedestrian's IoU (green dashed line) at the cost of reduced accuracy on non-critical objects (green solid line); Appendix C provides the corresponding sAP comparison within each object category. Notably, several objects missed under critical-object priority are distant and less relevant for immediate safety; the key benefit is preserving accurate localization for the near-term risk object. This example shows that object-level, priority-weighted feedback enables scenario-appropriate control decisions not captured by aggregate metrics alone.

5 Conclusion

We presented a lightweight per-object IoU decay model that predicts accuracy degradation and derives closed-form deadline estimates for real-time object detection. By providing object-level, deadline-aware feedback, the proposed approach enables more informed runtime control decisions than conventional frame-level metrics. We further designed a closed-loop, priority-aware detector selection framework that leverages these predictions to guide runtime scheduling. Across heterogeneous hardware platforms, integrating the proposed model into the controller consistently improves sAP over state-of-the-art runtime control and best static baselines, while incurring acceptable overhead. Overall, our results indicate that object-level deadline modeling effectively supports runtime controller design for real-time perception under resource constraints.

Acknowledgements

This work was supported by the U.S. NSF under Grant CNS2421244.

References

1. Bolme, D.S., Beveridge, J.R., Draper, B.A., Lui, Y.M.: Visual object tracking using adaptive correlation filters. In: 2010 IEEE Computer Society conference on Computer Vision and Pattern Recognition (CVPR). pp. 2544–2550. IEEE (2010)

2. Chang, M.F., Lambert, J., Sangkloy, P., Singh, J., Bak, S., Hartnett, A., Wang, D., Carr, P., Lucey, S., Ramanan, D., et al.: Argoverse: 3D tracking and forecasting with rich maps. In: Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR). pp. 8748–8757 (2019)
3. Crouse, D.F.: On implementing 2D rectangular assignment algorithms. IEEE Transactions on Aerospace and Electronic Systems **52**(4), 1679–1696 (2016)
4. Dobson, A.J., Barnett, A.G.: An introduction to generalized linear models. Chapman and Hall/CRC (2018)
5. Fang, W., Wang, L., Ren, P.: Tinier-YOLO: A real-time object detection method for constrained environments. IEEE Access **8**, 1935–1944 (2019)
6. Ge, W., Wang, X., Mao, Z., Ren, J., Shen, J.: StreamTrack: Real-time meta-detector for streaming perception in full-speed domain driving scenarios. Applied Intelligence **54**(23), 12177–12193 (2024)
7. Ghosh, A., Balloli, V., Nambi, A., Singh, A., Ganu, T.: Chanakya: Learning runtime decisions for adaptive real-time perception. Advances in Neural Information Processing Systems (NeurIPS) **36**, 55668–55680 (2023)
8. Henriques, J.F., Caseiro, R., Martins, P., Batista, J.: High-speed tracking with kernelized correlation filters. IEEE Transactions on Pattern Analysis and Machine Intelligence **37**(3), 583–596 (2014)
9. Hu, Y., Yang, J., Chen, L., Li, K., Sima, C., Zhu, X., Chai, S., Du, S., Lin, T., Wang, W., et al.: Planning-oriented autonomous driving. In: Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR). pp. 17853–17862 (2023)
10. Jiang, Z., Zhao, L., Li, S., Jia, Y.: Real-time object detection method based on improved YOLOv4-tiny. Preprint arXiv:2011.04244 (2020)
11. Jocher, G., Qiu, J., Chaurasia, A.: Ultralytics YOLO (2023), version 8.0.0. Available at: `https://github.com/ultralytics/ultralytics`. Accessed: 2026-06-30
12. Khanam, R., Hussain, M.: YOLOv11: An overview of the key architectural enhancements. Preprint arXiv:2410.17725 (2024)
13. Kong, Z.J., Xu, Q., Hu, Y.C.: ARISE: High-capacity ar offloading inference serving via proactive scheduling. In: Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services (MobiSys). pp. 451–464 (2024)
14. Lee, J., Hwang, K.i.: YOLO with adaptive frame control for real-time object detection applications. Multimedia Tools and Applications **81**(25), 36375–36396 (2022)
15. Lee, J., Varghese, B., Vandierendonck, H.: ROMA: Run-time object detection to maximize real-time accuracy. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV). pp. 6405–6414 (2023)
16. Li, B., Huang, Z., Ye, J., Li, Y., Scherer, S., Zhao, H., Fu, C.: PVT++: A simple end-to-end latency-aware visual tracking framework. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 10006–10016 (2023)
17. Li, M., Wang, Y.X., Ramanan, D.: Towards streaming perception. In: European Conference on Computer Vision (ECCV). pp. 473–488. Springer (2020)
18. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common objects in context. In: European Conference on Computer Vision (ECCV). pp. 740–755. Springer (2014)
19. Liu, K.Z., Yao, Y.: Robust control: Theory and applications. John Wiley & Sons (2016)

20. Liu, M., Ding, X., Du, W.: Continuous, real-time object detection on mobile devices without offloading. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS). pp. 976–986. IEEE (2020)
21. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: Single shot multibox detector. In: European Conference on Computer Vision (ECCV). pp. 21–37. Springer (2016)
22. Lou, W., Xun, L., Sabet, A., Bi, J., Hare, J., Merrett, G.V.: Dynamic-OFA: Runtime dnn architecture switching for performance scaling on heterogeneous embedded platforms. In: Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR). pp. 3110–3118 (2021)
23. Lyu, C., Zhang, W., Huang, H., Zhou, Y., Wang, Y., Liu, Y., Zhang, S., Chen, K.: RTMDet: An empirical study of designing real-time object detectors. Preprint arXiv:2212.07784 **7** (2022)
24. Milan, A., Leal-Taixé, L., Reid, I., Roth, S., Schindler, K.: MOT16: A benchmark for multi-object tracking. Preprint arXiv:1603.00831 (2016)
25. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence **39**(6), 1137–1149 (2016)
26. Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., Savarese, S.: Generalized intersection over union: A metric and a loss for bounding box regression. In: Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR). pp. 658–666 (2019)
27. Sela, G.E., Gog, I., Wong, J., Agrawal, K.K., Mo, X., Kalra, S., Schafhalter, P., Leong, E., Wang, X., Balaji, B., et al.: Context-aware streaming perception in dynamic environments. In: European Conference on Computer Vision (ECCV). pp. 621–638. Springer (2022)
28. Subramaniyan, S., Wang, X.: FC-GPU: Feedback control GPU scheduling for real-time embedded systems. ACM Transactions on Embedded Computing Systems **24**(5s), 1–25 (2025)
29. Tan, M., Pang, R., Le, Q.V.: EfficientDet: Scalable and efficient object detection. In: Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR). pp. 10781–10790 (2020)
30. Tian, R., Li, B., Weng, X., Chen, Y., Schmerling, E., Wang, Y., Ivanovic, B., Pavone, M.: Tokenize the world into object-level knowledge to address long-tail events in autonomous driving. Preprint arXiv:2407.00959 (2024)
31. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al.: SciPy 1.0: Fundamental algorithms for scientific computing in python. Nature Methods **17**(3), 261–272 (2020)
32. Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., et al.: YOLOv10: Real-time end-to-end object detection. Advances in Neural Information Processing Systems (NeurIPS) **37**, 107984–108011 (2024)
33. Yang, K., Yi, J., Lee, K., Lee, Y.: FlexPatch: Fast and accurate object detection for on-device high-resolution live video analytics. In: IEEE Conference on Computer Communications (INFOCOM). pp. 1898–1907. IEEE (2022)
34. Zhao, Y., Lv, W., Xu, S., Wei, J., Wang, G., Dang, Q., Liu, Y., Chen, J.: DETRs beat YOLOs on real-time object detection. In: Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR). pp. 16965–16974 (2024)