Adaptive Detection Scheduling for Multi-Object Tracking on Resource-Constrained Edge Devices

Sofia Bouazizi
VidOpt LLC Intern
ITKAN institute
Dallas, TX, USA
Sofia.bouazizi@gmail.com

Imed Bouazizi
VidOpt LLC
IEEE Senior Member
Dallas, TX, USA
ibouazizi@gmail.com

[1]

***Abstract*—Real-time multi-object tracking on low-cost edge devices is limited by the compute cost of deep object detection, which is conventionally run on every frame. A lightweight tracker can propagate object states between detections, but the detection frequency that best balances speed and accuracy, and whether that frequency should adapt to scene content, is not well characterized for constrained hardware. This paper presents a systematic study of detection scheduling for a YOLOv8n detector coupled with a SORT tracker. Sweeping the fixed detection interval N over a pedestrian sequence in a proof-of-concept implementation, we find that throughput scales almost linearly with N, reaching a 16-fold speedup and a 95 percent reduction in detector invocations at N equal to 20, while fidelity to the detect-every-frame baseline degrades with a sharp knee: object coverage stays above 0.90 through N equal to 3 and then collapses to 0.46 by N equal to 5. We further propose a content-adaptive scheduler that triggers detection from run-time motion signals, and we report the honest finding that a global maximum-displacement trigger over-fires in dense scenes and does not outperform a well-chosen fixed interval. We identify per-track and confidence-aware scheduling, on-device energy measurement, and ground-truth evaluation as the path to a deployable policy.**

***Keywords*—Adaptive scheduling, edge computing, energy-efficient computer vision, multi-object tracking, object detection, real-time systems.**

I. INTRODUCTION

Modern computer vision pipelines run object detection on every frame of a video stream. This maximizes per-frame accuracy but makes the deep detector the dominant consumer of computation and energy, which blocks deployment on low-cost embedded platforms that lack a discrete GPU.

Object trackers estimate object positions between detections at a fraction of a deep detector cost, so detection need not run every frame. A fixed detection cadence, however, is rarely optimal. Detecting too often wastes energy on redundant inference, while detecting too rarely lets tracking error accumulate and identities drift. The cadence that best trades speed for accuracy depends on scene dynamics, which vary within and across videos.

Prior work studies detection and tracking largely in isolation, or fixes the keyframe interval as a static hyperparameter. Detect or Track frames per-frame skipping as a baseline and argues it is suboptimal because detection frequency should depend on tracking quality [4]. Few studies, however, characterize the joint tradeoff between detection frequency, tracking fidelity, and throughput on constrained hardware, and fewer make that frequency adapt to scene content.

This paper makes three contributions.

1) A systematic characterization of the detection-interval tradeoff for a YOLOv8n plus SORT pipeline, identifying a usable operating region and a sharp accuracy knee.

2) A first content-adaptive detection scheduler driven by run-time motion signals, with an honest analysis of where it succeeds and where it fails.

3) A reproducible protocol and proof-of-concept implementation, with a clearly defined path to on-device energy measurement and ground-truth evaluation.

II. RELATED WORK

A. Object Detection on Edge Devices

The YOLO family provides a strong speed-accuracy balance for real-time detection. We use YOLOv8n, the smallest anchor-free variant, as a representative lightweight detector suitable for embedded inference [1]. Prior edge studies typically run such detectors on every frame and report throughput, without examining how far detection can be down-sampled before tracking degrades.

B. Joint Detection and Tracking

Tracking-by-detection couples a detector with a lightweight data-association tracker. SORT uses a Kalman filter and Hungarian assignment and runs at very high rates without appearance features [2]. DeepSORT adds a learned appearance metric to reduce identity switches under occlusion [3]. We adopt SORT for its low cost, which matches the edge setting, and accept its weaker identity handling as a known limitation.

[1]

C. Keyframe and Adaptive Scheduling

The idea of detecting sparsely and propagating between detections is established. Detect or Track learns a scheduler network that decides per frame whether to detect or track [4]. Looking Fast and Slow interleaves heavy and lightweight feature extractors and learns an inference policy with reinforcement learning [5]. These methods target server or mobile GPUs and do not report rail-level energy on a GPU-free device, which is the gap we move toward.

III. METHOD

A. Detection-Tracking Pipeline

Each frame is either a detection frame or a coast frame. On a detection frame the pipeline runs YOLOv8n, filters to the target class, and feeds the boxes to SORT, which performs Kalman prediction, IoU-based Hungarian association, and state update. On a coast frame the detector is skipped and SORT advances each track by Kalman prediction only. A measurement layer logs throughput and per-frame track boxes for analysis.

B. Periodic Detection Scheduling (Baseline)

The baseline runs detection every N frames and coasts on the intervening frames. We sweep N over the set 1, 2, 3, 5, 10, 15, and 20, where N equal to 1 is the detect-every-frame reference. Track lifetime is sized to the interval so that a track survives one full coast gap.

C. Adaptive Detection Scheduling

The adaptive scheduler replaces the fixed N with a run-time trigger. Detection fires on frame t when the maximum normalized track displacement since the last detection exceeds a threshold τ, or a staleness cap N_{max} is reached, or no active tracks remain, as in (1). Normalized displacement is the predicted center shift divided by the square root of the track area, which ties the trigger to the motion-versus-size bound that governs association.

$$fire(t) = [\max d_t^{\wedge} > \tau] \vee [s_t \geq N_{max}] \vee [\texttt{no active tracks}] \quad (1)$$

D. Implementation

The pipeline is implemented in Python using the Ultralytics YOLOv8n model and a compact SORT tracker with a NumPy Kalman filter and SciPy Hungarian assignment. The current results are obtained on a general-purpose CPU as a hardware-independent proof of concept. Deployment on a Raspberry Pi with the detector exported to NCNN, and energy measured at the supply rail with an INA219 sensor, is the next step and is not yet reported.

IV. EXPERIMENTAL SETUP

A. Data and Reference

Experiments use a 500-frame pedestrian sequence with multiple people, including crossings and partial occlusions. Because this clip has no tracking ground truth, we use the detect-every-frame run as a reference and measure how far each schedule departs from it. This is a fidelity proxy, not a ground-truth metric, and is the main threat to validity addressed in Section VI.

B. Metrics

We report throughput in frames per second, the number of detector invocations, coverage (the fraction of reference objects matched by the schedule at IoU at least 0.3), and box fidelity (the mean IoU of matched boxes). Coverage and fidelity are hardware-independent. Throughput depends on the platform, so its absolute values are indicative and its scaling shape is the transferable result.

C. Protocol

Each fixed interval and the adaptive scheduler at four thresholds are run once over the full clip under identical settings. The adaptive staleness cap is set to 20 frames.

V. RESULTS

A. Throughput Versus Detection Interval

Throughput scales almost linearly with N, from about 10 FPS at N equal to 1 to about 166 FPS at N equal to 20 on the test platform, a 16-fold speedup, while detector invocations fall from 500 to 25, a 95 percent reduction (Fig. 1). The trend confirms that the detector dominates pipeline cost and that coasting is nearly free.

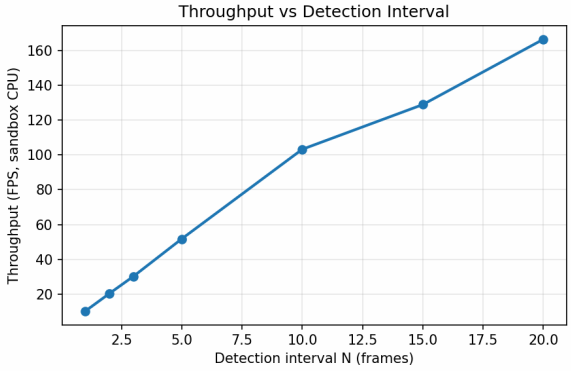

Fig. 1. Throughput versus detection interval. Throughput scales almost linearly with N because the detector dominates cost.

B. Fidelity Versus Detection Interval

Fidelity to the every-frame reference degrades with a clear knee (Fig. 2). Coverage stays high through N equal to 3, at 0.96 and 0.91 for N equal to 2 and 3, then collapses to 0.46 at N equal to 5 and continues to fall. Box fidelity degrades more gradually, from 0.90 at N equal to 2 to 0.30 at N equal to 20. The usable region for this clip is therefore N at most 3.

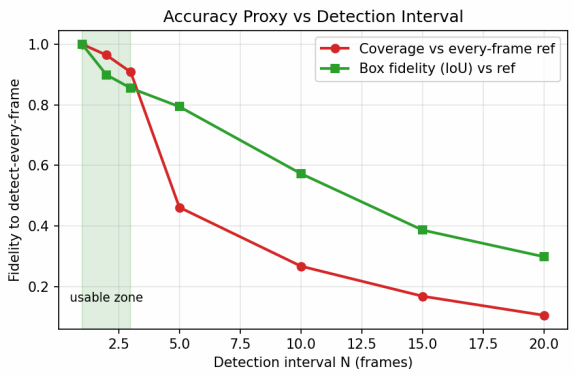

Fig. 2. Coverage and box fidelity versus detection interval. Coverage collapses between N equal to 3 and N equal to 5.

C. Speed-Accuracy Tradeoff

Plotting coverage against throughput (Fig. 3) shows the operating curve. The knee near N equal to 3 gives roughly a 3-fold speedup and a two-thirds reduction in detector calls while retaining above 0.90 coverage, which is the practical sweet spot for a fixed schedule on this sequence.

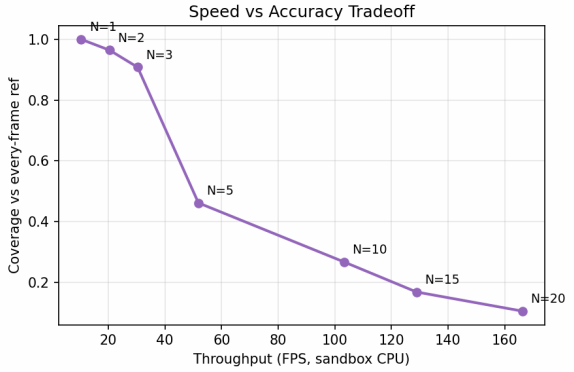

Fig. 3. Speed versus accuracy. The knee near N equal to 3 is the practical fixed-schedule operating point.

D. Adaptive Versus Fixed

The adaptive scheduler does not outperform the fixed schedule on this clip (Fig. 4). Across thresholds it issues 329 to 427 detector calls for coverage between 0.87 and 0.91, whereas fixed N equal to 3 reaches 0.91 coverage in only 167 calls. The trigger over-fires: because it keys on the maximum displacement over all tracks, a single fast-moving pedestrian in a crowd forces detection nearly every frame, giving an effective interval of only 1.2 to 1.5. This is a negative result for the naive design, and it points directly to the fix.

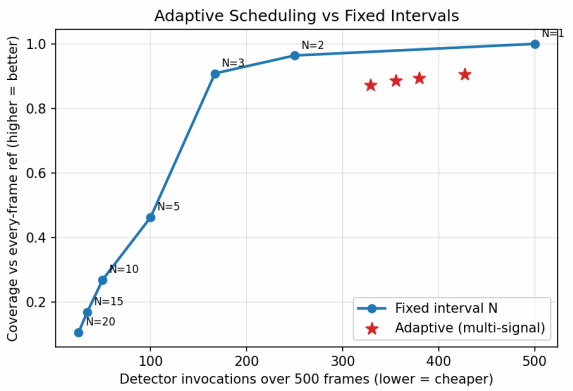

Fig. 4. Adaptive scheduler versus the fixed-interval frontier. The global maximum-displacement trigger over-fires and is dominated by fixed N equal to 3.

TABLE I. Summary of All Configurations

Config	FPS	Det. calls	Det. red. %
Fixed N=1	10.3	500	0.0
Fixed N=2	20.4	250	50.0
Fixed N=3	30.3	167	66.6
Fixed N=5	51.8	100	80.0
Fixed N=10	103.2	50	90.0
Fixed N=15	128.9	34	93.2
Fixed N=20	166.4	25	95.0
Adaptive τ=0.12	11.8	427	14.6
Adaptive τ=0.18	13.8	380	24.0
Adaptive τ=0.25	14.3	355	29.0
Adaptive τ=0.35	15.4	329	34.2

VI. DISCUSSION

Two findings stand out. First, a fixed interval near N equal to 3 is a strong, simple operating point for this scene, and the coverage cliff between N equal to 3 and N equal to 5 marks where Kalman coasting can no longer bridge object motion. Second, the cliff location is scene-dependent, which is exactly why a content-adaptive policy should help in principle, even though our first design does not. The naive trigger fails because the global maximum over many pedestrians is almost always high, so the policy collapses to near per-frame detection. A per-track schedule, where each object is re-detected on its own motion budget, plus hysteresis and a confidence signal, is the clear next design.

Threats to validity are significant and stated plainly. Results use a single clip and a fidelity proxy rather than ground-truth MOTA, IDF1, or HOTA [6], [7], [8]. Throughput is measured on a CPU, not the target Raspberry Pi, so absolute speeds will differ. No energy is measured yet. SORT lacks an appearance model, which bounds identity performance under occlusion. These do not affect the hardware-independent fidelity trends, but they bound the claims.

VII. CONCLUSION AND FUTURE WORK

We characterized the detection-scheduling tradeoff for a YOLOv8n plus SORT pipeline, showing near-linear throughput gains with a sharp accuracy knee near N equal to 3, and we presented a first adaptive scheduler with an honest

account of why a global motion trigger over-fires. Future work is concrete: per-track and confidence-aware scheduling with hysteresis, deployment on a Raspberry Pi with INA219 rail-level energy measurement, and evaluation on MOT17 and MOT20 with true MOTA, IDF1, and HOTA [9]. Modern trackers such as ByteTrack [10] are candidate drop-in replacements for the association stage.

ACKNOWLEDGMENT

The authors thank the home laboratory for compute access.

References

[1] G. Jocher, A. Chaurasia, and J. Qiu, Ultralytics YOLOv8, version 8.0.0, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[2] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in Proc. IEEE Int. Conf. Image Process. (ICIP), 2016, pp. 3464-3468.

[3] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in Proc. IEEE Int. Conf. Image Process. (ICIP), 2017, pp. 3645-3649.

[4] H. Luo, W. Xie, X. Wang, and W. Zeng, "Detect or track: towards cost-effective video object detection/tracking," in Proc. AAAI Conf. Artif. Intell., vol. 33, no. 1, 2019, pp. 8803-8810.

[5] M. Liu, M. Zhu, et al., "Looking fast and slow: memory-guided mobile video object detection," arXiv:1903.10172, 2019.

[6] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the CLEAR MOT metrics," EURASIP J. Image Video Process., vol. 2008, art. 246309, 2008.

[7] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," in Proc. ECCV Workshops, LNCS 9914, 2016, pp. 17-35.

[8] J. Luiten et al., "HOTA: a higher order metric for evaluating multi-object tracking," Int. J. Comput. Vis., vol. 129, 2021, pp. 548-578.

[9] P. Dendorfer et al., "MOTChallenge: a benchmark for single-camera multiple target tracking," Int. J. Comput. Vis., vol. 129, 2021, pp. 845-881.

[10] Y. Zhang et al., "ByteTrack: multi-object tracking by associating every detection box," in Proc. Eur. Conf. Comput. Vis. (ECCV), 2022.