

# Towards Streaming Perception

Mengtian Li<sup>1</sup>, Yu-Xiong Wang<sup>1,2</sup>, and Deva Ramanan<sup>1,3</sup>

<sup>1</sup>CMU, <sup>2</sup>UIUC, and <sup>3</sup>Argo AI

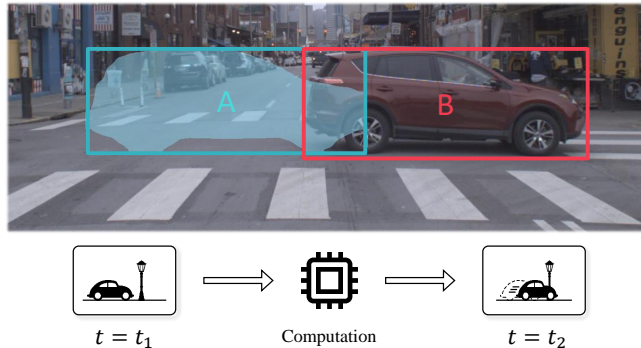
<https://www.cs.cmu.edu/~mengtial/proj/streaming/>

**Abstract.** Embodied perception refers to the ability of an autonomous agent to perceive its environment so that it can (re)act. The responsiveness of the agent is largely governed by latency of its processing pipeline. While past work has studied the algorithmic trade-off between latency and accuracy, there has not been a clear metric to compare different methods along the Pareto optimal latency-accuracy curve. We point out a discrepancy between standard offline evaluation and real-time applications: by the time an algorithm finishes processing a particular image frame, the surrounding world has changed. To these ends, we present an approach that coherently integrates latency and accuracy into a single metric for real-time online perception, which we refer to as “streaming accuracy”. The key insight behind this metric is to jointly evaluate the output of the entire perception stack at every time instant, forcing the stack to consider the amount of streaming data that should be ignored while computation is occurring. More broadly, building upon this metric, we introduce a meta-benchmark that systematically converts any image understanding task into a streaming perception task. We focus on the illustrative tasks of object detection and instance segmentation in urban video streams, and contribute a novel dataset with high-quality and temporally-dense annotations. Our proposed solutions and their empirical analysis demonstrate a number of surprising conclusions: (1) there exists an optimal “sweet spot” that maximizes streaming accuracy along the Pareto optimal latency-accuracy curve, (2) asynchronous tracking and future forecasting naturally emerge as internal representations that enable streaming image understanding, and (3) dynamic scheduling can be used to overcome temporal aliasing, yielding the paradoxical result that latency is sometimes minimized by sitting idle and “doing nothing”.

## 1 Introduction

Embodied perception refers to the ability of an autonomous agent to perceive its environment so that it can (re)act. A crucial quantity governing the responsiveness of the agent is its reaction time. Practical applications, such as self-driving vehicles or augmented reality and virtual reality (AR/VR), may require reaction time that rivals that of humans, which is typically 200 milliseconds (ms) for visual stimuli [14]. In such settings, low-latency algorithms are imperative to ensure safe operation or enable a truly immersive experience.

Historically, the computer vision community has not particularly focused on algorithmic latency. This is one reason why a disparate set of techniques



**Fig. 1.** Latency is inevitable in a real-world perception system. The system takes a snapshot of the world at  $t_1$  (the car is at location A), and when the algorithm finishes processing this observation, the surrounding world has already changed at  $t_2$  (the car is now at location B, and thus there is a mismatch between prediction A and ground truth B). If we define streaming perception as a task of continuously reporting back the current state of the world, then how should one evaluate vision algorithms under such a setting? We invite the readers to watch a video on the project website that compares a standard frame-aligned visualization with our latency-aware visualization [\[Link\]](#).

(and conference venues) have been developed for robotic vision. Interestingly, latency has been well studied recently (*e.g.*, fast but not necessarily state-of-the-art accurate detectors such as [25,18,16]). But it has still been primarily explored in an *offline* setting. Vision-for-online-perception imposes quite different latency demands as shown in Fig. 1, because by the time an algorithm finishes processing a particular image frame — say, after 200ms — the surrounding world has changed! This forces perception to be ultimately predictive of the future. In fact, such predictive forecasting is a fundamental property of human vision (*e.g.*, as required whenever a baseball player strikes a fast ball [22]). So we argue that streaming perception should be of interest to general computer vision researchers.

**Contribution (meta-benchmark)** To help explore embodied vision in a truly online streaming context, we introduce a general meta-benchmark that systematically converts *any* image understanding task into a streaming image understanding task. Our key insight is that streaming perception requires understanding the state of the world at all time instants — *when a new frame arrives, streaming algorithms must report the state of the world even if they have not done processing the previous frame*. Within this meta-benchmark, we introduce an approach to measure the real-time performance of perception systems. The approach is as simple as querying the state of the world at all time instants, and the quality of the response is measured by the *original* task metric. Such an approach naturally merges latency and accuracy into a single metric. Therefore, the trade-off between accuracy versus latency can now be measured quantitatively. Interestingly, our meta-benchmark naturally evaluates the perception stack *as a whole*.

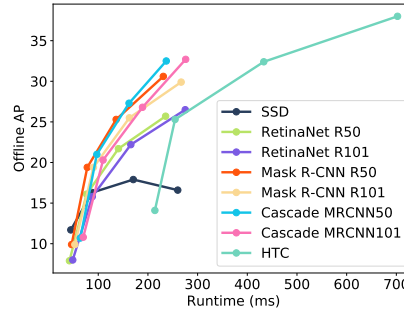
For example, a stack may include detection, tracking, and forecasting modules. Our meta-benchmark can be used to directly compare such modular stacks to end-to-end black-box algorithms [19]. In addition, our approach addresses the issue that overall latency of concurrent systems is hard to evaluate (*e.g.*, latency cannot be simply characterized by the runtime of a single module).

**Contribution (analysis)** Motivated by perception for autonomous vehicles, we instantiate our meta-benchmark on the illustrative tasks of object detection and instance segmentation in urban video streams. Accompanied with our streaming evaluation is a novel dataset with high-quality, high-frame-rate, and temporally-dense annotations of urban videos. Our evaluation on these tasks demonstrates a number of surprising conclusions. (1) Streaming perception is significantly more challenging than offline perception. Standard metrics like object-detection average precision (AP) dramatically drop (from 38.0 to 6.2), indicating the need for the community to focus on such problems. (2) Decision-theoretic scheduling, asynchronous tracking, and future forecasting naturally *emerge* as internal representations that enable accurate streaming image understanding, recovering much of the performance drop (boosting performance to 17.8). With simulation, we can verify that infinite compute resources modestly improves performance to 20.3, implying that our conclusions are fundamental to streaming processing, no matter the hardware. (3) It is well known that perception algorithms can be tuned to trade off accuracy versus latency. Our analysis shows that there exists an optimal “sweet spot” that uniquely maximizes streaming accuracy. This provides a different perspective on such well-explored trade-offs. (4) Finally, we demonstrate the effectiveness of decision-theoretic reasoning that dynamically schedules which frame to process at what time. Our analysis reveals the paradox that latency is minimized by sometimes sitting idle and “doing nothing”! Intuitively, it is sometimes better to wait for a fresh frame rather than to begin processing one that will soon become “stale”.

## 2 Related Work

**Latency evaluation** Latency is a well-studied subject in computer vision. One school of research focuses on reducing the FLOPS of backbone networks [12,28], while another school focuses on reducing the runtime of testing time algorithms [25,18,16]. We follow suit and create a latency-accuracy plot under our experiment setting (Fig. 2). While such a plot is suggestive of the trade-off for offline data processing (*e.g.*, archived video footage), it fails to capture the fact that *when the algorithm finishes processing, the surrounding world has already changed*. Therefore, we believe that existing plots do not reveal the streaming performance of these algorithms. Aside from computational latency, prior work has also investigated algorithmic latency [21], evaluated by running algorithms on a video in the *offline* fashion and measuring how many frames are required to detect an object after it appears. In comparison, our evaluation is done in the more realistic online real-time setting, and applies to any single image understanding task, instead of just object detection.

**Real-time evaluation** There has not been much prior effort to evaluate vision algorithms in the real-time fashion in the research community. Notable exceptions include work on real-time tracking and real-time simultaneous localization and mapping (SLAM). First, the VOT2017 tracking benchmark specifically included a real-time challenge [15]. Its benchmark toolkit sends out frames at 20 FPS to participants’ trackers and asks them to report back results before the next frame arrives. If the tracker fails to respond in time, the last reported result is used. This is equivalent to applying zero-order hold to trackers’ outputs. In our benchmarks, we adopt a similar zero-order hold strategy, but extend it to a broader context of arbitrary image understanding tasks and allow for a more delicate interplay between detection, tracking, and forecasting. Second, the literature on real-time SLAM also considers benchmark evaluation under a “hard-enforced” real-time requirement [4,8]. Our analysis suggests that hard-enforcement is too stringent of a formulation; algorithms should be allowed to run longer than the frame rate, but should still be scored on their ability to report the state of the world (*e.g.*, localized map) at frame rate.

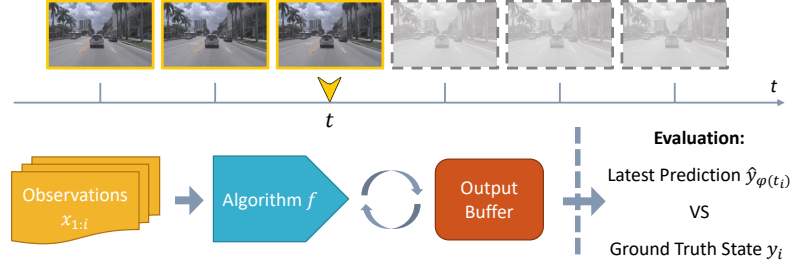


**Fig. 2.** Prior art routinely explores the trade-off between detection accuracy versus runtime. We generate the above plot by varying the input resolution of each detection network. We argue that such plots are exclusive to offline processing and fail to capture latency-accuracy trade-offs in streaming perception. AP stands for average precision, and is a standard metric for object detection [17].

**Progressive and anytime algorithms** There exists a body of work on progressive and anytime algorithms that can generate outputs with lower latency. Such work can be traced back to classic research on intelligent planning under resource constraints [3] and flexible computation [11], studied in the context of AI with bounded rationality [26]. Progressive processing [30] is a paradigm that splits up an algorithm into sequential modules that can be dynamically scheduled. Often, scheduling is formulated as a decision-theoretic problem under resource constraints, which can be solved in some cases with Markov decision processes (MDPs) [29,30]. Anytime algorithms are capable of returning a solution at any point in time [29]. Our work revisits these classic computation paradigms in the context of streaming perception, specifically demonstrating that classic visual tasks (like tracking and forecasting) naturally emerge in such bounded resource settings.

### 3 Proposed Evaluation

In the previous section, we have shown that existing latency evaluation fails to capture the streaming performance. To address this issue, here we propose a new



**Fig. 3.** Our proposed streaming perception evaluation. A streaming algorithm  $f$  is provided with (timestamped) observations up until the current time  $t$  and refreshes an output buffer with its latest prediction of the current state of the world. At the same time, the benchmark constantly queries the output buffer for estimates of world states. Crucially,  $f$  must consider the amount of streaming observations that should be ignored while computation is occurring.

method of evaluation. Intuitively, a streaming benchmark no longer evaluates a function, but a piece of executable code over a continuous time frame. The code has access to a sensor input buffer that stores the most recent image frame. The code is responsible for maintaining an output buffer that represents the up-to-date estimate of the state of the world (*e.g.*, a list of bounding boxes of objects in the scene). The benchmark examines this output buffer, comparing it with a ground truth stream of the actual world state (Fig. 3).

### 3.1 Formal definition

We model a data stream as a set of sensor observations, ground-truth world states, and timestamps, denoted respectively as  $\{(x_i, y_i, t_i)\}_{i=1}^T$ . Let  $f$  be a streaming algorithm to be evaluated. At any *continuous* time  $t$ , the algorithm  $f$  is provided with observations (and timestamps) that have appeared so far:

$$\{(x_i, t_i) | t_i \leq t\} \quad [\text{accessible input at time } t] \quad (1)$$

We allow the algorithm  $f$  to generate an output prediction at *any time*. Let  $s_j$  be the timestamp that indicates when a particular prediction  $\hat{y}_j$  is produced. The subscript  $j$  indexes over the  $N$  outputs generated by  $f$  over the entire stream:

$$\{(\hat{y}_j, s_j)\}_{j=1}^N \quad [\text{all outputs by } f] \quad (2)$$

Note that this output stream is *not* synchronized with the input stream, and  $N$  has no direct relationship with  $T$ . Generally speaking, we expect algorithms to run slower than the frame rate ( $N < T$ ).

We benchmark the algorithm  $f$  by comparing its most recent output at time  $t_i$  to the ground-truth  $y_i$ . We first compute the index of the most recent output:

$$\varphi(t) = \arg \max_j s_j < t \quad [\text{real-time constraint}] \quad (3)$$

This is equivalent to the benchmark applying a *zero-order hold* for the algorithm’s outputs to produce continuous estimation of the world states. Given an arbitrary single-frame loss  $L$ , the benchmark formally evaluates:

$$L_{\text{streaming}} = L(\{(y_i, \hat{y}_{\varphi(t_i)})\}_{i=1}^T) \quad [\text{evaluation}] \quad (4)$$

By construction, the streaming loss above can be applied to *any* single-frame task that computes a loss over a set of ground truth and prediction pairs.

### 3.2 Emergent tracking and forecasting

At first glance, “instant” evaluation may seem unreasonable: the benchmark at time  $t$  queries the state at time  $t$ . Although  $x_t$  is made available to the algorithm, any finite-time algorithm cannot make use of it to generate its prediction. For example, if the algorithm takes time  $\Delta t$  to perform its computation, then to make a prediction at time  $t$ , it can only use data before time  $t - \Delta t$ . We argue that this is the *realistic* setting for streaming perception, both in biological and robotic systems. Humans and autonomous vehicles must react to the instantaneous state of the world when interacting with dynamic scenes. Such requirements strongly suggest that perception should be inherently predictive of the future. Our benchmark similarly “forces” algorithms to reason and forecast into the future, to compensate for the mismatch between the last processed observation and the present.

One may also wish to take into account the inference time of downstream actuation modules (that say, need to optimize a motion plan that will be executed given the perceived state of the world). It is straightforward to extend our benchmark to require algorithms to generate a forecast of the world state when the downstream module finishes its processing. For example, at time  $t$  the benchmark queries the state of the world at time  $t + \eta$ , where  $\eta > 0$  represents the inference time of the downstream actuation module.

In order to forecast, the algorithms need to reason temporally through tracking (in the case of object detection). For example, constant velocity forecasting requires the tracks of each object over time in order to compute the velocity. Generally, there are two categories of trackers — post-hoc association [2] and template-based visual tracking [20]. In this paper, we refer them in short as “association” and “tracking”, respectively. Association of previously computed detections can be made extremely lightweight with simple linking of bounding boxes (*e.g.*, based on the overlap). However, association does not make use of the image itself as done in (visual) tracking. We posit that trackers may produce better streaming accuracy for scenes with highly unpredictable motion. As part of emergent solutions to our streaming perception problem, we include both association and tracking in our experiments in the next section.

Finally, it is natural to seek out an end-to-end system that directly optimizes streaming perception accuracy. We include one such method in Appendix C.2 to show that tracking and forecasting-based representations may also emerge from gradient-based learning.

### 3.3 Computational constraints



**Fig. 4.** Two computation models considered in our evaluation. Each block represents an algorithm running on a device and its length indicates its runtime.

Because our metric is runtime dependent, we need to specify the computational constraints to enable a fair comparison between algorithms. We first investigate a single GPU model (Fig. 4a), which is used for existing latency analysis in prior art. In the single GPU model, only a single GPU job (*e.g.*, detection or visual tracking) can run at a time. Such a restriction avoids multi-job interference and memory capacity issues. Note that a reasonable number of CPU jobs are allowed to run concurrently with the GPU job. For example, we allow bounding box association and forecasting modules to run on the CPU in Fig. 7.

Nowadays, it is common to have multiple GPUs in a single system. We investigate an *infinite* GPU model (Fig. 4b), with no restriction on the number of GPU jobs that can run concurrently. We implement this infinite computation model with *simulation*, described in the next subsection.

### 3.4 Challenges for practical implementation

While our benchmark is conceptually simple, there are several practical hurdles. First, we require high-frame-rate ground truth annotations. However, due to high annotation cost, most existing video datasets are annotated at rather sparse frame rates. For example, YouTube-VIS is annotated at 6 FPS, while the video data rate is 30 FPS [27]. Second, our evaluation is *hardware dependent* — the same algorithm on different hardware may yield different streaming performance. Third, stochasticity in actual runtimes yields stochasticity in the streaming performance. Note that the last two issues are also prevalent in *existing* offline runtime analyses. Here we present high-level ideas for the solutions and leave additional details to Appendix A.2 & A.3.

**Pseudo ground truth** We explore the use of pseudo ground truth labels as a surrogate to manual high-frame-rate annotations. The pseudo labels are obtained by running state-of-the-art, *arbitrarily expensive* offline algorithms on each frame of a benchmark video. While the absolute performance numbers (when benchmarked on ground truth and pseudo ground truth labels) differ, we find that the rankings of algorithms are remarkably stable. The Pearson correlation coefficient of the scores of the two ground truth sets is 0.9925, suggesting that the real score is literally a linear function of the pseudo score. Moreover, we find

that offline pseudo ground truth could also be used to self-supervise the training of streaming algorithms.

**Simulation** While streaming performance is hardware dependent, we now demonstrate that the benchmark can be evaluated on simulated hardware. In simulation, the benchmark assigns a runtime to each module of the algorithm, instead of measuring the wall-clock time. Then based on the assigned runtime, the simulator generates the corresponding output timestamps. The assigned runtime to each module provides a layer of abstraction on the hardware.

The benefit of simulation is to allow us to assess the algorithm performance on non-existent hardware, *e.g.*, a future GPU that is 20% faster or infinite GPUs in a single system. Simulation also allows our benchmark to inform practitioners about computation platforms necessary to obtain a certain level of accuracy.

**Runtime-induced variance** Due to algorithmic choice and system scheduling, different runs of the same algorithm may end up with different runtimes. This variation across runs also affects the overall streaming performance. Fortunately, we empirically find that such variance causes a standard deviation of up to 0.5% under our experiment setting. Therefore, we omit variance report in our experiments.

## 4 Solutions and Analysis

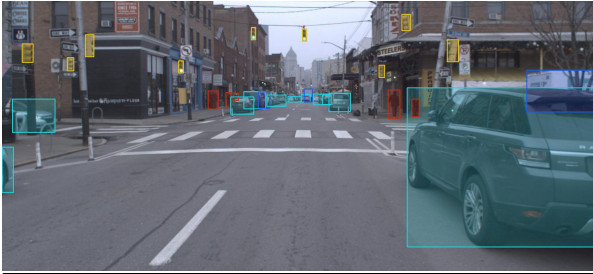
In this section, we instantiate our meta-benchmark on the illustrative task of object detection. While we show results on streaming detection, several key ideas also generalize to other tasks. An instantiation on instance segmentation can be found in Appendix A.6. We first explain the setup and present the solutions and analysis. For the solutions, we first consider single-frame detectors, and then add forecasting and tracking one by one into the discussion. We focus on the most effective combination of detectors, trackers, and forecasters which we have evaluated, but include additional methods in Appendix C.

### 4.1 Setup

We extend the publicly available video dataset Argoverse 1.1 [5] with our own annotations for streaming evaluation, which we name Argoverse-HD (High-frame-rate Detection). It contains diverse urban outdoor scenes from two US cities. We select Argoverse for its embodied setting (autonomous driving) and its high-frame-rate sensor data (30 FPS). We focus on the task of 2D object detection for our streaming evaluation. Under this setting, the state of the world  $y_t$  is a list of bounding boxes of the objects of interest. While Argoverse has multiple sensors, we only use the center RGB camera for simplicity. We collect our own annotations since the dataset does not provide dense 2D annotations<sup>1</sup>. For the

<sup>1</sup> It is possible to derive 2D annotations from the provided 3D annotations, but we find that such derived annotations are highly imprecise.





Dataset	AP	AP <sub>L</sub>	AP <sub>M</sub>	AP <sub>S</sub>	AP <sub>50</sub>	AP <sub>75</sub>
MS COCO	37.6	50.3	41.4	20.7	59.8	40.5
Argoverse-HD (Ours)	30.6	52.4	33.1	12.2	52.3	31.2

**Fig. 5.** Comparison between our dataset and MS COCO [17]. Top shows an example image from Argoverse 1.1 [5], overlaid with our dense 2D annotation (at 30 FPS). Bottom presents results of Mask R-CNN [10] (ResNet 50) evaluated on the two datasets. AP<sub>L</sub>, AP<sub>M</sub> and AP<sub>S</sub> denote AP for large, medium and small objects respectively. AP<sub>50</sub>, AP<sub>75</sub> denote AP with IoU (Intersection over Union) thresholds at 0.5 and 0.75 respectively. We first observe that the APs are roughly comparable, showing that our annotation is reasonable in evaluating object detection performance. Second, we see a significant drop in AP<sub>S</sub> from COCO to ours, suggesting that the detection of small objects is more challenging in our setting. For self-driving vehicle applications, those small objects are important to identify when the ego-vehicle is traveling at a high speed or making unprotected turns.

annotations, we follow MS COCO [17] class definitions and format. For example, we include the “iscrowd” attribute for ambiguous cases where each instance cannot be identified, and therefore the algorithms will not be wrongfully penalized. We use only a subset of 8 classes (from 80 MS COCO classes) that are directly relevant to autonomous driving: person, bicycle, car, motorcycle, bus, truck, traffic light, and stop sign. This definition allows us to evaluate off-the-shelf models trained on MS COCO. No training is involved in the following experiments unless otherwise specified. All numbers are computed on the validation set, which contains 24 videos ranging from 15–30 seconds each (the total number of frames is 15k). Figure 5 shows a comparison of our annotation with that of MS COCO. Additional comparison with other related datasets can be found in Appendix A.4. All output timing is measured on a single Geforce GTX 1080 Ti GPU (a Tesla V100 counterpart is provided in Appendix A.7).

## 4.2 Detection-Only

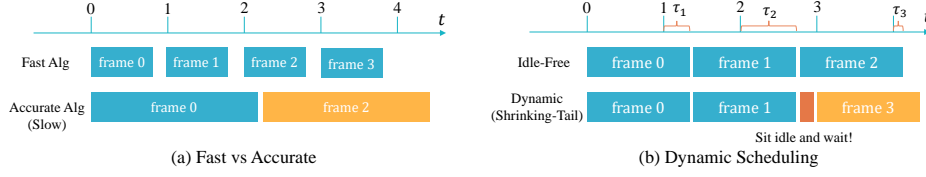
Table 1 includes the main results of using just detectors for streaming perception. We first examine the case of running a state-of-the-art detector — Hybrid Task Cascade (HTC) [6], both in the offline and the streaming settings. The AP drops significantly in the streaming setting. Such a result is not entirely surprising due to its high runtime (700ms). A commonly adopted strategy for real-time applications is to run a detector that is within the frame rate. We point out that

**Table 1.** Performance of existing detectors for streaming perception. The number after @ is the input scale (the full resolution is  $1920 \times 1200$ ). \* means using GPU for image pre-processing as opposed to using CPU in the off-the-shelf setting. The last column is the mean runtime of the detector for a single frame in milliseconds (mask branch disabled if applicable). The first baseline is to run an accurate detector (row 1), and we observe a significant drop of AP in the online real-time setting (row 2). Another commonly adopted baseline for embodied perception is to run a fast detector (row 3–4), whose runtime is smaller than unit time interval (33ms for 30 FPS streams). Neither of these baselines achieves good performance. Searching over a wide suite of detectors and input scales, we find that the optimal solution is Mask R-CNN (ResNet 50) operating at 0.5 input scale (row 5–6). In addition, our scheduling algorithm (Alg. 1) boosts the performance by 1.0/2.3 for AP/ $AP_L$  (row 7). In the hypothetical infinite GPU setting, a more expensive detector yields better trade-off (input scale switching from 0.5 to 0.75, almost doubling the runtime), and it further boosts the performance to 14.4 (row 8), which is the optimal solution achieved by just running the detector. Simulation suggests that 4 GPUs suffice to maximize streaming accuracy for this solution

ID	Method	Detector	AP	$AP_L$	$AP_M$	$AP_S$	$AP_{50}$	$AP_{75}$	Runtime
1	Accurate (Offline)	HTC @ s1.0	38.0	64.3	40.4	17.0	60.5	38.5	700.5
2	Accurate	HTC @ s1.0	6.2	9.3	3.6	0.9	11.1	5.9	700.5
3	Fast	RetinaNet R50 @ s0.2	5.5	14.9	0.4	0.0	9.9	5.6	36.4
4	Fast*	RetinaNet R50 @ s0.2	6.0	18.1	0.5	0.0	10.3	6.3	<b>31.2</b>
5	Optimized	Mask R-CNN R50 @ s0.5	10.6	21.2	6.3	0.9	22.5	8.8	77.9
6	Optimized*	Mask R-CNN R50 @ s0.5	<b>12.0</b>	<b>24.3</b>	<b>7.9</b>	<b>1.0</b>	<b>25.1</b>	<b>10.1</b>	56.7
7	+ Scheduling (Alg. 1)	Mask R-CNN R50 @ s0.5	13.0	26.6	9.2	1.1	26.8	11.1	56.7
8	+ Infinite GPUs	Mask R-CNN R50 @ s0.75	14.4	24.3	11.3	2.8	30.6	12.1	92.7

this strategy may be problematic, since such a hard-constrained time budget results in poor accuracy for challenging tasks (Table 1 row 3–4). In addition, we find that many existing network implementations are optimized for throughput rather than latency, reflecting the bias of the community for offline versus online processing! For example, image pre-processing (*e.g.*, resizing and normalizing) is often done on CPU, where it can be pipelined with data pre-fetching. By moving it to GPU, we save 21ms in latency (for an input of size  $960 \times 600$ ).

In our benchmarks, it is a *choice* for the streaming algorithm to decide when and what to process. Figure 6 compares a straight-forward schedule with our dynamic schedule (Alg. 1). Such subtlety is the result of temporal quantization. While spatial quantization has been studied in computer vision [10], temporal quantization in the streaming setting has not been well explored. Notably, it is difficult to pre-compute the optimal schedule because of the stochasticity of actual runtimes. Our proposed scheduling policy (Alg. 1) minimizes the expected temporal mismatch of the output stream and the data stream, thus increasing the overall streaming performance. Empirically, we find that it raises the AP for the detector (Table 1 row 7). We provide *theoretical reasoning showing its superiority* and results for a wide suite of detectors in Appendix B.1. Note that Alg. 1 is by construction task agnostic (not specific to object detection).



**Fig. 6.** Algorithm scheduling for streaming perception with a single GPU. (a) A fast detector finishes processing the current frame before the next frame arrives. An accurate (and thus slow) detector does not process every frame due to high latency. In this example, frame 1 is skipped. Note that the goal of streaming perception is not to process every frame but to produce accurate state estimations in a timely manner. (b) A straight-forward schedule for slow algorithms (runtime > unit time interval) is to always process the latest available frame upon the completion of the previous processing (idle-free). However, the latest available frame might be stale, and we find that it might be better to sit idle and wait (our dynamic schedule, Alg. 1). In this illustration, when the algorithm finishes processing frame 1, Alg. 1 determines that frame 2 is stale and decides to wait for frame 3 by comparing the tails  $\tau_2$  and  $\tau_3$ .

---

**Algorithm 1** Shrinking-tail policy

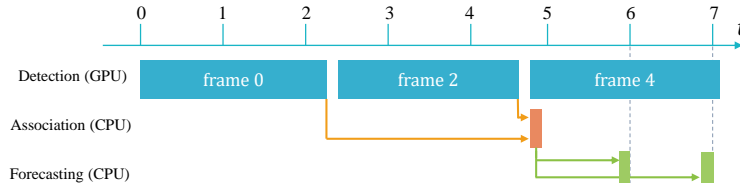
---

- 1: Given finishing time  $s$  and algorithm runtime  $r$  in the unit of frames (assuming  $r > 1$ ), this policy returns whether the algorithm should wait for the next frame
  - 2: Define tail function  $\tau(t) = t - \lfloor t \rfloor$
  - 3: **return**  $[\tau(s + r) < \tau(s)]$  (Iverson bracket)
- 

### 4.3 Forecasting

Now we expand our solution space to include forecasting methods. We experimented with both constant velocity models and first-order Kalman filters. We find good performance with the latter, given a small modification to handle asynchronous sensor measurements (Fig. 7). The classic Kalman filter [13] operates on uniform time steps, coupling prediction and correction updates at each step. In our case, we perform correction updates only when a sensor measurement is available, but predict at every step. Second, due to frame-skipping, the Kalman filter should be time-varying (the transition and the process noise depend on the length of the time interval, details can be found in Appendix B.2). Association for bounding boxes across frames is required to update the Kalman filter, and we apply IoU-based greedy matching. For association and forecasting, the computation involves only bounding box coordinates and therefore is very lightweight (< 2ms on CPU). We find that such overhead has little influence on the overall AP. The results are summarized in Table 2.

**Streamer (meta-detector)** Note that our dynamic scheduler (Alg. 1) and asynchronous Kalman forecaster can be applied to *any* off-the-shelf detector, regardless of its underlying latency (or accuracy). This means that we can assemble these modules into a *meta-detector* – which we call *Streamer* – that converts any detector into a streaming detection system that reports real-time



**Fig. 7.** Scheduling for association and forecasting. Association takes place immediately after a new detection result becomes available, and it links the bounding boxes in two consecutive detection results. Forecasting takes place right before the next time step and it uses an asynchronous Kalman filter to produce an output as the estimation of the current world state. By default, the prediction step also updates internal states in the Kalman filter and is always called before the update step. In our case, we perform multiple update-free predictions (green blocks) until we receive a frame result.

**Table 2.** Streaming perception with joint detection, association, and forecasting. Association is done by IoU-based greedy matching, while forecasting is done by an asynchronous Kalman filter. First, we observe that forecasting greatly boosts the performance (from Table 1 row 7’s 13.0 to row 1’s 16.7). Also, with forecasting compensating for algorithm latency, it is now desirable to run a more expensive detector (row 2). Searching again over a large suite of detectors after adding forecasting, we find that the optimal detector is still Mask R-CNN (ResNet 50), but at input scale 0.75 instead of 0.5 (runtime 93ms and 57ms)

ID	Method	AP	AP <sub>L</sub>	AP <sub>M</sub>	AP <sub>S</sub>	AP <sub>50</sub>	AP <sub>75</sub>
1	Detection + Scheduling + Association + Forecasting	16.7	39.9	14.9	1.2	31.2	16.0
2	+ Re-optimize Detection (s0.5 → s0.75)	17.8	33.3	16.3	3.2	35.2	16.5
3	+ Infinite GPUs	20.3	38.5	19.9	4.0	39.1	18.9

detections at an arbitrary framerate. Appendix B.4 evaluates the improvement in streaming AP across 80 different settings (8 detectors  $\times$  5 image scales  $\times$  2 compute models), which vary from 4% to 80% with an average improvement of 33%.

#### 4.4 Visual tracking

Visual tracking is an alternative for low-latency inference, due to its faster speed than a detector. For our experiments, we adopt the state-of-the-art multi-object tracker [1] (which is second place in the MOT’19 challenge [7] and is open sourced), and modify it to only track previously identified objects to make it faster than the base detector (see Appendix B.3). This tracker is built upon a two-stage detector and for our experiment, we try out the configurations of Mask R-CNN with different backbones and with different input scales. Also, we need a scheduling scheme for this detection plus tracking setting. For simplicity, we only explored running detection at fixed strides of 2, 5, 15, and 30. For example, stride 30 means that we run the detector once and then run the tracker 29 times, with the tracker getting reset after each new detection. Table 3 row 1 contains the best configuration over backbone, input scale, and detection stride.

**Table 3.** Streaming perception with joint detection, visual tracking, and forecasting. We see that initially visual trackers do not outperform simple association (Table 2) with the corresponding setting in the single GPU case. But that is reversed if the tracker can be optimized to run faster (2x) while maintaining the same accuracy (row 6). Such an assumption is not unreasonable given the fact that the tracker’s job is as simple as updating locations of previously detected objects

ID	Method	AP	AP <sub>L</sub>	AP <sub>M</sub>	AP <sub>S</sub>	AP <sub>50</sub>	AP <sub>75</sub>
1	Detection + Visual Tracking	12.0	29.7	11.2	0.5	23.3	11.3
2	+ Forecasting	13.7	38.2	14.2	0.5	24.6	13.6
3	+ Re-optimize Detection (s0.5 → s0.75)	16.5	31.0	14.5	2.8	33.4	14.8
4	+ Infinite GPUs w/o Forecasting	14.4	24.2	11.2	2.8	30.6	12.0
5	+ Forecasting	20.1	38.3	19.7	3.9	38.9	18.7
6	Detection + Simulated Fast Tracker (2x) + Forecasting + Single GPU	19.8	39.2	20.2	3.4	38.6	18.1

## 5 Discussion

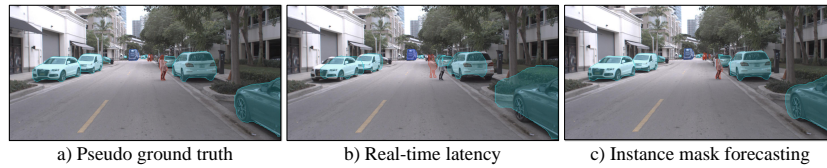
**Streaming perception remains a challenge** Our analysis suggests that streaming perception involves careful integration of detection, tracking, forecasting, and dynamic scheduling. While we present several strong solutions for streaming perception, the gap between the streaming performance and the offline performance remains significant (20.3 versus 38.0 in AP). This suggests that there is considerable room for improvement by building a better detector, tracker, forecaster, or even an end-to-end model that blurs boundary of these modules.

**Formulations of real-time computation** Common folk wisdom for real-time applications like online detection requires that detectors run within the sensor frame rate. Indeed, classic formulations of anytime processing require algorithms to satisfy a “contract” that they will finish under a compute budget [29]. Our analysis suggests that this view of computation might be too myopic as evidenced by contemporary robotic systems [24]. Instead, we argue that the sensor rate and compute budget should be seen as design choices that can be tuned to optimize a downstream task. Our streaming benchmark allows for such a global perspective.

**Generalization to other tasks** By construction, our meta-benchmark and dynamic scheduler (Alg. 1) are not restricted to object detection. We illustrate such generalization with an additional task of instance segmentation (Fig. 9). However, there are several practical concerns that need to be addressed. Densely annotating video frames for instance segmentation is almost prohibitively expensive. Therefore, we adopt offline pseudo ground truth (Section 3.4) to evaluate streaming performance. Another concern is that the forecasting module is task-specific. In the case of instance segmentation, we implement it as forecasting the bounding boxes and then warping the masks accordingly. Please refer to Appendix A.6 for the complete streaming instance segmentation benchmark.



**Fig. 8.** Qualitative results. Video results can be found on the project website [\[Link\]](#).



**Fig. 9.** Generalization to instance segmentation. (a) The offline pseudo ground truth we adopt for evaluation is of high quality. (b) A similar latency pattern can be observed for instance segmentation as in object detection. (c) Forecasting for instance segmentation can be implemented as forecasting the bounding boxes and then warping the masks accordingly.

## 6 Conclusion and Future Work

We introduce a meta-benchmark for systematically converting any image understanding task into a streaming perception task that naturally trades off computation between multiple modules (*e.g.*, detection versus tracking). We instantiate this meta-benchmark on tasks of object detection and instance segmentation. In general, we find online perception to be dramatically more challenging than its offline counterpart, though significant performance can be recovered by incorporating forecasting. We use our analysis to develop a simple meta-detector that converts any detector (with any internal latency) into a streaming perception system that can operate at any frame rate dictated by a downstream task (such as a motion planner). We hope that our analysis will lead to future endeavor in this under-explored but crucial aspect of real-time embodied perception. For example, streaming benchmarks can be used to motivate attentional processing; by spending more compute only on spatially [9] or temporally [23] challenging regions, one may achieve even better efficiency-accuracy tradeoffs.

**Acknowledgements:** This work was supported by the CMU Argo AI Center for Autonomous Vehicle Research and was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0051. Annotations for ArgoVerse-HD were provided by Scale AI.

## References

1. Bergmann, P., Meinhardt, T., Leal-Taixé, L.: Tracking without bells and whistles. In: ICCV (2019) [12](#)
2. Bewley, A., Ge, Z., Ott, L., Ramos, F., Upcroft, B.: Simple online and realtime tracking. In: ICIP (2016) [6](#)
3. Boddy, M., Dean, T.L.: Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence* (1994) [4](#)
4. Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., Leonard, J.J.: Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics* (2016) [4](#)
5. Chang, M.F., Lambert, J.W., Sangkloy, P., Singh, J., Bak, S., Hartnett, A., Wang, D., Carr, P., Lucey, S., Ramanan, D., Hays, J.: Argoverse: 3D tracking and forecasting with rich maps. In: CVPR (2019) [8](#), [9](#)
6. Chen, K., Pang, J., Wang, J., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Shi, J., Ouyang, W., Loy, C.C., Lin, D.: Hybrid task cascade for instance segmentation. In: CVPR (2019) [9](#)
7. Dendorfer, P., Rezatofighi, H., Milan, A., Shi, J., Cremers, D., Reid, I., Roth, S., Schindler, K., Leal-Taixé, L.: CVPR19 tracking and detection challenge: How crowded can it get? *arXiv:1906.04567* (2019) [12](#)
8. Engel, J., Koltun, V., Cremers, D.: Direct sparse odometry. *TPAMI* (2017) [4](#)
9. Gao, M., Yu, R., Li, A., Morariu, V.I., Davis, L.S.: Dynamic zoom-in network for fast object detection in large images. In: CVPR (2018) [14](#)
10. He, K., Gkioxari, G., Dollár, P., Girshick, R.B.: Mask R-CNN. In: ICCV (2017) [9](#), [10](#)
11. Horvitz, E.J.: Computation and action under bounded resources. Ph.D. thesis, Stanford University (1990) [4](#)
12. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv abs/1704.04861* (2017) [3](#)
13. Kalman, R.E.: A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering* **82**(Series D), 35–45 (1960) [11](#)
14. Kosinski, R.J.: A literature review on reaction time. *Clemson University* **10** (2008) [1](#)
15. Kristan, M., Leonardis, A., Matas, J., Felsberg, M., Pflugfelder, R., Čehovin Zajc, L., Vojir, T., Häger, G., Lukežič, A., Eldesokey, A., Fernandez, G.: The visual object tracking VOT2017 challenge results (2017) [4](#)
16. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: ICCV (2017) [2](#), [3](#)
17. 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: ECCV (2014) [4](#), [9](#)
18. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C.Y., Berg, A.C.: SSD: Single shot multibox detector. In: ECCV (2016) [2](#), [3](#)
19. Luc, P., Couprie, C., LeCun, Y., Verbeek, J.: Predicting future instance segmentations by forecasting convolutional features. In: ECCV (2018) [3](#)
20. Lukežic, A., Vojir, T., Zajc, L.C., Matas, J., Kristan, M.: Discriminative correlation filter with channel and spatial reliability. In: CVPR (2017) [6](#)



21. Mao, H., Yang, X., Dally, W.J.: A delay metric for video object detection: What average precision fails to tell. In: ICCV (2019) [3](#)
22. McLeod, P.: Visual reaction time and high-speed ball games. *Perception* (1987) [2](#)
23. Mullapudi, R.T., Chen, S., Zhang, K., Ramanan, D., Fatahalian, K.: Online model distillation for efficient video inference. In: ICCV (2019) [14](#)
24. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: ICRA workshop on open source software. Kobe, Japan (2009) [13](#)
25. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: CVPR (2016) [2](#), [3](#)
26. Russell, S.J., Wefald, E.: Do the right thing: studies in limited rationality. MIT press (1991) [4](#)
27. Yang, L., Fan, Y., Xu, N.: Video instance segmentation. In: ICCV (2019) [7](#)
28. Zhang, X., Zhou, X., Lin, M., Sun, J.: ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In: CVPR (2018) [3](#)
29. Zilberstein, S.: Using anytime algorithms in intelligent systems. *AI magazine* (1996) [4](#), [13](#)
30. Zilberstein, S., Mouaddib, A.I.: Optimal scheduling of progressive processing tasks. *International Journal of Approximate Reasoning* (2000) [4](#)