# AROD: Adaptive Real-Time Object Detection Based on Pixel Motion Speed

Yu Liu, Kyoung-Don Kang
Department of Computer Science
State University of New York at Binghamton
{yliu456, kang}@binghamton.edu

***Abstract*—Real-time object detection is essential for AI-based intelligent traffic management. However, growing complexities of deep learning models for object detection cause increased latency and resource requirements. To tackle the challenge, we introduce a new approach, named AROD (Adaptive Real-Time Object Detection), that infers the pixel motion speed in continuous traffic video frames and skips redundant frames when the pixel velocity is low. Thereby, AROD aims to significantly enhance the efficiency and scalability, sustaining the accuracy of object detection. Our evaluation using real-world traffic videos reveals that our method for pixel velocity inference via lightweight deep learning reduces the RMSE (Root Mean Square Error) by up to two orders of magnitude compared to state-of-the-art approaches. AROD improves the frame processing rate of YOLOv5, SSD, and EfficientDet by approximately 32-61%, 110-174%, and 120-213%, respectively. AROD considerably enhances scalability by supporting real-time object detection for up to three concurrent traffic video streams on a commodity machine. Moreover, AROD demonstrates its generalizability by supporting competitive accuracy in object detection for a separate traffic video that was fully hidden during training.**

***Index Terms*—AI-based Intelligent Traffic Management, Pixel Motion Speed Inference, Adaptive Real-Time Object Detection**

## I. Introduction

Real-time object detection is crucial in AI-based intelligent traffic management in a smart city [1]. Although deep learning models, such as [2]–[8], have significantly improved the quality of inference for computer vision tasks, their increasing complexities result in higher latency and resource requirements. Therefore, it is challenging to improve the efficiency and scalability of object detection, supporting the desired processing rate of 30 fps (frames per second) for real-time object detection.

To tackle the challenge, we introduce AROD (Adaptive Real-Time Object Detection) that is *not yet another object detection model but an adaptive framework for efficient real-time object detection.* AROD infers the speed of pixel movements (measured in pixels per frame), indicating the average movement of objects between successive frames in a traffic video. AROD then dynamically reduces the object detection rate and drops redundant frames when the pixel velocity is low. A summary of our **contributions** follows:

- To minimize errors in pixel motion speed inference, we carefully design and train a new lightweight MLP (Multiple Layer Perceptron) model for cost-effective regression, avoiding overfitting.
- We devise a new, flexible object detection framework that can utilize any effective object detection model. The execution rate of the plugged-in object detection model is dynamically adapted according to the inferred pixel velocity for efficient object detection.
- To further enhance efficiency, AROD utilizes both the CPU and GPU via pipelining. In the CPU that is mostly idle otherwise, AROD infers the average pixel speed of moving objects and adapts the object detection rate accordingly. At the same time, object detection is performed for the frames that are forwarded to the GPU, instead of getting dropped due to the slow pixel velocity. To conceal the latency of AROD and improve the overall frame processing rate in terms of fps, we ensure that the latency of AROD run in the CPU is shorter than that of object detection in the GPU.

Our evaluation is conducted using four real-world traffic videos in a cost-effective video analytics system consisting of commodity hardware components, much less powerful and expensive than its cloud counterparts. Using the test sets of the first three traffic videos captured under diverse environmental lighting conditions, we analyze the accuracy of pixel velocity inference, frame processing rate, and scalability in terms of the RMSE, fps, and number of traffic video streams processed concurrently, respectively. Our evaluation reveals that our MLP model decreases the RMSE of pixel velocity inference by up to two orders of magnitude compared to several SOTA (state-of-the-art) methods. To demonstrate the flexibility of AROD, we combine it with three popular object detection models: YOLOv5 [4], SSD [5], and EfficientDet [6]. AROD improves their fps by approximately 32-61%, 110-174%, and 120-213%, respectively. When AROD runs alongside YOLOv5 (fastest among the three models above), it can process up to three traffic video streams in real-time, supporting 30-34 fps for each stream. Thus, AROD could considerably reduce the number of machines needed for AI-based intelligent traffic management in a smart city. Moreover, the integration of AROD with YOLOv5 effectively generalizes to the fourth traffic video entirely unseen during training. It maintains an approximately equivalent mean average precision (mAP), which serves as the standard metric for assessing object detection accuracy, in comparison to the mAP attained by the standalone YOLOv5 model without AROD.

The rest of the paper is organized as follows. Related work is discussed in Section II. In Section III, background information on optical flow analysis is followed by a discussion of its limitations. Section IV describes AROD introduced in this paper. Design and training of the MLP model for pixel velocity inference, data sets, and system configurations for evaluation are described in Section V. Section VI discusses the evaluation results. Finally, Section VII concludes the paper and discusses future work. Our source code is available at https://github.com/Real-Time-Lab/AROD-Adaptive-Real-Time-Object-Detestion-Aware-of-the-Traffic-Flow-Speed.

## II. Related Work

**Object detection:** Deep learning-based object detection models have significantly outperformed traditional approaches. Single-stage object detection models based on CNNs (Convolutional Neural Networks), such as YOLO [4], SSD [5], and EfficientDet [6], detect objects, produce their bounding boxes, and classify them in one stage. Thus, they are more suitable for real-time applications than two-stage object detection models, such as [2], [3], [9]. Novel vision transformers, such as [7], [8], have improved the accuracy of computer vision tasks. However, self-attention in vision transformers suffers from quadratic complexity. Generally speaking, increasing complexities of newer models tend to improve object detection accuracy, while increasing the latency and resource consumption.
**Efficiency Improvements:** In [10], [11], structural similarity in successive frames is analyzed to drop redundant frames. In [12], a new CNN model, called ERD (Empty Road Detection), is used to preprocess every traffic video frame to infer if there exists any predefined object of interest (e.g., a vehicle). ERD then performs object detection only for nonempty frames with at least one object. Also, L-filter [13] reduces the overhead of ERD via hybrid time series analysis. However, the overhead for structural similarity analysis [10], [11] and road occupancy (empty or nonempty) inference [12], [13] cannot be amortized when successive traffic video frames are dissimilar or nonempty, for example, on a busy highway. Consequently, they can actually decrease the frame processing rate essential for real-time object detection. Different from [10]–[13], AROD hides its latency via efficient CPU-GPU pipelining. Thus, it supports at least as high fps as the corresponding standalone object detection model without AROD does. Moreover, AROD can use any object detection model, e.g., a single-stage YOLO model, and therefore it is complementary to most existing work on object detection, including the ones discussed above. This contrasts to [10], [11] that depend on a relatively slow two-stage object detector [3]. Although model compression [14], [15] reduces model complexity, we do not utilize a compressed object detection model in this paper, since it often decreases accuracy.

## III. Background on Optical Flow Analysis

The Lucas-Kanade method (L-K method) [16] is a widely used optical flow analysis algorithm. It assumes 1) brightness constancy, 2) temporal continuity, and 3) spatial coherence in a sequence of images (frames). Brightness constancy assumes that a pixel's grayscale value does not change between frames. Temporal continuity assumes that the displacement of image contents between consecutive frames is small and approximately constant. The first two assumptions allow taking partial derivatives of grayscale values with respect to the pixel position:

$$I_x v_x + I_y v_y = -I_t \tag{1}$$

where $v_x = dx/dt$ is the optical flow velocity of a pixel in the horizontal direction and $v_y = dy/dt$ is the optical flow velocity in the vertical direction. Additionally, $I_x = dI/dx$ and $I_y = dI/dy$ are the gradients (partial derivatives) of image intensity at position $(x, y)$, while $I_t = dI/dt$ is the gradient of the intensity at time $t$.

In Equation 1, we encounter two unknown variables: $v_x$ and $v_y$. Thus, Equation 1 is under-constrained and cannot be solved directly. The L-K algorithm addresses this challenge using the third assumption of spatial coherence. It assumes that adjacent pixels on the same surface in the scene share similar motion and, therefore, their projections onto the image plane and velocities are closely aligned and consistent. Based on this assumption, the L-K method formulates multiple equations for $n$ neighboring pixels in matrix form, $Av = b$, to solve for $v_x$ and $v_y$:

$$A = \begin{bmatrix} I_{x_1} & I_{y_1} \\ \vdots & \vdots \\ I_{x_i} & I_{y_i} \\ \vdots & \vdots \\ I_{x_n} & I_{y_n} \end{bmatrix}, v = \begin{bmatrix} v_x \\ v_y \end{bmatrix}, b = \begin{bmatrix} -I_{t_1} \\ \vdots \\ -I_{t_i} \\ \vdots \\ -I_{t_n} \end{bmatrix} \tag{2}$$

In the equation, $(I_{x_1}, I_{y_1}) \ldots (I_{x_n}, I_{y_n})$ denote the positional gradients at $(x, y)$ coordinates of the $n$ pixels, while $I_{t_1} \ldots I_{t_n}$ represent the gradients of the pixels at time $t$.

Finally, the least square method is applied to find a solution for the following linear equation:

$$v = \left(\mathrm{A}^T\ \mathrm{A}\right)^{-1}\ \mathrm{A}^T\ \mathrm{b} \tag{3}$$

Unfortunately, the L-K method has **significant limitations.** First, the assumptions of the L-K method may not hold in reality. A pixel's gray scale value may change substantially in nearby frames. Image contents between consecutive frames may vary considerably if objects move fast. Thus, it is susceptible to large errors in the presence of noise or fast-moving objects [17], [18]. Second, the L-K method suffers from the depth compression effect, where the pixel motion velocity is incorrectly considered higher when objects are closer to the camera compared to when they are farther away, even if the actual speed is the same. Moreover, solving Equations 1, 2, and 3 for every pixel is prohibitively costly.

## IV. AROD: Adaptive Real-Time Object Detection

Figure 1 gives **an overview of the AROD framework**. As depicted in Figure 1, the positions of at most $K$ moving

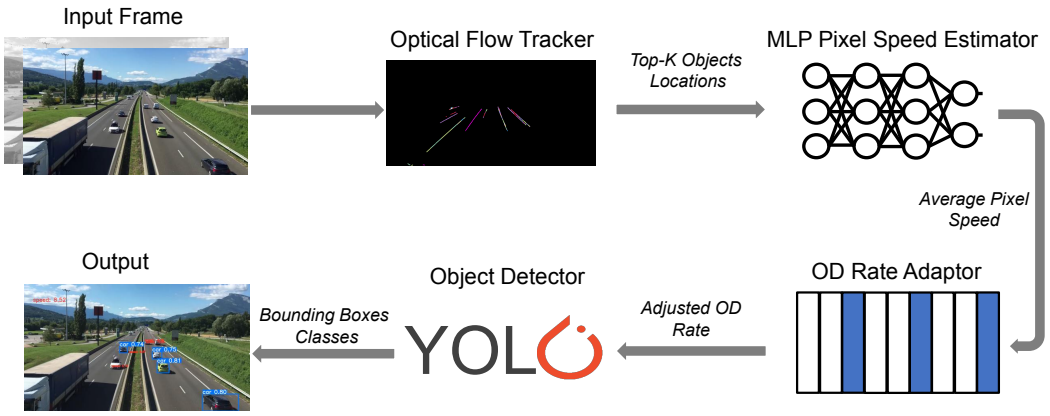


Fig. 1: Overall structure of AROD.

objects estimated via optical flow analysis are inputted to our MLP model to infer the average pixel motion speed, where $K$ is a predetermined maximum threshold for efficient pixel velocity inference. Given the inferred pixel motion speed, the rate adaptor in Figure 1 dynamically adjusts the rate of object detection and feeds traffic video frames into the adopted object detection model, such as a YOLO model, according to the adjusted rate. Then, the object detection model generates bounding boxes around all objects it detects, while classifying them as illustrated in Figure 1.

To address the limitations of optical flow analysis, we design a new lightweight MLP model that significantly reduces potential errors in pixel velocity inference. Furthermore, we substantially reduce computational costs for optical flow analysis as follows:

- Utilizing the Shi-Tomasi method [19], the optical flow tracker of AROD in Figure 1 identifies the corners of a maximum of $K$ moving objects (e.g., $K = 10$). If there are a total of $m$ objects in a frame, it tracks only one corner of each of the $n = \min(m, K)$ objects. By doing this, we limit the number of objects and decrease the number of pixels to track in consecutive frames.
- To further enhance the efficiency, our optical flow tracker tracks $n$ object corners using the L-K [16] method within the region of measurements (RoM) only, where an arbitrary object appears big enough for visual identification, as illustrated in Figure 2.

Given that, Algorithm 1 formally describes **how AROD works**. In line 1, we initialize the frame number $t = 1$. Lines 2-12 are applied to incoming video frames for efficient object detection. In line 3, we perform efficient optical flow analysis to estimate the positions of $n$ objects (i.e., their corners) in the RoM of frame $t$ as discussed above.

If $t > 1$, AROD executes lines 4-12. In lines 5-6, the positions of $n$ objects in frames $t-1$ and $t$ are stored in

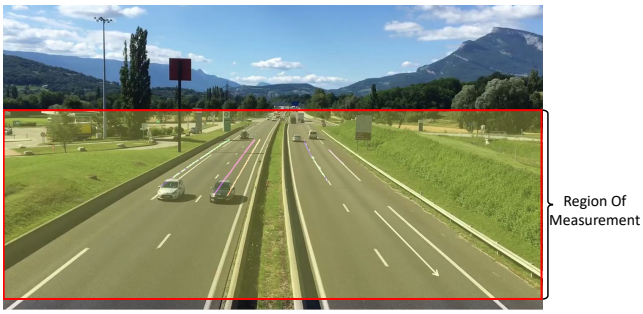


Fig. 2: Region of measurement for optical flow analysis

vectors $L(t-1)$ and $L(t)$, respectively:

$$L(t-1) = [(x_1(t-1), y_1(t-1)) \dots (x_n(t-1), y_n(t-1))]$$
$$L(t) = [(x_1(t), y_1(t)) \dots (x_n(t), y_n(t))] \quad (4)$$

In line 6, we concatenate $L(t-1)$ and $L(t)$ into a vector, $\ell(t)$. In line 7, given $n$ and $\ell(t)$, the MLP model of AROD infers the average pixel motion speed of the $n$ objects, $\hat{v}(t)$. The design and training of the MLP model for efficient, high-accuracy inference of average pixel velocity is described in Section V.

In line 8, we dynamically adapt the object detection rate, $\gamma(t)$, according to $\hat{v}(t)$. In this paper, we assume that the input frame rate $\gamma_c$ from the camera, e.g., 30 fps, is constant. Given that, $\gamma(t) \leq \gamma_c, \forall t$; that is, *the current object detection rate is less than or equal to the input frame rate, depending on the pixel motion speed of objects*. In particular, in line 8, the object detection rate is adapted using a set of prespecified pixel velocity thresholds: $\tau_1, \dots, \tau_{max}$ (pixels per frame), where $\tau_1 < \dots < \tau_{max}$. Using them, we dynamically adapt the object detection rate, $\gamma(t)$, according to $\hat{v}(t)$:

$$\gamma(t) = \begin{cases} \frac{1}{\phi(1)}\gamma_c & if\ \hat{v}(t) < \tau_1 \\ \frac{1}{\phi(j+1)}\gamma_c & if\ \tau_j \leq \hat{v}(t) < \tau_{j+1}\ (1 \leq j < max) \\ \gamma_c & if\ \hat{v}(t) \geq \tau_{max} \end{cases} \quad (5)$$

**Algorithm 1:** Adaptive Real-Time Object Detection

**input :** Incoming video frames
**output:** Object detection results

1 $t$ = 1
2 **while** ***true*** **do**
    // optical flow analysis
3 $L(t)$ = OFA(*frame t*)
4 **if** $t > 1$ **then**
        // $n$ objects in frames $t-1$ and $t$
5 $n = \min(m, K)$
6 $\ell(t)$ = $L(t-1) + L(t)$
        // infer pixel motion speed
7 $\hat{v}(t)$ = MLP($n, \ell(t)$)
        // adapt obj. detection rate
8 $\gamma(t)$ = adapt($\hat{v}(t)$)
        // Do obj. detection at rate $\gamma(t)$
9 results = OD($\gamma(t)$)
        // Bounding boxes & obj. classes
10 display(results)
11 **end**
12 $t$++;
13 **end**

where $\phi(1) > \ldots > \phi(max) > 1$ to use a lower object detection rate for a lower pixel motion speed, $\hat{v}(t)$. If $\hat{v}(t) \geq \tau_{max}$, object detection is set to $\gamma_c$ and therefore AROD performs object detection for every input frame. Otherwise, object detection rate is reduced according to the pixel motion speed using Equation 5. For example, if $\phi(j+1) = 2$, AROD skips every other frame when $\tau_j \leq \hat{v}(t) < \tau_{j+1}$.

In lines 9-10, we perform object detection at the rate of $\gamma(t)$ and display the result. In line 12, we increment the frame number, and repeat lines 2-12 for incoming traffic video frames.

## V. MLP Model Design, Datasets, and Training

### A. Objectives of Model Design

A main challenge for high-accuracy inference of pixel velocity is how to minimize the risk of overfitting and improve generalizability with little computational overhead. To face the challenge, we take systematic approaches to design, train, and evaluate the MLP model of AROD:

- **Per-Weight Regularization:** During training, we utilize an effective optimization algorithm that significantly reduces overfitting via per-weight regularization [20].
- **Early Stopping:** To avoid overfitting, we analyze training and validation losses over training epochs and terminate training early as soon as both losses converge to small values and stabilize.
- **MLP Architecture Search:** In an iterative feedback loop, we vary the depth and width of MLP models and analyze their losses. Thereby, we find an MLP model with as low complexity as possible that converges to minimal validation and training losses, reducing the risk of overfitting due to a more complex structure.
- **Generalizability Analysis using a Different Dataset:** We also verify the generalizability of AROD to a traffic video entirely unseen during training (in Section VI).

### B. Datasets

In this paper, four HD (1280×720 pixels) traffic monitoring videos are used for evaluation. Video 1 [21], Video 2 [22], and Video 3 [23] were captured at different locations in bright sunlight, under a cloudy condition, and at night, respectively. Moreover, Video 4 [24], annotated with ground-truth bounding boxes, is set aside for assessing the risk of overfitting and the generalization capability of AROD. None of the frames in Video 4 are utilized in training our MLP model. If the MLP model becomes overfitted and fails to generalize to a new video, it could result in a degradation of the mAP (mean Average Precision) by AROD.

In this paper, an object (e.g., a vehicle) in a traffic video frame is one sample used for training. Videos 1, 2, and 3 have 2,833,699 samples in total. We split the videos so that the samples in the first 30% of the videos are reserved for testing, while the samples in the remaining 70% of the videos are used for training and validation. (The 70% of the videos is further divided into training and validation datasets with the split ratio of 0.8:0.2.)

### C. MLP Model Search and Training

For the sake of training only, we track a large number of possible vehicle corners (e.g., $K = 200$). To reliably derive the reference speed, we manually identify vehicle corners that pass through the entire RoM (Figure 2), while rejecting noisy ones that fail to go through the RoM. Subsequently, we analyze the position of the identified vehicle $i$ over $w$ successive frames (e.g., $w$=120 frames for 4 seconds at 30 fps) using the Lucas-Kanade [16] and Shi-Tomasi [19] methods (Section III). Based on that, we derive the *reference pixel velocity* of an arbitrary vehicle $i$, $v(i)$, for all $i$.

To train the MLP model, we use the following loss function, where $n$ denotes the number of samples in each mini-batch and $\hat{v}(i;\theta)$ is the *inferred pixel speed* of the vehicle $i$ using the MLP with the current set of weights $\theta$:

$$Loss = \frac{1}{n}\sum_{i=1}^{n}(\hat{v}(i;\theta) - v(i))^2 \tag{6}$$

To minimize the loss, we apply the Adam with weight decay (AdamW) optimizer [20]. AdamW improves upon Adam [25] by decoupling weight decay from gradient updates, supporting per-weight regularization. In this way, it avoids overfitting and enhances generalizability compared to Adam. In Equation 7, at the $j^{th}$ iteration, each weight is updated based on its gradient mean (moving average), $m_j$, and squared gradient (uncentered variance), $\sigma_j^2$. In the equation, $\alpha$ is the learning rate, $\epsilon$ is a

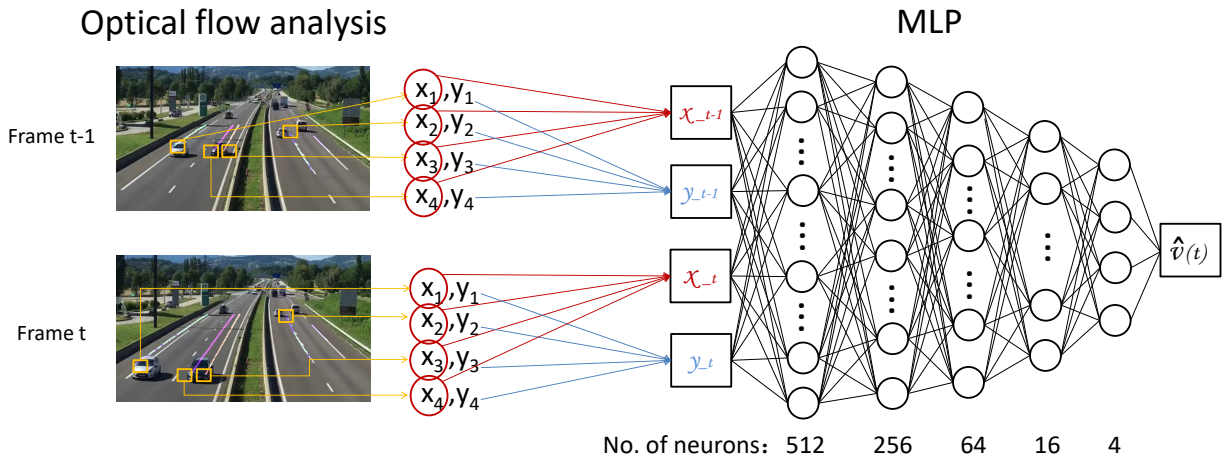


Fig. 3: Pixel motion speed inference for $n$ objects via MLP-based regression

small constant used to prevent division by zero, and $\lambda$ is the weight decay factor used to avoid overfitting via regularization:

$$\theta_{j+1} = \theta_j - \alpha \cdot \frac{m_j}{\sqrt{\sigma_j^2 + \epsilon}} - \lambda \cdot \theta_j \tag{7}$$

Thus, AdamW updates a weight by a smaller step size, if its squared gradient is large or vice versa. In this paper, we set $\alpha = 0.001$ and $\lambda = 0.0005$.

Based on this, we have analyzed various MLP model architectures with different depths and widths, finding the cost-effective MLP model in Figure 3 that minimizes the loss efficiently. As depicted in the figure, our MLP model has 5 hidden layers with 512, 256, 64, 16, and 4 neurons, respectively. The final layer in Figure 3 has a single neuron that produce the average pixel velocity of $n$ objects, $\hat{v}(t)$. As illustrated in Figure 4, both the validation and training loss curves of the MLP model exhibit a downward trend until they plateau in the vicinity of the 20th epoch. The figure demonstrates a favorable pattern of consistently decreasing losses that swiftly stabilize. Hence, to avoid overfitting, we terminate the training at epoch 20.

In addition, we empirically tune several parameters in Algorithm 1 to optimize inference accuracy with minimal complexity. For efficient inference, we set $K$=10 to track at most 10 moving objects, e.g., vehicles, with the highest confidence. The corner size is set to $15 \times 15$ pixels to track fewer pixels than the entire pixels of a moving object. Finally, in Equation 5, we set $\tau_{max} = \tau_6 = 10$ pixels/frame and $\tau_1 = 1.7$ pixels/frame, configuring $\phi(1) > \ldots > \phi(6)$.

### D. Hardware Platform and Implementation

We assess AROD on a personal computer configured to mimic a relatively inexpensive video analytics server, in contrast to a high-end cloud server, for performing real-time object detection near data sources (cameras) [1]. It consists of commodity hardware components: an Intel® Core i7-7820X CPU, 64 GB of RAM, and an NVIDIA GeForce GTX 3080Ti GPU. The operating system is Ubuntu 18.04.6 LTS. All methods are implemented using Python 3.10, PyTorch 2.0.1, Scikit-learn 1.0.2, and OpenCV-python 4.7.0. For evaluation, we employ three pretrained one-stage object detection models: EfficientDet [6], SSD [5], and YOLOv5 [26], which provide superior fps and similar detection accuracy, compared to two-stage object detection models, such as [2], [3], [9].

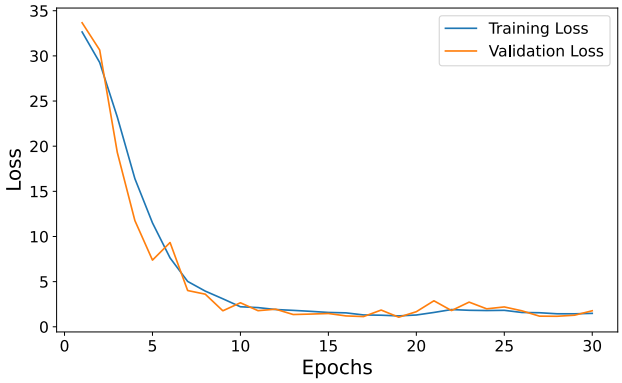


Fig. 4: Training and Validation Losses of MLP for Training Epochs

## VI. Evaluation

In Subsections VI-A through VI-D, we describe the evaluation results acquired using the test sets of Videos 1, 2, and 3 unseen during the training. Furthermore, in Subsection VI-E, we assess the mAP and generalizability of AROD using entire Video 4 completely unseen during the training.

### A. Pixel Speed Inference Errors

We compare the RMSE of our MLP model to those of the Lucas-Kanade method [16] and five other SOTA machine learning methods in Table I. Our model provides the **most robust inference** in terms of RMSE. It achieves the smallest

RMSE for Video 1 and the second smallest RMSE values for the other videos. For Videos 2 and 3, polynomial linear regression achieves the smallest RMSE; however, its RMSE for Video 1 is over 6.25× the RMSE of our model. In contrast to the L-K method, our model exhibits an RMSE that is one or two orders of magnitude smaller for all three videos.

TABLE I: RMSEs (pixels/frame) in pixel velocity inference (**boldface**: best; underlined: second best)

| Models | Video 1 | Video 2 | Video 3 |
|---|---|---|---|
| L-K method | 79.4 | 25.7 | 26.6 |
| Linear Regression | 4.02 | 1.37 | 1.86 |
| Polynomial LR | 5.38 | **1.24** | **0.89** |
| ElasticNet | 4.02 | 2.17 | 2.18 |
| XGBoost | 0.91 | 1.36 | 1.45 |
| CatBoost | 1.03 | 1.61 | 1.49 |
| MLP of AROD | **0.86** | 1.26 | 1.26 |

### B. Model Size, GFLOP, and Latency Comparisons

In Table II, the **model size, GFLOP, and latency of AROD are one or more orders-of-magnitude smaller** than those of the object detection models due to the lightweight design. Also, the **latency of AROD is concealed** via effective CPU-GPU pipelining: AROD runs in the CPU and its latency is significantly shorter than that of object detection simultaneously performed in the GPU, as shown in Table II.

TABLE II: AROD vs. the three object detection models utilized for evaluation. AROD is executed in the CPU, but each object detection model is executed in the GPU.

| Models | Parameters | GFLOP | Latency |
|---|---|---|---|
| YOLOv5 | 7.2 M | 7.7 | 21.1 ms |
| SSD300 | 22.9 M | 18.9 | 45.2 ms |
| EfficientDet-B0 | 3.9 M | 2.5 | 114.4 ms |
| AROD | **0.6** M | **0.002** | **3.8 ms** |

### C. Frame Processing Rate for A Traffic Video Stream

As illustrated in Table III, **AROD improves the fps by approximately 32-61%, 110-174%, and 120-213%** for YOLOv5, SSD300, and EfficientDet-B0, respectively. AROD significantly outperforms ERD [12] and L-filter [13] that enhance the fps of the three object detection models by 10-44% and 31-47%, respectively, on the same hardware platform. This is because, unlike ERD [12], L-filter [13], and the methods based on structural similarity analysis [10], [11] (discussed in Section II), the fps of AROD is at least as high as the fps of the standalone models without AROD, regardless of the road occupancy status or structural similarity between frames, due to efficient CPU-GPU pipelining.[1]

In Table III, AROD+object detection achieves the lowest fps for Video 1 and highest fps for Video 2, because Video 1 and Video 2 exhibit the highest and lowest pixel velocity, respectively. However, the fps values of the object detection models without AROD in the "Baseline" column do not noticeably change across videos, since they are not adaptive to the pixel motion speed.

[1]ERD [12] and L-filter [13] are much faster than structural similarity analysis used in [10], [11]. Detailed results are omitted due to space limitations.

TABLE III: Average fps of the baseline object detection models without AROD in the 2nd column and AROD+object detection models in the 3rd though 5th columns.

| Models | Baseline | Video 1 | Video 2 | Video 3 |
|---|---|---|---|---|
| YOLOv5 | 46.1 | **61.1** | **74.3** | **70.6** |
| SSD300 | 11.5 | 24.2 | 31.5 | 27.1 |
| EfficientDet-B0 | 8.7 | 19.1 | 27.2 | 26.2 |

### D. Scalability of AROD for Concurrent Streams

In this subsection, we assess the scalability of AROD when it works alongside YOLOv5, which is the fastest among the tested object detection models. As shown in Table IV, AROD+YOLOv5 supports approximately **30-34 fps per stream, when it processes three concurrent streams**. This represents a significant scalability advancement, particularly given the high resource demands of real-time object detection. For instance, even a high-end NVIDIA V100 GPU can only process two 30 fps video streams for object detection using YOLOv5 [27].

TABLE IV: Per-stream fps of AROD+YOLOv5 when one or more streams are processed concurrently

| Number of Streams | 1 | 2 | 3 |
|---|---|---|---|
| Video 1 | 61.1 | 41.5 | 30.2 |
| Video 2 | **74.3** | **49.7** | **34.2** |
| Video 3 | 70.6 | 48.5 | 31.6 |

### E. Generalizability

For Video 4, AROD+YOLOv5 in Table V achieves comparable mAP to YOLOv5 with respect to different IoU (Intersection over Union) thresholds used to measure the mAP. This reveals that our MLP model is **not overfitted and effectively generalizes** to Video 4 totally unseen during training. Moreover, AROD+YOLOv5 produces **good visual results**, as captured in Figure 5.

TABLE V: mAP of object detection for different IoU thresholds

| IoU threshold | 0.1 | 0.2 | 0.5 |
|---|---|---|---|
| YOLOv5 | 0.443 | 0.442 | 0.401 |
| AROD+YOLOv5 | **0.445** | **0.444** | **0.406** |

## VII. Conclusions and Future Work

Efficient real-time object detection is essential for AI-based intelligent traffic management. However, increasing complexities of deep learning models for object detection result in significant latency and resource requirements. In this paper, we introduce a new framework for adaptive object detection that dynamically reduces the execution rate of a plugged-in object detection model during periods of low pixel velocity.

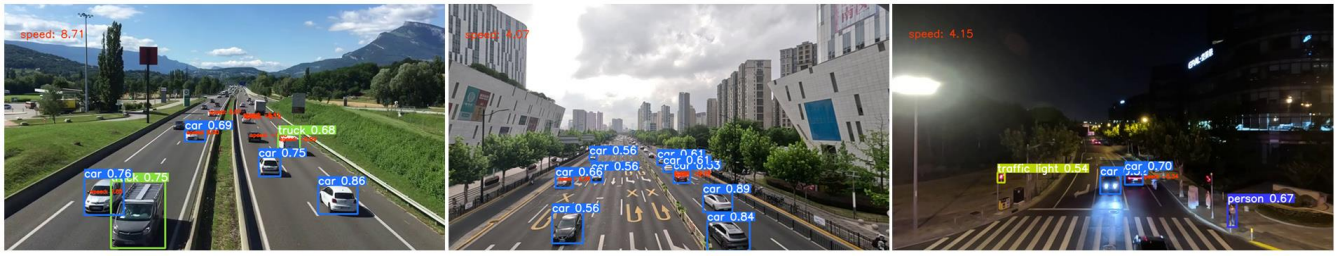


Fig. 5: Visualization: AROD+YOLOv5 effectively detects moving objects under different lighting conditions.

We rigorously design and train a new lightweight MLP model to infer pixel velocity with high accuracy, avoiding overfitting. Moreover, AROD hides its latency via efficient CPU-GPU pipelining. Our evaluation verifies the cost-effectiveness of AROD: 1) AROD supports robust pixel velocity inference; 2) it enhances the fps of YOLOv5, SSD, and EfficientDet by approximately 32-61%, 110-174%, and 120-213%, respectively; and 3) AROD+YOLOv5 can concurrently process up to three traffic video streams, supporting at least 30 fps per stream without requiring expensive cloud GPUs. Moreover, it effectively generalizes to a separate traffic video set aside by providing the comparable mean average precision to that of YOLOv5. In the future, we will investigate more advanced approaches to further improve the efficiency and scalability of real-time object detection, sustaining accuracy. Moreover, we will explore how to raise the accuracy of real-time object detection without largely increasing latency and resource demands.

## ACKNOWLEDGEMENT

This work was partially supported by National Science Foundation grants CNS-2007854 and CNS-2326796.

## REFERENCES

[1] P. Arthurs, L. Gillam, P. Krause, N. Wang, K. Halder, and A. Mouzakitis, "A taxonomy and survey of edge cloud computing for intelligent transportation systems and connected vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6206–6221, 2021.

[2] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[4] "YOLOv5." https://github.com/ultralytics/yolov5/wiki [Online] Accessed in April 2024.

[5] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pp. 21–37, Springer, 2016.

[6] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10781–10790, 2020.

[7] H. Yin, A. Vahdat, J. M. Alvarez, A. Mallya, J. Kautz, and P. Molchanov, "AdaViT: Adaptive tokens for efficient vision transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10809–10818, 2022.

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[10] U. Ahmed, J. C.-W. Lin, and G. Srivastava, "Multi-aspect detection and classification with multi-feed dynamic frame skipping in vehicle of internet things," *Wireless Networks*, pp. 1–12, 2022.

[11] U. Ahmed, J. C.-W. Lin, and G. Srivastava, "Efficient Multimedia Frame-Skipping Architecture Using Deep Learning in Vehicular Networks," *IEEE MultiMedia*, vol. 29, no. 2, pp. 66–73, 2022.

[12] Y. Liu and K.-D. Kang, "Preprocessing via Deep Learning for Enhancing Real-Time Performance of Object Detection," in *IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*, 2023.

[13] Y. Liu and K.-D. Kang, "Filtering empty video frames for efficient real-time object detection," *Sensors*, vol. 24, no. 10, 2024.

[14] A. Taheri Tajar, A. Ramazani, and M. Mansoorizadeh, "A lightweight Tiny-YOLOv3 vehicle detection approach," *Journal of Real-Time Image Processing*, vol. 18, no. 6, pp. 2389–2401, 2021.

[15] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[16] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI'81: 7th international joint conference on Artificial intelligence*, vol. 2, pp. 674–679, 1981.

[17] J. Lee, C.-J. Park, and I.-H. Lee, "An arbitrary point tracking using multi-scale refined optical flow," in *The 9th International Conference on Advanced Communication Technology*, vol. 1, pp. 373–377, IEEE, 2007.

[18] C. Pan, D. Xue, Y. Xu, J. Wang, and R. Wei, "Evaluating the accuracy performance of Lucas-Kanade algorithm in the circumstance of PIV application," *Science China Physics, Mechanics & Astronomy*, vol. 58, pp. 1–16, 2015.

[19] J. Shi *et al.*, "Good features to track," in *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, pp. 593–600, IEEE, 1994.

[20] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019.

[21] V. Kiraly, "Relaxing highway traffic." https://www.youtube.com/watch?v=nt3D26lrkho, 2017. [Online] Accessed in April 2024.

[22] R. Liu, "A nighttime traffic video of Guangfuxi Road, Shanghai, China for object detection and tracking." https://youtu.be/ZI-tcEbklks, 2023. [Online] Accessed in April 2024.

[23] R. Liu, "A traffic video of Jinshajiang Road, Shanghai, China." https://youtu.be/KD_9g1FIJYc, 2023. [Online] Accessed in April 2024.

[24] I. Sboukraa, "Car Object Detection in Road Traffic." https://www.kaggle.com/datasets/boukraailyesali/traffic-road-object-detection-dataset-using-yolo, 2023. [Online] Accessed in April 2024.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[26] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

[27] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," 2021.