IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

*Digital Object Identifier 10.1109/ACCESS.2024.0429000*

# Improving Token-based Object Detection with Video

**ABHINEET SINGH[1], and NILANJAN RAY[2]**
[1]Department of Computing Science, University of Alberta (e-mail: asingh1@ualberta.ca)
[2]Department of Computing Science, University of Alberta (e-mail: nray1@ualberta.ca)

Corresponding author: Abhineet Singh (e-mail: asingh1@ualberta.ca).

**ABSTRACT** This paper improves upon the Pix2Seq object detector [1] by extending it for videos. In the process, it introduces a new way to perform end-to-end video object detection that improves upon existing video detectors in two key ways. First, by representing objects as variable-length sequences of discrete tokens, we can succinctly represent widely varying numbers of video objects, with diverse shapes and locations, without having to inject any localization cues in the training process. This eliminates the need to sample the space of all possible boxes that constrains conventional detectors and thus solves the dual problems of loss sparsity during training and heuristics-based postprocessing during inference. Second, it conceptualizes and outputs the video objects as fully integrated and indivisible 3D boxes or tracklets instead of generating image-specific 2D boxes and linking these boxes together to construct the video object, as done in most conventional detectors. This allows it to scale effortlessly with available computational resources by simply increasing the length of the video subsequence that the network takes as input, even generalizing to multi-object tracking if the subsequence can span the entire video. We compare our video detector with the baseline Pix2Seq static detector on several datasets and demonstrate consistent improvement, although with strong signs of being bottlenecked by our limited computational resources. We also compare it with several video detectors on UA-DETRAC to show that it is competitive with the current state of the art even with the computational bottleneck. We make our code and models publicly available [2].

**INDEX TERMS** object detection, video object detection, autoregression, language modeling, tokenization, transformer

## I. INTRODUCTION

DEEP learning based object detection models conventionally produce continuous-valued and fixed-sized outputs, i.e. their output consists of real numbers rather than discrete integers and has an architecturally-determined size that is independent of the contents of the input image. This is true of models based on both convolutional neural networks (CNNs) [3] and transformers [4]–[6]. This is not well suited to tasks like object detection and multi-object tracking (MOT) where the outputs are inherently discrete and variable-sized in nature. In both cases, the output size depends on the input and is highly variable across inputs since the number of objects in an image can vary widely between images. In addition, the space of all possible objects is not only *much* larger than the number of objects that might realistically be present in an image but it is also far too large to be sampled densely by any fixed-sized representation. This problem is greatly compounded in case of video detection and MOT since the space of all possible objects or trajectories in a video increases exponentially with the length of the video while the actual number of such instances remains the same as in a single image. Any fixed-sized modeling of this output space must therefore resort to not only sampling this space sparsely but employing an output size that is much larger than the actual number of output instances that the model needs to produce. An important example of this problem is provided by anchor boxes or similar methods of sampling the space of possible boxes, that are used in both RCNN [7]–[12] and YOLO [13]–[21] families of detectors.

This in turn leads to two main problems. Firstly, it causes the loss function to become very sparse since the training signal only comes from objects that are actually present in the images which constitute a tiny fraction of the space of all possible objects that the network output, and therefore the loss function, needs to represent. As a consequence, a vast majority of this output corresponds to the background and the resultant class imbalance needs to be handled through complex loss engineering. Examples include careful domain-

specific anchor box design, one-to-many ground truth (GT) to anchor box associations [8], focal loss [10] and hard example mining [22]. Secondly, we need to perform heuristics-based postprocessing like confidence thresholding and non-maximum suppression on the raw network output to extract the small number of discrete outputs that we actually need for downstream processing. This introduces a disconnect between what the network learns from the training data and how it actually performs during inference. In some cases, especially with MOT, this postprocessing can in fact have a greater impact on the overall performance than the trained model itself [23], [24]. Further, when the model is used as one component of a larger system that is otherwise fully differentiable, the presence of these non-differentiable heuristics makes it difficult to train the entire system end-to-end.

These issues can be largely resolved by modeling the output space with a discrete variable-length representation. There has in fact been a trend towards such discretization in the vision literature over the last few years, whereby the outputs are represented by sequences of discrete tokens that are produced one at a time by autoregression [4]. Pix2Seq [25] is a popular language-modeling framework for computer vision that was originally introduced for object detection [1] and then extended with a multi-task version [26] that added support for instance segmentation, keypoint detection and image captioning. There have also been follow-up works that replace autoregression with diffusion, first for image generation and captioning [27], and then for panoptic segmentation [28]. Other examples of tokenization in vision include SeqTrack [29] for single-object tracking, MMTrack [30] for vision-language tracking, PolyFormer [31] and SeqTR [32] for referring image segmentation, and UniTab [33] for grounded image captioning. There have also been related works that discretize the output space but use set prediction [34], [35] similar to instead of language modeling. Two examples are Point2Seq [36] for 3D object detection, and Obj2Seq [37] for pose estimation, both of which are inspired by DETR [38], [39]. Finally, there have also been attempts to learn the tokenization from data instead of designing it manually, including UViM [40] and AiT [41], both of which are multi-task models that support depth estimation and instance segmentation, among other tasks. This trend seems to have been inspired by the remarkable success of transformer-based natural language processing (NLP) models [42] in recent years, especially large language models (LLMs) [43], [44] like Chat-GPT [45].

This paper proposes another step in this direction by tokenizing video object detection and potentially also MOT when future computational resources allow the video subsequence to become long enough to incorporate entire trajectories. This single-step approach to video detection also offers an adavantage, albeit largely theoretical for now, over many existing video detectors that conceptualize video detection as a two-step problem. These detectors first run a static detector on each individual frame and then aggregate these frame-specific outputs to produce their video-level output. For example, VSTAM [46], PTSEFormer [47], TransVOD [48], ClipVID [49], and TGBFormer [50] use DETR [38], [39] as their base detector, while MEGA [51], MST-FA [52], GMLCN [53], DAFA [54], and CFA-Net [55] have Faster-RCNN [8], SparseVOD [56] and QueryProp [57] have Sparse RCNN [8], [12], [58], REPP [59] and YOLOV [60], [61] have some version of YOLO [15], [62], and SALISA [63] has EfficientDet [64] as the base detector. This two-step approach restricts most of these detectors to producing objects that span a predetermined and very small number of frames (usually only two) and there is no easy way to increase this number.

While our proposed detector can also be seen as an extension of the static Pix2Seq detector [1], it does not build upon frame-level outputs from the latter; instead, it conceptualizes and outputs the video objects as indivisible 3D boxes or tracklets that are produced by consolidating global video-level information without involving any frame-specific components. This allows our detector to scale up to arbitrary number of frames as long as sufficient computational resources are available. In theory, it can even generalize to MOT if the number of frames can be made large enough to span entire trajectories.

We have implemented the video detector as a part of the Pix2Seq framework [25] and make our code and trained models publicly available [2] to ensure wide compatibility and ease of reproducibility. Note that many of the tokenization concepts in this paper can be better illustrated with animations rather than still images. Since it is not possible to include animations here, we have created a website for this project [65] which contains the videos corresponding to many of the images included here. The links to the actual videos are included in the caption to each such image.

The remainder of this paper is organized as follows. Sec. II first describes the tokenization strategy in Pix2Seq static detector (Sec. II-A), followed by its proposed extension for video detection (Sec. II-B). Sec. III provides details of the network architectures, again starting with the static image input (Sec. III-A) and followed by its adaptation for video inputs (Sec. III-B). Sec. IV details the evaluation process including datasets (Sec. IV-A), metrics (Sec. IV-B), training IV-C and quantitative results (Sec. IV-D) comparing the proposed detector with the static version as well as several state of the art video detectors from literature. Finally, Sec. VI concludes with a summary of the results and ideas for future improvements.

## II. TOKENIZATION

### A. STATIC OBJECT DETECTION

Pix2Seq static detector represents each object by five tokens - four for the bounding box corner coordinates and one for the class: $ty, lx, by, rx, cls$. Here, $(lx, ty)$ and $(rx, by)$ are respectively the top left and bottom right bounding box corners coordinates, while $cls$ is the class token. These coordinates are in image-space which is discretized into bins. The number of such bins $H$ can be either greater than the image resolution to achieve sub-pixel accuracy or less than it to reduce the

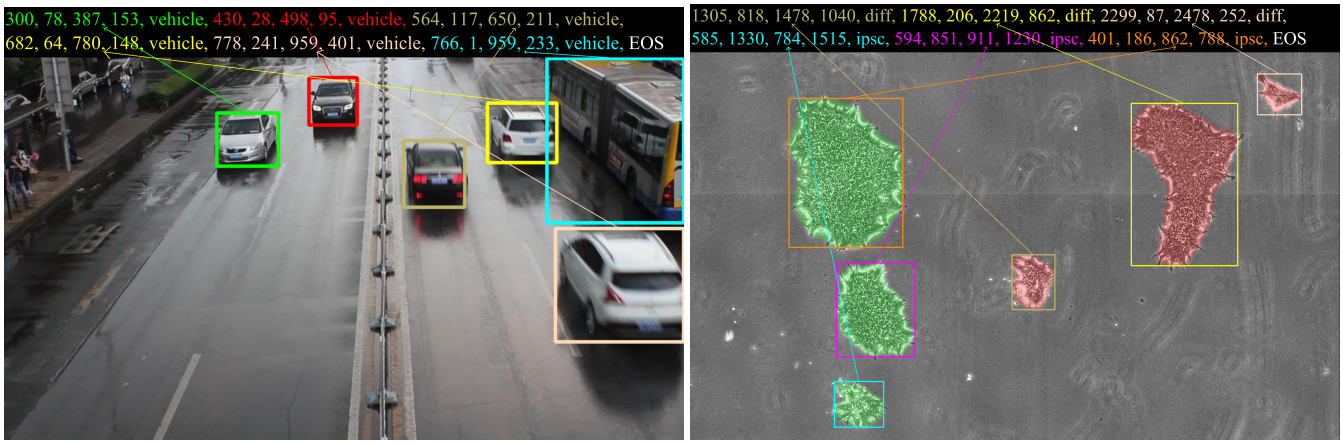


FIGURE 1: Visualization of Pix2Seq object tokenization on images from (left) UA-DETRAC [66] and (right) IPSC [67] datasets. UA-DETRAC has a single class, represented here by the *vehicle* token, while IPSC has two classes of cells, shown here with green and red masks and represented with *ipsc* and *diff* tokens respectively. Animated versions of these images are available here and here respectively.

vocabulary size. Both $x$ and $y$ coordinates are represented by the same shared set of tokens so that the total number of tokens in the vocabulary $V$ is given by $V = H + C + r$, where $C$ is the number of classes and $r$ is number of reserved tokens (e.g. EOS and padding token). Pix2Seq static detector uses $H = 2K$ and $V = 3K$ by default, although the framework supports $V$ upto $32K$ out-of-the-box and we have successfully trained models with $V$ upto $68K$, which seems to be limited only by the available computational resources.

The tokens for all the objects in the image are produced sequentially, so that the total number of output tokens = $5 \times n + 1$, where $n$ is the number of objects in the image and the extra token at the end is the EOS token that marks the end of all objects. The order of objects is not defined and is randomized each time the same image is shown to the network during training so the network is able to learn an order-agnostic representation of the objects. Fig. 1 shows examples of static detection tokenization for both single- and multi-class cases.

### B. VIDEO OBJECT DETECTION

We adapted this tokenization strategy for video detection by considering all the bounding boxes of an object in an $N$-frame temporal window as a single 3D polygon or tubelet [68] that can be represented by a sequence of $N$ 4-tuples, one for each bounding box, followed by the class token. Therefore, each object can be represented by $4 \times N + 1$ tokens and all $n$ objects by $n \times (4 \times N + 1) + 1$ tokens. For example, with $N = 2$, an object can be represented with 9 tokens:

$ty_1, lx_1, by_1, rx_1, ty_2, lx_2, by_2, rx_2, cls$

Here, $(lx_i, ty_i)$ and $(rx_i, by_i)$ are respectively the top left and bottom right bounding box corner coordinates in the $i^{th}$ frame $F_i$, $1 \leq i \leq N$. Fig. 2 shows two examples of video detection tokenization.

#### 1) *NA* Token

It is possible that an object is not present in each of the $N$ frames in the temporal window. We account for this possibility by adding a special *NA* token to the vocabulary to denote non-existence. Missing objects happen because an object either enters or leaves the scene in the middle of the temporal window, or it becomes briefly occluded in the middle of the window. Following are examples of all three cases:

- $N = 3$, object leaves the scene in the third frame:
  $ty_1, lx_1, by_1, rx_1, ty_2, lx_2, by_2, rx_2, NA, NA, NA, NA, cls$
- $N = 4$, object enters the scene in the third frame:
  $NA, NA, NA, NA, NA, NA, NA, NA, ty_3, lx_3, by_3, rx_3,$
  $ty_4, lx_4, by_4, rx_4, cls$
- $N = 5$, object occluded in the third and fourth frames:
  $ty_1, lx_1, by_1, rx_1, ty_2, lx_2, by_2, rx_2, NA, NA, NA, NA,$
  $NA, NA, NA, NA, ty_5, tx_5, by_5, rx_5, cls$

*NA* tokens become more common as $N$ increases, especially in scenarios where the camera field of view is limited and objects are fast-moving. An animated example from UA-DETRAC with $N = 6$ is available here.

#### 2) Limits on *N*

In theory, $N$ could be large enough to cover the entire video, in which case video detection progresses into MOT. In practice, however, it is severely limited by the amount of available RAM in the graphics processing units (GPUs) that are necessary to accelerate the training to make it time-feasible. Using the smallest ResNet-50 based Pix2Seq architecture (Sec. III-A) and without freezing any layers, $N = 16$ is the maximum that can be trained on an RTX 3090 or 4090 GPU with 24 GB RAM, which is the maximum available on a consumer-grade GPU as of this writing. It should be possible to get close to $N = 50$ on a Tesla A100 with 80 GB RAM, which is the maximum available on a workstation GPU. With the backbone frozen, it is possible to go as high as $N = 64$ on 24 GB RAM, though this limits the batch size to 1 per GPU or

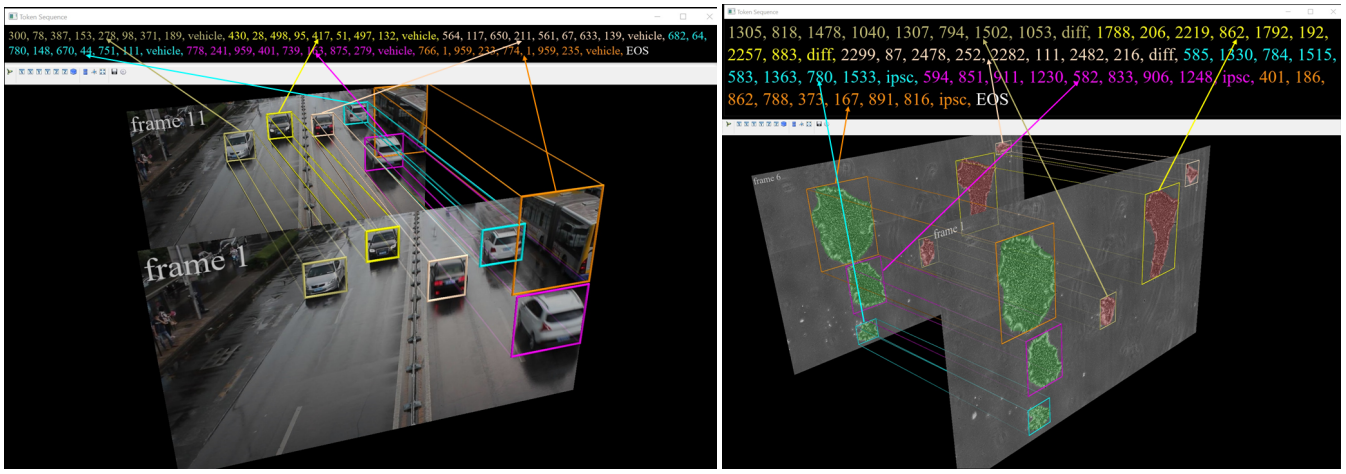


FIGURE 2: Visualization of video object tokenization on video clips from (left) UA-DETRAC and (right) IPSC datasets, both with $N = 2$. UA-DETRAC shows $G = 10$ while IPSC shows $G = 5$. Animated versions of these images are available here and here respectively.

total 6 over the 3 dual-GPU servers available to us (Table 4). This is far too small to be able to learn something as complex as the the set of all objects in a 64-frame temporal window.

We have successfully trained models upto $N = 32$, although performance drops sharply beyond $N = 8$ (Sec. V-D), indicating that such models cannot be successfully trained on our existing hardware. This is very likely due to the above problem of too small batch sizes, exacerbated by the frozen backbone and the relatively small number of videos in the datasets (Table 1). Another constraint on how large $N$ can be made in practice is the the number of training iterations required for the model to reach convergence, which increases rapidly with $N$. Finally, the maximum sequence length $L$, that must be $\geq$ number of tokens required to represent all the objects in each temporal window, imposes another limit on $N$. For example, with $N = 64$, we need $64 \times 4 + 1 = 257$ tokens to represent a single object and therefore several thousand tokens for tens of objects which are not unusual in many real-world scenarios. Pix2Seq has a default limit of $L = 512$ on the maximum number of tokens that can be produced by the network. We have managed to increase this to as high as $L = 4096$ but any increase in $L$ causes both GPU RAM consumption and training time to rise quickly, even if $N$ itself is not changed.

### 3) 1D Coordinate Tokens

The problem of $L$ increasing rapidly with $N$ can be partially ameliorated by flattening the image-space and using the corresponding 1D coordinate tokens instead of the standard 2D $(x, y)$ tokens. Each bounding box can then be represented by only 2 tokens instead of 4 and each object by $2 \times N + 1$ tokens, thus reducing $L$ nearly by half. For example, with $N = 3$, an object that leaves the scene in the third frame is tokenized as:
$tl_1, br_1, tl_2, br_2, NA, NA, cls$
Here, $tl_i$ and $br_i$ are respectively the flattened top left and bottom right corner coordinates of the bounding box in $F_i$.

However, using 1D coordinates greatly increases $V$ since now we need a total of $H^2$ different coordinates tokens instead of $H$ and therefore $V = H^2 + C + r$. This severely limits the number of coordinate bins $H$ that we can use, which in turn significantly reduces the localization accuracy of the detector. Also, while increasing $V$ does not affect the GPU memory consumption and training time to the same extent as $L$, it does cause both to increase too, thereby limiting the batch size we can use and the number of epochs we can train for. We did perform some experiments with $H = 160$ and $H = 256$, using respective vocabulary sizes of $V = 28K$ and $V = 68K$ (Sec. V-E), but found the models to be training far too slowly to be practicable. It appears that the 1D tokenization is too much of a change from the standard 2D version used in the pretrained weights for them to be fine-tuned successfully with the limited data and computational resources available to us.

### 4) Temporal Windows

Since we cannot process the entire video at once, we divide it into $N$-frame temporal windows which are processed one at a time in a sliding window manner. There are two hyper-parameters we can adjust to construct more varied temporal windows, both as a form of training data augmentation and to improve inference results.

The first parameter is the temporal stride $T$ which is the number of frames that separate two consecutive windows. We can use a stride $T < N$ to introduce redundancy during inference which can help to deal with false negatives. For example, with $N = 3$ and $T = 1$, we get frames 1, 2, 3 in the first temporal window; 2, 3, 4 in the second one; 3, 4, 5 in the third one and so on:
$(F_1, F_2, F_3)$, $(F_2, F_3, F_4)$, $(F_3, F_4, F_5)$, $(F_4, F_5, F_6)$, ...
As a result, we have detection outputs for $F_2$ from two different temporal windows while the outputs for $F_3$ onwards come from three different windows. For any $N$ in general, $T = 1$ provides $N$-way redundancy for all but the first $N - 1$ frames

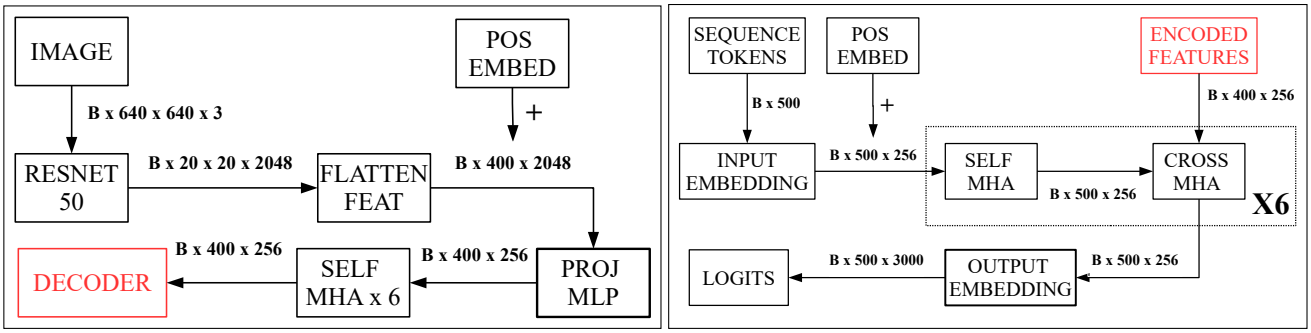


FIGURE 3: Flow diagram representing the high-level processing in the Pix2Seq encoder on the left and decoder on the right. Please refer Sec. III-A1 and III-A2 for details of individual modules.

in the video. These redundant outputs can be combined using non-maximum suppression to fill-in objects that might have been missed in some windows but not all of them.

The second parameter is the frame gap $G$ between successive frames in the same temporal window. We can make $G > 1$ as a form of training data augmentation to allow the network to learn to deal with faster inter-frame motions, which would be useful, for example, in handling dropped frames during live inference. Even for offline inference, $G > 1$ can provide us with even more overlapping temporal windows for each frame, thereby increasing the redundancy further. For example, with $N = 6, T = 3$ and $G = 2$, we get temporal windows: $(F_1, F_3, F_5, F_7, F_9, F_{11})$, $(F_4, F_6, F_8, F_{10}, F_{12}, F_{14})$,... Another application of $G > 1$ is to effectively utilize a much larger value of $N$ than can be fit in the available GPU RAM. This can be done by training the network to predict the complete tracklet, including portions that correspond to missing frames. For example, we can get temporal windows spanning 13 frames: $(F_1, F_7, F_{13})$, $(F_2, F_8, F_{14})$,... with $N = 3, T = 1$ and $G = 6$. Now, instead of only predicting portions of the tracklets corresponding to the 3 frames in the input:
$ty_1, lx_1, by_1, rx_1, ty_7, lx_7, by_7, rx_7, ty_{13}, lx_{13}, by_{13}, rx_{13}, cls$
we train the network to predict the entire 13-frame tracklet:
$ty_1, lx_1, by_1, rx_1, ty_2, lx_2, by_2, rx_2, ty_3, lx_3, by_3, rx_3, ...,$
$ty_{13}, lx_{13}, by_{13}, rx_{13}, cls$
This will require the network to be able to interpolate tracklets over the missing frames, but, as shown in Sec. V-C2, Pix2Seq seems pretty good at doing this. All our experiments in this paper have been restricted to $G = 1$ but $G > 1$ provides an interesting avenue for future exploration. We have likewise only used $T = 1$ for training, along with $T = 1$ and $T = N$ for inference, and leave experiments with other values of $T$ as future work.

## III. NETWORK ARCHTECTURES

### A. STATIC INPUT

Pix2Seq supports two backbone architectures, namely ResNet-50 [69] and VIT [5], each with three input sizes - $640 \times 640$, $1024 \times 1024$ and $1333 \times 1333$. We have used the smallest version - $640 \times 640$ ResNet-50 - for most of our experiments so as to maximize $N$ and the training batch size $B$. We did perform a few experiments using $1024 \times 1024$ and $1333 \times 1333$ ResNet-50 as well as $640 \times 640$ VIT but did not observe any significant performance improvements. Note that we have only adapted the ResNet-50 version for video modeling since the video fusion architectures proposed in Sec. III-B are incompatible with VIT. We leave the adaptation of VIT for video processing as future work since it requires far too much GPU memory for any video version to be practicably trainable with our computational resources. The remainder of this section is therefore restricted to describing the ResNet-50 variant of the Pix2Seq static architecture. The network itself has the standard transformer-based encoder-decoder architecture [4] where the encoder performs only self-attention with image features while the autoregressive decoder does self-attention with sequence features as well as cross-attention between sequence and image features.

#### 1) Encoder

The encoder takes the $640 \times 640$ RGB image as input and applies multi-headed self-attention [4] to convert it into $400 \times 256$ features which are used as input for the decoder. This diagram in Fig. 3 (left) summarizes the encoder architecture. Here,

- *FLATTEN FEAT* is the spatial flatenning of the $20 \times 20$ ResNet-50 feature maps into 1D feature vectors of size 400 each
- *POS EMBED* adds positional embedding to the flattened features
- *PROJ MLP* is a multi-layer perceptron (MLP) that projects the 2048 flattened feature vectors to $400 \times 256$
- *Self MHA X 6* is the multi-headed attention (MHA) operation that applies pairwise self-attention between each of the 400 image features, which is repeated 6 times.

#### 2) Decoder

The decoder takes the $400 \times 256$ image features from the encoder as input along with the sequence tokens. It then applies multi-headed self-attention to the sequence embedding features, followed by cross-attention between the sequence and image features. This self + cross MHA operation is repeated 6 times just like the self-MHA operation in the encoder. Fig.

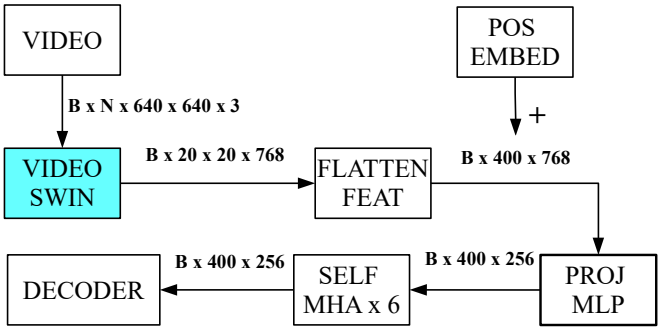


FIGURE 4: Flow diagram representing high-level processing in the early-fusion video encoder with the video Swin transformer backbone. Please refer Sec. III-B1 for details.

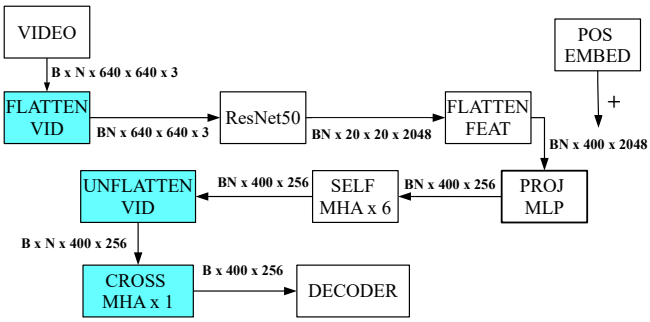


FIGURE 5: Flow diagram representing high-level processing in the middle-fusion video encoder. Please refer Sec. III-B2 for details.

3 (right) summarizes the decoder architecture. Here,

- *SEQUENCE TOKENS* refers to the sequence of target tokens constructed from the GT objects and padded to size $L = 500$
- *INPUT EMBEDDING* is obtained by table-lookup into the $V \times 256$ weight matrix of a single linear layer, where each row corresponds to one token in the vocabulary of size $V = 3000$
- *OUTPUT EMBEDDING* is obtained by projecting the 256-dimensional feature vectors to $V$-dimensional ones using the same linear layer whose weights are used in the input embedding module
- *SELF MHA* applies pairwise self-attention between each of the 500 sequence embedding features
- *CROSS MHA* applies pairwise cross-attention between each of the 500 sequence features from self-MHA and 400 image features from the encoder.

### B. VIDEO INPUT

We have determined experimentally that, like most transformer-based language models, Pix2Seq is very difficult to train from scratch on the relatively small datasets that we are working with. Therefore, we want to be able to use as much of the pre-trained weights as possible. This in turn requires that the baseline architecture is modified as little as possible. With this objective, we propose three ways to adapt the architecture for processing videos. These differ in the stage of the encoder-decoder pipeline at which the features from individual video frames are fused together.

#### 1) Early Fusion

This method replaces the ResNet-50 backbone with a video-specific backbone such as the video Swin transformer [70] or 3D-ResNet [71] so that the feature fusion happens within the backbone itself. We have only experimented with Video Swin Transformer so far. Fig. 4 summarizes the early-fusion architecture. This method only changes the number of backbone feature maps from 2048 to 768 while the rest of the pipeline remains unchanged. This means that we are unable to use pretrained weights only for the projection MLP that projects the flattened backbone features from $400 \times 768$ to $400 \times 256$. Of course, we also lose the pretrained weights for the ResNet50 backbone itself but the video Swin transformer implementation we are using [72] comes with its own weights which seem to work well enough as long as the backbone is not frozen while training. Unlike the Pix2Seq pretrained weights which were trained for token-based object detection, this backbone was trained on the Kinetics action recognition dataset [73] with conventional modeling. As a result, the overall network fails to generalize to token-based tasks if the backbone is kept frozen (Sec. V-C1). We have experimented with both tiny and base variants of this backbone but found them to have similar performance inspite of the latter having more than three times the number of parameters (87.64M versus 27.85M), probably because of the much smaller $B$ necessitated by this larger network size.

#### 2) Middle Fusion

As shown in Fig. 5, we first flatten the temporal dimension along the batch dimension to replace $B$ with $B \times N$ images (and feature maps), while leaving the rest of the encoder pipeline unchanged. The result of this reshaping is that all the subsequent operations up to and including self-MHA are performed to each one of the video frames independently as if we had a batch size of $B \times N$ instead of $B$. Once we have the $BN \times 400 \times 256$ output from the self-MHA module, we unflatten the temporal dimension to separate out the $400 \times 256$ features for each one of the $N$ video frames.

We then apply some form of cross-attention between the features from different video frames to fuse them together. The specific technique we have used is pairwise compositional cross-MHA as shown in Fig. 6. We first apply cross-MHA between the features of $F_1$ and $F_2$, then between the output of this operation and features of $F_3$, then between the output of this operation and features of $F_4$ and so on. For the sake of simplicity, we share weights between all the cross-MHA operations although it is also possible to have separate weights for each one. Another way to perform video cross-MHA is hierarchical consecutive cross-MHA as shown in Fig. 16. There are many other ways to perform this operation but

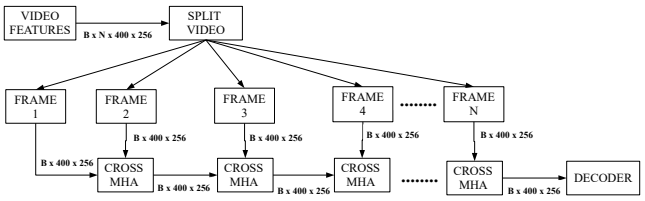


FIGURE 6: Flow diagram for the pairwise compositional variant of the cross-MHA module in the middle-fusion video encoder. Please refer Sec. III-B2 for details.

these are the only two that we have tried and the pairwise compositional variant is the one we have used for all of our experiments since the hierarchical version is less stable to train. This is likely due the excessive number of cross-MHA operations - $N \times (N-1)/2$ - that that latter requires compared to the former which only needs $N-1$ operations.

Note that, unline the self-MHA operation, the video cross-MHA operation cannot be repeated multiple times since the input and output shapes of each such operation are different.

Middle-fusion allows us to use all the pretrained weights but we have to learn the compositional cross-MHA weights from scratch. This can sometimes lead to a bit of instability during training, especially for larger values of $N$. Nevertheless, middle fusion outperforms the other two methods in most cases (Sec. V-C), albeit by small margins, so this is the one we have used for most of our experiments.

#### 3) Late Fusion

This is the only method where feature fusion happens in the decoder, unlike the other two methods where it happens in the encoder so that the decoder remains exactly the same as in the static architecture. Fig. 7 shows the late-fusion encoder and decoder. The encoder here is identical to middle-fusion as far as generating the $BN \times 400 \times 2048$ backbone features, after which the temporal dimension is unflattened to separate frame-specific features. 3D position embedding is then added to these to encode information about the position of each frame within the video. The 3D position embedding parameters are the only ones for which we cannot use pre-trained weights and therefore have to learn from scratch. The $400$ features for each of the $N$ frames in each video are then concatenated together into $400 \times N$ features. This creates an overall feature map of size $B \times 400N \times 2048$ which is processed normally for the remainder of the encoder pipeline, except that now we have $400 \times N$ features instead of $400$. Each of these $400 \times N$ image features is then cross-attended with each of the $500$ sequence features in the decoder so that every single frame is directly able to attend to every single output token.

In theory, we would expect that the flexibility of every frame being able to directly affect every token would make it possible to train better models since visual information from any frame can be used to improve the prediction of tokens corresponding to both past and future frames. This should be particularly useful for handling occlusions since the frame where an object is actually occluded contains little to no visual cues about this occluded state but this information can be obtained from past and future frames where the object is not occluded. Late-fusion is also the method with the fewest parameters for which we are unable to use the Pix2Seq pretrained weights (409K in late fusion versus 800K in middle-fusion and 24M in early-fusion) However, in practice, late-fusion has not shown any consistent performance improvement over the other two methods.

TABLE 1: Quantitative details of the datasets used for testing video detection. Number of objects is the total number of frame-level objects (ignoring instance information) while the number of trajectories is a total number of video-level object instances. For example, if an object enters the scene in frame 1 and leaves the scene in the frame 151 without being occluded in any frame, this will count as a single trajectory but 150 objects.

| Dataset | | ACAD | | | IPSC | | UA-DETRAC |
|---|---|---|---|---|---|---|---|
| | | #1 | #3 | #4 | Early-stage | Late-stage | |
| Classes | | 8 | 6 | 6 | 2 | 2 | 1 |
| Sequences | Train | 33 | 218 | 528 | 31 | 31 | 60 |
| | Test | 539 | 317 | 535 | 31 | 31 | 40 |
| | Total | 572 | 535 | 535 | 31 | 31 | 100 |
| Images | Train | 8,001 | 60,003 | 7,392 | 1,178 | 2,263 | 82,085 |
| | Test | 150,117 | 88,023 | 140,628 | 496 | 496 | 56,167 |
| | Total | 158,118 | 148,026 | 148,020 | 1,674 | 2,759 | 138,252 |
| Trajectories | Train | 38 | 240 | 578 | 288 | 265 | 5,952 |
| | Test | 612 | 358 | 598 | 240 | 240 | 2,337 |
| | Total | 650 | 598 | 1,176 | 528 | 505 | 8,289 |
| Objects | Train | 8,226 | 65,731 | 7,937 | 7,463 | 11,847 | 598,281 |
| | Test | 163,157 | 94,364 | 152,152 | 3,248 | 3,248 | 675,774 |
| | Total | 171,383 | 160,095 | 160,089 | 10,711 | 15,095 | 1,274,055 |

## IV. EVALUATION

### A. DATASETS

We have evaluated our detector the folowing three datasets:

- ACAMP Canadian Animal Detection (ACAD) [74]: We have trained on only 3 of the 8 configurations [74, Table 4] - $\#1$, $\#3$ and $\#4$ - since these are the only ones that contain video information.
- IPSC [67]: We have trained on both early and late-stage training configurations [67, Sec. 2.3] and tested both sets of models on the same test set containing the first 16 images from each sequence.
- UA-DETRAC [66]: This is a class-agnostic large-scale dataset with long and diverse sequences containing relatively long-term motion information that is missing from the previous two datasets. This allows us to at least partially rule out dataset limitations when comparing the static Pix2Seq detector with our proposed video version, especially for larger values of $N$.

Table 1 provides quantitative details for these datasets.

### B. METRICS

We have used a wide range of object detection metrics since different metrics are more suited to different application domains. These metrics include mAP, mRP and cRP for ACAD [74, Sec. 4.1], and ROC-AUC, RP-AUC, FN-DET, FP-DUP,

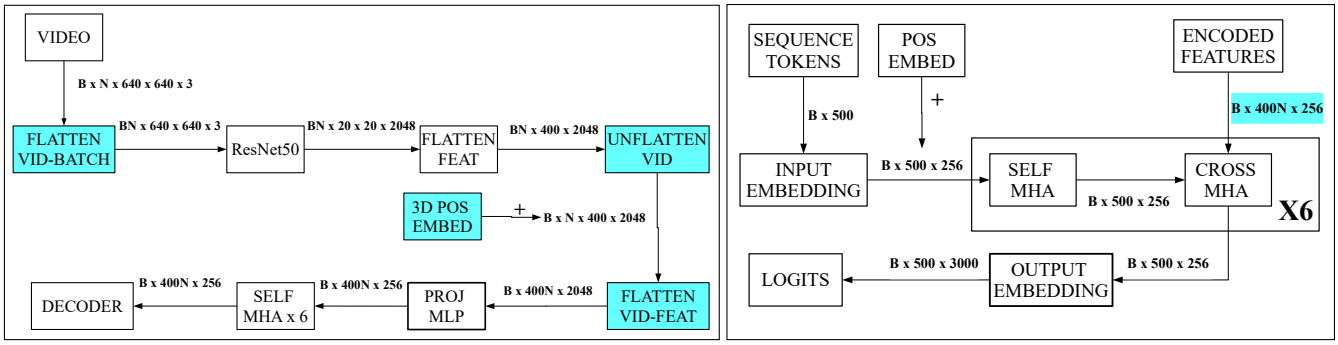


FIGURE 7: Flow diagram representing high-level processing in the late-fusion video encoder and decoder on the left and right respectively. Please refer Sec. III-B3 for details.

and FP-NEX for IPSC [67, Sec. 3.1, 3.2]. Although several of these detection metrics are strongly correlated with each other in many scenarios, we have empirically found RP-AUC to correspond best with the overall detection performance for most practical purposes. Our principal objective in this paper is to compare the baseline Pix2Seq static detection model with our video detection model as well as compare between different variants of the latter, so we use RP-AUC as the single main metric for this purpose. We do still use the domain-specific metrics proposed in [74, Sec. 4.1] and [67, Sec. 3] when comparing the token-based models with the conventional models from those chapters.

RP-AUC measures both localization and classification performance so we also use a class-agnostic version of RP-AUC which we have termed **cRP-AUC**. Similar to the cRP metric in [74, Sec. 4.1], cRP-AUC is computed by considering all the predicted and GT objects to belong to the same class so that misclassifications are not penalized and we are able to measure the localization performance alone. In many practical applications, especially those involving human-in-the-loop systems, being able to correctly detect the presence or absence of an object is more important than classifying it correctly and this metric allows us to measure the suitability of a detector for such systems.

### C. TRAINING

#### 1) Setup

We have performed most of the training on four GPU servers, each with 2 × Geforce RTX 3090 24GB GPUs. A few of the larger models were trained on a cloud server with 2 × Tesla A100 80GB GPUs. We have also trained some of the smaller models on a fourth GPU server with 3 × Geforce GTX 1080Ti 11GB GPUs. Finally, we used a fifth GPU server with a Geforce RTX 3090 24GB and a Geforce RTX 3060 12GB for running inference. Note that expensive resources such as the Tesla A100 server were used for a very limited time in this study so most of the models reported here could not benefit from these. More details of these servers are provided in Table 4. We trained all the models with the default hyperparameter settings provided by the Pix2Seq authors, except for adjusting $B$ to the maximum that would fit on the GPUs, along with $V$ and $L$ as needed for each model configuration.

#### 2) Validation

Pix2Seq codebase does not support performing validation as part of the training run. While we did add support for this, we found that it not only slowed down training significantly, but also caused random crashes due to running out of GPU memory. As a workaround, we implemented a remote validation pipeline where the inference server periodically polls the training server for new checkpoints and runs inference on the latest checkpoint thus found. We set the time period between successive polling attempts to the maximum of two hours or the inference time. Since inference requires much less GPU memory than training, it is possible to simultaneously perform validation for multiple training runs on the same inference server.

We performed validation directly on the test set rather than a subset of the training set, as is considered the standard practice. We had to do this both in order to reduce the total training time as well as to utilize as many of the limited number of labeled images for training as possible. A separate validation set is principally needed to prevent overfitting but it turned out that each of our test sets is sufficiently similar to the corresponding training set that the test performance reached a plateau in every single case and did not decline even when training was continued for several hundreds of thousands of iterations after this plateauing. Some of the test sets are too large to complete inference within a reasonable timeframe (e.g. $< 10$ hours) In such cases, we validated on a small representative subset of the test set (e.g. 10 frames per sequence) after empirically confirming that the performance trend on this subset was consistent with the full test set.

#### 3) Distributed Training

Pix2Seq codebase does support multi-machine distributed training and we used it to train a few models over two or more of the four dual RTX 3090 servers in order to use the combined 96 to 192 GB of GPU memory. It would have been extremely beneficial to have been able to do this for all (or at least most) of the models, especially on larger datasets like ACAD and UA-DETRAC. However, Pix2Seq distributed

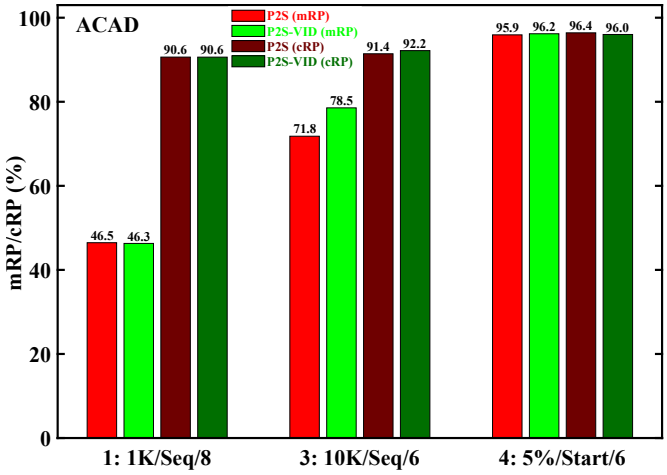


FIGURE 8: Results on ACAD configurations #1, #3 and #4 [74, Table 4] in terms of both mRP and cRP [74, Sec. 4.1] shown respectively with lighter and darker shades of red and green. P2S and P2S-VID respectively refer to the static and video detection Pix2Seq models. P2S-VID uses middle fusion architecture with $N = 2$ and was trained with frozen backbone and class token weight equalization.

training implementation is optimized for Tensor Processing Units (TPUs) rather than GPUs and this, combined with the relatively slow network connection between our GPU servers, made the overhead so high that the GPU usage during these runs was $< 50\%$ nearly the entire time, and the GPUs spent a good fraction of that time idling. In addition, two of our servers suffer from hardware incompatibility between their motherboards and RTX 3090 GPUs which causes them to restart randomly after a while when used in a distributed training setup. These issues in turn extended the training time to such an extent (often to several weeks) that distributed training was feasible for only a few models.

### *D. RESULTS*

This section presents results comparing our proposed token-based video detection model with the baseline Pix2Seq static detection model [1] along with some state of the art video detection models. Unless otherwise specified, all Pix2Seq models have the $640 \times 640$ ResNet-50 as their backbone and the video models use the middle-fusion video architecture (Sec. III-B2) with $N = 2$. For the sake of brevity, Pix2Seq static and video detection models are referred to as **P2S** and **P2S-VID** respectively for the remainder of this paper. We experimented with many different configurations or variants of P2S and P2S-VID (V) but this section only summarizes the best results we found. The specific model configurations that we have included here for each dataset are detailed in Table 5. Note that we were only able to train a small fraction of all the models we would have liked to have trained due to limited time and computational resources, so these results very likely do not indicate the best performance that these models are capable of, especially in the case of the video models with larger values of $N$.

#### 1) Summary

Following are the key takeaways from the results presented in the remainder of this paper:

- P2S-VID models exhibit strong signs of being bottlenecked, especially for larger values of $N$, by the low batch sizes (V-B) and possibly also the relatively small amounts of training data we had to use.
- Pix2Seq models are better at localizing objects than classifying them correctly so that the class-agnostic performance of P2S-VID tends to compare more favourably with P2S than its overall performance.
- Related to the last point is that Pix2Seq models are relatively less robust to class imbalance and tend to overfit to the more numerous class.
- This problem of poor classification performance can be partially ameliorated by equalizing the weights assigned to class tokens during training so that each class token has the same weight as all of the corresponding bounding box coordinate tokens combined (V-A).
- P2S-VID performs appreciably better overall than the baseline P2S but this improvement is mainly due to the output redundancy (Sec. II-B4) obtained by using $T < N$ and this performance advantage mostly disappears by setting $T = N$.
- There is no consistent improvement in performance with increase in $N$ (V-D).
- Static models trained to predict video outputs by processing only the first frame in each video temporal window (V-C2) are able to keep up with the video models surprisingly well, even for large values of $N$, indicating that the latter are not able to make sufficient use of the video information.

#### 2) ACAD

Fig. 8 summarizes performance on all three configurations of the ACAD dataset. Note that, even though config #4 has the smallest training set, it is the easiest to handle from a classification standpoint because the training set contains frames from nearly every sequence in the test set and therefore the model is able to train on every combination of backgrounds and foregrounds available therein. While the other two configs contain more frames overall, they have no overlap between the training and test sequences and offer the models access to far fewer combinations of foregrounds and backgrounds in the test set. P2S-VID mostly performs about the same as P2S, but it does significantly outperform the latter in the challenging config #3 which is the only one that has enough frames to at least partially alleviate the bottleneck imposed by the relatively small datasets.

#### 3) IPSC

Fig. 9 shows the results for both configurations of this dataset in terms of both high-level and low-level detection metrics. P2S-VID significantly outperforms P2S on nearly every

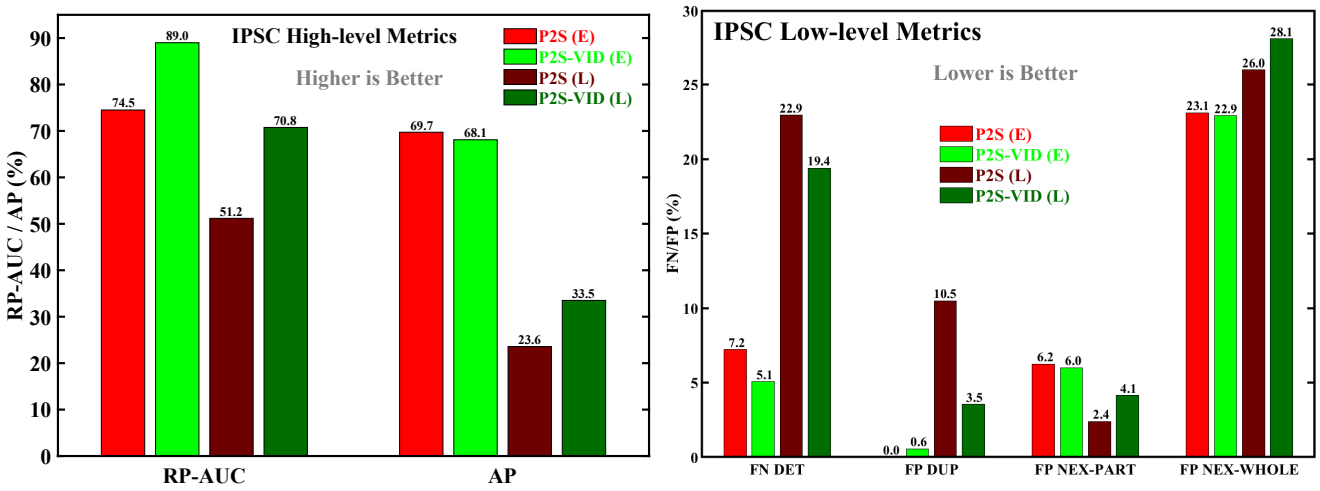


FIGURE 9: Comparing P2S and P2S-VID on IPSC dataset in terms of (left) high-level and (right) low-level detection metrics. Results are provided for both early and late-stage training configuration, shown respectively with lighter and darker shades of red and green and denoted in the legend with the suffixes E and L.

metric especially on the much more challenging late-stage configuration. The only minor exception is FP-NEX which can be attributed to the redundancy involved in combining detections from multiple temporal windows. As mentioned in [67, Sec. 3.4.2], FP-NEX essentially measures how many of the unlabeled cells are incorrectly detected and classified as IPSCs, something which the relatively poor classification ability of the language models makes them susceptible to.

#### 4) UA-DETRAC

Fig. 14 shows a summary of results on the UA-DETRAC dataset for P2S and P2S-VID with $N = 2$ to $N = 32$. We had hoped that the size of this dataset would allow the severe bottleneck on the video models to be at least partially overcome but that turned out not to be the case. All the models, including P2S, appear to be limited by $B$ here, judging by the identical ceiling of around $85\%$ that they all reach before $N$ becomes too large. As seen in the example of ACAD #3 (Fig. 11), $B$ seems to be an even more important bottleneck than the size of the dataset. In fact, the optimal $B$ probably increases with the size and complexity of the dataset, so the models are likely to be even more bottlenecked on UA-DETRAC than ACAD. It is true that the ACAD example demonstrated performance limitation only on the classification task and this is not relevant with UA-DETRAC since it has only one class. However, UA-DETRAC has much longer and more complex trajectories than ACAD. This makes its localization task a lot more challenging and therefore just as likely to be bottlenecked as classification on ACAD. P2S-VID models upto $N = 8$ are able to mostly match P2S with $T = 1$ but the performance drops sharply after that, thus confirming the inadequacy of our existing hardware for training such large models. We would expect that all the models, including P2S, would be able to reach $> 90\%$ if trained with large enough B.

We would like to conclude this section with a recent result which is a promising step in this direction. We generated this by training models on two Tesla A100 80 GB GPUs that we rented to produce publication-quality results that are competitive with the current state of the art. VSTAM [46] is the top-ranked model on the leaderboard [75] at the time of this writing and has been so since it was released nearly 3 years ago. However, as shown in Table 2, one of our P2S-VID models was able to outperform VSTAM by $0.75\%$. This model uses the late-fusion architecture which turns out to significantly outperform the other two fusion schemes once the batch size bottleneck is alleviated, as we had hypothesized in Sec. III-B3 based on its theoretical advantage of cross-attending every token with each of the $N$ frames. Also, this model was trained with the smallest possible video length of $N = 2$ because even 160 GB GPU RAM is apparently insufficient to relieve the bottleneck on models with higher $N$ (Table 3). We were was hoping to rent 4 or even 8 of these GPUs to better exploit the video length but these configurations were unfortunately not available. This result is perhaps the most convincing evidence we have that most of the P2S-VID results in this paper are of bottlenecked models and do not represent their true potential.

TABLE 2: Comparing P2S and P2S-VID with the current state of the art video detection models on UA-DETRAC.

| Model | mAP (%) |
|---|---|
| TFEN [76] | 82.42 |
| YOLOv3-SPP [77] | 84.96 |
| MSVD_SPP [78] | 85.29 |
| SpotNet [79] | 86.8 |
| FFAVOD-SpotNet [80] | 88.1 |
| VSTAM [46] | 90.39 |
| P2S | 88.62 |
| **P2S-VID** | **91.14** |

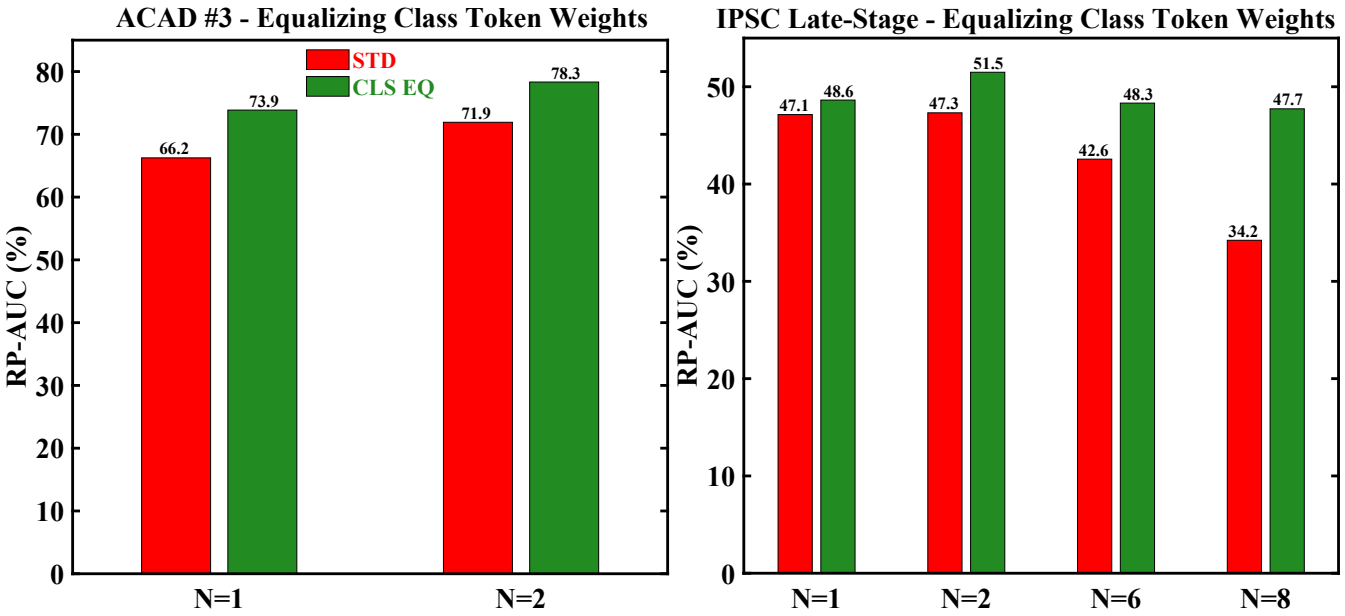


FIGURE 10: Performance impact of equalizing the class token weights on (left) ACAD #3 and (right) IPSC late-stage datasets. *N=1* denotes the baseline P2S model. Models trained with and without class token weight equalization are respectively shown in green and red and denoted with *CLS EQ* and *STD* in the legend. Note the different Y-axis limits in the two plots.

## V. ABLATION STUDY

This section presents the results of our experimentation with some of the important model parameters that were useful in finding the optimal models that are reported in Sec. IV-D1. We used the IPSC late-stage dataset for most of these experiments since it is small enough to allow training a large number of models while at the same time also being challenging enough to be able to discriminate between these models.

### A. EQUALIZING CLASS TOKEN WEIGHTS

A possible reason for the poor classification performance of Pix2Seq models, especially in object detection, might be that the weight assigned to the class tokens during training is equal to that assigned to every single coordinate token. Since the number of coordinate tokens is $4 \times N$ times greater than the number of class tokens, this effectively means that the localization task is assigned $4 \times N$ times greater weightage than the classification task. In order to address this disparity, we trained detection models with each class token assigned the same weight as all of the corresponding coordinate tokens combined.

As shown in Fig. 10, this does improve the classification performance appreciably, particularly on large datasets and with higher vlues of $N$. The degree of improvement also increases with $N$, especially in the IPSC case. This makes sense since the factor by which coordinate tokens are over-weighted with respect to class tokens without equalization increases linearly with $N$ as does the overemphasis on the localization task. This improvement does come at the cost of significantly slower training, at least in the case of larger datasets. For example, ACAD #3 took 700K iterations and close to 12 days to reach convergence with class equalization and only 300K iterations and just over 5 days without it. These disparities were not as strongly marked on the IPSC dataset, where both sets of models took approximately the same amount of time to converge.

### B. TRAINING BATCH SIZE (B)

Extensive experiments have shown to us that the training batch size matters a lot more for large datasets like ACAD and UA-DETRAC rather than smaller ones like IPSC. It also has a greater impact on video models than static ones, especially with larger values of $N$. Finally, it has far greater impact on the classification task (i.e. RP-AUC) than localization (i.e. cRP-AUC). Fig. 11 shows an example for both P2S and P2S-VID training on ACAD #3. It can be seen that the peak RP-AUC on the validation set nearly doubles from 37% to 66% for P2S and nearly triples from 28% to 78% for P2S-VID. On the other hand, cRP-AUC peaks at just above 90% in all four cases and this is reached in just a few thousand iterations as opposed to RP-AUC which takes from 200K to 700K iterations to attain its plateau. The significantly greater relative increase in the case of P2S-VID (2.76 times vs. 1.75 times for P2S), along with the much lower absolute peak with the smaller batch size (28% vs. 37%) confirms the much greater bottleneck faced by the video models.

P2S-VID here uses only $N = 2$ and this bottleneck only gets worse as $N$ increases. In our experience, and subject to the size and complexity of the dataset, the optimal batch size required for a video model to achieve its full potential increases at least linearly with $N$. However, the GPU memory required to train a video model also increases linearly with

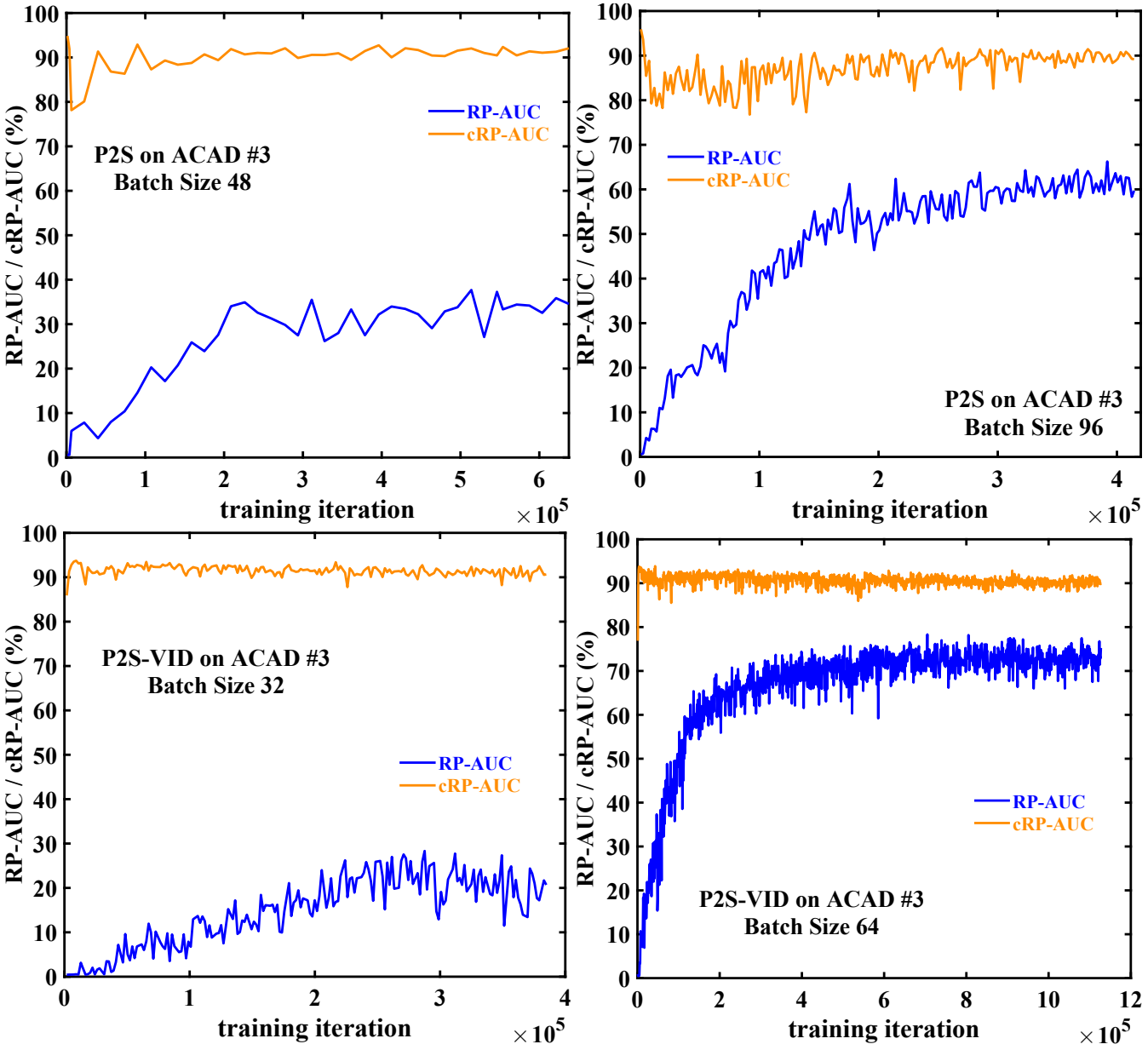


FIGURE 11: Impact of batch size on validation performance when training (top) P2S and (bottom) P2S-VID on ACAD #3. The left plots show the results for training on a single RTX 3090 GPU with batch sizes 48 and 32 while the ones on the right show dual-GPU training with batch sizes 96 and 64.

$N$ even if the batch size remains unchanged. Since the total amount of available GPU memory is fixed, we have to actually *decrease* the batch size linearly with $N$.

### C. VIDEO ARCHITECTURE

Fig. 12 shows comparative results for the three video architectures (Sec. III-B) with $N = 2$. Middle-fusion outperforms the other two models in terms of mRP while early-fusion performs best in terms of cRP. We have seen this trend of early-fusion models performing much better on the localization task than classification in other models we have trained as well. The video Swin transformer backbone we are using for early-fusion was pre-trained for human action-recognition and therefore unsurprisingly finds it easier to localize objects than to classify them correctly. Although late-fusion seems slightly inferior to middle-fusion in these results, we have seen it outperform the latter with similar margins on other values of $N$ so, on the whole, the two models can be said to perform at par.

#### 1) Frozen Backbone

We would have expected that the entire Pix2Seq network would need to be retrained in order to work well on a new task – either video detection or semantic segmentation - but

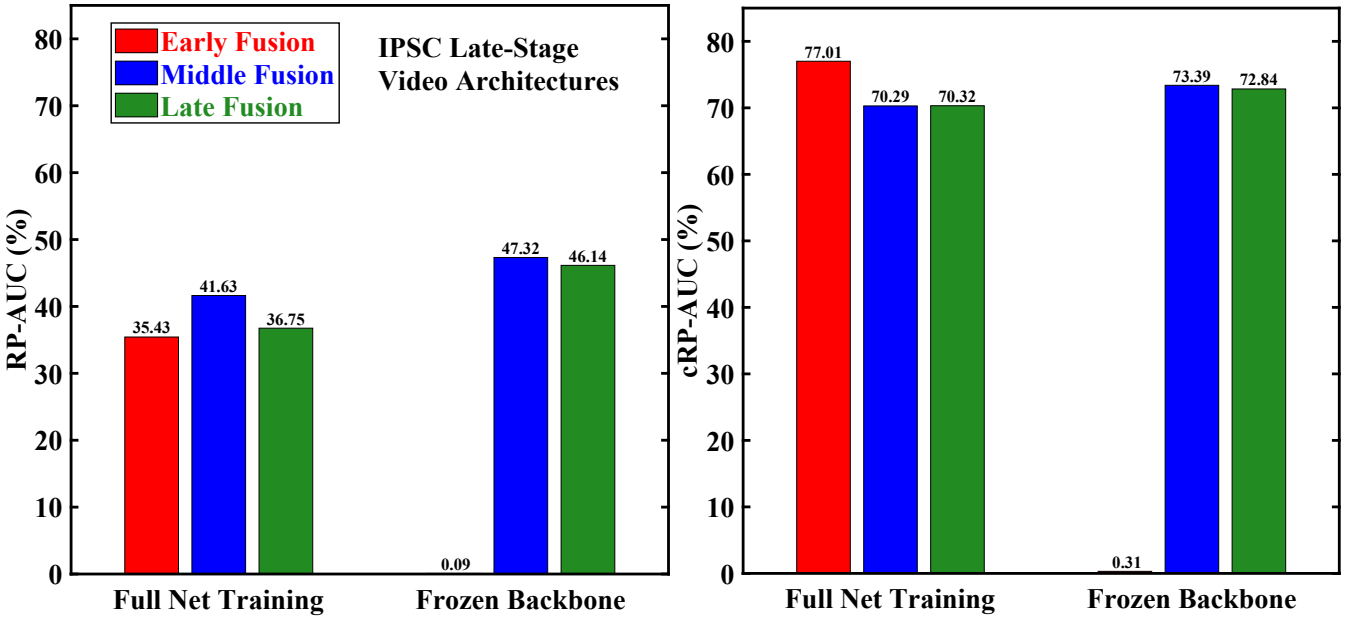


FIGURE 12: Comparing the three video architectures (Sec. III-B) with $N = 2$ and trained with and without frozen backbone. The left and right plots show RP-AUC and cRP-AUC respectively. All models were trained on the IPSC late-stage dataset.

this turned out not to be the case. As shown in Fig. 12, keeping the backbone frozen actually leads to significantly better video detection performance for both middle and late-fusion architectures. However, this is not true for early-fusion, which completely fails to learn anything useful with its video Swin transformer backbone frozen. This backbone was pre-trained for action recognition with conventional modeling and it seems that weights optimized for conventional modeling cannot generalize to language modeling without retraining. Nevertheless, these weights are useful in fine-tuning the network for language modeling since we found that training the early-fusion models without loading the pretrained weights for its backbone leads to significantly worse performance. Note that freezing the backbone allows us to use much higher batch sizes (e.g. 80 vs. 20 with $N = 2$) which must partially account for this improvement as well.

### 2) Video Output with Static Input

We wanted to find out how much useful information the network is able to learn from video frames so we trained static models to predict video output (either detection or segmentation) using only the first frame $F_1$ in each video temporal window as input. This means that the model is trained to produce the same output as a P2S-VID or P2S-VIDSEG model but it only has access to the first frame in each temporal window, rather than all $N$ frames. It therefore needs to use the first frame to predict the contents of the future $N - 1$ frames in the sequence. Since there is no more video input, we can use the baseline P2S architecture for these models. We trained models with $N = 2$, $N = 4$, $N = 6$ and $N = 8$, all with the backbone frozen, and all on the IPSC late-stage dataset. Note that these static-video models can be trained with the same (and much larger) batch size as P2S, irrespective of $N$. This gives them an advantage over the true video models whose batch size decreases linearly with $N$. The video stride $T$ is important in evaluating these models since $T = 1$ ensures that each frame would be the first frame in some temporal window so that the static input models can output valid boxes only for the first frame and still not be penalized during inference. This is unlikely to happen in practice since the model is trained to output boxes for all $N$ frames and therefore will be penalized during training for learning a simple strategy like this.

Results are shown in Fig. 13. Static input models show surprisingly little performance loss over the video models even for $N$ as high as 8, though this loss is greater for $T = N$ than $T = 1$ as expected. Similarly, the performance loss predictably increases with $N$, except for the odd case of $N = 8$ with $T = 1$ (Sec. V-D). The performance loss is also greater for cRP-AUC than RP-AUC, which makes sense since the first frame is usually sufficient to classify an object but we need the remaining frames to localize it correctly in those frames. The fact that static input models are able to perform so well even with $T = N$ is explained at least partly by the fact the cells do not move a lot and this makes the bounding boxes in the IPSC dataset relatively easy to predict. It is probably also another indicator of the bottleneck on the video models that are unable to make full use of the video information themselves. It is also possible, though we consider it unlikely, that the video information fusion schemes we have come up with are not good enough to take full adavantage of this information.

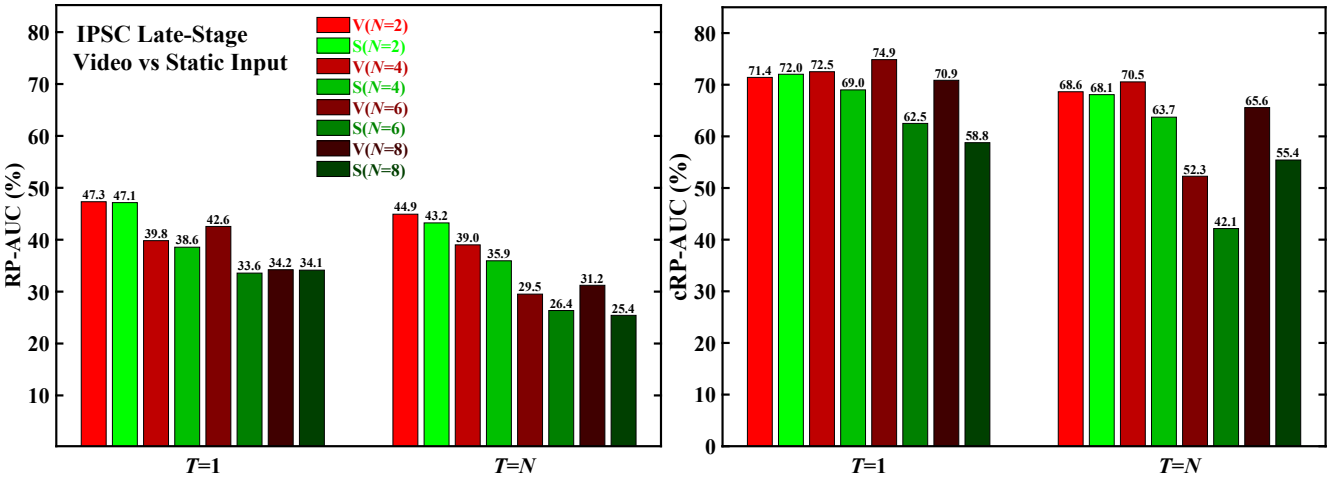


FIGURE 13: Performance impact of replacing $N$ video frames with only the first frame in the sequence as input to P2S-VID models. The two cases are respectively shown in shades of red and green and denoted with *V* and *S* in the legend. Darker shades represent higher $N$. All video input models used the middle-fusion architecture without class token weight equalization. The left and and right plots show RP-AUC and cRP-AUC respectively. All models were trained on the IPSC late-stage dataset.

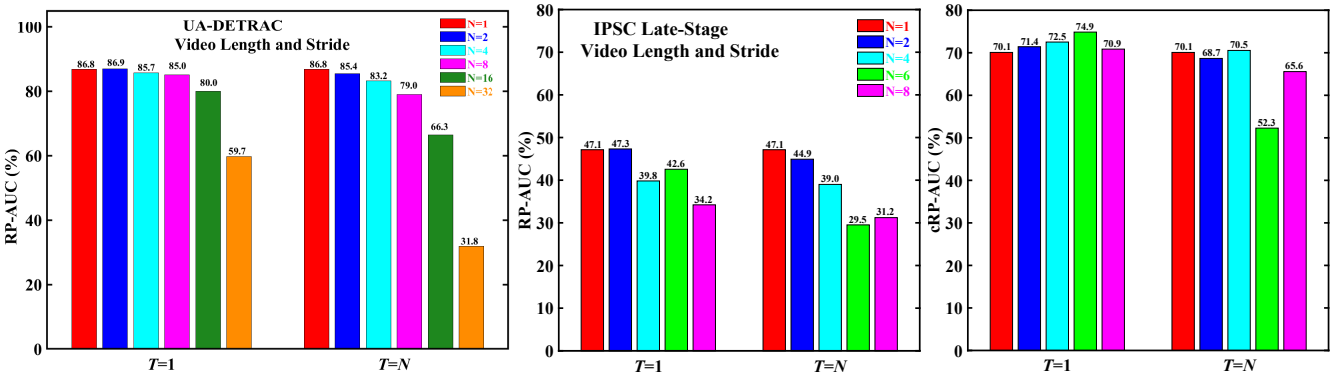


FIGURE 14: Impact of video length $N$ on P2S-VID pereformance over (left) UA-DETRAC and (center and right) IPSC late-stage datasets. *N=1* denotes the baseline P2S model and is included for comparison. Best viewed under high magnification.

### D. VIDEO LENGTH AND STRIDE

Fig. 14 summarizes the impact of video length $N$ and stride $T$ on video detection performance over both UA-DETRAC and IPSC late-stage datasets. Neither dataset shows any consistent improvement in performance with $N$. Quite the contrary, in fact. The only case that shows any signs of steady improvement is IPSC cRP-AUC with $T = 1$, at least as far as $N = 6$. There is, however, fairly consistent improvement in going from $T = N$ to $T = 1$ and the degree of this improvement also increases with $N$, which is to be expected because of the increasing redundancy.

### E. 1D COORDINATE TOKENS

As mentioned in Sec. II-B3, 1D coordinate tokens can be used to partially solve the problem of $L$ becoming too large as $N$ increases. We trained a few models to judge the practicability of this approach. As shown in Fig. 15, it turned out to be not doable, at least on our existing hardware. The 1D model was able to achieve barely half the performance of the standard 2D model on the late-stage dataset, although it fared slightly better on the easier early-stage variant. An interesting finding here was that training the 1D model without the backbone frozen causes a further sharp decline in performance. This is surprising since one would expect that learning a new coordinate tokenization different from the one that was used for pretraining the backbone would benefit from fine-tuning the backbone on the new tokenization, but that is not the case. Note that the 2D model was trained with the default Pix2Seq vocabulary parameters $H = 2K$ and $V = 3K$ while the 1D version had $H = 160$ and $V = 28K$. Although both 2D and 1D models were trained with the same $B$, it is possible that this poor performance might be another case of a bottleneck introduced by $B$ since the much higher $V$ in the 1D case probably requires much larger $B$ to work. This is also supported by the much greater performance advantage of frozen-backbone models with 1D tokenization than with 2D tokenization. Freezing the backbone allowed the 1D models to be trained with $B = 64$ while the non-frozen models were

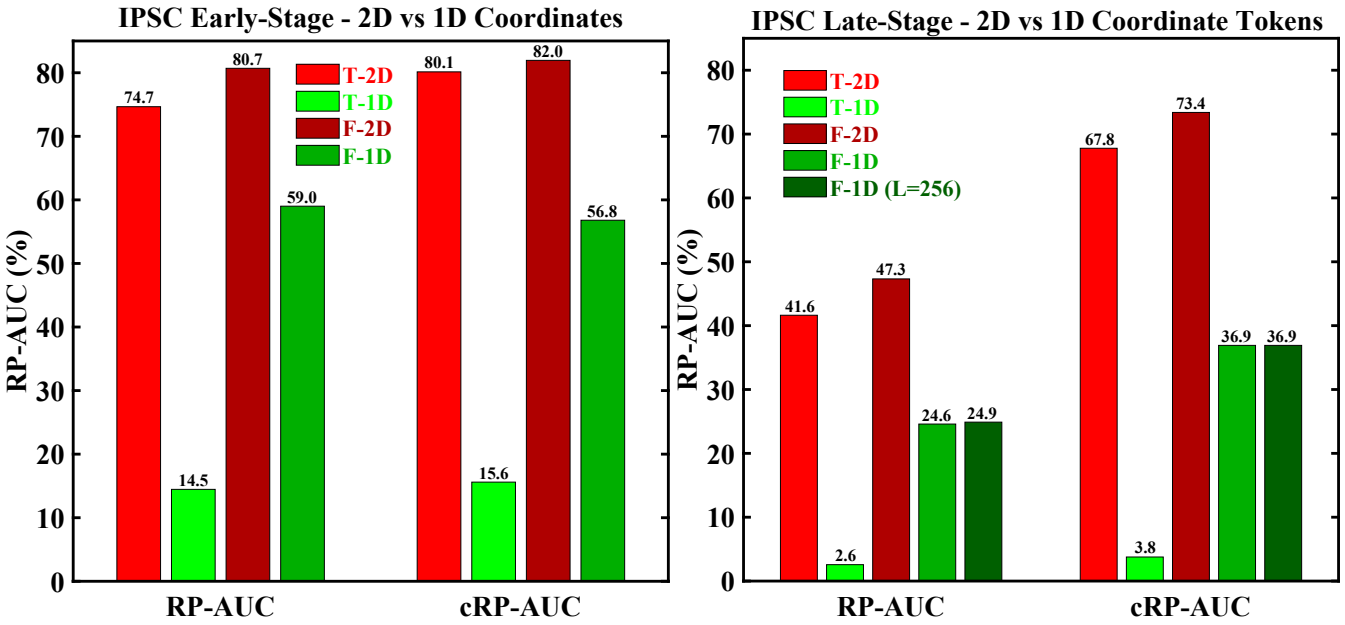


FIGURE 15: Comparing the standard 2D coordinate tokenization of P2S-VID with its 1D variant on the (left) early and (right) late-stage IPSC datasets. 2D and 1D tokenization models are respectively shown in shades of red and green. Models trained with and without frozen backbone are shown respectively in darker and lighter shades and denoted with the prefixes *F* and *T* in the legend. All models use $N = 2$.

restricted to $B = 18$. The performance drop can also be partially attributed to the decrease in localization accuracy due to the drop in $H$ by a factor of more than 10 from $H = 2000$ to $H = 160$. We also trained a 1D model with $L$ reduced by half to 256 to take advantage of the shorter sequences but this turned out to have no impact on the overall performance.

## VI. CONCLUSIONS

This paper has introduced a new way to perform object detection in videos by modeling the outputs of these tasks as sequences of discrete tokens. We have proposed these new methods as another step in the direction of the more general tokenization of visual recognition tasks that has been happening over the last few years through the paradigm of language modeling. We have presented theoretical arguments for why such tokenization can help to solve the problems consequent upon trying to model the inherently discrete and variable-length outputs that are common in vision tasks with the continuous-valued and fixed-length representations in conventional modeling. We have tested these models on a wide range of real-world problems with in-depth experiments to demonstrate their competitiveness with the state of the art in conventional modeling. Although we have not been able to demonstrate that our method offers significant and consistent performance advantage over conventional models, we do present strong evidence to suggest that this is not due to any intrinsic weakness in the models themselves. Rather, it is very likely to be a consequence of the bottleneck that is imposed upon these models by the small training batch sizes that we have been constrained to use by our limited computational resources. Once these constraints are lifted and the models can be trained to their full potential, we are confident that they will be able to justify their theoretical advantages with practical performance benefits.

## APPENDIX. HIERARCHICAL VIDEO CROSS-MHA

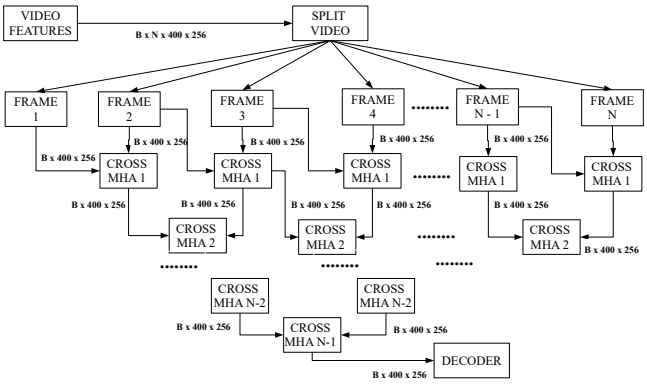


FIGURE 16: Flow diagram for the hierarchical variant of the cross-MHA module in the middle-fusion video encoder. Best viewed under high magnification.

This appendix provides an alternative way to perform the video cross-MHA operation in the middle-fusion video encoder. As shown in fig. 16, we first apply cross-MHA between pairs of consecutive frames, i.e. $(F_1, F_2)$, $(F_2, F_3)$, $(F_3, F_4)$ and so on to obtain $N-1$ feature maps, each of size $400 \times 256$, corresponding to the $N - 1$ frame-pairs. Consecutive pairs

from these $N-1$ feature maps are then cross-attended in a second level cross-MHA operation to yield $N-2$ feature maps, again with the same $400 \times 256$ size. This process is repeated for a total of $N-1$ levels of cross-MHA operations to finally yield a single $400 \times 256$ feature map that is passed to the decoder.

## APPENDIX. MODEL TRAINING

This appendix provides configuration details for the models reported in Sec. IV-D as well as the GPU servers used for training these models.

TABLE 3: Performance on UA-DETRAC dataset of P2S and P2S-VID models trained with more GPU RAM to partially alleviate batch size bottleneck. Please refer Sec. IV-D4 and Table 2 for more details.

| Model | | *N* | mAP (%) | *B* | GPUs | GPU RAM |
|---|---|---|---|---|---|---|
| P2S-VID | Early Fusion | 2 | 75.129 | 80 | 2 x Tesla A100 | 160 GB |
| | Middle Fusion | 2 | 89.55 | 320 | 2 x Tesla A100 | 160 GB |
| | | 3 | 87.97 | 216 | 2 x Tesla A100 | 160 GB |
| | | 4 | 88.17 | 160 | 8 x RTX 3090 | 192 GB |
| | Late Fusion | **2** | **91.14** | **256** | **2 x Tesla A100** | **160 GB** |
| | | 3 | 89.76 | 168 | 2 x Tesla A100 | 160 GB |
| | | 4 | 89.47 | 96 | 8 x RTX 3090 | 192 GB |
| P2S | | 1 | 88.62 | 288 | 6 x RTX 3090 | 144 GB |

TABLE 4: Details of the GPU servers used for model training and inference. Best viewed under high magnification.

| Type | CPU | | | RAM | GPUs | GPU RAM |
|---|---|---|---|---|---|---|
| | Model | Speed | Cores / Threads | | | |
| Primary Training Servers | Intel Core i7-6800K | 3.4 GHz | 6 / 12 | 64 GB | 2 x Geforce RTX 3090 | 48 GB |
| | Intel Xeon E5-2620v4 | 2.1 GHz | 8 / 16 | 32 GB | 2 x Geforce RTX 3090 | 48 GB |
| | Intel Core i7-6700K | 4.0 GHz | 4 / 8 | 32 GB | 2 x Geforce RTX 3090 | 48 GB |
| | Intel Core i7-6800K | 3.4 GHz | 6 / 12 | 64 GB | 2 x Geforce RTX 3090 | 48 GB |
| Rented Training Server | AMD EPYC-Milan | 2.45 GHz | 16 / 16 | 64 GB | 2 x Tesla A100 80 GB | 160 GB |
| Secondary Training / Inference Server | Intel Core i7-3820 | 3.6 GHz | 4 / 8 | 32 GB | 3 x Geforce GTX 1080 Ti | 33 GB |
| Inference Server | Intel Core i5-10400 | 2.9 GHz | 6 / 12 | 48 GB | Geforce RTX 3090<br>Geforce RTX 3060 | 36 GB |

## REFERENCES


[1] T. Chen, S. Saxena, L. Li, D. J. Fleet, and G. E. Hinton, "Pix2seq: A Language Modeling Framework for Object Detection," in *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022.

[2] A. Singh, "P2S-Video: Extension of Pix2Seq for Video Detection and Segmentation," online: https://github.com/abhineet123/p2s-video.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in neural information processing systems*, vol. 25, 2012.

[4] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," *NIPS*, 2017.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *ICLR*, 2021.

[6] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," *ICCV*, pp. 9992–10 002, 2021.

[7] R. Girshick, "Fast R-CNN," in *ICCV*, Dec 2015, pp. 1440–1448.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, June 2017.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," in *ECCV*, 2016.

[10] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," in *ICCV*, Oct 2017, pp. 2999–3007.

[11] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *TPAMI*, vol. 42, pp. 386–397, 2020.

[12] Z. Cai and N. Vasconcelos, "Cascade R-CNN: High Quality Object Detection and Instance Segmentation," *TPAMI*, vol. 43, pp. 1483–1498, 2021.

[13] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *CVPR*, pp. 779–788, 2015.

[14] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *CVPR*, pp. 6517–6525, 2017.

[15] ——, "YOLOv3: An Incremental Improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[16] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," *ArXiv*, vol. abs/2004.10934, 2020.

[17] G. R. Jocher, A. Stoken, J. Borovec, NanoCode, A. Chaurasia, TaoXie, C. Liu, Abhiram, Laughing, tkianai, yxNONG, A. Hogan, lorenzomammana, AlexWang, J. Hájek, L. Diaconu, Marc, Y. Kwon, Oleg, wanghaoyang, Y. Defretin, A. Lohia, ml ah, B. Milanko, B. Fineran, D. P. Khromov, D. Yiwei, Doug, Durgesh, and F. Ingham, "ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations," 2021. [Online]. Available: https://github.com/ultralytics/yolov5

[18] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors," *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7464–7475, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:250311206

[19] G. R. Jocher, A. Stoken, J. Borovec, NanoCode, A. Chaurasia, TaoXie, C. Liu, Abhiram, Laughing, tkianai, yxNONG, A. Hogan, lorenzomammana, AlexWang, J. Hájek, L. Diaconu, Marc, Y. Kwon, Oleg, wanghaoyang, Y. Defretin, A. Lohia, ml ah, B. Milanko, B. Fineran, D. P. Khromov, D. Yiwei, Doug, Durgesh, and F. Ingham, "Ultralytics YOLOv8: A computer vision model architecture for detection, classification, segmentation, and more," 2023. [Online]. Available: https://yolov8.com/

[20] C.-Y. Wang, I.-H. Yeh, and H. Liao, "YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information," *ArXiv*, vol. abs/2402.13616, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:267770251

[21] J. R. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Mach. Learn. Knowl. Extr.*, vol. 5, pp. 1680–1716, 2023.

[22] A. Shrivastava, A. K. Gupta, and R. B. Girshick, "Training Region-Based Object Detectors with Online Hard Example Mining," *CVPR*, pp. 761–769, 2016.

[23] A. Bewley, Z. Ge, L. Ott, F. T. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," *ICIP*, pp. 3464–3468, 2016.

[24] E. Bochinski, V. Eiselein, and T. Sikora, "High-Speed Tracking-by-Detection without using Image Information," *AVSS*, pp. 1–6, 2017.

[25] T. Chen, "Pix2Seq Codebase: Multi-tasks with generative modeling (autoregressive and diffusion)," online: https://github.com/google-research/pix2seq.

[26] T. Chen, S. Saxena, L. Li, T.-Y. Lin, D. J. Fleet, and G. Hinton, "A Unified Sequence Interface for Vision Tasks," in *Proceedings of the 36th Annual Conference on Neural Information Processing Systems (NIPS)*, 2022.

[27] T. Chen, R. Zhang, and G. Hinton, "Analog bits: Generating discrete data using diffusion models with self-conditioning," in *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, 2023.

[28] T. Chen, L. Li, S. Saxena, G. E. Hinton, and D. J. Fleet, "A Generalist Framework for Panoptic Segmentation of Images and Videos," *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 909–919, 2023.

[29] X. Chen, H. Peng, D. Wang, H. Lu, and H. Hu, "SeqTrack: Sequence to Sequence Learning for Visual Object Tracking," *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14 572–14 581, 2023.

[30] Y. Zheng, B. Zhong, Q. Liang, G. Li, R. Ji, and X. Li, "Toward Unified Token Learning for Vision-Language Tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 34, pp. 2125–2135, 2023.

[31] J. Liu, H. Ding, Z. Cai, Y. Zhang, R. K. Satzoda, V. Mahadevan, and R. Manmatha, "Polyformer: Referring image segmentation as sequential

TABLE 5: Details of the models whose results are reported in Sec. IV-D1 except a few models in Sec. IV-D4 whose details are in Table 3.

**P2S**

| Dataset | | Backbone | Input Size | Frozen Backbone | Class Equalization | Coord Tokens | *H* | *V* | *L* | GPUs | *B* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ACAD | #1 | ResNet-50 | 640 | Yes | No | 2D | 2K | 3K | 512 | 1 x RTX 3090 | 48 |
| | #3 | ResNet-50 | 640 | Yes | Yes | 2D | 2K | 3K | 512 | 2 x RTX 3090 | 96 |
| | #4 | ResNet-50 | 640 | Yes | No | 2D | 2K | 3K | 512 | 1 x RTX 3090 | 48 |
| IPSC | Early-Stage | VIT-B | 640 | Yes | No | 2D | 2K | 3K | 512 | 1 x RTX 3090 | 4 |
| | Late-Stage | VIT-B | 640 | Yes | No | 2D | 2K | 3K | 512 | 1 x RTX 3090 | 4 |
| UA-DETRAC | | ResNet-50 | 640 | Yes | No | 2D | 2K | 3K | 512 | 3 x GTX 1080 Ti | 60 |

**P2S-VID**

| Dataset | | *N* | Video Architecture | Backbone | Input Size | Frozen Backbone | Class Equalization | Coord Tokens | *H* | *V* | *L* | GPUs | *B* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ACAD | #1 | 2 | Middle Fusion | ResNet-50 | 640 | Yes | Yes | 2D | 2K | 3K | 512 | 2 x RTX 3090 | 64 |
| | #3 | 2 | Middle Fusion | ResNet-50 | 640 | Yes | Yes | 2D | 2K | 3K | 512 | 2 x RTX 3090 | 64 |
| | #4 | 2 | Middle Fusion | ResNet-50 | 640 | Yes | No | 2D | 2K | 3K | 512 | 2 x RTX 3090 | 72 |
| IPSC | Early-Stage | 6 | Late Fusion | ResNet-50 | 640 | Yes | No | 2D | 2K | 3K | 512 | 2 x RTX 3090 | 16 |
| | Late-Stage | 8 | Middle Fusion | ResNet-50 | 640 | Yes | Yes | 2D | 2K | 3K | 512 | 3 x GTX 1080 Ti | 12 |
| UA-DETRAC | | 2 | Middle Fusion | ResNet-50 | 640 | Yes | No | 2D | 2K | 3K | 512 | 2 x RTX 3090 | 80 |
| | | 4 | Middle Fusion | ResNet-50 | 640 | Yes | No | 2D | 2K | 3K | 512 | 2 x RTX 3090 | 40 |
| | | 8 | Middle Fusion | ResNet-50 | 640 | Yes | No | 2D | 2K | 3K | 2048 | 2 x RTX 3090 | 24 |
| | | 16 | Middle Fusion | ResNet-50 | 640 | Yes | No | 2D | 2K | 3K | 3072 | 6 x RTX 3090 | 6 |
| | | 32 | Middle Fusion | ResNet-50 | 640 | Yes | No | 2D | 2K | 3K | 5120 | 6 x RTX 3090 | 6 |


polygon generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 18 653–18 663.

[32] C. Zhu, Y. Zhou, Y. Shen, G. Luo, X. Pan, M. Lin, C. Chen, L. Cao, X. Sun, and R. Ji, "SeqTR: A simple yet universal network for visual grounding," in *European Conference on Computer Vision*. Springer, 2022, pp. 598–615.

[33] Z. Yang, Z. Gan, J. Wang, X. Hu, F. Ahmed, Z. Liu, Y. Lu, and L. Wang, "Unitab: Unifying text and box outputs for grounded vision-language modeling," in *European Conference on Computer Vision*. Springer, 2022, pp. 521–539.

[34] S. H. Rezatofighi, V. K. Bg, A. Milan, E. Abbasnejad, A. Dick, and I. Reid, "Deepsetnet: Predicting sets with deep neural networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 5257–5266.

[35] L. Pineda, A. Salvador, M. Drozdzal, and A. Romero, "Elucidating Image-to-Set Prediction: An Analysis of Models, Losses and Datasets," *arXiv preprint arXiv:1904.05709*, 2019.

[36] Y. Xue, J. Mao, M. Niu, H. Xu, M. B. Mi, W. Zhang, X. Wang, and X. Wang, "Point2Seq: Detecting 3D Objects as Sequences," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8511–8520, 2022.

[37] Z. Chen, Y. Zhu, Z. Li, F. Yang, W. Li, H. Wang, C. Zhao, L. Wu, R. Zhao, J. Wang *et al.*, "Obj2seq: Formatting objects as sequences with class prompt for visual tasks," *Advances in Neural Information Processing Systems*, vol. 35, pp. 2494–2506, 2022.

[38] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-End Object Detection with Transformers," *ECCV*, 2020.

[39] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable Transformers for End-to-End Object Detection," *ICLR*, 2021.

[40] A. Kolesnikov, A. Susano Pinto, L. Beyer, X. Zhai, J. Harmsen, and N. Houlsby, "UViM: A unified modeling approach for vision with learned guiding codes," *Advances in Neural Information Processing Systems*, vol. 35, pp. 26 295–26 308, 2022.

[41] J. Ning, C. Li, Z. Zhang, C. Wang, Z. Geng, Q. Dai, K. He, and H. Hu, "All in tokens: Unifying output space of visual tasks via soft token," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 19 900–19 910.

[42] G. Țucudean, M. Bucos, B. Drăgulescu, and C. D. Căleanu, "Natural Language Processing with Transformers: A Review," *PeerJ Computer Science*, vol. 10, 2024.

[43] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J. Nie, and J. rong Wen, "A Survey of Large Language Models," *ArXiv*, vol. abs/2303.18223, 2023.

[44] S. Minaee, T. Mikolov, N. Nikzad, M. A. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large Language Models: A Survey," *ArXiv*, vol. abs/2402.06196, 2024.

[45] A. Radford and K. Narasimhan, "Improving Language Understanding by Generative Pre-Training," 2018.

[46] M. Fujitake and A. Sugimoto, "Video sparse transformer with attention-guided memory for video object detection," *IEEE Access*, vol. 10, pp. 65 886–65 900, 2022.

[47] H. Wang, J. Tang, X. Liu, S. Guan, R. Xie, and L. Song, "PTSEFormer: Progressive Temporal-Spatial Enhanced TransFormer Towards Video Object Detection," in *European Conference on Computer Vision*. Springer, 2022, pp. 732–747.

[48] Q. Zhou, X. Li, L. He, Y. Yang, G. Cheng, Y. Tong, L. Ma, and D. Tao, "TransVOD: End-to-end video object detection with spatial-temporal transformers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 6, pp. 7853–7869, 2022.

[49] C. Deng, D. Chen, and Q. Wu, "Identity-Consistent Aggregation for Video Object Detection," *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 13 388–13 398, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:260900188

[50] Q. Qi and X. Wang, "Tgbformer: Transformer-graphformer blender network for video object detection," in *AAAI Conference on Artificial Intelligence*, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:277103597

[51] Y. Chen, Y. Cao, H. Hu, and L. Wang, "Memory enhanced global-local aggregation for video object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 337–10 346.

[52] C. Xu, J. Zhang, M. Wang, G. Tian, and Y. Liu, "Multilevel spatial-temporal feature aggregation for video object detection," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 11, pp.

7809–7820, 2022.
[53] L. Han and Z. Yin, "Global memory and local continuity for video object detection," *IEEE Transactions on Multimedia*, vol. 25, pp. 3681–3693, 2022.
[54] S.-D. Roh and K.-S. Chung, "DAFA: Diversity-Aware Feature Aggregation for Attention-Based Video Object Detection," *IEEE Access*, vol. 10, pp. 93 453–93 463, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252022543
[55] L. Han, P. Wang, Z. Yin, F. Wang, and H. Li, "Class-Aware Feature Aggregation Network for Video Object Detection," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, pp. 8165–8178, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:240730954
[56] K. A. Hashmi, D. Stricker, and M. Z. Afzal, "Spatio-Temporal Learnable Proposals for End-to-End Video Object Detection," in *British Machine Vision Conference*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252715466
[57] F. He, N. Gao, J. Jia, X. Zhao, and K. Huang, "QueryProp: Object Query Propagation for High-Performance Video Object Detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 1, 2022, pp. 834–842.
[58] P. Sun, R. Zhang, Y. Jiang, T. Kong, C. Xu, W. Zhan, M. Tomizuka, L. Li, Z. Yuan, C. Wang *et al.*, "Sparse R-CNN: End-to-end object detection with learnable proposals," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 14 454–14 463.
[59] A. Sabater, L. Montesano, and A. C. Murillo, "Robust and efficient post-processing for video object detection," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 536–10 542.
[60] Y. Shi, N. Wang, and X. Guo, "YOLOV: Making still image object detectors great at video object detection," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, no. 2, 2023, pp. 2254–2262.
[61] Y. Shi, T. Zhang, and X. Guo, "Practical Video Object Detection via Feature Selection and Aggregation," *arXiv preprint arXiv:2407.19650*, 2024.
[62] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430*, 2021.
[63] B. Ehteshami Bejnordi, A. Habibian, F. Porikli, and A. Ghodrati, "SALISA: Saliency-based input sampling for efficient video object detection," in *European Conference on Computer Vision*. Springer, 2022, pp. 300–316.
[64] 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*, 2020, pp. 10 781–10 790.
[65] A. Singh, "Video Detection and Segmentation with Language Modeling," online: https://webdocs.cs.ualberta.ca/~asingh1/p2s/.
[66] L. Wen, D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang, and S. Lyu, "UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking," *CVIU*, vol. 193, p. 102907, 2020.
[67] A. Singh, I. Jasra, O. Mouhammed, N. Dadheech, N. Ray, and J. Shapiro, "Towards Early Prediction of Human iPSC Reprogramming Success," *Machine Learning for Biomedical Imaging*, vol. 2, pp. 390–407, 2023. [Online]. Available: https://melba-journal.org/2023:014
[68] K. Kang, W. Ouyang, H. Li, and X. Wang, "Object Detection from Video Tubelets with Convolutional Neural Networks," *CVPR*, pp. 817–825, 2016.
[69] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CVPR*, pp. 770–778, 2016.
[70] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, "Video Swin Transformer," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3192–3201, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235624247
[71] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "SlowFast Networks for Video Recognition," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6201–6210, 2018.
[72] M. Innat, "Keras 3 Implementation of Video Swin Transformers for 3D Video Modeling," online: https://github.com/innat/VideoSwin, December 2024.
[73] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, A. Natsev, M. Suleyman, and A. Zisserman, "The Kinetics Human Action Video Dataset," *ArXiv*, vol. abs/1705.06950, 2017.
[74] A. Singh, M. Pietrasik, G. Natha, N. Ghouaiel, K. Brizel, and N. Ray, "Animal Detection in Man-made Environments," *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1427–1438, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:204900826
[75] "Object Detection on UA-DETRAC," online: https://paperswithcode.com/sota/object-detection-on-ua-detrac.
[76] M. Fujitake and A. Sugimoto, "Temporal feature enhancement network with external memory for object detection in surveillance video," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 7684–7691.
[77] K.-J. Kim, P.-K. Kim, Y.-S. Chung, and D.-H. Choi, "Performance enhancement of YOLOv3 by adding prediction layers with spatial pyramid pooling for vehicle detection," in *2018 15th IEEE international conference on advanced video and signal based surveillance (AVSS)*. IEEE, 2018, pp. 1–6.
[78] ——, "Multi-scale detector for accurate vehicle detection in traffic surveillance data," *IEEE Access*, vol. 7, pp. 78 311–78 319, 2019.
[79] H. Perreault, G.-A. Bilodeau, N. Saunier, and M. Héritier, "SpotNet: Self-attention multi-task network for object detection," in *2020 17th Conference on Computer and Robot Vision (CRV)*. IEEE, 2020, pp. 230–237.
[80] ——, "FFAVOD: Feature fusion architecture for video object detection," *Pattern Recognition Letters*, vol. 151, pp. 294–301, 2021.

**ABHINEET SINGH** received the B.Tech. degree in information technology from IIIT Allahabad, Prayagraj, India, in 2013, the M.Sc. degree in computing science from the University of Alberta, Edmonton, AB, Canada, in 2017 and the Ph.D. degree from the same department in 2025.

He is currently working as a machine learning developer for an Edmonton-based agricultural automation company named Mojow Autonomous Solutions. His research interests include computer vision and machine learning in general and application of deep learning for object detection, tracking, and segmentation in particular.

**NILANJAN RAY** received Bachelor of Mechanical Engineering from Jadavpur University, Calcutta, India, in 1995, M.Tech. in Computer Science from Indian Statistical Institute, Calcutta, in 1997, and Ph.D. in Electrical Engineering from the University of Virginia, USA, in 2003. After having two years of postdoctoral research and a year of industrial work experience he joined the Department of Computing Science, University of Alberta in 2006, where, currently, he is a full professor.

Nilanjan's research is in computer vision, image analysis and visual recognition with deep learning. He is interested in medical imaging and general computer vision applications including classification, recognition, semantic segmentation, object tracking, image registration and motion detection. He has published over 150 articles in these areas.

Nilanjan's professional activities include serving as a General Co-chair for AI/GI/CRV conference in 2017, Associate Editor for IEEE Transactions on Image Processing (2013-2017), IET Image Processing (2016-2021), BMVC 2022 PC member, Editorial board member of Discover Imaging, 2024-.