Putting the Object Back into Video Object Segmentation

Ho Kei Cheng[1] Seoung Wug Oh[2] Brian Price[2] Joon-Young Lee[2] Alexander Schwing[1]
[1]University of Illinois Urbana-Champaign [2]Adobe Research
{hokeikc2,aschwing}@illinois.edu, {seoh,bprice,jolee}@adobe.com

Abstract

*We present Cutie, a video object segmentation (VOS) network with object-level memory reading, which puts the object representation from memory back into the video object segmentation result. Recent works on VOS employ bottom-up pixel-level memory reading which struggles due to matching noise, especially in the presence of distractors, resulting in lower performance in more challenging data. In contrast, Cutie performs top-down object-level memory reading by adapting a small set of object queries. Via those, it interacts with the bottom-up pixel features iteratively with a **q**uery-based object **t**ransformer (**qt**, hence Cutie). The object queries act as a high-level summary of the target object, while high-resolution feature maps are retained for accurate segmentation. Together with foreground-background masked attention, Cutie cleanly separates the semantics of the foreground object from the background. On the challenging MOSE dataset, Cutie improves by 8.7 $\mathcal{J}\&\mathcal{F}$ over XMem with a similar running time and improves by 4.2 $\mathcal{J}\&\mathcal{F}$ over DeAOT while being three times faster. Code is available at:* hkchengrex.github.io/Cutie.

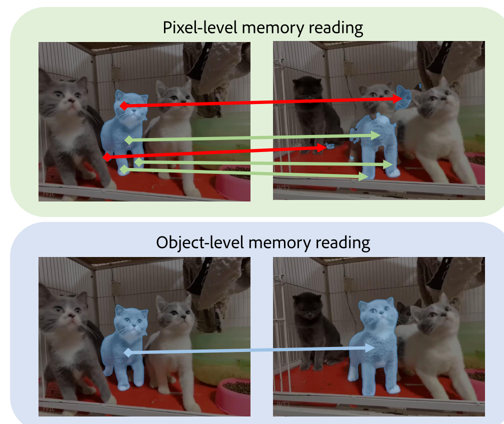

Figure 1. Comparison of pixel-level memory reading v.s. object-level memory reading. In each box, the left is the reference frame, and the right is the query frame to be segmented. Red arrows indicate wrong matches. Low-level pixel matching (e.g., XMem [9]) can be noisy in the presence of distractors. We propose object-level memory reading for more robust video object segmentation.

1. Introduction

Video Object Segmentation (VOS), specifically the "semi-supervised" setting, requires tracking and segmenting objects from an open vocabulary specified in a first-frame annotation. VOS methods are broadly applicable in robotics [1], video editing [2], reducing costs in data annotation [3], and can also be combined with Segment Anything Models (SAMs) [4] for universal video segmentation (e.g., Tracking Anything [5–7]).

Recent VOS approaches employ a memory-based paradigm [8–11]. A memory representation is computed from past segmented frames (either given as input or segmented by the model), and any new query frame "reads" from this memory to retrieve features for segmentation. Importantly, these approaches mainly use *pixel-level matching* for memory reading, either with one [8] or multiple matching layers [10], and generate the segmentation bottom-up from the pixel memory readout. Pixel-level matching maps every query pixel independently to a linear combination of memory pixels (e.g., with an attention layer). Consequently, pixel-level matching lacks high-level consistency and is prone to matching noise, especially in the presence of distractors. This leads to lower performance in challenging scenarios with occlusions and frequent distractors. Concretely, the performance of recent approaches [9, 10] is more than 20 points in $\mathcal{J}\&\mathcal{F}$ lower when evaluating on the recently proposed challenging MOSE [12] dataset rather than the simpler DAVIS-2017 [13] dataset.

We think this unsatisfactory result in challenging scenarios is caused by the lack of object-level reasoning. To address this, we propose *object-level memory reading*, which effectively puts the object from a memory back into the query frame (Figure 1). Inspired by recent query-based object detection/segmentation [14–18] that represent objects as "object queries," we implement our object-level memory reading with an object transformer. This object transformer uses a small set of end-to-end trained object queries to 1) iteratively probe and calibrate a feature map (initialized by a pixel-level memory readout), and 2) encode object-level information. This approach simultaneously keeps a high-level/global object query representation and a low-level/high-resolution feature map, enabling bidirectional top-down/bottom-up communication. This communication is parameterized with a sequence of attention layers, including a proposed *foreground-background masked attention*. The masked attention, extended from foreground-only masked attention [15], lets part of the object queries attend only to the foreground while the remainders attend only to the background – allowing both global feature interaction and clean separation of foreground/background semantics. Moreover, we introduce a compact *object memory* (in addition to a pixel memory) to summarize the features of target objects, enhancing end-to-end object queries with target-specific features.

In experiments, the proposed approach, *Cutie*, is significantly more robust in challenging scenarios (e.g., +8.7 $\mathcal{J}\&\mathcal{F}$ in MOSE [12] over XMem [9]) than existing approaches while remaining competitive in standard datasets (i.e., DAVIS [13] and YouTubeVOS [19]) in both accuracy and efficiency. In summary,

- We develop *Cutie*, which uses high-level top-down queries with pixel-level bottom-up features for robust video object segmentation in challenging scenarios.
- We extend masked attention to include foreground *and* background for both rich features and a clean semantic separation between the target object and distractors.
- We construct a compact *object memory* to summarize object features in the long term, which are retrieved as target-specific object-level representations during querying.

2. Related Works

Memory-Based VOS. Since semi-supervised Video Object Segmentation (VOS) involves a directional propagation of information, many existing approaches employ a feature memory representation that stores past features for segmenting future frames. This includes online learning that finetunes a network on the first-frame segmentation for every video during inference [20–24]. However, finetuning is slow during test-time. Recurrent approaches [25–31] are faster but lack context for tracking under occlusion. Recent approaches use more context [5, 8, 11, 32–64] via pixel-level feature matching and integration, with some exploring the modeling of background features – either explicitly [36, 65] or implicitly [51]. XMem [9] uses multiple types of memory for better performance and efficiency but still struggles with noise from low-level pixel matching. While we adopt the memory reading of XMem [9], we develop an object reading mechanism to integrate the pixel features at an object level which permits Cutie to attain much better performance in challenging scenarios.

Transformers in VOS. Transformer-based [66] approaches have been developed for pixel matching with memory in video object segmentation [10, 50, 53, 67–70]. However, they compute attention between spatial feature maps (as cross-attention, self-attention, or both), which is computationally expensive with $O(n^4)$ time/space complexity, where n is the image side length. SST [67] proposes sparse attention but performs worse than state-of-the-art methods. AOT approaches [10, 68] use an identity bank for processing multiple objects in a single forward pass to improve efficiency, but are not permutation equivariant with respect to object ID and do not scale well to longer videos. Concurrent approaches [69, 70] use a single vision transformer network to jointly model the reference frames and the query frame without explicit memory reading operations. They attain high accuracy but require large-scale pretraining (e.g., MAE [71]) and have a much lower inference speed (< 4 frames per second). Cutie is carefully designed to *not* compute any (costly) attention between spatial feature maps in our object transformer while facilitating efficient global communication via a small set of object queries – allowing Cutie to be real-time.

Object-Level Reasoning. Early VOS algorithms [59, 72, 73] that attempt to reason at the object level use either re-identification or k-means clustering to obtain object features and have a lower performance on standard benchmarks. HODOR [18], and its follow-up work TarViS [17], approach VOS with object-level descriptors which allow for greater flexibility (e.g., training on static images only [18] or extending to different video segmentation tasks [17, 74, 75]) but fall short on VOS segmentation accuracy (e.g., [74] is 6.9 $\mathcal{J}\&\mathcal{F}$ behind state-of-the-art methods in DAVIS 2017 [13]) due to under-using high-resolution features. ISVOS [76] proposes to inject features from a pre-trained instance segmentation network (i.e., Mask2Former [15]) into a memory-based VOS method [51]. Cutie has a similar motivation but is crucially different in three ways: 1) Cutie learns object-level information end-to-end, without needing to pre-train on instance segmentation tasks/datasets, 2) Cutie allows bi-directional communication between pixel-level features and object-level features for an integrated framework, and 3) Cutie is a one-stage method that does not perform separate instance segmentation while ISVOS does – this allows Cutie to run six times (estimated) faster. Moreover, ISVOS does not release code while we open source code for the community which facilitates follow-up work.

Automatic Video Segmentation. Recently, video object segmentation methods have been used as an integral component in automatic video segmentation pipelines, such as open-vocabulary/universal video segmentation (e.g., Tracking Anything [5, 6], DEVA [7]) and unsupervised video segmentation [77]. We believe the robustness and efficiency of Cutie are beneficial for these applications.

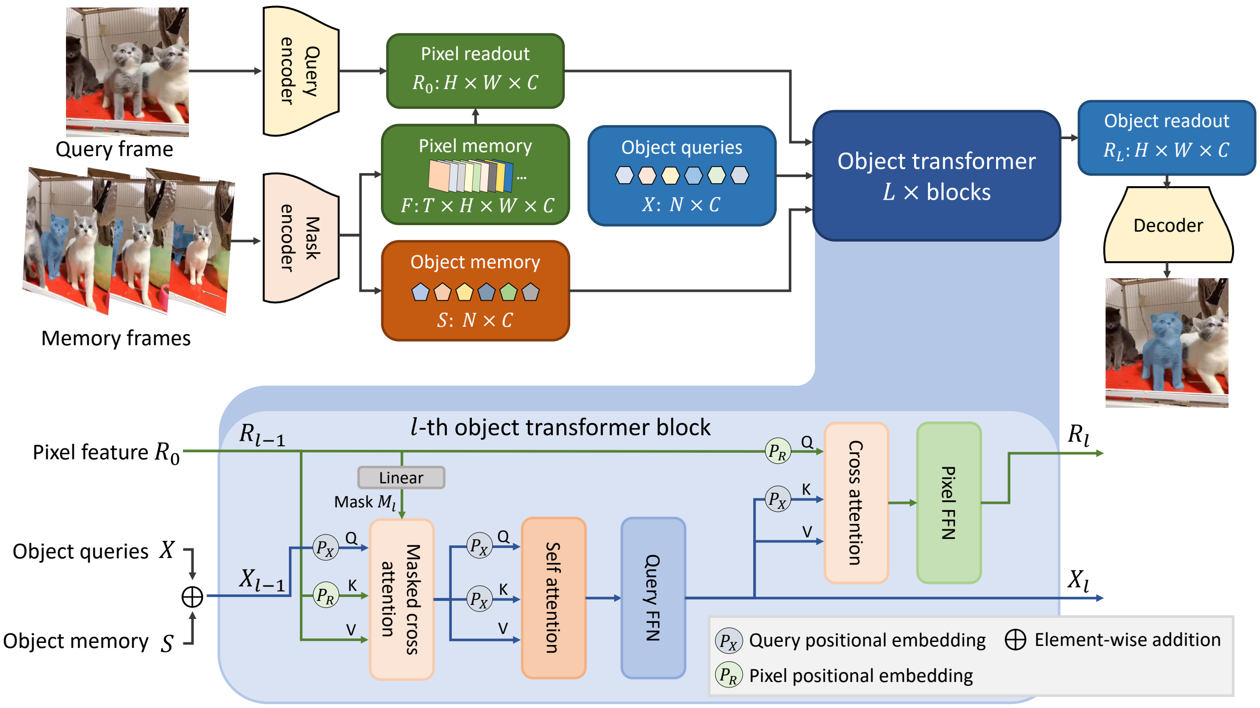

Figure 2. Overview of Cutie. We store pixel memory F and object memory S representations from past segmented (memory) frames. Pixel memory is retrieved for the query frame as pixel readout R_0, which bidirectionally interacts with object queries X and object memory S in the object transformer. The L object transformer blocks enrich the pixel feature with object-level semantics and produce the final R_L object readout for decoding into the output mask. Standard residual connections, layer normalization, and skip-connections from the query encoder to the decoder are omitted for readability.

3. Cutie

3.1. Overview

We provide an overview of Cutie in Figure 2. For readability, following prior works [8, 9], we consider a single target object as the extension to multiple objects is straightforward (see supplement). Following the standard semi-supervised video object segmentation (VOS) setting, Cutie takes a first-frame segmentation of target objects as input and segments subsequent frames sequentially in a streaming fashion. First, Cutie encodes segmented frames (given as input or segmented by the model) into a high-resolution pixel memory F (Section 3.4.1) and a high-level object memory S (Section 3.3) and stores them for segmenting future frames. To segment a new query frame, Cutie retrieves an initial pixel readout R_0 from the pixel memory using encoded query features. This initial readout R_0 is computed via low-level pixel matching and is therefore often noisy. We enrich it with object-level semantics by augmenting R_0 with information from the object memory S and a set of object queries X through an *object transformer* with L transformer blocks (Section 3.2). The enriched output of the object transformer, R_L, or the object readout, is passed to the decoder for generating the final output mask. In the following, we will first describe the three main contributions of Cutie: object transformer, masked attention, and object memory. Note, we derive the pixel memory from existing works [9], which we only describe as implementation details in Section 3.4.1 without claiming any contribution.

3.2. Object Transformer

3.2.1 Overview

The bottom of Figure 2 illustrates the object transformer. The object transformer takes an initial readout $R_0 \in \mathbb{R}^{HW \times C}$, a set of N end-to-end trained object queries $X \in \mathbb{R}^{N \times C}$, and object memory $S \in \mathbb{R}^{N \times C}$ as input, and integrates them with L transformer blocks. Note H and W are image dimensions after encoding with stride 16. Before the first block, we sum the static object queries with the dynamic object memory for better adaptation, i.e., $X_0 = X + S$. Each transformer block bidirectionally allows the object queries X_{l-1} to attend to the readout R_{l-1}, and vice versa, producing updated queries X_l and readout R_l as the output of the l-th block. The last block's readout, R_L, is the final output of the object transformer.

Within each block, we first compute masked cross-attention, letting the object queries X_{l-1} read from the pixel features R_{l-1}. The masked attention focuses half of the object queries on the foreground region while the other half is targeted towards the background (details in Section 3.2.2). Then, we pass the object queries into standard self-attention and feed-forward layers [66] for object-level reasoning. Next, we update the pixel features with a reversed cross-attention layer, *putting the object* semantics from object queries X_l *back* into pixel features R_{l-1}. We then pass the pixel features into a feed-forward network while skipping the computationally expensive self-attention in a standard transformer [66]. Throughout, positional embeddings are added to the queries and keys following [14, 15] (Section 3.2.3). Residual connections and layer normalizations are used in every attention and feed-forward layer following [78]. All attention layers are implemented with multi-head scaled dot product attention [66]. Importantly,

1. We carefully avoid any direct attention between high-resolution spatial features (e.g., R), as they are intensive in both memory and compute. Despite this, these spatial features can still interact globally via object queries, making each transformer block efficient and expressive.
2. The object queries restructure the pixel features with a residual contribution without discarding the high-resolution pixel features. This avoids irreversible dimensionality reductions (would be over $100\times$) and keeps those high-resolution features for accurate segmentation.

Next, we describe the core components in our object transformer blocks: foreground/background masked attention and the construction of the positional embeddings.

3.2.2 Foreground-Background Masked Attention

In our (pixel-to-query) cross-attention, we aim to update the object queries $X_l \in \mathbb{R}^{N\times C}$ by attending over the pixel features $R_l \in \mathbb{R}^{HW\times C}$. Standard cross-attention with the residual path finds

$$X_l' = A_l V_l + X_l = \text{softmax}(Q_l K_l^T) V_l + X_l, \tag{1}$$

where Q_l is a learned linear transformation of X_l, and K_l, V_l are learned linear transformations of R_l. The rows of the affinity matrix $A_l \in \mathbb{R}^{N\times HW}$ describe the attention of each object query over the entire feature map. We note that there are distinctly different attention patterns for different object queries – some focus on different foreground parts, some on the background, and some on distractors (top of Figure 3). These object queries collect information from different regions of interest and integrate them in subsequent self-attention/feed-forward layers. However, the soft nature of attention makes this process noisy and less reliable – queries that mainly attend to the foreground might have small weights distributed in the background and vice versa. Inspired by [15], we deploy masked attention to aid the clean separation of semantics between foreground and background. Different from [15], which only attends to the foreground, we find it helpful to also attend to the background, especially in challenging tracking scenarios with distractors. In practice, we let the first half of the object queries (i.e., foreground queries) always attend to the foreground and the second half (i.e., background queries) attend to the background. This masking is shared across all attention heads.

Formally, our foreground-background masked cross-attention finds

$$X_l' = \text{softmax}(\mathcal{M}_l + Q_l K_l^T) V_l + X_l, \tag{2}$$

where $\mathcal{M}_l \in \{0, -\infty\}^{N\times HW}$ controls the attention masking – specifically, $\mathcal{M}_l(q,i)$ determines whether the q-th query is allowed ($=0$) or not allowed ($=-\infty$) to attend to the i-th pixel. To compute $\mathcal{M}_l$, we first find a mask prediction at the current layer M_l, which is linearly projected from the last pixel feature R_{l-1} and activated with the sigmoid function. Then, $\mathcal{M}_l$ is computed as

$$\mathcal{M}_l(q,i) = \begin{cases} 0, & \text{if } q \le N/2 \text{ and } M_l(i) \ge 0.5 \\ 0, & \text{if } q > N/2 \text{ and } M_l(i) < 0.5 \\ -\infty, & \text{otherwise} \end{cases}, \tag{3}$$

where the first case is for foreground attention and the second is for background attention. Figure 3 (bottom) visualizes the attention maps after this foreground-background masking. Note, despite the hard foreground-background separation, the object queries communicate in the subsequent self-attention layer for potential global feature interaction. Next, we discuss the positional embeddings used in object queries and pixel features that allow location-based attention.

3.2.3 Positional Embeddings

Since vanilla attention operations are permutation equivariant, positional embeddings are used to provide additional features about the position of each token [66]. Following prior transformer-based vision networks [14, 15], we add the positional embedding to the query and key features at every attention layer (Figure 2), and not to the value.

For the object queries, we use a positional embedding $P_X \in \mathbb{R}^{N\times C}$ that combines an end-to-end learnable embedding $E_X \in \mathbb{R}^{N\times C}$ and the dynamic object memory $S \in \mathbb{R}^{N\times C}$ via

$$P_X = E_X + f_{\text{ObjEmbed}}(S), \tag{4}$$

where f_{ObjEmbed} is a trainable linear projection.

For the pixel feature, the positional embedding $P_R \in \mathbb{R}^{HW\times C}$ combines a fixed 2D sinusoidal positional embedding R_{sin} [14] that encodes absolute pixel coordinates and the initial readout $R_0 \in \mathbb{R}^{HW\times C}$ via

$$P_R = R_{\text{sin}} + f_{\text{PixEmbed}}(R_0), \tag{5}$$

where f_{PixEmbed} is another trainable linear projection. Note that the sinusoidal embedding R_{sin} operates on normalized coordinates and is scaled accordingly to different image sizes at test time.

Figure 3. Visualization of cross-attention weights (rows of A_L) in the object transformer. The middle cat is the target object. Top: without foreground-background masking – some queries mix semantics from foreground and background (framed in red). Bottom: with foreground-background masking. The leftmost three are foreground queries, and the rightmost three are background queries. Semantics is thus cleanly separated. The f.g./b.g. queries can communicate in the subsequent self-attention layer. Note the queries attend to different foreground regions, distractors, and background regions.

3.3. Object Memory

In the object memory $S \in \mathbb{R}^{N\times C}$, we store a compact set of N vectors which make up a high-level summary of the target object. This object memory is used in the object transformer (Section 3.2) to provide target-specific features. At a high level, we compute S by mask-pooling over all encoded object features with N different masks. Concretely, given object features $U \in \mathbb{R}^{THW\times C}$ and N pooling masks $\{W_q \in [0,1]^{THW}, 0 < q \leq N\}$, where T is the number of memory frames, the q-th object memory $S_q \in \mathbb{R}^{C}$ is computed by

$$S_q = \frac{\sum_{i=1}^{THW} U(i)W_q(i)}{\sum_{i=1}^{THW} W_q(i)}. \tag{6}$$

During inference, we use a classic streaming average algorithm such that this operation takes constant time and memory with respect to the video length. See the supplement for details. Note, an object memory vector S_q would not be modified if the corresponding pooling weights are zero, i.e., $\sum_{i=1}^{HW} W_q^t(i) = 0$, preventing feature drifting when the corresponding object region is not visible (e.g., occluded).

To find U and W for a memory frame, we first encode the corresponding image I and the segmentation mask M with the mask encoder for memory feature $F \in \mathbb{R}^{THW\times C}$. We use a 2-layer, C-dimensional MLP f_{ObjFeat} to obtain the object feature U via

$$U = f_{\text{ObjFeat}}(F). \tag{7}$$

For the N pooling masks $\{W_q \in [0,1]^{THW}, 0 < q \leq N\}$, we additionally apply foreground-background separation as detailed in Section 3.2.2 and augment it with a fixed 2D sinusoidal positional embedding R_{sin} (as mentioned in Section 3.2.3). The separation allows it to aggregate clean semantics during pooling, while the positional embedding enables location-aware pooling. Formally, we compute the i-th pixel of the q-th pooling mask via

$$W_q(i) = \begin{cases} 0, & \text{if } q \leq N/2 \text{ and } M(i) < 0.5 \\ 0, & \text{if } q > N/2 \text{ and } M(i) \geq 0.5 \\ \sigma(f_{\text{PoolWeight}}(F(i) + R_{\text{sin}}(i))), & \text{otherwise} \end{cases}, \tag{8}$$

where σ is the sigmoid function, $f_{\text{PoolWeight}}$ is a 2-layer, N-dimensional MLP, and the segmentation mask M is downsampled to match the feature stride of F.

3.4. Implementation Details

3.4.1 Pixel Memory

Our pixel memory design, which provides the pixel feature R_0 (see Figure 2), is derived from XMem [5, 9] working and sensory memory. We do not claim contributions. Here, we present the high-level algorithm and defer details to the supplementary material. The pixel memory is composed of an attentional component (with keys $\mathbf{k} \in \mathbb{R}^{THW\times C^k}$ and values $\mathbf{v} \in \mathbb{R}^{THW\times C}$) and a recurrent component (with hidden state $\mathbf{h}^{HW\times C}$). Long-term memory [9] can be optionally included in the attentional component without re-training for better performance on long videos. The keys and values consist of low-level appearance features for matching while the hidden state provides temporally consistent features. To retrieve a pixel readout R_0, we first encode the query frame to obtain query feature $\mathbf{q}^{HW\times C}$, and compute the query-to-memory affinity $A^{\text{pix}} \in [0,1]^{HW\times THW}$ via

$$A^{\text{pix}}_{ij} = \frac{\exp\left(d(\mathbf{q}_i, \mathbf{k}_j)\right)}{\sum_m \exp\left(d(\mathbf{q}_i, \mathbf{k}_m)\right)}, \tag{9}$$

where $d(\cdot,\cdot)$ is the anisotropic L2 function [9] which is proportional to the similarity between the two inputs. Finally, we find the pixel readout R_0 by combining the attention readout with the hidden state:

$$R_0 = f_{\text{fuse}}\left(A^{\text{pix}}\mathbf{v} + \mathbf{h}\right), \tag{10}$$

where f_{fuse} is a small network consisting of two C-dimension convolutional residual blocks with channel attention [79].

3.4.2 Network Architecture

We study two model variants: 'small' and 'base' with different query encoder backbones, otherwise sharing the same configuration: $C = 256$ channels with $L = 3$ object transformer blocks and $N = 16$ object queries.

ConvNets. We parameterize the query encoder and the mask encoder with ResNets [80]. Following [8, 9], we discard the last convolutional stage and use the stride 16 feature. For the query encoder, we use ResNet-18 for the small model and ResNet-50 for the base model. For the mask encoder, we use ResNet-18. 'Cutie-base' thus shares the same backbone configuration as XMem. We find that Cutie works well with a lighter decoder – we use a similar iterative upsampling architecture as in XMem but halve the number of channels in all upsampling blocks for better efficiency.

Feed-Forward Networks (FFN). We use query FFN and pixel FFN in our object transformer block (Figure 2). For the query FFN, we use a 2-layer MLP with a hidden size of $8C = 2048$. For the pixel FFN, we use two 3×3 convolutions with a smaller hidden size of $C = 256$ to reduce computation. As we do not use self-attention on the pixel features, we compensate by using efficient channel attention [79] after the second convolution of the pixel FFN. Layer normalizations are applied to the query FFN following [78] and not to the pixel FFN, as we observe no empirical benefits. ReLU is used as the activation function.

3.4.3 Training

Data. Following [8–10], we first pretrain our network on static images [81–85] by generating three-frame sequences with synthetic deformation. Next, we perform the main training on video datasets DAVIS [13] and YouTubeVOS [19] by sampling eight frames following [9]. We optionally also train on MOSE [12] (combined with DAVIS and YouTubeVOS), as we notice the training sets of YouTubeVOS and DAVIS have become too easy for our model to learn from (>93% IoU during training). For every setting, we use one trained model and do not finetune for specific datasets. We additionally introduce a 'MEGA' setting with BURST [3] and OVIS [86] included in training (+1.6 $\mathcal{J}\&\mathcal{F}$ in MOSE). Details are provided in the supplementary material.

Optimization. We use the AdamW [87] optimizer with a learning rate of $1\text{e}{-4}$, a batch size of 16, and a weight decay of 0.001. Pretraining lasts for 80K iterations with no learning rate decay. Main training lasts for 125K iterations, with the learning rate reduced by 10 times after 100K and 115K iterations. The query encoder has a learning rate multiplier of 0.1 following [5, 10, 15] to mitigate overfitting. Following the bag of tricks from DEVA [5], we clip the global gradient norm to 3 throughout and use stable data augmentation. The entire training process takes approximately 30 hours on four A100 GPUs for the small model.

Losses. Following [15], we adopt point supervision which computes the loss only at K sampled points instead of the whole mask. We use importance sampling [88] and set $K = 8192$ during pretraining and $K = 12544$ during main training. We use a combined loss function of cross-entropy and soft dice loss with equal weighting following [5, 9, 10]. In addition to the loss applied to the final segmentation output, we adopt auxiliary losses in the same form (scaled by 0.01) to the intermediate masks M_l in the object transformer.

3.4.4 Inference

During testing, we encode a memory frame for updating the pixel memory and the object memory every r-th frame. r defaults to 5 following [9]. For the keys $\mathbf{k}$ and values $\mathbf{v}$ in the attention component of the pixel memory, we always keep features from the first frame (as it is given by the user) and use a First-In-First-Out (FIFO) approach for other memory frames to ensure the total number of memory frames T is less than or equal to a pre-defined limit $T_{\max} = 5$. For processing long videos (e.g., BURST [3] or LVOS [89] with over a thousand frames per video), we use the long-term memory [9] instead of FIFO without re-training, following the default parameters in [9]. For the pixel memory, we use top-k filtering [2] with $k = 30$. Inference is fully online, can be streamed, and uses a constant amount of compute per frame and memory with respect to the sequence length.

4. Experiments

For evaluation, we use standard metrics: Jaccard index $\mathcal{J}$, contour accuracy $\mathcal{F}$, and their average $\mathcal{J}\&\mathcal{F}$ [13]. In YouTubeVOS [19], $\mathcal{J}$ and $\mathcal{F}$ are computed for "seen" and "unseen" categories separately. $\mathcal{G}$ is the averaged $\mathcal{J}\&\mathcal{F}$ for both seen and unseen classes. For BURST [3], we assess Higher Order Tracking Accuracy (HOTA) [90] on common and uncommon object classes separately. For our models, unless otherwise specified, we resize the inputs such that the shorter edge has no more than 480 pixels and rescale the model's prediction back to the original resolution.

4.1. Main Results

We compare with several state-of-the-art approaches on recent standard benchmarks: DAVIS 2017 validation/test-dev [13] and YouTubeVOS validation [19]. To assess the robustness of VOS algorithms, we also report results on MOSE validation [12], which contains heavy occlusions and crowded environments for evaluation. DAVIS 2017 [13] contains annotated videos at 24 frames per second (fps), while YouTubeVOS contains videos at 30fps but is only annotated

Method	MOSE			DAVIS-17 val			DAVIS-17 test			YouTubeVOS-2019 val					
	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$	$\mathcal{F}$	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$	$\mathcal{F}$	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$	$\mathcal{F}$	$\mathcal{G}$	$\mathcal{J}_s$	$\mathcal{F}_s$	$\mathcal{J}_u$	$\mathcal{F}_u$	FPS
Trained without MOSE															
STCN [51]	52.5	48.5	56.6	85.4	82.2	88.6	76.1	72.7	79.6	82.7	81.1	85.4	78.2	85.9	13.2
AOT-R50 [10]	58.4	54.3	62.6	84.9	82.3	87.5	79.6	75.9	83.3	85.3	83.9	88.8	79.9	88.5	6.4
RDE [55]	46.8	42.4	51.3	84.2	80.8	87.5	77.4	73.6	81.2	81.9	81.1	85.5	76.2	84.8	24.4
XMem [9]	56.3	52.1	60.6	86.2	82.9	89.5	81.0	77.4	84.5	85.5	84.3	88.6	80.3	88.6	22.6
DeAOT-R50 [68]	59.0	54.6	63.4	85.2	82.2	88.2	80.7	76.9	84.5	85.6	84.2	89.2	80.2	88.8	11.7
SimVOS-B [69]	-	-	-	81.3	78.8	83.8	-	-	-	-	-	-	-	-	3.3
JointFormer [70]	-	-	-	-	-	-	65.6	61.7	69.4	73.3	75.2	78.5	65.8	73.6	3.0
ISVOS [76]	-	-	-	80.0	76.9	83.1	-	-	-	-	-	-	-	-	5.8*
DEVA [5]	60.0	55.8	64.3	86.8	83.6	90.0	82.3	78.7	85.9	85.5	85.0	89.4	79.7	88.0	25.3
Cutie-small	62.2	58.2	66.2	87.2	84.3	90.1	84.1	80.5	87.6	**86.2**	85.3	89.6	**80.9**	**89.0**	**45.5**
Cutie-base	**64.0**	**60.0**	**67.9**	**88.8**	**85.4**	**92.3**	**84.2**	**80.6**	**87.7**	86.1	**85.5**	**90.0**	80.6	88.3	36.4
Trained with MOSE															
XMem [9]	59.6	55.4	63.7	86.0	82.8	89.2	79.6	76.1	83.0	85.6	84.1	88.5	81.0	88.9	22.6
DeAOT-R50 [68]	64.1	59.5	68.7	86.0	83.1	88.9	82.8	79.1	86.5	85.3	84.2	89.0	79.9	88.2	11.7
DEVA [5]	66.0	61.8	70.3	87.0	83.8	90.2	82.6	78.9	86.4	85.4	84.9	89.4	79.6	87.8	25.3
Cutie-small	67.4	63.1	71.7	86.5	83.5	89.5	83.8	80.2	87.5	86.3	85.2	89.7	81.1	89.2	**45.5**
Cutie-base	**68.3**	**64.2**	**72.3**	**88.8**	**85.6**	**91.9**	**85.3**	**81.4**	**89.3**	**86.5**	**85.4**	**90.0**	**81.3**	**89.3**	36.4

Table 1. Quantitative comparison on video object segmentation benchmarks. All algorithms with available code are re-run on our hardware for a fair comparison. We could not obtain the code for [69, 70, 76] at the time of writing, and thus they cannot be reproduced on datasets that they do not report results on. For a fair comparison, all methods in this table use ImageNet [91] pre-training only or are trained from scratch. We compare methods with external pre-training (e.g., MAE [71] pre-training) in the supplement. *estimated FPS.

Method		BURST val			BURST test			
		All	Com.	Unc.	All	Com.	Unc.	Mem.
DeAOT [68]	FIFO	51.3	56.3	50.0	53.2	53.5	53.2	10.8G
DeAOT [68]	INF	56.4	59.7	55.5	57.9	56.7	58.1	34.9G
XMem [9]	FIFO	52.9	56.0	52.1	55.9	57.6	55.6	3.03G
XMem [9]	LT	55.1	57.9	54.4	58.2	59.5	58.0	3.34G
Cutie-small	FIFO	56.8	61.1	55.8	61.1	62.4	60.8	**1.35G**
Cutie-small	LT	58.3	61.5	**57.5**	61.6	63.1	61.3	2.28G
Cutie-base	LT	**58.4**	**61.8**	**57.5**	**62.6**	**63.8**	**62.3**	2.36G

Table 2. Comparisons of performance on long videos on the BURST dataset [3]. Mem.: maximum GPU memory usage. FIFO: first-in-first-out memory bank; INF: unbounded memory; LT: long-term memory [9]. DeAOT [68] is not compatible with long-term memory. All methods are trained with the MOSE [12] dataset.

at 6fps. For a fair comparison, we evaluate all algorithms at full fps whenever possible, which is crucial for video editing and for having a smooth user-interaction experience. For this, we re-run (De)AOT [10, 68] with their official code at 30fps on YouTubeVOS. We also retrain XMem [9], DeAOT [68], and DEVA [5] with their official code to include MOSE as training data (in addition to YouTubeVOS and DAVIS). For long video evaluation, we test on BURST [3] and LVOS [89] and experiment with the long-term memory [9] in addition to our default FIFO memory strategy. See supplement for details. We compare with DeAOT [68] and XMem [9] under the same setting.

Table 1 and Table 2 list our findings. Our method is highlighted with lavender. FPS is recorded on YouTubeVOS with a V100. Results on YouTubeVOS-18 and LVOS [89] are provided in the supplement. Cutie achieves better results than state-of-the-art methods, especially on the challenging MOSE dataset, while remaining efficient.

4.2. Ablations

Here, we study various design choices of our algorithm. We use the small model variant with MOSE [12] training data. We highlight our default configuration with lavender. For ablations, we report the $\mathcal{J}\&\mathcal{F}$ for MOSE validation and FPS on YouTubeVOS-2019 validation when applicable. Due to resource constraints, we train a selected subset of ablations three times with different random seeds and report mean±std. The baseline is trained five times. In tables that do not report std, we present our performance with the default random seed only.

Hyperparameter Choices. Table 3 compares our results with different choices of hyperparameters: number of object transformer blocks L, number of object queries N, interval between memory frames r, and maximum number of memory frames $T_{\max}$. Note that $L=0$ is equivalent to not having an object transformer. We visualize the progression of pixel features in Figure 4. We find that the object transformer

Setting	$\mathcal{J}\&\mathcal{F}$	FPS
Number of transformer blocks		
$L=0$	65.2	**56.6**
$L=1$	66.0	51.1
$L=3$	67.4	45.5
$L=5$	**67.8**	37.1
Number of object queries		
$N=8$	**67.6**	**45.5**
$N=16$	**67.4**	**45.5**
$N=32$	67.2	**45.5**
Memory interval		
$r=3$	**68.9**	43.2
$r=5$	67.4	45.5
$r=7$	67.0	**46.4**
Max. memory frames		
$T_{\max}=3$	66.9	**48.5**
$T_{\max}=5$	**67.4**	45.5
$T_{\max}=10$	**67.6**	37.4

Table 3. Performance comparison with different choices of hyperparameters.

Setting	$\mathcal{J}\&\mathcal{F}$	FPS
Both	**67.3**±0.36	45.5
Bottom-up only	65.0±0.44	**56.6**
Top-down only	40.7±1.62	46.9

Table 4. Comparison of our approach with bottom-up-only (no object transformer) and top-down-only (no pixel memory).

Setting	$\mathcal{J}\&\mathcal{F}$
With both	**67.3**±0.36
No object memory (X)	66.9±0.26
No object query (S)	**67.2**±0.10

Table 5. Ablations on the dynamic object memory and the static object query. Running times are similar.

Setting	$\mathcal{J}\&\mathcal{F}$	FPS
f.g.-b.g. masked attn.	**67.3**±0.36	45.5
f.g. masked attn. only	66.7±0.21	45.5
No masked attn.	63.8±1.06	**46.3**

Table 6. Ablations on foreground-background masked attention and object memory.

Setting	$\mathcal{J}\&\mathcal{F}$
With both p.e.	**67.4**
Without query p.e.	66.5
Without pixel p.e.	66.2
With neither	66.1

Table 7. Ablations on positional embeddings. Running times are similar.

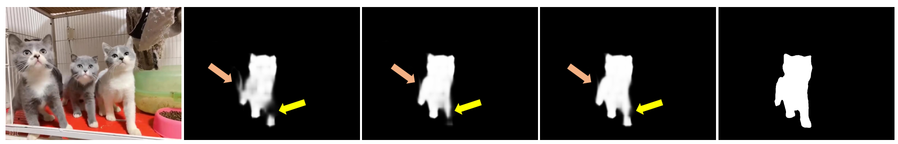

Figure 4. Visualization of auxiliary masks (M_l) at different layers of the object transformer. At every layer, noises are suppressed (pink arrows) and the target object becomes more coherent (yellow arrows).

blocks effectively suppress noises from distractors and produce more coherent object masks. Cutie is insensitive to the number of object queries – we think this is because 8 queries are sufficient to model the foreground/background of a single target object. As these queries execute in parallel, we find no noticeable differences in running time. Cutie benefits from having a shorter memory interval and a larger memory bank at the cost of a slower running time (e.g., +2.2 $\mathcal{J}\&\mathcal{F}$ on MOSE with half the speed) – we explore this speed-accuracy trade-off (as Cutie+) without re-training in the supplement.

Bottom-Up v.s. Top-Down Feature. Table 4 reports our findings. We compare a bottom-up-only approach (similar to XMem [9] with the training tricks [5] and a lighter backbone) without the object transformer, a top-down-only approach without the pixel memory, and our approach with both. Ours, integrating both features, performs the best.

Masked Attention Table 6 shows our results with different masked attention configurations. Masked attention is crucial for good performance – we hypothesize that using full attention produces confusing signals (especially in cluttered settings, see supplement), which leads to poor generalization. We note that using full attention also leads to rather unstable training. We experimented with different distributions of f.g./b.g. queries with no significant observed effects.

Object Memory and Positional Embeddings. Table 5 and Table 7 ablate on the object memory (X), the object query (S), and the positional embeddings in the object transformer. We note that the object query, while standard, is not useful for Cutie in the presence of the object memory. Positional embeddings are commonly used and do help.

4.3. Limitations

Despite being more robust, Cutie often fails when highly similar objects move in close proximity or occlude each other. This problem is not unique to Cutie. We suspect that, in these cases, neither the pixel memory nor the object memory is able to pick up sufficiently discriminative features for the object transformer to operate on. We provide visualizations in the supplementary material.

5. Conclusion

We present Cutie, an end-to-end network with object-level memory reading for robust video object segmentation in challenging scenarios. Cutie efficiently integrates top-down and bottom-up features, achieving new state-of-the-art results in several benchmarks, especially on the challenging MOSE dataset. We hope to draw more attention to object-centric video segmentation and to enable more accessible universal video segmentation methods via integration with segment-anything models [4, 5].

Acknowledgments. Work supported in part by NSF grants 2008387, 2045586, 2106825, MRI 1725729 (HAL [92]), and NIFA award 2020-67021-32799.

References

[1] Vladimír Petrík, Mohammad Nomaan Qureshi, Josef Sivic, and Makar Tapaswi. Learning object manipulation skills from video via approximate differentiable physics. In *IROS*, 2022. 1

[2] Ho Kei Cheng, Yu-Wing Tai, and Chi-Keung Tang. Modular interactive video object segmentation: Interaction-to-mask, propagation and difference-aware fusion. In *CVPR*, 2021. 1, 6

[3] Ali Athar, Jonathon Luiten, Paul Voigtlaender, Tarasha Khurana, Achal Dave, Bastian Leibe, and Deva Ramanan. Burst: A benchmark for unifying object recognition, segmentation and tracking in video. In *WACV*, 2023. 1, 6, 7

[4] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *arXiv*, 2023. 1, 8

[5] Ho Kei Cheng, Seoung Wug Oh, Brian Price, Alexander Schwing, and Joon-Young Lee. Tracking anything with decoupled video segmentation. In *ICCV*, 2023. 1, 2, 5, 6, 7, 8

[6] Jinyu Yang, Mingqi Gao, Zhe Li, Shang Gao, Fangjing Wang, and Feng Zheng. Track anything: Segment anything meets videos. In *arXiv*, 2023. 2

[7] Yangming Cheng, Liulei Li, Yuanyou Xu, Xiaodi Li, Zongxin Yang, Wenguan Wang, and Yi Yang. Segment and track anything. In *arXiv*, 2023. 1, 2

[8] Seoung Wug Oh, Joon-Young Lee, Ning Xu, and Seon Joo Kim. Video object segmentation using space-time memory networks. In *ICCV*, 2019. 1, 2, 3, 6

[9] Ho Kei Cheng and Alexander G Schwing. XMem: Long-term video object segmentation with an atkinson-shiffrin memory model. In *ECCV*, 2022. 1, 2, 3, 5, 6, 7, 8

[10] Zongxin Yang, Yunchao Wei, and Yi Yang. Associating objects with transformers for video object segmentation. In *NeurIPS*, 2021. 1, 2, 6, 7

[11] Maksym Bekuzarov, Ariana Bermudez, Joon-Young Lee, and Hao Li. Xmem++: Production-level video segmentation from few annotated frames. In *ICCV*, 2023. 1, 2

[12] Henghui Ding, Chang Liu, Shuting He, Xudong Jiang, Philip HS Torr, and Song Bai. MOSE: A new dataset for video object segmentation in complex scenes. In *arXiv*, 2023. 1, 2, 6, 7

[13] Federico Perazzi, Jordi Pont-Tuset, Brian McWilliams, Luc Van Gool, Markus Gross, and Alexander Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *CVPR*, 2016. 1, 2, 6

[14] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 1, 4

[15] Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *CVPR*, 2022. 2, 4, 6

[16] Junfeng Wu, Qihao Liu, Yi Jiang, Song Bai, Alan Yuille, and Xiang Bai. In defense of online models for video instance segmentation. In *ECCV*, 2022.

[17] Ali Athar, Alexander Hermans, Jonathon Luiten, Deva Ramanan, and Bastian Leibe. Tarvis: A unified approach for target-based video segmentation. *arXiv preprint arXiv:2301.02657*, 2023. 2

[18] Ali Athar, Jonathon Luiten, Alexander Hermans, Deva Ramanan, and Bastian Leibe. Hodor: High-level object descriptors for object re-segmentation in video learned from static images. In *CVPR*, 2022. 1, 2

[19] Ning Xu, Linjie Yang, Yuchen Fan, Dingcheng Yue, Yuchen Liang, Jianchao Yang, and Thomas Huang. Youtube-vos: A large-scale video object segmentation benchmark. In *ECCV*, 2018. 2, 6

[20] Sergi Caelles, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool. One-shot video object segmentation. In *CVPR*, 2017. 2

[21] Paul Voigtlaender and Bastian Leibe. Online adaptation of convolutional neural networks for video object segmentation. In *BMVC*, 2017.

[22] K-K Maninis, Sergi Caelles, Yuhua Chen, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool. Video object segmentation without temporal information. In *PAMI*, 2018.

[23] Goutam Bhat, Felix Järemo Lawin, Martin Danelljan, Andreas Robinson, Michael Felsberg, Luc Van Gool, and Radu Timofte. Learning what to learn for video object segmentation. In *ECCV*, 2020.

[24] Andreas Robinson, Felix Jaremo Lawin, Martin Danelljan, Fahad Shahbaz Khan, and Michael Felsberg. Learning fast and robust target models for video object segmentation. In *CVPR*, 2020. 2

[25] Federico Perazzi, Anna Khoreva, Rodrigo Benenson, Bernt Schiele, and Alexander Sorkine-Hornung. Learning video object segmentation from static images. In *CVPR*, 2017. 2

[26] Yuan-Ting Hu, Jia-Bin Huang, and Alexander Schwing. Maskrnn: Instance level video object segmentation. In *NIPS*, 2017.

[27] Ping Hu, Gang Wang, Xiangfei Kong, Jason Kuen, and Yap-Peng Tan. Motion-guided cascaded refinement network for video object segmentation. In *CVPR*, 2018.

[28] Seoung Wug Oh, Joon-Young Lee, Kalyan Sunkavalli, and Seon Joo Kim. Fast video object segmentation by reference-guided mask propagation. In *CVPR*, 2018.

[29] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. Fast online object tracking and segmentation: A unifying approach. In *CVPR*, 2019.

[30] Lu Zhang, Zhe Lin, Jianming Zhang, Huchuan Lu, and You He. Fast video object segmentation via dynamic targeting network. In *ICCV*, 2019.

[31] Carles Ventura, Miriam Bellver, Andreu Girbau, Amaia Salvador, Ferran Marques, and Xavier Giro-i Nieto. Rvos: End-to-end recurrent network for video object segmentation. In *CVPR*, 2019. 2

[32] Yuan-Ting Hu, Jia-Bin Huang, and Alexander G Schwing. Videomatch: Matching based video object segmentation. In *ECCV*, 2018. 2

[33] Paul Voigtlaender, Yuning Chai, Florian Schroff, Hartwig Adam, Bastian Leibe, and Liang-Chieh Chen. Feelvos: Fast end-to-end embedding learning for video object segmentation. In *CVPR*, 2019.

[34] Ziqin Wang, Jun Xu, Li Liu, Fan Zhu, and Ling Shao. Ranet: Ranking attention network for fast video object segmentation. In *ICCV*, 2019.

[35] Kevin Duarte, Yogesh S. Rawat, and Mubarak Shah. Capsulevos: Semi-supervised video object segmentation using capsule routing. In *ICCV*, 2019.

[36] Zongxin Yang, Yunchao Wei, and Yi Yang. Collaborative video object segmentation by foreground-background integration. In *ECCV*, 2020. 2

[37] Yu Li, Zhuoran Shen, and Ying Shan. Fast video object segmentation using the global context module. In *ECCV*, 2020.

[38] Yizhuo Zhang, Zhirong Wu, Houwen Peng, and Stephen Lin. A transductive approach for video object segmentation. In *CVPR*, 2020.

[39] Hongje Seong, Junhyuk Hyun, and Euntai Kim. Kernelized memory network for video object segmentation. In *ECCV*, 2020.

[40] Xiankai Lu, Wenguan Wang, Danelljan Martin, Tianfei Zhou, Jianbing Shen, and Van Gool Luc. Video object segmentation with episodic graph memory networks. In *ECCV*, 2020.

[41] Yongqing Liang, Xin Li, Navid Jafari, and Jim Chen. Video object segmentation with adaptive feature bank and uncertain-region refinement. In *NeurIPS*, 2020.

[42] Xuhua Huang, Jiarui Xu, Yu-Wing Tai, and Chi-Keung Tang. Fast video object segmentation with temporal aggregation network and dynamic template matching. In *CVPR*, 2020.

[43] Shuxian Liang, Xu Shen, Jianqiang Huang, and Xian-Sheng Hua. Video object segmentation with dynamic memory networks and adaptive object alignment. In *ICCV*, 2021.

[44] Xiaohao Xu, Jinglu Wang, Xiao Li, and Yan Lu. Reliable propagation-correction modulation for video object segmentation. In *AAAI*, 2022.

[45] Wenbin Ge, Xiankai Lu, and Jianbing Shen. Video object segmentation using global and instance embedding learning. In *CVPR*, 2021.

[46] Li Hu, Peng Zhang, Bang Zhang, Pan Pan, Yinghui Xu, and Rong Jin. Learning position and target consistency for memory-based video object segmentation. In *CVPR*, 2021.

[47] Haochen Wang, Xiaolong Jiang, Haibing Ren, Yao Hu, and Song Bai. Swiftnet: Real-time video object segmentation. In *CVPR*, 2021.

[48] Haozhe Xie, Hongxun Yao, Shangchen Zhou, Shengping Zhang, and Wenxiu Sun. Efficient regional memory network for video object segmentation. In *CVPR*, 2021.

[49] Hongje Seong, Seoung Wug Oh, Joon-Young Lee, Seongwon Lee, Suhyeon Lee, and Euntai Kim. Hierarchical memory matching network for video object segmentation. In *ICCV*, 2021.

[50] Yunyao Mao, Ning Wang, Wengang Zhou, and Houqiang Li. Joint inductive and transductive learning for video object segmentation. In *ICCV*, 2021. 2

[51] Ho Kei Cheng, Yu-Wing Tai, and Chi-Keung Tang. Rethinking space-time networks with improved memory coverage for efficient video object segmentation. In *NeurIPS*, 2021. 2, 7

[52] Yong Liu, Ran Yu, Fei Yin, Xinyuan Zhao, Wei Zhao, Weihao Xia, and Yujiu Yang. Learning quality-aware dynamic memory for video object segmentation. In *ECCV*, 2022.

[53] Ye Yu, Jialin Yuan, Gaurav Mittal, Li Fuxin, and Mei Chen. Batman: Bilateral attention transformer in motion-appearance neighboring space for video object segmentation. In *ECCV*, 2022. 2

[54] Bo Miao, Mohammed Bennamoun, Yongsheng Gao, and Ajmal Mian. Region aware video object segmentation with deep motion modeling. In *arXiv*, 2022.

[55] Mingxing Li, Li Hu, Zhiwei Xiong, Bang Zhang, Pan Pan, and Dong Liu. Recurrent dynamic embedding for video object segmentation. In *CVPR*, 2022. 7

[56] Kwanyong Park, Sanghyun Woo, Seoung Wug Oh, In So Kweon, and Joon-Young Lee. Per-clip video object segmentation. In *CVPR*, 2022.

[57] Yong Liu, Ran Yu, Jiahao Wang, Xinyuan Zhao, Yitong Wang, Yansong Tang, and Yujiu Yang. Global spectral filter memory network for video object segmentation. In *ECCV*, 2022.

[58] Yurong Zhang, Liulei Li, Wenguan Wang, Rong Xie, Li Song, and Wenjun Zhang. Boosting video object segmentation via space-time correspondence learning. In *CVPR*, 2023.

[59] Xiaohao Xu, Jinglu Wang, Xiang Ming, and Yan Lu. Towards robust video object segmentation with adaptive object calibration. In *ACM MM*, 2022. 2

[60] Suhwan Cho, Heansung Lee, Minjung Kim, Sungjun Jang, and Sangyoun Lee. Pixel-level bijective matching for video object segmentation. In *WACV*, 2022.

[61] Kai Xu and Angela Yao. Accelerating video object segmentation with compressed video. In *CVPR*, 2022.

[62] Roy Miles, Mehmet Kerim Yucel, Bruno Manganelli, and Albert Saa-Garriga. Mobilevos: Real-time video object segmentation contrastive learning meets knowledge distillation. In *CVPR*, 2023.

[63] Kun Yan, Xiao Li, Fangyun Wei, Jinglu Wang, Chenbin Zhang, Ping Wang, and Yan Lu. Two-shot video object segmentation. In *CVPR*, 2023.

[64] Rui Sun, Yuan Wang, Huayu Mai, Tianzhu Zhang, and Feng Wu. Alignment before aggregation: trajectory memory retrieval network for video object segmentation. In *ICCV*, 2023. 2

[65] Zongxin Yang, Yunchao Wei, and Yi Yang. Collaborative video object segmentation by multi-scale foreground-background integration. In *TPAMI*, 2021. 2

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 2, 4

[67] Brendan Duke, Abdalla Ahmed, Christian Wolf, Parham Aarabi, and Graham W Taylor. Sstvos: Sparse spatiotemporal transformers for video object segmentation. In *CVPR*, 2021. 2

[68] Zongxin Yang and Yi Yang. Decoupling features in hierarchical propagation for video object segmentation. In *NeurIPS*, 2022. 2, 7

[69] Qiangqiang Wu, Tianyu Yang, Wei Wu, and Antoni Chan. Scalable video object segmentation with simplified framework. In *ICCV*, 2023. 2, 7

[70] Jiaming Zhang, Yutao Cui, Gangshan Wu, and Limin Wang. Joint modeling of feature, correspondence, and a compressed memory for video object segmentation. In *arXiv*, 2023. 2, 7

[71] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Doll'ar, and Ross B Girshick. Masked autoencoders are scalable vision learners. 2022 ieee. In *CVPR*, 2021. 2, 7

[72] Xiaoxiao Li and Chen Change Loy. Video object segmentation with joint re-identification and attention-aware mask propagation. In *ECCV*, 2018. 2

[73] Jonathon Luiten, Paul Voigtlaender, and Bastian Leibe. Premvos: Proposal-generation, refinement and merging for video object segmentation. In *ACCV*, 2018. 2

[74] Bin Yan, Yi Jiang, Jiannan Wu, Dong Wang, Ping Luo, Zehuan Yuan, and Huchuan Lu. Universal instance perception as object discovery and retrieval. In *CVPR*, 2023. 2

[75] Bin Yan, Yi Jiang, Peize Sun, Dong Wang, Zehuan Yuan, Ping Luo, and Huchuan Lu. Towards grand unification of object tracking. In *ECCV*, 2022. 2

[76] Junke Wang, Dongdong Chen, Zuxuan Wu, Chong Luo, Chuanxin Tang, Xiyang Dai, Yucheng Zhao, Yujia Xie, Lu Yuan, and Yu-Gang Jiang. Look before you match: Instance understanding matters in video object segmentation. In *CVPR*, 2023. 2, 7

[77] Shubhika Garg and Vidit Goel. Mask selection and propagation for unsupervised video object segmentation. In *WCAV*, 2021. 2

[78] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *ICLR*, 2020. 4, 6

[79] Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. Eca-net: efficient channel attention for deep convolutional neural networks. In *CVPR*, 2020. 6

[80] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 6

[81] Jianping Shi, Qiong Yan, Li Xu, and Jiaya Jia. Hierarchical image saliency detection on extended cssd. In *TPAMI*, 2015. 6

[82] Lijun Wang, Huchuan Lu, Yifan Wang, Mengyang Feng, Dong Wang, Baocai Yin, and Xiang Ruan. Learning to detect salient objects with image-level supervision. In *CVPR*, 2017.

[83] Xiang Li, Tianhan Wei, Yau Pun Chen, Yu-Wing Tai, and Chi-Keung Tang. Fss-1000: A 1000-class dataset for few-shot segmentation. In *CVPR*, 2020.

[84] Yi Zeng, Pingping Zhang, Jianming Zhang, Zhe Lin, and Huchuan Lu. Towards high-resolution salient object detection. In *ICCV*, 2019.

[85] Ho Kei Cheng, Jihoon Chung, Yu-Wing Tai, and Chi-Keung Tang. Cascadepsp: Toward class-agnostic and very high-resolution segmentation via global and local refinement. In *CVPR*, 2020. 6

[86] Jiyang Qi, Yan Gao, Yao Hu, Xinggang Wang, Xiaoyu Liu, Xiang Bai, Serge Belongie, Alan Yuille, Philip HS Torr, and Song Bai. Occluded video instance segmentation: A benchmark. In *IJCV*, 2022. 6

[87] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 6

[88] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *CVPR*, 2020. 6

[89] Lingyi Hong, Wenchao Chen, Zhongying Liu, Wei Zhang, Pinxue Guo, Zhaoyu Chen, and Wenqiang Zhang. Lvos: A benchmark for long-term video object segmentation. In *ICCV*, 2023. 6, 7

[90] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. Hota: A higher order metric for evaluating multi-object tracking. In *IJCV*, 2021. 6

[91] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 7

[92] Volodymyr Kindratenko, Dawei Mu, Yan Zhan, John Maloney, Sayed Hadi Hashemi, Benjamin Rabe, Ke Xu, Roy Campbell, Jian Peng, and William Gropp. Hal: Computer system for scalable deep learning. In *PEARC*, 2020. 8