Supplementary Material
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

The supplementary material is structured as follows:

1. We first provide visual comparisons of Cutie with state-of-the-art methods in Section A.
2. We then show some highly challenging cases where both Cutie and state-of-the-art methods fail in Section B.
3. We analyze the running time of XMem and Cutie in Section C.
4. To elucidate the workings of the object transformer, we visualize the difference in attention patterns of pixel readout v.s. object readout, feature progression within the object transformer, and the qualitative benefits of masked attention/object transformer in Section D.
5. We present additional details on BURST evaluation in Section E.
6. We list options for adjusting the speed-accuracy trade-off without re-training, comparisons with methods that use external training, additional quantitative results on YouTube-VOS 2018 [1]/LVOS [2], and the performance variations with respect to different random seeds in Section F.
7. We give more implementation details on the training process, decoder architecture, and pixel memory in Section G.
8. Lastly, we showcase an interactive video segmentation tool powered by Cutie in Section H. This tool will be open-sourced to help researchers and data annotators.

A. Visual Comparisons

We provide visual comparisons of Cutie with DeAOT-R50 [3] and XMem [4] at youtu.be/LGbJ11GT8Ig. For a fair comparison, we use Cutie-base and train all models with the MOSE dataset. We visualize the comparisons using sequences from YouTubeVOS-2019 validation, DAVIS 2017 test-dev, and MOSE validation. Only the first-frame (not full-video) ground-truth annotations are available in these datasets. At the beginning of each sequence, we pause for two seconds to show the first-frame segmentation that initializes all the models. Our model is more robust to distractors and generates more coherent masks.

B. Failure Cases

We visualize some failure cases of Cutie at youtu.be/PIjXUYRzQ8Q, following the format discussed in Section A. As discussed in the main paper, Cutie fails in some of the challenging cases where similar objects move in close proximity or occlude each other. This problem is not unique to Cutie, as current state-of-the-art methods also fail as shown in the video.

In the first sequence "elephants", all models have difficulty tracking the elephants (e.g., light blue mask) behind the big (unannotated) foreground elephant. In the second sequence "birds", all models fail when the bird with the pink mask moves and occludes other birds.

We think that this is due to the lack of useful features from the pixel memory and the object memory, as they fail to disambiguate objects that are similar in both appearance and position. A potential future work direction for this is to encode three-dimensional spatial understanding (i.e., the bird that occludes is closer to the camera).

C. Running Time Analysis

We analyze the total runtime of XMem and Cutie in Tab. S1, testing on a single video with a 2080Ti. We synchronize and warm up properly to get accurate timing; small deviations might arise from minor implementation differences and run-time variations. We note that the speedup is mostly achieved by using a lighter decoder.

	XMem	Cutie-base	Cutie-small
Query encoder	0.861	0.851	0.295
Mask encoder	0.143	0.145	0.142
Pixel memory read	0.758	0.514	0.514
Object memory read	-	0.913	0.894
Decoding	2.749	0.725	0.700

Table S1. Total running time (s) of each component in XMem and Cutie.

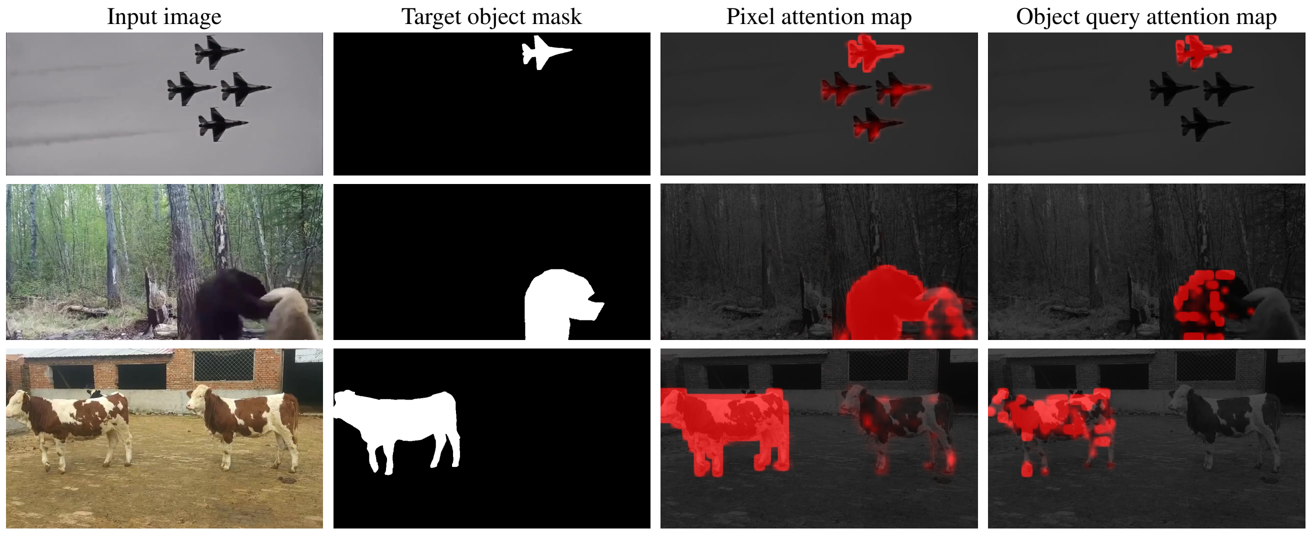


Figure S1. Comparison of foreground attention patterns between pixel attention with object query attention. In each of the three examples, the two leftmost columns show the input image and the ground-truth (annotated by us for reference). The two rightmost columns show the attention patterns for pixel attention and object query attention respectively. Ideally, the attention weights should focus on the foreground object. As shown, the pixel attention has a broader coverage but is easily distracted by similar objects. The object query attention's attention is more sparse (as we use a small number of object queries), and is more focused on the foreground. Our object transformer makes use of both: it first initializes with pixel attention and restructures the features iteratively with object query attention.

D. Additional Visualizations

D.1. Attention Patterns of Pixel Attention v.s. Object Query Attention

Here, we visualize the attention maps of pixel memory reading and of the object transformer, showing the qualitative difference between the two.

To visualize attention in pixel memory reading, we use "self-attention", i.e., by setting $\mathbf{k} = \mathbf{q} \in \mathbb{R}^{HW \times C^{\mathbf{k}}}$. We compute the pixel affinity $A^{\text{pix}} \in [0, 1]^{HW \times HW}$, as in Equation (9) of the main paper. We then sum over the foreground region along the rows, visualizing the affinity of every pixel to the foreground. Ideally, all the affinity should be in the foreground – others are distractors that cause erroneous matching. The foreground region is defined by the last auxiliary mask M_L in the object transformer.

To visualize the attention in the object transformer, we inspect the attention weights $A_L \in \mathbb{R}^{N \times HW}$ of the first (pixel-to-query) cross-attention in the last object transformer block. Similar to how we visualize the pixel attention, we focus on the foreground queries (i.e., first $N/2$ object queries) and sum over the corresponding rows in the affinity matrix.

Figure S1 visualizes the differences between these two types of attention. The pixel attention is more spread out and is easily distracted by similar objects. The object query attention focuses on the foreground without being distracted. Our object transformer makes use of both types of attention by using pixel attention for initialization and object query attention for restructuring the feature in every transformer block.

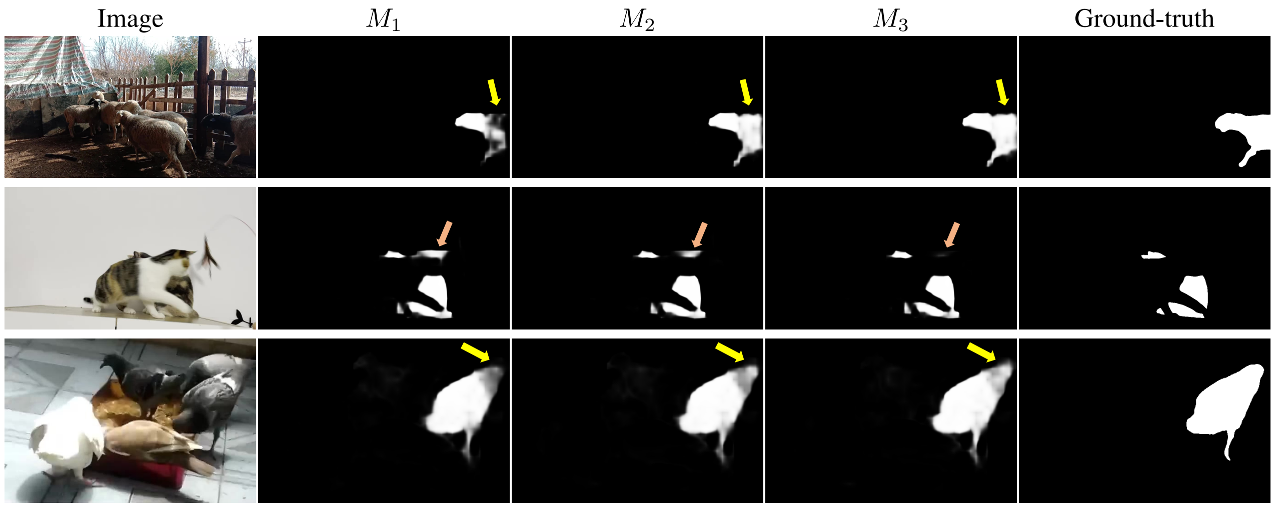


Figure S2. Visualization of auxiliary masks M_l in the l-th object transformer block. At every layer, matching errors are suppressed (pink arrows) and the target object becomes more coherent (yellow arrows). The ground-truth is annotated by us for reference.

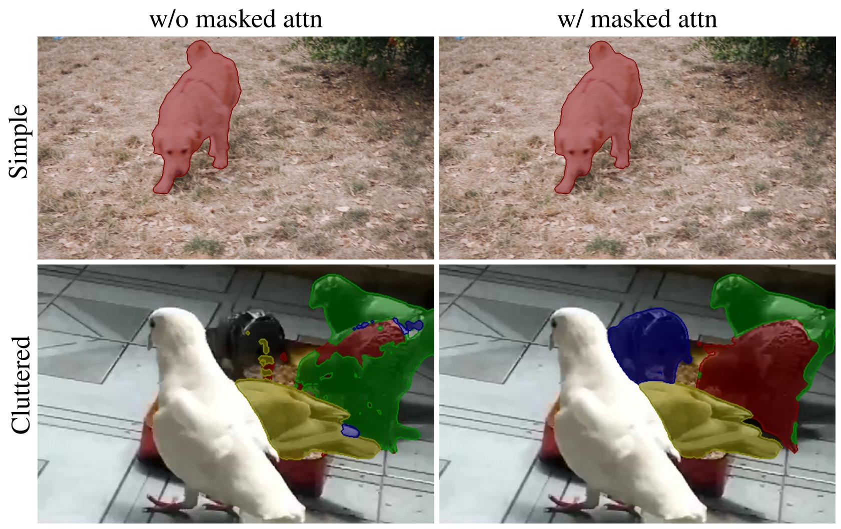


Figure S3. Comparisons of Cutie with/without masked attention. While both work well in simple cases, masked attention helps to differentiate similarly-looking objects.

D.2. Feature Progression in the Object Transformer

Figure S2 visualizes additional feature progressions within the object transformer (in addition to Figure 4 in the main paper). The object transformer helps to suppress noises from low-level matching and produces more coherent object-level features.

D.3. Benefits of Masked Attention/Object Transformer

Figure S3 qualitatively compares results with/without using masked attention – while both work well for simpler cases, masked attention helps in challenging cases with similar objects. Figure S4 visualizes the benefits of the object transformer. Using the object transformer leads to more complete and accurate outputs.

E. Details on BURST Evaluation

In BURST [5], we update the memory every 10th frame following [4]. Since BURST contains high-resolution images (e.g., 1920×1200), we downsize the images such that the shorter edge has no more than 600 pixels instead of the default 480 pixels for all methods. Following [5], we assess Higher Order Tracking Accuracy (HOTA) [6] on common and uncommon object classes separately.

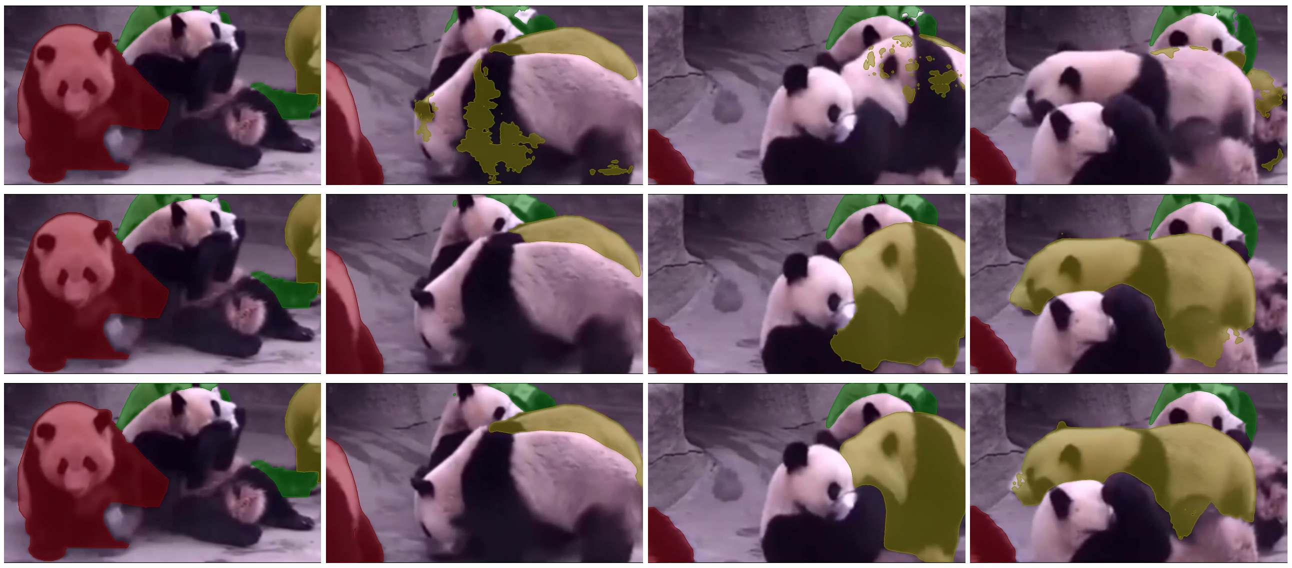


Figure S4. Top-to-bottom: Without object queries, Cutie's default model, and ground-truth. The leftmost frame is a reference frame.

For better performance on long videos, we experiment with the long-term memory [4] in addition to our default FIFO memory strategy. The long-term memory is a plug-in addition to our pixel memory – it routinely compresses the attentional "working memory" into a long-term memory storage instead of discarding them as in our first-in-first-out approach. The long-term memory can be adopted without any re-training. We follow the default long-term memory parameters in XMem [4] and present the improvement in the main paper.

F. Additional Quantitative Results

F.1. Speed-Accuracy Trade-off

We note that the performance of Cutie can be further improved by changing hyperparameters like memory interval and the size of the memory bank during inference, at the cost of a slower running time. Here, we present "Cutie+", which adjusts the following hyperparameters without re-training:

1. Maximum memory frames $T_{\max} = 5 \rightarrow T_{\max} = 10$
2. Memory interval $r = 5 \rightarrow r = 3$
3. Maximum shorter side resolution during inference $480 \rightarrow 720$ pixels

These settings apply to DAVIS [7] and MOSE [8]. For YouTubeVOS, we keep the memory interval $r = 5$ and set the maximum shorter side resolution during inference to 600 for two reasons: 1) YouTubeVOS is annotated every 5 frames, and aligning the memory interval with annotation avoids adding unannotated objects into the memory as background, and 2) YouTubeVOS has lower video quality and using higher resolution makes artifacts more apparent. The results of Cutie+ are tabulated in the bottom portion of Table S2.

F.2. Comparisons with Methods that Use External Training

Here, we present comparisons with methods that use external training: SimVOS [17], JointFormer [11], and ISVOS [13] in Table S2. Note, we could not obtain the code for these methods at the time of writing. ISVOS [13] does not report running time – we estimate to the best of our ability with the following information: 1) For the VOS branch, it uses XMem [4] as the baseline with a first-in-first-out 16-frame memory bank, 2) for the instance branch, it uses Mask2Former [15] with an unspecified backbone. Beneficially for ISVOS, we assume the lightest backbone (ResNet-50), and 3) the VOS branch and the instance branch share a feature extraction backbone. Our computation is as follows:

1. Time per frame for XMem with a 16-frame first-in-first-out memory bank (from our testing): 75.2 ms
2. Time per frame for Mask2Former with ResNet-50 backbone (from Mask2Former paper): 103.1 ms
3. Time per frame of the doubled-counted feature extraction backbone (from our testing): 6.5 ms

Thus, we estimate that ISVOS would take (75.2+103.1-6.5) = 171.8 ms per frame, which translates to 5.8 frames per second.

In an endeavor to reach a better performance with Cutie by adding more training data, we devise a "MEGA" training

Method		MOSE $\mathcal{J}\&\mathcal{F}$	MOSE $\mathcal{J}$	MOSE $\mathcal{F}$	DAVIS-17 val $\mathcal{J}\&\mathcal{F}$	DAVIS-17 val $\mathcal{J}$	DAVIS-17 val $\mathcal{F}$	DAVIS-17 test $\mathcal{J}\&\mathcal{F}$	DAVIS-17 test $\mathcal{J}$	DAVIS-17 test $\mathcal{F}$	YouTubeVOS-2019 val $\mathcal{G}$	YouTubeVOS-2019 val $\mathcal{J}_s$	YouTubeVOS-2019 val $\mathcal{F}_s$	YouTubeVOS-2019 val $\mathcal{J}_u$	YouTubeVOS-2019 val $\mathcal{F}_u$	FPS
SimVOS-B [9]		-	-	-	81.3	78.8	83.8	-	-	-	-	-	-	-	-	3.3
SimVOS-B [9]	w/ MAE [10]	-	-	-	88.0	85.0	91.0	80.4	76.1	84.6	84.2	83.1	-	79.1	-	3.3
JointFormer [11]		-	-	-	-	-	-	65.6	61.7	69.4	73.3	75.2	78.5	65.8	73.6	3.0
JointFormer [11]	w/ MAE [10]	-	-	-	89.7	86.7	92.7	87.6	84.2	91.1	87.0	86.1	90.6	82.0	89.5	3.0
JointFormer [11]	w/ MAE [10] + BL30K [12]	-	-	-	90.1	87.0	**93.2**	**88.1**	**84.7**	**91.6**	**87.4**	**86.5**	**90.9**	82.0	**90.3**	3.0
ISVOS [13]		-	-	-	80.0	76.9	83.1	-	-	-	-	-	-	-	-	5.8*
ISVOS [13]	w/ COCO [14]	-	-	-	87.1	83.7	90.5	82.8	79.3	86.2	86.1	85.2	89.7	80.7	88.9	5.8*
ISVOS [13]	w/ COCO [14] + BL30K [12]	-	-	-	88.2	84.5	91.9	84.0	80.1	87.8	86.3	85.2	89.7	81.0	89.1	5.8*
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
Cutie-small	w/ MOSE [8]	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	w/ MOSE [8]	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
Cutie-small	w/ MEGA	68.6	64.3	72.9	87.0	84.0	89.9	85.3	81.4	89.2	86.8	85.2	89.6	82.1	**90.4**	**45.5**
Cutie-base	w/ MEGA	69.9	65.8	74.1	87.9	84.6	91.1	86.1	82.4	89.9	87.0	86.0	90.5	82.0	89.6	36.4
Cutie-small+		64.3	60.4	68.2	88.7	86.0	91.3	85.7	82.5	88.9	86.7	85.7	89.8	81.7	89.6	20.6
Cutie-base+		66.2	62.3	70.1	**90.5**	**87.5**	**93.4**	85.9	82.6	89.2	86.9	86.2	90.7	81.6	89.2	17.9
Cutie-small+	w/ MOSE [8]	69.0	64.9	73.1	89.3	86.4	92.1	86.7	83.4	90.1	86.5	85.4	89.7	81.6	89.2	20.6
Cutie-base+	w/ MOSE [8]	70.5	66.5	74.6	90.0	87.1	93.0	86.3	82.9	89.7	86.8	85.7	90.0	81.8	89.6	17.9
Cutie-small+	w/ MEGA	70.3	66.0	74.5	89.3	86.2	92.5	87.1	83.8	90.4	86.8	85.4	89.5	82.3	90.0	20.6
Cutie-base+	w/ MEGA	**71.7**	**67.6**	**75.8**	88.1	85.5	90.8	**88.1**	**84.7**	**91.4**	**87.5**	**86.3**	90.6	**82.7**	**90.5**	17.9

Table S2. Quantitative comparison on common video object segmentation benchmarks, including methods that use external training data. Recent vision-transformer-based methods [9, 11, 13] depend largely on pretraining, either with MAE [10] or pretraining a separate Mask2Former [15] network on COCO instance segmentation [14]. Note they do not release code at the time of writing, and thus they cannot be reproduced on datasets that they do not report results on. Cutie performs competitively to those recent (slow) transformer-based methods, especially with added training data. MEGA is the aggregated dataset consisting of DAVIS [7], YouTubeVOS [1], MOSE [8], OVIS [16], and BURST [5]. *estimated FPS.

Method		YouTubeVOS-2018 val $\mathcal{G}$	YouTubeVOS-2018 val $\mathcal{J}_s$	YouTubeVOS-2018 val $\mathcal{F}_s$	YouTubeVOS-2018 val $\mathcal{J}_u$	YouTubeVOS-2018 val $\mathcal{F}_u$	LVOS val $\mathcal{J}\&\mathcal{F}$	LVOS val $\mathcal{J}$	LVOS val $\mathcal{F}$	LVOS test $\mathcal{J}\&\mathcal{F}$	LVOS test $\mathcal{J}$	LVOS test $\mathcal{F}$	FPS
DEVA [18]		85.9	85.5	90.1	79.7	88.2	58.3	52.8	63.8	54.0	49.0	59.0	25.3
DEVA [18]	w/ MOSE [8]	85.8	85.4	90.1	79.7	88.2	55.9	51.1	60.7	56.5	52.2	60.8	25.3
DDMemory [2]		84.1	83.5	88.4	78.1	86.5	**60.7**	55.0	**66.3**	55.0	49.9	60.2	18.7
Cutie-small		**86.3**	85.5	90.1	**80.6**	**89.0**	58.8	54.6	62.9	**57.2**	**53.7**	**60.7**	**45.5**
Cutie-base		86.1	**85.8**	**90.5**	80.0	88.0	60.1	**55.9**	64.2	56.2	51.8	60.5	36.4
Cutie-small	w/ MOSE [8]	86.8	85.7	90.4	81.6	89.7	60.7	55.6	65.8	56.9	53.5	60.2	**45.5**
Cutie-base	w/ MOSE [8]	86.6	85.7	90.6	80.8	89.1	63.5	59.1	67.9	63.6	59.1	68.0	36.4
Cutie-small	w/ MEGA	**86.9**	85.5	90.1	**81.7**	**90.2**	62.9	58.3	67.4	66.4	61.9	70.9	**45.5**
Cutie-base	w/ MEGA	**87.0**	**86.4**	**91.1**	81.4	89.2	**66.0**	**61.3**	**70.6**	**66.7**	**62.4**	**71.0**	36.4

Table S3. Quantitative comparison on YouTubeVOS-2018 [1] and LVOS [2]. DDMemory [2] is the baseline method presented in LVOS [2] with no available official code at the time of writing. Note, we think LVOS is significantly different than other datasets because it contains a lot more tiny objects. See Section F.3 for details. MEGA is the aggregated dataset consisting of DAVIS [7], YouTubeVOS [1], MOSE [8], OVIS [16], and BURST [5].

scheme that includes training on BURST [5] and OVIS [16] in addition to DAVIS [7], YouTubeVOS [1], and MOSE [8]. We train for an additional 50K iterations in the MEGA setting. The results are tabulated in the bottom portion of Table S2.

F.3. Results on YouTubeVOS-2018 and LVOS

Here, we provide additional results on the YouTubeVOS-2018 validation set and LVOS [2] validation/test sets in Table S3. FPS is measured on YoutubeVOS-2018/2019 following the main paper. YouTubeVOS-2018 is the old version of YouTubeVOS-2019 – we present our main results using YouTubeVOS-2019 and provide results on YouTubeVOS-2018 for reference. Note that

Method			BURST val All	BURST val Com.	BURST val Unc.	BURST test All	BURST test Com.	BURST test Unc.	Memory usage
DeAOT [3]	FIFO	w/ MOSE [8]	51.3	56.3	50.0	53.2	53.5	53.2	10.8G
DeAOT [3]	INF	w/ MOSE [8]	56.4	59.7	55.5	57.9	56.7	58.1	34.9G
XMem [4]	FIFO	w/ MOSE [8]	52.9	56.0	52.1	55.9	57.6	55.6	3.03G
XMem [4]	LT	w/ MOSE [8]	55.1	57.9	54.4	58.2	59.5	58.0	3.34G
Cutie-small	FIFO	w/ MOSE [8]	56.8	61.1	55.8	61.1	62.4	60.8	**1.35G**
Cutie-small	LT	w/ MOSE [8]	58.3	61.5	57.5	61.6	63.1	61.3	2.28G
Cutie-base	LT	w/ MOSE [8]	58.4	61.8	57.5	62.6	63.8	62.3	2.36G
Cutie-small	LT	w/ MEGA	**61.6**	**65.3**	**60.6**	64.4	63.7	64.6	2.28G
Cutie-base	LT	w/ MEGA	61.2	65.0	60.3	**66.0**	**66.5**	**65.9**	2.36G

Table S4. Extended comparisons of performance on long videos on the BURST dataset [5], including our results when trained in the MEGA setting. Com. and Unc. stand for common and uncommon objects respectively. Mem.: maximum GPU memory usage. FIFO: first-in-first-out memory bank; INF: unbounded memory; LT: long-term memory [4]. DeAOT [3] is not compatible with long-term memory.

these results are ready at the time of paper submission and are referred to in the main paper. The complete tables are listed here due to space constraints in the main paper.

LVOS [2] is a recently proposed long-term video object segmentation benchmark, with 50 videos in its validation set and test set respectively. Note, we have also presented results in another long-term video object segmentation benchmark, BURST [5] in the main paper, which contains 988 videos in the validation set and 1419 videos in the test set. We test Cutie on LVOS *after* completing the design of Cutie, adopt long-term memory [4], and perform no tuning. We note that our method (Cutie-base) performs better than DDMemory, the baseline presented in LVOS [2], on the test set and has a comparable performance on the validation set, while running about twice as fast. Upon manual inspection of the results, we observe that one of the unique challenges in LVOS is the prevalence of tiny objects, which our algorithm has not been specifically designed to handle. We quantify this observation by analyzing the first frame annotations of all the videos in the validation sets of DAVIS [7], YouTubeVOS [1], MOSE [8], BURST [5], and LVOS [2], as shown in Figure S5. Tiny objects are significantly more prevalent on LVOS [2] than on other datasets. We think this makes LVOS uniquely challenging for methods that are not specifically designed to detect small objects.

F.4. Performance Variations

To assess performance variations with respect to different random seeds, we train Cutie-small with five different random seeds (including both pretraining and main training with the MOSE dataset) and report mean±standard deviation on the MOSE [8] validation set and the YouTubeVOS 2019 [1] validation set in Table S5. Note, the improvement brought by our model (i.e., 8.7 $\mathcal{J}\&\mathcal{F}$ on MOSE and 0.9 $\mathcal{G}$ on YouTubeVOS over XMem [4]) corresponds to +24.2 s.d. and +8.2 s.d. respectively.

MOSE val $\mathcal{J}\&\mathcal{F}$	MOSE val $\mathcal{J}$	MOSE val $\mathcal{F}$	YouTubeVOS-2019 val $\mathcal{G}$	YouTubeVOS-2019 val $\mathcal{J}_s$	YouTubeVOS-2019 val $\mathcal{F}_s$	YouTubeVOS-2019 val $\mathcal{J}_u$	YouTubeVOS-2019 val $\mathcal{F}_u$
67.3±0.36	63.1±0.36	71.6±0.35	86.2±0.11	85.1±0.20	89.6±0.27	81.1±0.19	89.3±0.13

Table S5. Performance variations (median±standard deviation) across five different random seeds.

G. Implementation Details

Here, we include more implementation details for completeness. Our training and testing code will be released for reproducibility.

G.1. Extension to Multiple Objects

We extend Cutie to the multi-object setting following [4, 18–20]. Objects are processed independently (in parallel as a batch) except for 1) the interaction at the first convolutional layer of the mask encoder, which extracts features corresponding to a target object with a 5-channel input concatenated from the image (3-channel), the mask of the target object (1-channel), and the sum of masks of all non-target objects (1-channel); 2) the interaction at the soft-aggregation layers [19] used to generate segmentation logits – where the object probability distributions at every pixel are normalized to sum up to one. Note these are

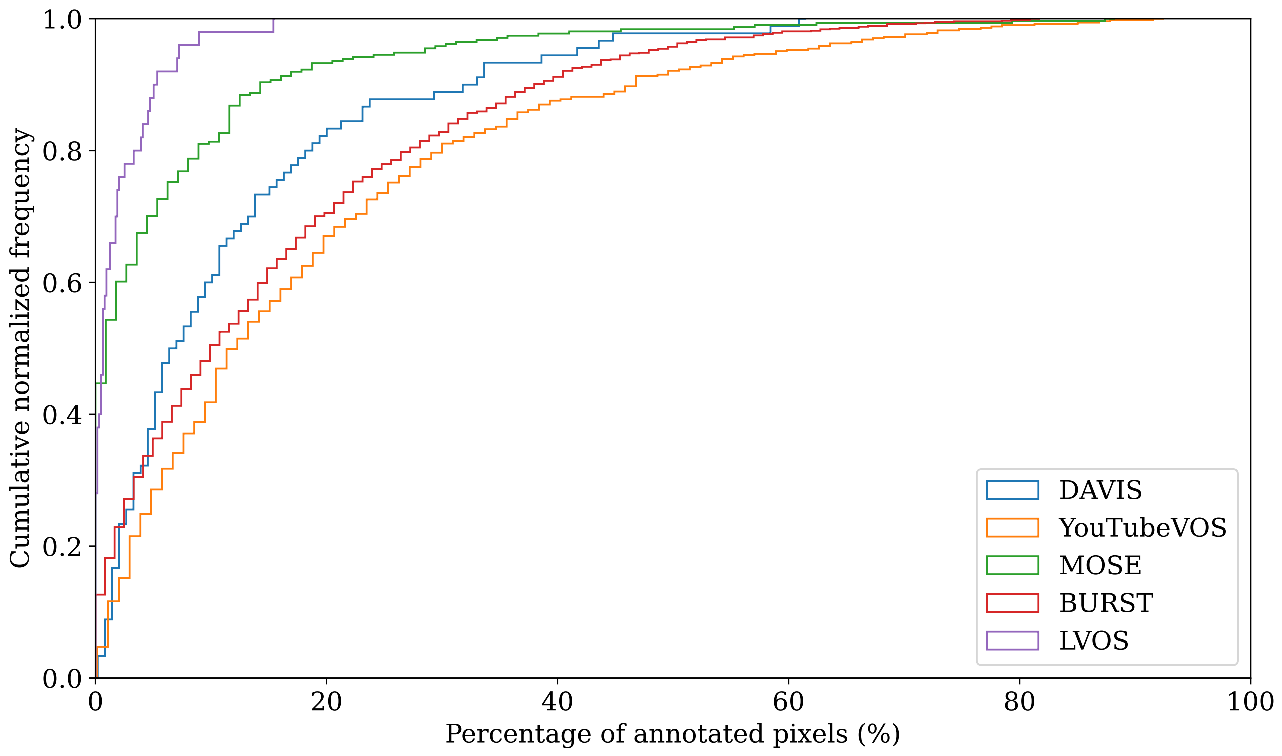


Figure S5. Cumulative frequency graph of annotated pixel areas (as percentages of the total image area) for different datasets. Tiny objects are significantly more prevalent on LVOS [2] than on other datasets.

standard operations from prior works [4, 18–20]. Parts of the computation (i.e., feature extraction from the query image and affinity computation) are shared between objects while the rest are not. We experimented with object interaction within the object transformer in the early stage of this project but did not obtain positive results.

Figure S6 plots the FPS against the number of objects. Our method slows down with more objects but remains real-time when handling a common number of objects in a scene (29.9 FPS with 5 objects). For instance, the BURST [5] dataset averages 5.57 object tracks per video and DAVIS-2017 [7] averages just 2.03.

Additionally, we plot the memory usage with respect to the number of processed frames during inference in Figure S7.

G.2. Streaming Average Algorithm for the Object Memory

To recap, we store a compact set of N vectors which make up a high-level summary of the target object in the object memory $S \in \mathbb{R}^{N\times C}$. 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

$$S_q = \frac{\sum_{i=1}^{THW} U(i)W_q(i)}{\sum_{i=1}^{THW} W_q(i)}. \tag{S1}$$

During inference, we use a classic streaming average algorithm such that this operation takes constant time and memory with respect to the memory length. Concretely, for the q-th object memory at time step t, we keep track of a cumulative memory $\sigma_{S_q}^t \in \mathbb{R}^C$ and a cumulative weight $\sigma_{W_q}^t \in \mathbb{R}$. We update the accumulators and find S_q via

$$\sigma_{S_q}^t = \sigma_{S_q}^{t-1} + \sum_{i=1}^{THW} U(i)W_q(i), \qquad \sigma_{W_q}^t = \sigma_{W_q}^{t-1} + \sum_{i=1}^{THW} W_q(i), \qquad \text{and} \qquad S_q = \frac{\sigma_{S_q}^t}{\sigma_{W_q}^t}, \tag{S2}$$

where U and W_q can be discarded after every time step.

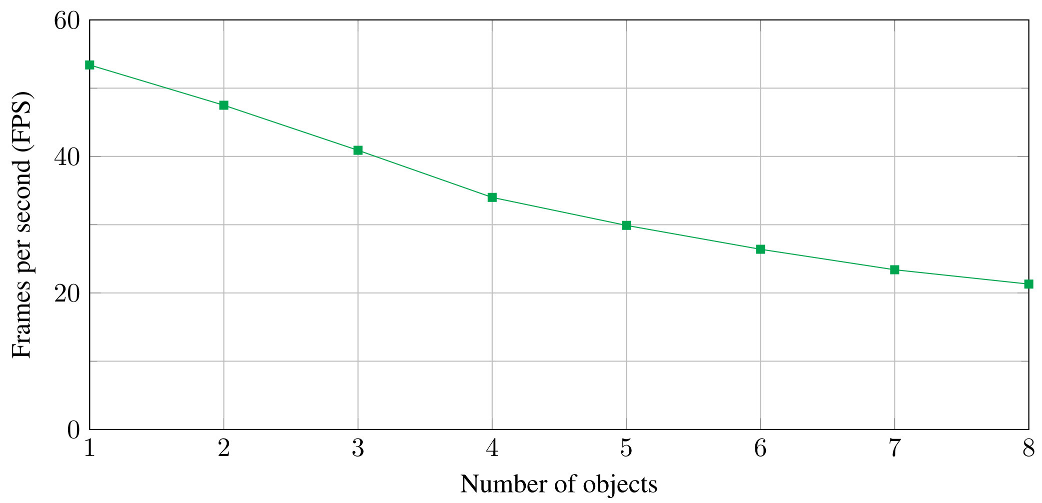


Figure S6. Cutie-small's processing speed with respect to the number of objects in the video. Common benchmarks (DAVIS [7], YouTubeVOS [1], and MOSE [8]) average 2-3 objects per video with longer-term benchmarks like BURST [5] averaging 5.57 objects per video – our model remains real-time (25+ FPS) in these scenarios. For evaluation, we use standard 854 $\times$ 480 test videos with 100 frames each.

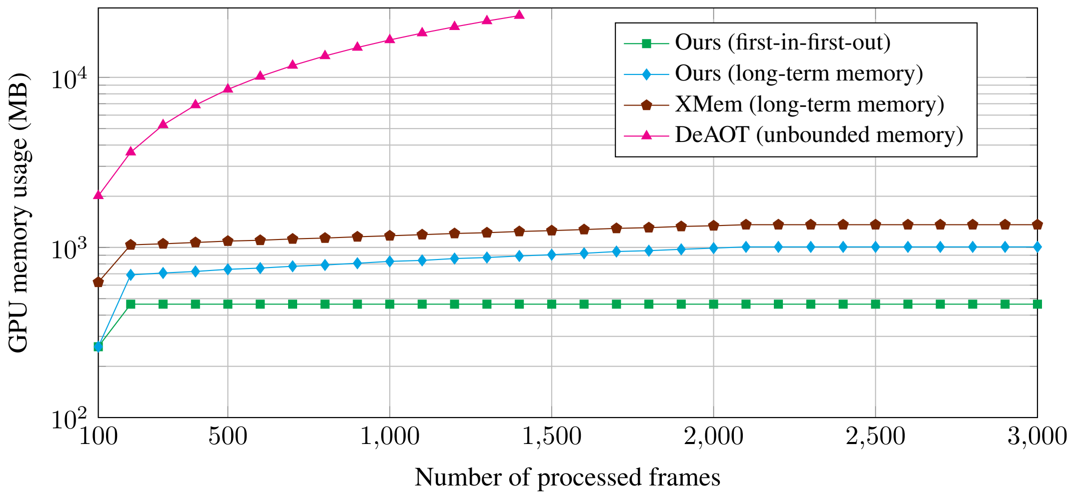


Figure S7. Running GPU memory usage (log-scale) comparisons with respect to the number of processed frames during inference. By default, we use a first-in-first-out (FIFO) memory bank which leads to constant memory usage over time. Optionally, we include the long-term memory from XMem [4] in our method for better performance on long videos. Our method (with long-term memory) uses less memory than XMem because of a smaller channel size (256 in our model; 512 in XMem). DeAOT [3] has an unbounded memory size and is impractical for processing long videos – our hardware (32GB V100, server-grade GPU) cannot process beyond 1,400 frames.

G.3. Training Details

As mentioned in the main paper, we train our network in two stages: static image pretraining and video-level main training following prior works [4, 19, 21]. Backbone weights are initialized from ImageNet [22] pretraining, following prior work [4, 19, 21]. We implement our network with PyTorch [23] and use automatic mixed precision (AMP) for training.

G.3.1 Pretraining

Our pretraining pipeline follows the open-sourced code of [4, 12, 20], and is described here for completeness. For pretraining, we use a set of image segmentation datasets: ECSSD [17], DUTS [24], FSS-1000 [25], HRSOD [26], and BIG [27]. We mix these datasets and sample HRSOD [26] and BIG [27] five times more often than the others as they are more accurately annotated. From a sampled image-segmentation pair, we generate synthetic sequences of length three by deforming the pair with random affine transformation, thin plate spline transformation [28], and cropping (with crop size 384×384). With the generated sequence, we use the first frame (with ground-truth segmentation) as the memory frame to segment the second frame. Then, we encode the second frame with our predicted segmentation and concatenate it with the first-frame memory to segment the third frame. Loss is computed on the second and third frames and back-propagated through time.

G.3.2 Main Training

For main training, we use two different settings. The "without MOSE" setting mixes the training sets of DAVIS-2017 [7] and YouTubeVOS-2019 [1]. The "with MOSE" setting mixes the training sets of DAVIS-2017 [7], YouTubeVOS-2019 [1], and MOSE [8]. In both settings, we sample DAVIS [7] two times more often as its annotation is more accurate. To sample a training sequence, we first randomly select a "seed" frame from all the frames and randomly select seven other frames from the same video. We re-sample if any two consecutive frames have a temporal frame distance above D. We employ a curriculum learning schedule following [4] for D, which is set to $[5, 10, 15, 5]$ correspondingly after $[0\%, 10\%, 30\%, 80\%]$ of training iterations.

For data augmentation, we apply random horizontal mirroring, random affine transformation, cut-and-paste [29] from another video, and random resized crop (scale $[0.36, 1.00]$, crop size 480×480). We follow stable data augmentation [18] to apply the same crop and rotation to all the frames in the same sequence. We additionally apply random color jittering and random grayscaling to the sampled images following [4, 20].

To train on a sampled sequence, we follow the process of pretraining, except that we only use a maximum of three memory frames to segment a query frame following [4]. When the number of past frames is smaller or equal to 3, we use all of them, otherwise, we randomly sample three frames to be the memory frames. We compute the loss at all frames except the first one and back-propagate through time.

G.3.3 Point Supervision

As mentioned in the main paper, we adopt point supervision [15] for training. As reported by [15], using point supervision for computing the loss has insignificant effects on the final performance while using only one-third of the memory during training. In Cutie, we note that using point supervision reduces the memory cost during loss computation but has an insignificant impact on the overall memory usage. We use importance sampling with default parameters following [15], i.e., with an oversampling ratio of 3, and sample 75% of all points from uncertain points and the rest from a uniform distribution. We use the uncertainty function for semantic segmentation (by treating each object as an object class) from [30], which is the logit difference between the top-2 predictions. Note that using point supervision also focuses the loss in uncertain regions but this is not unique to our framework. Prior works XMem [4] and DeAOT [3] use bootstrapped cross-entropy to similarly focus on difficult-to-segment pixels. Overall, we do not notice significant segmentation accuracy differences in using point supervision vs. the loss in XMem [4].

G.4. Decoder Architecture

Our decoder design follows XMem [4] with a reduced number of channels. XMem [4] uses 256 channels while we use 128 channels. This reduction in the number of channels improves the running time. We do not observe a performance drop from this reduction which we think is attributed to better input features (which are already refined by the object transformer).

The inputs to the decoder are the object readout feature R_L at stride 16 and skip-connections from the query encoder at strides 8 and 4. The skip-connection features are first projected to C dimensions with a 1×1 convolution. We process the object readout features with two upsampling blocks and incorporate the skip-connections for high-frequency information in each block. In each block, we first bilinearly upsample the input feature by two times, then add the upsampled features with the skip-connection features. This sum is processed by a residual block [31] with two 3×3 convolutions as the output of the upsample block. In the final layer, we use a 3×3 convolution to predict single-channel logits for each object. The logits are bilinearly upsampled to the original input resolution. In the multi-object scenario, we use soft-aggregation [19] to merge the object logits.

G.5. Details on Pixel Memory

As discussed in the main paper, we derive our pixel memory design from XMem [4] without claiming contributions. Namely, the attentional component is derived from the working memory, and the recurrent component is derived from the sensory memory of XMem [4]. Long-term memory [4], which compresses the working memory during inference, can be adopted without re-training for evaluation on long videos.

G.5.1 Attentional Component

For the attentional component, we store memory keys $\mathbf{k} \in \mathbb{R}^{THW \times C^{\text{k}}}$ and memory value $\mathbf{v} \in \mathbb{R}^{THW \times C}$ and later retrieve features using a query key $\mathbf{q} \in \mathbb{R}^{HW \times C^{\text{k}}}$. Here, T is the number of memory frames and H, W are image dimensions at stride 16. As we use the anisotropic L2 similarity function [4], we additionally store a memory shrinkage $\mathbf{s} \in [1, \infty]^{THW}$ term and use a query selection term $\mathbf{e} \in [0, 1]^{HW \times C^{\text{k}}}$ during retrieval.

The anisotropic L2 similarity function $d(\cdot, \cdot)$ is computed as

$$d(\mathbf{q}_i, \mathbf{k}_j) = -\mathbf{s}_j \sum_{c}^{C^{\text{k}}} \mathbf{e}_{ic}(\mathbf{k}_{ic} - \mathbf{q}_{jc}). \tag{S3}$$

We compute memory keys $\mathbf{k}$, memory shrinkage terms $\mathbf{s}$, query keys $\mathbf{q}$, and query selection terms $\mathbf{e}$ by projecting features encoded from corresponding RGB images using the query encoder. Since these terms are only dependent on the image, they, and thus the affinity matrix A^{pix} can be shared between multiple objects with no additional costs. The memory value $\mathbf{v}$ is computed by fusing features from the mask encoder (that takes both image and mask as input) and the query encoder. This fusion is done by first projecting the input features to C dimensions with 1×1 convolutions, adding them together, and processing the sum with two residual blocks, each with two 3×3 convolutions.

G.5.2 Recurrent Component

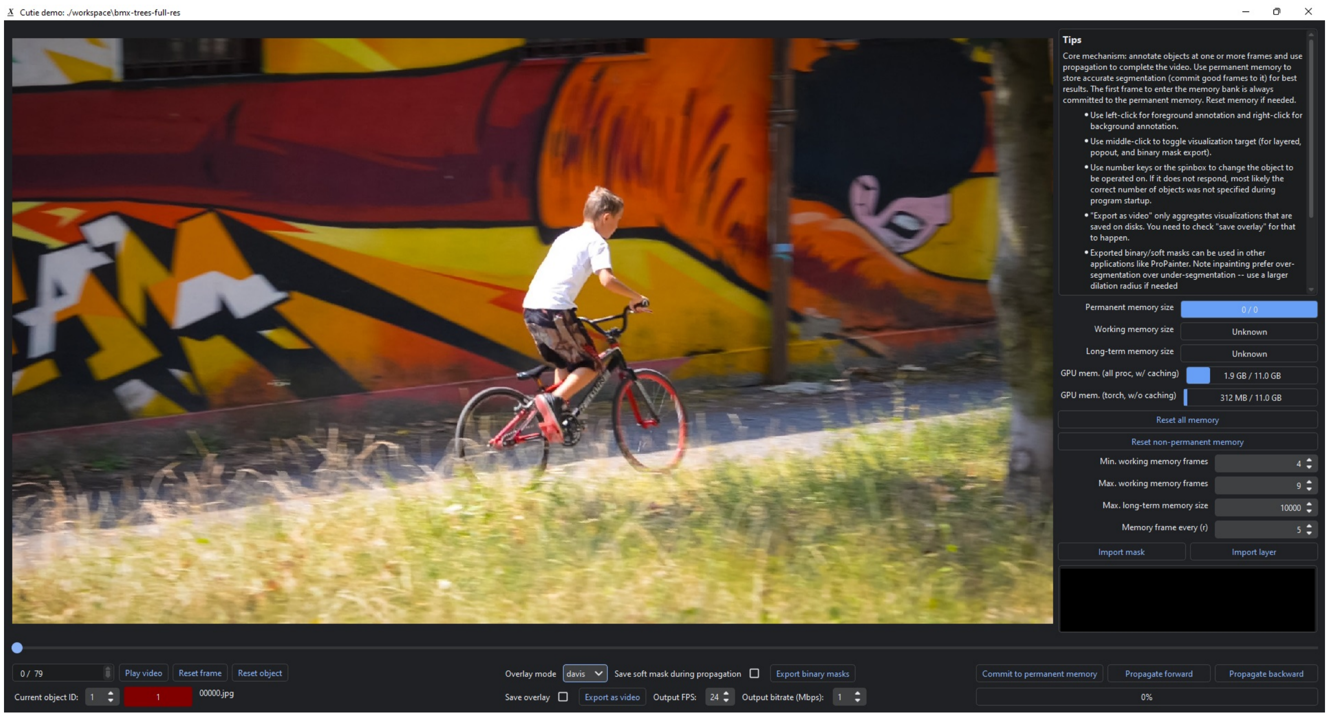
The recurrent component stores a hidden state $\mathbf{h}^{HW \times C}$ which is updated by a Gated Recurrent Unit (GRU) [32] every frame. This GRU takes multi-scale inputs (from stride 16, 8, and 4) from the decoder to update the hidden state $\mathbf{h}$. We first area-downsample the input features to stride 16, then project them to C dimensions before adding them together. This summed input feature, together with the last hidden state, is fed into a GRU as defined in XMem [4] to generate a new hidden state.

Every time we insert a new memory frame, i.e., every r-th frame, we apply a *deep update* as in XMem [4]. Deep update uses a separate GRU that takes the output of the mask encoder as its input feature. This incurs minimal overhead as the mask encoder is invoked during memory frame insertion anyway. Deep updates refresh the hidden state and allow it to receive updates from a deeper network.

H. Interactive Tool for Video Segmentation

Based on the video object segmentation capability of Cutie, we build an interactive video segmentation tool to facilitate research and data annotation. We follow the decoupled paradigm of MiVOS [12] – users annotate one or more frames using an interactive image segmentation tool such as RITM [33] and use Cutie for propagating these image segmentations through the video. Users can also include multiple permanent memory frames (as in XMem++ [34]) to increase the segmentation robustness. Figure S8 shows a screenshot of this tool.

This interactive tool is open-source to benefit researchers, data annotators, and video editors.



Tips

Core mechanism: annotate objects at one or more frames and use propagation to complete the video. Use permanent memory to store accurate segmentation (commit good frames to it) for best results. The first frame to enter the memory bank is always committed to the permanent memory. Reset memory if needed.

- Use left-click for foreground annotation and right-click for background annotation.
- Use middle-click to toggle visualization target (for layered, popout, and binary mask export).
- Use number keys or the spinbox to change the object to be operated on. If it does not respond, most likely the correct number of objects was not specified during program startup.
- "Export as video" only aggregates visualizations that are saved on disks. You need to check "save overlay" for that to happen.
- Exported binary/soft masks can be used in other applications like ProPainter. Note inpainting prefer over-segmentation over under-segmentation -- use a larger dilation radius if needed

Permanent memory size: 0 / 0
Working memory size: Unknown
Long-term memory size: Unknown
GPU mem. (all proc, w/ caching): 1.9 GB / 11.0 GB
GPU mem. (torch, w/o caching): 312 MB / 11.0 GB
Min. working memory frames: 4
Max. working memory frames: 9
Max. long-term memory size: 10000
Memory frame every (r): 5
0 / 79
Current object ID: 1
00000.jpg
Overlay mode: davis
Save soft mask during propagation: ☐
Save overlay: ☐
Output FPS: 24
Output bitrate (Mbps): 1
0%

Figure S8. Screenshot of our interactive video segmentation tool.

References

[1] 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. 1, 5, 6, 8, 9

[2] 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. 1, 5, 6, 7

[3] Zongxin Yang and Yi Yang. Decoupling features in hierarchical propagation for video object segmentation. In *NeurIPS*, 2022. 1, 6, 8, 9

[4] Ho Kei Cheng and Alexander G Schwing. XMem: Long-term video object segmentation with an atkinson-shiffrin memory model. In *ECCV*, 2022. 1, 3, 4, 6, 7, 8, 9, 10

[5] 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. 3, 5, 6, 7, 8

[6] 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. 3

[7] 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. 4, 5, 6, 7, 8, 9

[8] 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. 4, 5, 6, 8, 9

[9] Qiangqiang Wu, Tianyu Yang, Wei Wu, and Antoni Chan. Scalable video object segmentation with simplified framework. In *ICCV*, 2023. 5

[10] 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. 5

[11] 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. 4, 5

[12] 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. 5, 8, 10

[13] 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. 4, 5

[14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 5

[15] Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *CVPR*, 2022. 4, 5, 9

[16] 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. 5

[17] Jianping Shi, Qiong Yan, Li Xu, and Jiaya Jia. Hierarchical image saliency detection on extended cssd. In *TPAMI*, 2015. 4, 8

[18] Ho Kei Cheng, Seoung Wug Oh, Brian Price, Alexander Schwing, and Joon-Young Lee. Tracking anything with decoupled video segmentation. In *ICCV*, 2023. 5, 6, 7, 9

[19] Seoung Wug Oh, Joon-Young Lee, Ning Xu, and Seon Joo Kim. Video object segmentation using space-time memory networks. In *ICCV*, 2019. 6, 8, 9

[20] 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. 6, 7, 8, 9

[21] Zongxin Yang, Yunchao Wei, and Yi Yang. Associating objects with transformers for video object segmentation. In *NeurIPS*, 2021. 8

[22] 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. 8

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 8

[24] 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. 8

[25] 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. 8

[26] Yi Zeng, Pingping Zhang, Jianming Zhang, Zhe Lin, and Huchuan Lu. Towards high-resolution salient object detection. In *ICCV*, 2019. 8

[27] 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. 8

[28] Jean Duchon. Splines minimizing rotation-invariant semi-norms in sobolev spaces. In *Constructive Theory of Functions of Several Variables*, 1977. 9

[29] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation. In *CVPR*, 2021. 9

[30] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *CVPR*, 2020. 9

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 9

[32] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *arXiv*, 2014. 10

[33] Konstantin Sofiiuk, Ilya A Petrov, and Anton Konushin. Reviving iterative training with mask guidance for interactive segmentation. In *ICIP*, 2022. 10

[34] Maksym Bekuzarov, Ariana Bermudez, Joon-Young Lee, and Hao Li. Xmem++: Production-level video segmentation from few annotated frames. In *ICCV*, 2023. 10