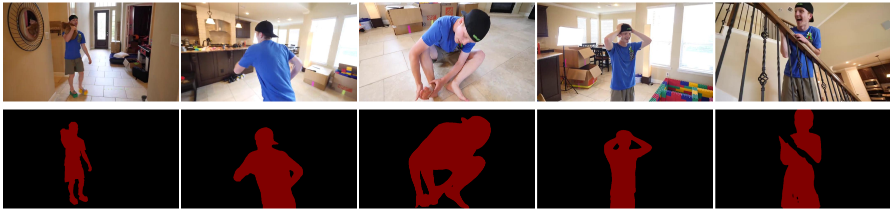
XMem: Long-Term Video Object Segmentation with an Atkinson-Shiffrin Memory Model

Anonymous ECCV submission

Paper ID 2



Frame 0 (input) Frame 295 Frame 460 Frame 1285 Frame 2327

Abstract. We present XMem, a video object segmentation architecture for long videos with unified feature memory stores inspired by the Atkinson-Shiffrin memory model. Prior work on video object segmentation typically only uses one type of feature memory. For videos longer than a minute, a single feature memory model tightly links memory consumption and accuracy. In contrast, following the Atkinson-Shiffrin model, we develop an architecture that incorporates multiple independent yet deeply-connected feature memory stores: a rapidly updated *sensory memory*, a high-resolution *working memory*, and a compact thus sustained *long-term memory*. Crucially, we develop a memory potentiation algorithm that routinely consolidates actively used working memory elements into the long-term memory, which avoids memory explosion and minimizes performance decay for long-term prediction. Combined with a new memory reading mechanism, XMem greatly exceeds state-of-the-art performance on long-video datasets while being on par with state-of-the-art methods (that do not work on long videos) on short-video datasets.[1]

1 Introduction

Video object segmentation (VOS) highlights specified target objects in a given video. Here, we focus on the semi-supervised setting where a first-frame annotation is provided by the user, and the method segments objects in all other frames as accurately as possible while preferably running in real-time, online, and while having a small memory footprint even when processing long videos.

As information has to be propagated from the given annotation to other video frames, most VOS methods employ a *feature memory* to store relevant deep-net representations of an object. Recent state-of-the-art VOS methods use attention [10,6,15,3,19] to link representations of past frames stored in the feature memory with features extracted from the newly observed query frame which needs to be segmented. Despite the high performance of these methods, they require a large amount of GPU memory to store past frame representations.

[1] Code is available at `hkchengrex.github.io/XMem`

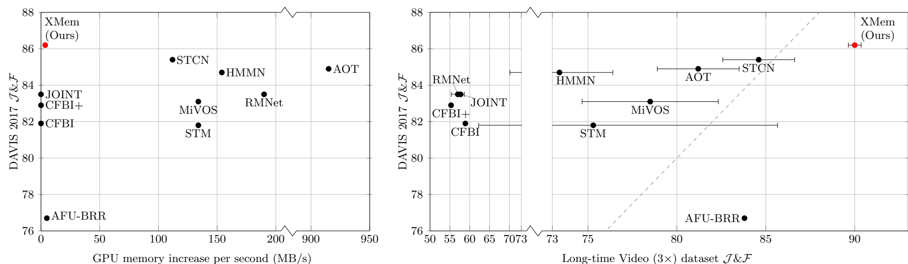


Fig. 1. Do state-of-the-art VOS algorithms scale well? **Left**: Memory scaling with respect to short-term segmentation quality. **Right**: Segmentation quality scaling from standard short videos (y-axis) to long videos (x-axis) – the dashed line indicates a 1:1 performance ratio. Error bars show standard deviations in memory sampling if applicable. See Section 3.1 for details.

In practice, they usually struggle to handle videos longer than a minute on consumer-grade hardware.

Methods that are specifically designed for VOS in long videos exist [8,7]. However, they often sacrifice segmentation quality. Specifically, these methods reduce the size of the representation during feature memory insertion by merging new features with those already stored in the feature memory. As high-resolution features are compressed right away, they produce less accurate segmentations. Figure 1 shows the relation between GPU memory consumption and segmentation quality in short/long video datasets (details are given in Section 3.1).

We think this undesirable connection of performance and GPU memory consumption is a direct consequence of using a single feature memory type. To address this limitation we propose a unified memory architecture, dubbed XMem. Inspired by the Atkinson–Shiffrin memory model [1] which hypothesizes that the human memory consists of three components, XMem maintains three independent yet deeply-connected feature memory stores: a rapidly updated *sensory memory*, a high-resolution *working memory*, and a compact thus sustained *long-term memory*. In XMem, the sensory memory corresponds to the hidden representation of a GRU [4] which is updated every frame. It provides temporal smoothness but fails for long-term prediction due to representation drift. To complement, the working memory is agglomerated from a subset of historical frames and considers them equally [10,3] without drifting over time. To control the size of the working memory, XMem routinely consolidates its representations into the long-term memory, inspired by the consolidation mechanism in the human memory [14]. XMem stores long-term memory as a set of highly compact prototypes. For this, we develop a memory potentiation algorithm that aggregates richer information into these prototypes to prevent aliasing due to sub-sampling. To read from the working and long-term memory, we devise a space-time memory reading operation. The three feature memory stores combined permit handling long videos with high accuracy while keeping GPU memory usage low.

We find XMem to greatly exceed prior state-of-the-art results on the Long-time Video dataset [8]. Importantly, XMem is also on par with current state-of-

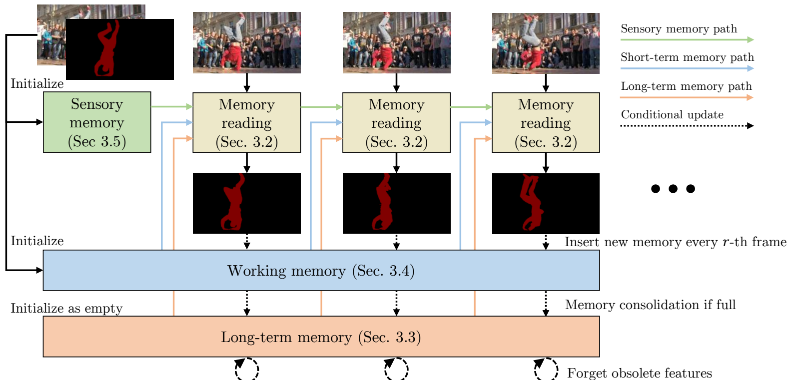


Fig. 2. Overview of XMem. The memory reading operation extracts relevant features from all three memory stores and uses those features to produce a mask. To incorporate new memory, the sensory memory is updated every frame while the working memory is only updated every r-th frame. The working memory is consolidated into the long-term memory in a compact form when it is full, and the long-term memory will forget obsolete features over time.

the-art (that cannot handle long videos) on short-video datasets [12,17]. Related works are discussed in the appendix.

2 XMem

2.1 Overview

Figure 2 provides an overview of XMem. For readability, we consider a single target object. However, note that XMem is implemented to deal with multiple objects, which is straightforward. Given the image and target object mask at the first frame (top-left of Figure 2), XMem tracks the object and generates corresponding masks for subsequent query frames. For this, we first initialize the different feature memory stores using the inputs. For each subsequent query frame, we perform memory reading (Section 2.2) from long-term memory (Section 2.3), working memory , and sensory memory respectively. The readout features are used to generate a segmentation mask. Then, we update each of the feature memory stores at different frequencies. We update the sensory memory every frame and insert features into the working memory at every r-th frame. When the working memory reaches a pre-defined maximum of $T_{\max}$ frames, we consolidate features from the working memory into the long-term memory in a highly compact form. When the long-term memory is also full (which only happens after processing thousands of frames), we discard obsolete features to bound the maximum GPU memory usage. These feature memory stores work in conjunction to provide high-quality features with low GPU memory usage even for very long videos.

In the following, we will first describe the memory reading operation before discussing each feature memory store in detail.

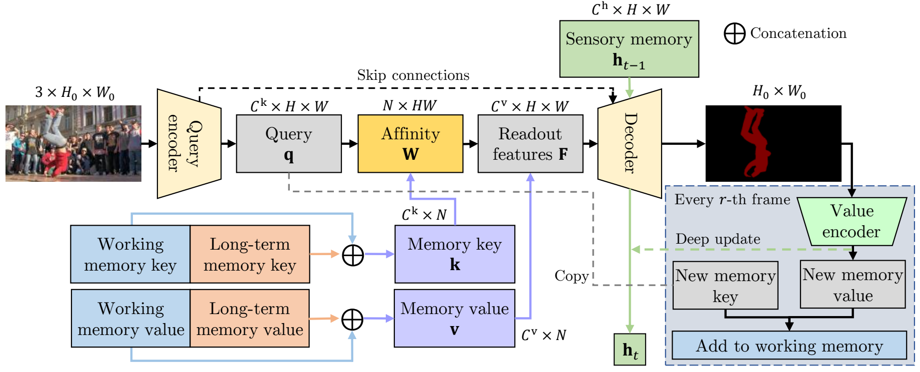


Fig. 3. Process of memory reading and mask decoding of a single query frame. We extract query $\mathbf{q}$ from the image and perform attention-based memory reading from the working/long-term memory to obtain features F. Together with the sensory memory, it is fed into the decoder to generate a mask. For every r-th frame, we store new features into the working memory and perform a deep update to the sensory memory.

2.2 Memory Reading

Figure 3 illustrates the process of memory reading and mask generation for a single frame. The mask is computed via the decoder which uses as input the short-term sensory memory $\mathbf{h}_{t-1} \in \mathbb{R}^{C^{\text{h}} \times H \times W}$ and a feature $\mathbf{F} \in \mathbb{R}^{C^{\text{v}} \times H \times W}$ representing information stored in both the long-term and the working memory.

The feature $\mathbf{F}$ representing information stored in both the long-term and the working memory is computed via the readout operation

$$\mathbf{F} = \mathbf{v}\mathbf{W}(\mathbf{k}, \mathbf{q}). \tag{1}$$

Here, $\mathbf{k} \in \mathbb{R}^{C^{\text{k}} \times N}$ and $\mathbf{v} \in \mathbb{R}^{C^{\text{v}} \times N}$ are C^{k}- and C^{v}-dimensional keys and values for a total of N memory elements which are stored in both the long-term and working memory. Moreover, $\mathbf{W}(\mathbf{k}, \mathbf{q})$ is an affinity matrix of size $N \times HW$, representing a readout operation that is controlled by the key $\mathbf{k}$ and a query $\mathbf{q} \in \mathbb{R}^{C^{\text{k}} \times HW}$ obtained from the query frame through the query encoder. The readout operation maps every query element to a distribution over all N memory elements and correspondingly aggregates their values $\mathbf{v}$.

The affinity matrix $\mathbf{W}(\mathbf{k}, \mathbf{q})$ is obtained by applying a softmax on the memory dimension (rows) of a similarity matrix $\mathbf{S}(\mathbf{k}, \mathbf{q})$ which contains the pairwise similarity between every key element and every query element. For computing the similarity matrix, we use a newly proposed similarity function (*anisotropic L2*) by introducing two new scaling terms to the L2 similarity proposed in STCN [3].

Concretely, the key is associated with a <u>s</u>hrinkage term $\mathbf{s} \in [1, \infty)^N$ and the query is associated with a s<u>e</u>lection term $\mathbf{e} \in [0, 1]^{C^{\text{k}} \times HW}$. Then, the similarity between the i-th key element and the j-th query element is computed via

$$\mathbf{S}(\mathbf{k}, \mathbf{q})_{ij} = -\mathbf{s}_i \sum_{c}^{C^{\text{k}}} \mathbf{e}_{cj} \left(\mathbf{k}_{ci} - \mathbf{q}_{cj}\right)^2, \tag{2}$$

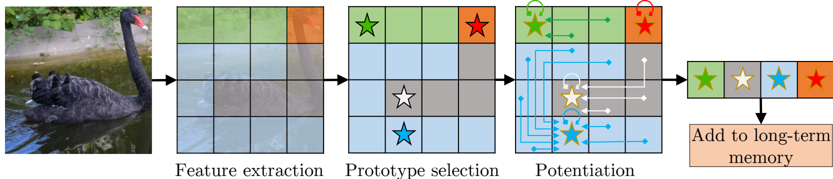


Fig. 4. Memory consolidation procedure. Given an image, we extract features as memory keys (image stride exaggerated). We visualize these features with colors. For memory consolidation, we first select prototype keys (stars) from the candidates (all grids). Then, we invoke potentiation which non-locally aggregates values from all the candidates to generate more representative prototype values (golden outline). The resultant prototype keys and values are added to the long-term memory. Only one frame is shown here – in practice multiple frames are used in a single consolidation.

which equates to the original L2 similarity [3] if $\mathbf{s}_i = \mathbf{e}_{cj} = 1$ for all i, j, and c. The <u>s</u>hrinkage term $\mathbf{s}$ directly scales the similarity and explicitly encodes confidence – a high shrinkage represents low confidence and leads to a more local influence. Differently, the s<u>e</u>lection term $\mathbf{e}$ controls the relative importance of each channel in the key space such that attention is given to the more discriminative channels.

The selection term $\mathbf{e}$ is generated together with the query $\mathbf{q}$ by the query encoder. The shrinkage term $\mathbf{s}$ is collected together with the key $\mathbf{k}$ and the value $\mathbf{v}$ from the working and the long-term memory.[2] The collection is simply implemented as a concatenation in the last dimension: $\mathbf{k} = \mathbf{k}^{\mathrm{w}} \oplus \mathbf{k}^{\mathrm{lt}}$ and $\mathbf{v} = \mathbf{v}^{\mathrm{w}} \oplus \mathbf{v}^{\mathrm{lt}}$, where superscripts 'w' and 'lt' denote working and long-term memory respectively. The working memory consists of key $\mathbf{k}^{\mathrm{w}} \in \mathbb{R}^{C^{\mathrm{k}} \times THW}$ and value $\mathbf{v}^{\mathrm{w}} \in \mathbb{R}^{C^{\mathrm{v}} \times THW}$, where T is the number of working memory frames. The long-term memory similarly consists of keys $\mathbf{k}^{\mathrm{lt}} \in \mathbb{R}^{C^{\mathrm{k}} \times L}$ and values $\mathbf{v}^{\mathrm{lt}} \in \mathbb{R}^{C^{\mathrm{v}} \times L}$, where L is the number of long-term memory prototypes. Thus, the total number of elements in the working/long-term memory is $N = THW + L$.

Next, we discuss the feature memory stores in detail.

2.3 Long-Term Memory

Motivation. A long-term memory is crucial for handling long videos. With the goal of storing a set of compact (consume little GPU memory) yet representative (lead to high segmentation quality) memory features, we design a *memory consolidation* procedure that selects *prototypes* from the working memory and enriches them with a *memory potentiation* algorithm, as illustrated in Figure 4.

We perform memory consolidation when the working memory reaches a pre-defined size $T_{\max}$. The first frame (with user-provided ground-truth) and the most recent $T_{\min} - 1$ memory frames will be kept in the working memory as a high-resolution buffer while the remainder ($T_{\max} - T_{\min}$ frames) are *candidates* for being converted into long-term memory representations. We refer to the keys

[2] For brevity, we omit the handling of these two scaling terms in memory updates for the rest of the paper. They are updated in the same way as the value.

and values of these candidates as $\mathbf{k}^{\mathrm{c}} \subset \mathbf{k}^{\mathrm{w}}$ and $\mathbf{v}^{\mathrm{c}} \subset \mathbf{v}^{\mathrm{w}}$ respectively. In the following, we describe the prototype selection process that picks a compact set of prototype keys $\mathbf{k}^{\mathrm{p}} \subset \mathbf{k}^{\mathrm{c}}$, and the memory potentiation algorithm that generates enriched prototype values $\mathbf{v}^{\mathrm{p}}$ associated with these prototype keys. Finally, these prototype keys and values are appended to the long-term memory $\mathbf{k}^{\mathrm{lt}}$ and $\mathbf{v}^{\mathrm{lt}}$.

Prototype Selection. In this step, we sample a small representative subset $\mathbf{k}^{\mathrm{p}} \subset \mathbf{k}^{\mathrm{c}}$ from the candidates as *prototypes*. It is essential to pick only a small number of prototypes, as their amount is directly proportional to the size of the resultant long-term memory. Inspired by human memory which moves frequently accessed or studied patterns to a long-term store, we pick candidates with high *usage*. Concretely, we pick the top-P frequently used candidates as prototypes. "Usage" of a memory element is defined by its cumulative total affinity (probability mass) in the affinity matrix $\mathbf{W}$ (Eq. (1)), and normalized by the duration that each candidate is in the working memory. Note that the duration for each candidate is at least $r \cdot (T_{\min} - 1)$, leading to stable usage statistics. We obtain the keys of these prototypes as $\mathbf{k}^{\mathrm{p}} \in \mathbb{R}^{C^{\mathrm{k}} \times P}$.

Memory Potentiation. Note that, so far, our sampling of prototype keys $\mathbf{k}^{\mathrm{p}}$ from the candidate keys $\mathbf{k}^{\mathrm{c}}$ is both *sparse* and *discrete*. If we were to sample the prototypes values $\mathbf{v}^{\mathrm{p}}$ in the same manner, the resultant prototypes would inevitably under-represent other candidates and would be prone to *aliasing*. The common technique to prevent aliasing is to apply an anti-aliasing (e.g., Gaussian) filter [5]. Similarly motivated, we perform filtering and aggregate more information into every sampled prototype. While standard filtering can be easily performed on the image plane (2D) or the spatial-temporal volume (3D), it leads to blurry features – especially near object boundaries. To alleviate, we instead construct the neighbourhood for the filtering in the high dimensional (C^{k}) key space, such that the highly expressive adjacency information given by the keys $\mathbf{k}^{\mathrm{p}}$ and $\mathbf{k}^{\mathrm{c}}$ is utilized. As these keys have to be computed and stored for memory reading anyway, it is also economical in terms of run-time and memory consumption.

Concretely, for each prototype, we aggregate values from all the value candidates $\mathbf{v}^{\mathrm{c}}$ via a weighted average. The weights are computed using a softmax over the key-similarity. For this, we conveniently re-use Eq. (2). By substituting the memory key $\mathbf{k}$ with the candidate key $\mathbf{k}^{\mathrm{c}}$, and the query $\mathbf{q}$ with the prototype keys $\mathbf{k}^{\mathrm{p}}$, we obtain the similarity matrix $\mathbf{S}(\mathbf{k}^{\mathrm{c}}, \mathbf{k}^{\mathrm{p}})$. As before, we use a softmax to obtain the affinity matrix $\mathbf{W}(\mathbf{k}^{\mathrm{c}}, \mathbf{k}^{\mathrm{p}})$ (where every prototype corresponds to a distribution over candidates). Then, we compute the prototype values $\mathbf{v}^{\mathrm{p}}$ via

$$\mathbf{v}^{\mathrm{p}} = \mathbf{v}^{\mathrm{c}} \mathbf{W}(\mathbf{k}^{\mathrm{c}}, \mathbf{k}^{\mathrm{p}}). \tag{3}$$

Finally, $\mathbf{k}^{\mathrm{p}}$ and $\mathbf{v}^{\mathrm{p}}$ are appended to the long-term memory $\mathbf{k}^{\mathrm{lt}}$ and $\mathbf{v}^{\mathrm{lt}}$ respectively – concluding the memory consolidation process. Note, similar prototypical approximations have been used in transformers [16,11]. Differently, our approach uses a novel prototype selection scheme suitable for video object segmentation.

Removing Obsolete Features. Although the long-term memory is extremely compact with a high ($> 6000\%$) compression ratio, memory can still overflow since we are continuously appending new features. Empirically, with a 6GB memory budget (e.g., a consumer-grade mid-end GPU), we can process up to 34,000 frames before running into any memory issues. To handle even longer videos, we introduce a least-frequently-used (LFU) eviction algorithm similar to [8]. Long-term memory elements with the least usage will be evicted when a pre-defined memory limit is reached.

The long-term memory is key to enabling efficient and accurate segmentation of long videos. Next, we discuss the working memory, which is crucial for accurate short-term prediction. It acts as the basis for the long-term memory.

2.4 Working and Sensory Memory

The working memory mainly follows STCN [3], and the sensory memory is implemented as a Gated Recurrent Unit (GRU) [4]. Please refer to to the appendix for details.

3 Experiments

Please refer to the appendix for detailed experimental settings, results on short-term datasets, and ablation studies.

3.1 Long-Time Video Dataset

To evaluate long-term performance, we test models on the Long-time Video dataset [8] which contains three videos with more than 7,000 frames in total. We also synthetically extend it to even longer variants by playing the video back and forth. $n\times$ denotes a variant that has n times the number of frames. For comparison, we select state-of-the-art methods with available implementation as we need to re-run their models. Most SOTA methods cannot handle long videos natively. We first measure their GPU memory increase per frame by averaging the memory consumption difference between the 100-th and 200-th frame in 480p.[3] Figure 1 (left) shows our findings, assuming 24FPS. For methods with prohibitive memory usage on long videos, we limit their feature memory insertion frequency accordingly, using 50 memory frames in STM as a baseline following [8]. Our method uses less memory than this baseline. We note that a low memory insertion frequency leads to high variances in performance, thus we run these experiments with 5 evenly-spaced offsets to the memory insertion routine and show "mean $\pm$ standard deviation" if applicable. In this dataset, we use $r = 10$. We do not find BL30K [2] pretraining to help here.

Table 1 tabulates the quantitative results, and Figure 1 (right) plots the short-term performance against the long-term performance. Methods that use a

[3] We make sure to exclude any caching or input buffering overhead.

Table 1. Quantitative comparisons on the Long-time Video dataset [8].

	Long-time Video (1×)			Long-time Video (3×)			$\Delta_{1\times\rightarrow 3\times}$
Method	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$	$\mathcal{F}$	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$	$\mathcal{F}$	$\mathcal{J}\&\mathcal{F}$
CFBI+ [20]	50.9	47.9	53.8	55.3	54.0	56.5	4.4
RMNet [15]	59.8 ±3.9	59.7 ±8.3	60.0 ±7.5	57.0 ±1.6	56.6 ±1.5	57.3 ±1.8	-2.8
JOINT [9]	67.1 ±3.5	64.5 ±4.2	69.6 ±3.9	57.7 ±0.2	55.7 ±0.3	59.7 ±0.2	-9.4
CFBI [18]	53.5	50.9	56.1	58.9	57.7	60.1	5.4
HMMN [13]	81.5 ±1.8	79.9 ±1.2	83.0 ±1.5	73.4 ±3.3	72.6 ±3.1	74.3 ±3.5	-8.1
STM [10]	80.6 ±1.3	79.9 ±0.9	81.3 ±1.0	75.3 ±13.0	74.3 ±13.0	76.3 ±13.1	-5.3
MiVOS* [2]	81.1 ±3.2	80.2 ±2.0	82.0 ±3.1	78.5 ±4.5	78.0 ±3.7	79.0 ±5.4	-2.6
AOT [19]	84.3 ±0.7	83.2 ±3.2	85.4 ±3.3	81.2 ±2.5	79.6 ±3.0	82.8 ±2.1	-3.1
AFB-URR [8]	83.7	82.9	84.5	83.8	82.9	84.6	0.1
STCN [3]	87.3 ±0.7	85.4 ±1.1	89.2 ±1.1	84.6 ±1.9	83.3 ±1.7	85.9 ±2.2	-2.7
XMem (Ours)	**89.8**±0.2	**88.0**±0.2	**91.6**±0.2	**90.0**±0.4	**88.2**±0.3	**91.8**±0.4	0.2

temporally local feature window (CFBI(+) [18,20], JOINT [9]) have a constant memory cost but fail when they lose track of the context. Methods with a fast-growing memory bank (e.g., STM [10], AOT [19], STCN [3]) are forced to use a low feature memory insertion frequency and do not scale well to long videos. Figure 5 shows the scaling behavior of STCN vs. XMem in more detail.

3.2 Limitations

Our method sometimes fails when the target object moves too quickly or has severe motion blur as even the fastest updating sensory memory cannot catch up. See the appendix for examples. We think a sensory memory with a large receptive field that is more powerful than our baseline instantiation could help.

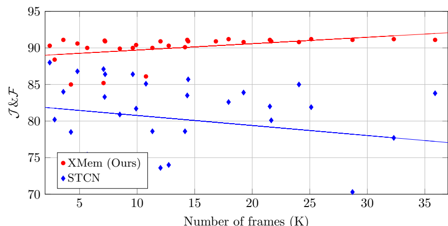


Fig. 5. Least-square fits of performance over video length for XMem and STCN [3] on variants of the Long-time Video dataset [8] from 1× to 10×. In longer videos, STCN decays due to missing context while ours stabilizes as we gain sufficient context.

4 Conclusion

We present XMem – to our best knowledge the first multi-store feature memory model used for video object segmentation. XMem achieves excellent performance with minimal GPU memory usage for both long and short videos. We believe XMem is a good step toward accessible VOS on mobile devices, and we hope to draw attention to the more widely-applicable long-term VOS task.

References

1. Atkinson, R.C., Shiffrin, R.M.: Human memory: A proposed system and its control processes. In: Psychology of learning and motivation, vol. 2, pp. 89–195. Elsevier (1968)
2. Cheng, H.K., Tai, Y.W., Tang, C.K.: Modular interactive video object segmentation: Interaction-to-mask, propagation and difference-aware fusion. In: CVPR (2021)
3. Cheng, H.K., Tai, Y.W., Tang, C.K.: Rethinking space-time networks with improved memory coverage for efficient video object segmentation. In: NeurIPS (2021)
4. Cho, K., Van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. In: arXiv (2014)
5. Forsyth, D., Ponce, J.: Computer vision: A modern approach. Prentice hall (2011)
6. Hu, L., Zhang, P., Zhang, B., Pan, P., Xu, Y., Jin, R.: Learning position and target consistency for memory-based video object segmentation. In: CVPR (2021)
7. Li, Y., Shen, Z., Shan, Y.: Fast video object segmentation using the global context module. In: ECCV (2020)
8. Liang, Y., Li, X., Jafari, N., Chen, J.: Video object segmentation with adaptive feature bank and uncertain-region refinement. In: NeurIPS (2020)
9. Mao, Y., Wang, N., Zhou, W., Li, H.: Joint inductive and transductive learning for video object segmentation. In: ICCV (2021)
10. Oh, S.W., Lee, J.Y., Xu, N., Kim, S.J.: Video object segmentation using space-time memory networks. In: ICCV (2019)
11. Patrick, M., Campbell, D., Asano, Y.M., Metze, I.M.F., Feichtenhofer, C., Vedaldi, A., Henriques, J., et al.: Keeping your eye on the ball: Trajectory attention in video transformers. In: NeurIPS (2021)
12. Pont-Tuset, J., Perazzi, F., Caelles, S., Arbeláez, P., Sorkine-Hornung, A., Van Gool, L.: The 2017 davis challenge on video object segmentation. In: arXiv:1704.00675 (2017)
13. Seong, H., Oh, S.W., Lee, J.Y., Lee, S., Lee, S., Kim, E.: Hierarchical memory matching network for video object segmentation. In: ICCV (2021)
14. Squire, L.R., Genzel, L., Wixted, J.T., Morris, R.G.: Memory consolidation. In: Cold Spring Harbor perspectives in biology. Cold Spring Harbor Lab (2015)
15. Xie, H., Yao, H., Zhou, S., Zhang, S., Sun, W.: Efficient regional memory network for video object segmentation. In: CVPR (2021)
16. Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., Singh, V.: Nyströmformer: A nyström-based algorithm for approximating self-attention. In: AAAI (2021)
17. Xu, N., Yang, L., Fan, Y., Yue, D., Liang, Y., Yang, J., Huang, T.: Youtube-vos: A large-scale video object segmentation benchmark. In: ECCV (2018)
18. Yang, Z., Wei, Y., Yang, Y.: Collaborative video object segmentation by foreground-background integration. In: ECCV (2020)
19. Yang, Z., Wei, Y., Yang, Y.: Associating objects with transformers for video object segmentation. In: NeurIPS (2021)
20. Yang, Z., Wei, Y., Yang, Y.: Collaborative video object segmentation by multi-scale foreground-background integration. In: PAMI (2021)