Appendices

A. PUMaVOS Dataset

PUMaVOS is a dataset that covers visually challenging segmentation scenarios, including, but not limited to high appearance variability due to changes in lighting, viewing angles, deformation, and scale changes; partial segmentation (where only a part of the object is being segmented), often with limited to no low-level image cues (e.g. half of a person's face) and occlusion. It consists of 24 videos, from 13.5 to 60 seconds long, 29 on average, with 480p resolution with different aspect ratios (both vertical and horizontal). There is a total of 21K densely annotated frames in **PUMaVOS**.

Examples of the videos from the dataset are illustrated in Fig. 9. Sequences "Half Face", "Guitar" and "Dog Tail" depict challenging partial segmentation use-case where only the left part of a person's face, the dog's tail or just the body of the guitar body is segmented, without following any image cues such as edges or corners. "Queen car" and "Full Face" contain changes in scale and heavy occlusion throughout the sequence, while "Pimples" and "Ice Cream" both contain very small objects that also move a lot throughout the scene. "Pimples" and "Caps" are also examples of ambiguity - there are multiple objects with similar appearances present in the video, but only some of them are supposed to be segmented. "Vlog" is one of the most challenging sequences in the dataset, as it not only contains partial segmentation with arbitrarily-defined boundaries (irrespective of the low-level image cues) but also has multiple target regions to segment, which are located on the same physical object, as well as having a lot of variability in lighting, static and dynamic background scenes, and frequent deformations, which are also present in "Shirt" and "Tattoo" sequences.

***PUMaVOS** and the source code of **XMem++** and the annotation algorithm are going to be released to the public after the paper is accepted. We also provide pseudocode for our frame selection algorithm.*

B. Frame Selection User Study

A user study was performed to analyze the effectiveness of the automatic frame selection module. A total of seven participants were selected - two experienced and two novice users. The users were given three videos each and asked to select 5 most representative frames (besides frame #0) that capture the variation in the appearance of the target object. The videos chosen from **PUMaVOS** are: sequences "Half Face" (shortened), "Ice Cream", and "Vlog". Videos last from 22s (673 frames) to 32s (960 frames).

The experienced users were explained how **XMem++** works internally in detail, as well as shown its predictions on a few sample videos, while novice users treated it as a blackbox. The participants were shown the full video and the annotation of the target object(s) for the first frame and were free to rewatch it as many times as they deemed necessary, then asked to pick the frames. They were told that the relative position of the frame does not matter, only its content. We recorded the time it took the users to select the frames (excluding the initial video viewing time) and compared it to the running time of the frame selection algorithm. We then took the chosen annotations and performed a quantitative comparison of speed and final segmentation accuracy using **XMem++**, presented in Table 4.

Group/Method	**IoU**	**F-score**	Average time, s
Algorithm	0.777	0.828	**1.7**
Experts	**0.805**	**0.855**	47.1
Non-experts	0.779	0.825	54.8

Table 4. Comparison of performance metrics across expert and non-expert with the frame annotation candidate selection algorithm.

Our algorithm is 27× faster than the expert users, and 32× faster than non-expert ones, while providing comparable performance to non-expert users' results. This makes it a practical tool for large-scale environments where it is infeasible to have a lot of trained experts to work on videos, while also having a practical application for expert users, who can use the algorithm to very quickly obtain a lot of potentially valuable annotation candidates and then select the best ones with their expertise, saving a lot of time in the process.

C. Additional Results

Please refer to the accompanying video to see all the video results and method comparisons. We highlight a number of highly complex scenes, where existing methods are challenged with varying scales, appearance, occlusions, and ambiguous objects. For example, in Fig. 16 row 1 the subject's face is segmented (without ears or hair), while they are going through a variety of poses, rotate around, occlude the target region, and move both closer to and farther from the camera.

Our method successfully segments the target across multiple scales and poses, resulting in a smooth, temporally coherent, and accurate segmentation. Rows 4 and 5 depict scenes with similar challenges - a multitude of objects present, that have a similar appearance, frequently occlude each other and move, both around the scene and with their individual body parts. In both cases, our method successfully segments all of the targets, without confusing them, "bleeding" the mask into neighboring objects, or merging multiple targets into one.

Moreover, in the last picture of row 5, Fig. 16 it can be observed that the flower the person marked with the blue mask is holding was correctly not segmented, since in the

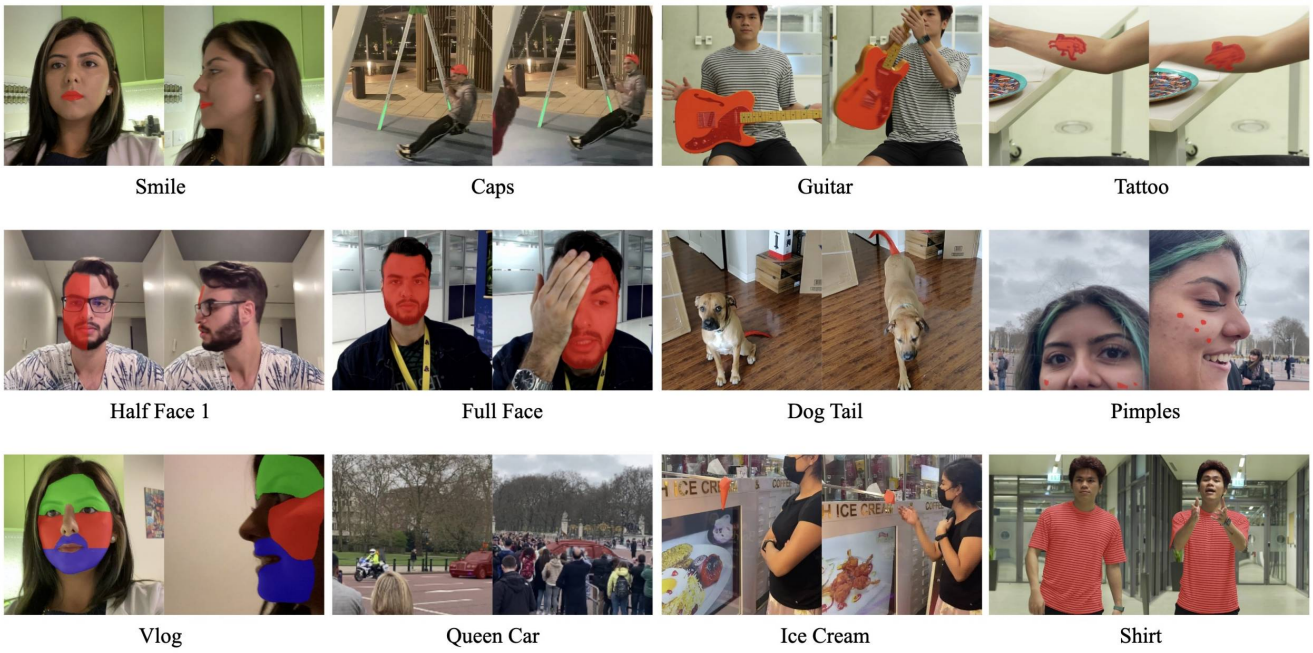

Figure 9. **PUMaVOS** dataset overview

provided annotations, it is not included, as not being a part of the target object. This illustrates that **XMem++** can work correctly with a large number of targets in the scene while preserving a high level of detail about their appearance.

Comparisons. We provide additional comparisons between our method and the current SOTA interactive segmentation model XMem [4]. XMem is a resource-efficient and fast memory-based segmentation method introduced in 2022 by Ho Kei Cheng and Alexander G. Schwing. Additional comparison results are provided in Fig. 11 and Fig. 10. For each video 6 frames were selected for annotation by uniformly sampling the video (refer to Eq. 1), starting from frame 0. Row 1 shows the same video as in "Full Face" sequence from **PUMaVOS**, but now with only half of the face being segmented, thus providing the same challenges as discussed earlier, but with even more difficult segmentation. Row 2 is a "guitar" example from **PUMaVOS**, where only the frontal part of the guitar's body is the target. In both cases we see **XMem++** resulting in a noticeably better segmentation, in particular with the "Half face" sequence, where it manages to produce the correct segmentation mask throughout the extreme variations in pose, expression, and scale, while XMem often "bleeds" the mask into neighboring regions, sticking more to the visual cues of the object. The results for the "Guitar" sequence demonstrate a similar outcome - **XMem++** correctly segmenting the front of the guitar, but not the sides, while XMem segments the whole object, again overfitting to visual cues instead of correct segmentation boundaries.

Figure 10. Illustration of smooth interpolation between different object's appearance that the permanent memory module in **XMem++** provides. The green frames are the ground truth annotations given to the model. It is noticeable that **XMem++** (row b) smoothly interpolates the mask across the change in the face's orientation, but the original XMem (row a) only fixes its predictions after processing the second ground truth annotation, resulting in a sharp "jump" in visual quality.

Limitations. Some examples are challenging even for our method. For example, while some motion blur is fine, with extreme motion blur it can't, which is a common challenge for all existing methods. Furthermore, memory-based models generally struggle when provided "negative masks" - empty annotations where the target object is not present, but the model has a false-positive segmentation prediction, illustrated in Fig. 13

Our frame selection algorithm does not work well when there is too much dynamics in the scene, with a lot of objects moving chaotically at the same time. Equivalently,

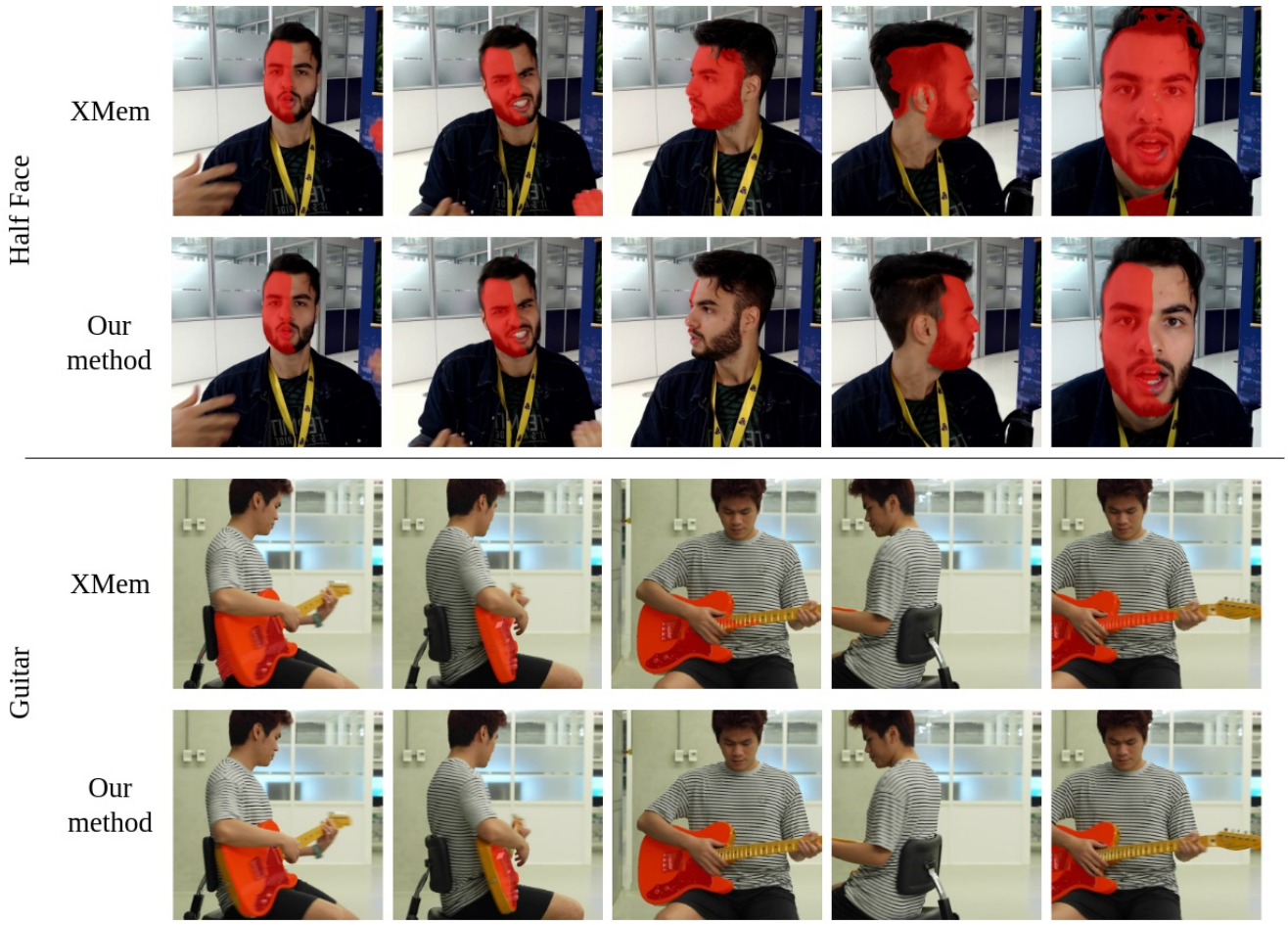

Figure 11. Comparison models

if there is too little movement, no clear scene boundaries, or very little variation in the target object's appearance. In both of these cases, the importance of selecting the right candidates for annotations is significantly reduced, as most of the frames result in a similar accuracy improvement. In this case, our algorithm wouldn't necessarily perform better than randomly/uniformly selected frames. To prove this, we evaluate our frame selection algorithm with **XMem++**, and a uniform baseline, calculated by the formula in the Eq. 1 on LVOS dataset [14], in which the videos typically have one of the aforementioned traits, and we show that the performance of the uniform baseline and our algorithm is very similar (Fig. 12).

$$F_A = \lfloor linspace(0, N-1, k) \rfloor \tag{1}$$

Given a video with N total frames, we select k candidates for both uniform and our frame annotation candidate selection algorithm. We then run inference on LVOS validation set with 49 videos, described in the Results section in the main paper, and illustrate the distribution of F-score ($\mathcal{F}$) and Intersection-over-Union ($\mathcal{J}$) metrics in Fig. 12.

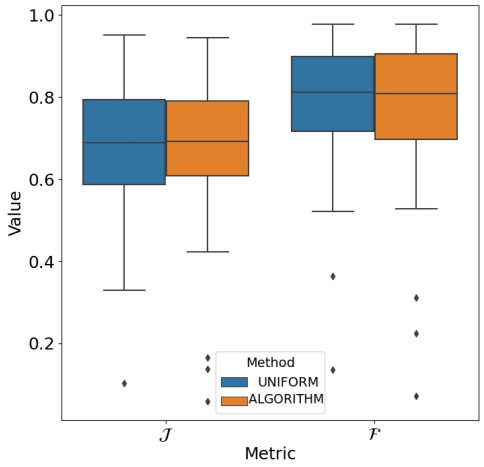

Figure 12. Comparison of segmentation quality with frames chosen by uniform baseline and our candidate selection algorithm on LVOS validation dataset.

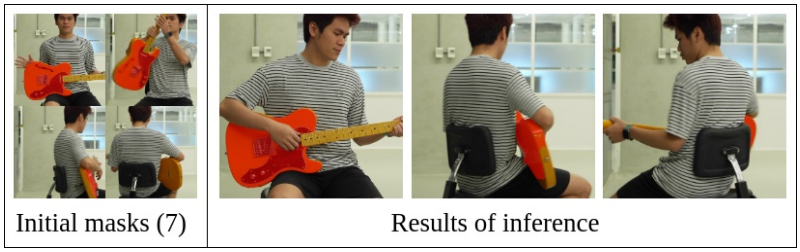

Figure 13. Limitation of **XMem++**: Partial failure (the back of the guitar is segmented in some frames)

D. Additional Evaluation

We analyze the performance increase of XMem and **XMem++** with the number of annotations available, by providing both qualitative (Fig. 15) and quantitative results (Fig. 14). In Rows 1-3 of the "Caps" sequence in Fig. 15, we see that providing just 5 frames is enough to completely resolve the ambiguity problem when segmenting one of the two identical caps in the frame. Providing 10 annotated frames further improves segmentation quality in challenging scenes, such as in Columns 4-5, where there is a lot of motion blur on the target object, as well as lighting variation. **XMem++** demonstrates similar results for "Queen car" sequence where just 5 provided annotated frames drastically improve the quality in the scenes with extreme occlusion, where the car is hardly visible behind lots of people (Columns 2-4).

We furthermore evaluate our method's quality scaling performance on LVOS validation dataset, from 1 to 10 annotated frames provided, illustrated in Fig. 14. As expected, given only 1 frame both models yield equivalent results, however **XMem++** (drawn with orange line) demonstrates significantly higher scalability potential and efficiency starting at 2 annotated frames and keeps the advantage throughout the whole comparison, up to +13% difference at 10 frames (0.63 XMem vs 0.76 **XMem++**)

In-Memory Augmentations. We address a possible use-case where annotations could be very sparsely available or too expensive to produce, by exploring in-memory augmentations for provided annotated frames. Through rigorous testing, we select the **11** best augmentations that, when combined, lead to the highest possible segmentation quality improvement for **XMem++**, shown in Fig. 5. When processing and adding provided frames and their annotations to the permanent memory, each of the augmentations is applied to every frame (and their corresponding mask, where necessary) and stored in the permanent memory as well. This can be a practical way of increasing the segmentation accuracy without any extra work done by the end-user, at the cost of higher memory usage and potentially slower inference speed.

Utility of Permanent Memory. We further demonstrate the capabilities of our introduced permanent memory module by disabling updates to the temporary memory in

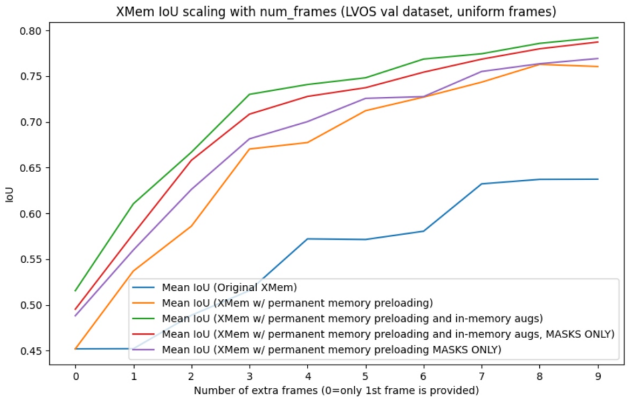

Figure 14. Demonstration of superior segmentation quality scaling of **XMem++** compared to original XMem with the number of annotated frames available. Blue line is the original XMem model, orange is **XMem++** as used in all other evaluations and comparison. Purple line shows a modification of **XMem++** with disabled temporary memory, and green and red indicate the usage of in-memory augmentations, with and without temporary memory correspondingly.

Increase brightness	0.721	+0.009
Decrease brightness	0.725	+0.013
Grayscale	0.707	-0.005
Reduce bits to 3	0.717	+0.005
Make sharp	0.718	+0.006
Gaussian blur	0.731	+0.019
Rotate right 45 deg†	0.723	+0.011
Translate right +100 px	0.675	-0.037
Zoom out 0.5×	0.715	+0.003
Zoom in 1.5×	0.727	+0.015
Shear right by 20†	0.730	+0.018
Crop mask region	0.704	-0.008

Table 5. In-memory augmentations in their individual effect on the overall segmentation quality on LVOS dataset. Only transformations named in **bold** were considered for experiments. For transformation with †the equivalent symmetrical transform was used as well. A total of 11 augmentations were used for the experiments in Fig. 14.

XMem++, effectively keeping it empty and frozen throughout the inference. We observe that for LVOS dataset (Fig. 14) this results in an *increase* in the overall segmentation quality, both with (red) and without (purple) using in-memory augmentations. This shows that for certain types of videos, especially when predicted masks are prone to have errors, the best strategy is to not use them at all, and very few high-quality references in the memory result in higher segmentation quality than dozens, potentially hundreds, but containing errors. Temporary memory plays has an important role in **XMem++** architecture, allowing it to adapt better to changes in the target appearance, but through our experiments we show that for some videos it can be safely disabled (for example, if the target object's appearance stays relatively consistent throughout the video), leading to higher inference speed and lower memory footprint.

Figure 15. Visual comparison of the segmentation results by **XMem++** with 1, 5, and 10 uniformly-sampled annotation candidates provided.

Figure 16. Results of XMem++

Algorithm 1 select-next-candidates($\mathbf{K}$, $\mathbf{M}$, k, $\mathbf{PC}$, $\alpha = 0.5$, $\beta = 9$)

Here is the pseudo-code for the annotation candidate selection algorithm. The $\odot$ is a pointwise multiplication operation. Symbol [ ] denotes an empty list, and symbols $\mathbf{S}$ and $\neg\mathbf{S}$ are used to denote similarity and dissimilarity correspondingly (the negation symbol $\neg$ is used as a visual cue)

Require: $\mathbf{K}$: list of "key" feature maps for all frames of the video
Require: $\mathbf{M}$: list of masks for each frame (predicted or user-provided)
Require: k: number of candidate frames to select
Require: $\mathbf{PC}$: list of previously chosen candidate indices (default is [0])
Require: α: weight of mask regions (default is 0.5), $\alpha \in [0..1]$
Require: β: minimum number of pixels for a valid mask, to explicitly filter out frames without the target object or where it is too small (default is 9px)
Ensure: $\mathbf{PC}$ not empty, $0 \leq \alpha \leq 1.0$, $k > 0$

function SELECT-NEXT-CANDIDATES($\mathbf{K}$, $\mathbf{M}$, k, $\mathbf{PC}$, α, β)
  $\mathbb{K} \leftarrow [\,]$, $N \leftarrow |\mathbf{K}|$ ▷ Composite keys, Number of frames
  for i in $[0, N-1]$ **do**
    $\hat{k} \leftarrow \mathbf{K}[i] \odot \mathbf{M}[i] \cdot \alpha + \mathbf{K}[i] \cdot (1-\alpha)$ ▷ $\hat{k}$ is a "composite" key
    ▷ Equivalent to alpha-blending operation
    $\mathbb{K}$.add_to_end($\hat{k}$)
  end for
  $\mathbf{CC} \leftarrow \mathbf{PC}$ ▷ Chosen candidates, initialize with previous candidates
  $\mathbf{C\hat{K}} \leftarrow [\mathbb{K}[i] \mid i \in \mathbf{PC}]$ ▷ Chosen candidates composite keys
  for i in [0, k] **do**
    $\neg\mathbf{S} \leftarrow [\,]$ ▷ Dissimilarities between candidates and other frames
    for j in [0, N] **do**
      if $|\mathbf{M}[i] > 0| < \beta$ **then** ▷ Mask empty or too small, ignore
        $\neg S_{min} \leftarrow 0$ ▷ Minimum dissimilarity of frame i to all in $\mathbf{CC}$
      else
        $\neg\mathbf{S_K} \leftarrow [\,]$ ▷ Dissimilarities of $i \rightarrow j, \forall j \in \mathbf{CC}$
        for j in $\mathbf{CC}$ **do**
          $S_{j \rightarrow i} \leftarrow$ similarity($\mathbf{C\hat{K}}[j]$, $\mathbb{K}[i]$)
          $S_{i \rightarrow j} \leftarrow$ similarity($\mathbb{K}[i]$, $\mathbf{C\hat{K}}[j]$)
          $\neg\mathbf{S}_{cycle} \leftarrow (S_{j \rightarrow i} - S_{i \rightarrow j})$ ▷ Pixel-wise cycle dissimilarity
          $\neg S_{cycle} \leftarrow \frac{\sum \max(0, \neg\mathbf{S}_{cycle})}{|\neg\mathbf{S}_{cycle}|}$ ▷ Only non-negative mappings
          $\neg\mathbf{S_K}$.add_to_end($\neg S_{cycle}$)
        end for
        $\neg S_{min} \leftarrow \min(\neg\mathbf{S_K})$
      end if
      $\neg\mathbf{S}$ .add_to_end($\neg S_{min}$)
    end for
    $c \leftarrow$ argmax($\neg\mathbf{S}$) ▷ New selected candidate
    $\mathbf{CC}$.add_to_end(c)
    $\mathbf{C\hat{K}}$.add_to_end($\mathbb{K}[c]$)
  end for
  return $[\mathbf{CC}[i] \mid i \geq |\mathbf{PC}|]$ ▷ Return new candidates, from index $|\mathbf{PC}|$
end function