A Datasets

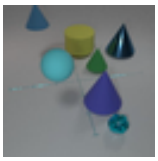
	Sample Frame	Textured FG	Textured BG	Static Objects	Camera Motion	Natural
CATER (Girdhar & Ramanan, 2020)		No	No	**Yes**	**Yes**	No
CATERTex		**Yes**	**Yes**	**Yes**	No	No
MOVi-Solid		No	**Yes**	No	No	No
MOVi-Tex		**Yes**	**Yes**	No	No	No
MOVi-D (Greff et al., 2022)		**Yes**	**Yes**	**Yes**	No	No
MOVi-E (Greff et al., 2022)		**Yes**	**Yes**	**Yes**	**Yes**	No
YT-Traffic		**Yes**	**Yes**	**Yes**	No	**Yes**
YT-Aquarium		**Yes**	**Yes**	No	No	**Yes**

Table 1: Characteristics of the video datasets used in our evaluation.

Dataset Descriptions. We generated MOVi-Solid dataset using Kubric (Greff et al., 2022) which introduces textured background and more complex shapes to the MOVi dataset Kipf et al. (2021). For this, we used 100 HDR backgrounds and Kubric's KuBasic shapes. We also generate two test sets: one which has the same number of objects as the training set (3-10 objects) and other which has higher out-of-distribution number of objects (11-12 objects). Similarly to MOVi-Solid, for MOVi-Tex, we use Kubric to generate the dataset. In MOVi-Tex, we modify our MOVi-Solid dataset to have complex textures applied to the objects taken from the Describable Textures dataset Cimpoi et al. (2014). Describable Textures dataset provides several categories of textures. We used 240 textures each for generating the training set. For this, we used 120 textures that are from the banded category and 120 textures that are from the chequered category. We also generated two out-of-distribution test sets: one which has the same number of objects as the training set (3-10 objects) but never-before-seen textures and the other which as the same textures but higher number of objects (11-12 objects). For the out-of-distribution test set with never-before-seen textures, 120 textures were taken from the grid category and 120 were taken from the grooved category of the Describable Textures dataset. For CATER, MOVi-D and MOVi-E, we used the standard training and test splits provided in the respective releases of CATER (with masks) and MOVi datasets. We generated CATERTex by integrating the codes for CATER and CLEVRTex (Girdhar & Ramanan, 2020; Karazija et al., 2021). We generated two sets: the training and the IID test sets which contain 3-8 objects and an out-of-distribution test set which contains 9-10 objects. We collected the Youtube datasets by downloading a 6-hour long video streams, center-cropping and resizing them. The Youtube-Traffic dataset was generated from a video feed collected by a traffic camera that watches a fixed intersection. The camera does not move during the video. The lighting conditions change slightly over time and this has a slight effect on the background and the overall scene from one clip to another. The vehicles appear to show diversity in size, color, and type. The Youtube-Aquarium dataset was generated from a video collected by a static camera mounted inside a fish exhibit. The background, lighting conditions, and other characteristics remain stationary throughout the video. The size, color, and types of fish show some diversity. As these were collected from Youtube and did not have ground truth masks, we manually annotated 50 videos of the test set. We will release the training and the test splits of all the datasets.

B STEVE: Architecture Details

Recurrent Slot Encoder. The design of our slot-based recurrent encoder $f_\phi^{\text{slot-rnn}}$ is inspired by Kipf et al. (2021). At the start of an episode, we provide initial slots $\mathbf{s}_0$ by randomly sampling from a Gaussian distribution with learned mean and variance.

$$\mathbf{s}_{0,n} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}), \qquad \forall n \in \{1, \ldots, N\}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are learned parameters. For each input frame $\mathbf{x}_t$, we compute an encoding in the form of a feature map using a backbone CNN. The architecture of this CNN is described in Table 2. The resulting feature map has size $H_{\text{enc}} \times W_{\text{enc}} \times D_{\text{enc}}$. It is then flattened on spatial dimensions to obtain

Layer	Kernel Size	Stride	Padding	Channels	Activation
Conv	5×5	1 (2)	2	32 (64)	ReLU
Conv	5×5	1	2	32 (64)	ReLU
Conv	5×5	1	2	32 (64)	ReLU
Conv	5×5	1	2	32 (64)	None

Table 2: Configuration of the CNN encoder used in our model. The values in parentheses are applicable for image size 128.

a set of $H_{\text{enc}} \cdot W_{\text{enc}}$ features. A positional embedding $\mathbf{p}$ is added to these followed by a 2-layer MLP. With these, we obtain a set of input features $\mathbf{e}_t = \{\mathbf{e}_{t,1}, \mathbf{e}_{t,2}, \ldots, \mathbf{e}_{t,HW}\}$. The positional embedding is computed by applying a linear map to the 4-dimensional spatial position as introduced by Locatello et al. (2020) and Kipf et al. (2021). Next, the slots of the previous time-step $\mathbf{s}_{t-1}$ perform attention on the features $\mathbf{e}_t$ and use the attention result to update the slots to $\mathbf{s}_t$. For this, the slots in $\mathbf{s}_{t-1}$ first compute the attention weights $\mathcal{A}_t = \{\mathcal{A}_{t,1}, \ldots, \mathcal{A}_{t,N}\}$ over the input features. Using the attention weights, the attention result $\mathbf{r}_{t,n}$ is computed for each slot via an attention-weighted sum of input

features.

$$\mathcal{A}_t = \texttt{softmax}\left(\frac{q(\mathbf{s}_{t-1}) \cdot k(\mathbf{e}_t)^T}{\sqrt{D}}, \text{dim} = \texttt{slots}\right) \qquad \mathbf{r}_{t,n} = \frac{\sum_{i=1}^{HW} \mathcal{A}_{t,n,i} \cdot v(\mathbf{e}_{t,i})}{\sum_{j=1}^{HW} \mathcal{A}_{t,n,j}}.$$

where q, k and v are linear projections and D is the output size of the projections. The attention result $\mathbf{r}_{t,n}$ is then used to update the respective slot: $\tilde{\mathbf{s}}_{t,n} \leftarrow f_{\text{RNN}}(\mathbf{r}_{t,n}, \mathbf{s}_{t-1,n})$ where the update function f_{RNN} is implemented as a GRU (Cho et al., 2014). For additional expressiveness, a 2-layer MLP (with residual connection) and LayerNorm is applied on the output of the GRU. In practice, these attention and update steps can be executed multiple times per frame. We use 2 iterations per frame in our experiments. Lastly, the slots are made to interact using an interaction function $\mathbf{s}_t \leftarrow f_\phi^{\text{interact}}(\tilde{\mathbf{s}}_t)$ implemented using a 1-layer transformer with 4 heads. Following Kipf et al. (2021), we use the slots $\tilde{\mathbf{s}}_t$ before the interaction step for reconstruction. For fairness of comparison between our model STEVE and SAVi, we use the same implementation of this backbone recurrent slot encoder in both models.

Discrete VAE. The implementation of our discrete VAE is based on that proposed by (Singh et al., 2022). It consists of an encoder network f_ϕ^{dVAE} and a decoder network g_θ^{dVAE}.

The encoder network essentially divides the whole image $\mathbf{x}_t$ into patches of size 4×4 and encodes each patch as one discrete token belonging to a vocabulary $\mathcal{V}$. We take the size of the vocabulary to be 4096 in our experiments. This is achieved in parallel for all patches applying a convolutional neural network on the image of size $H \times W \times 3$ as follows.

Layer	Kernel Size	Stride	Padding	Channels	Activation
Conv	4×4	4	0	64	ReLU
Conv	1×1	1	0	64	ReLU
Conv	1×1	1	0	64	ReLU
Conv	1×1	1	0	64	ReLU
Conv	1×1	1	0	64	ReLU
Conv	1×1	1	0	64	ReLU
Conv	1×1	1	0	64	ReLU
Conv	1×1	1	0	$\lvert\mathcal{V}\rvert$	None

Table 3: Configuration of discrete VAE encoder used in our model.

This results in a feature map of size $(H/4) \times (W/4) \times |\mathcal{V}|$. Each cell $i \in 1, \ldots, L$ (where $L = HW/16$) of this feature map can be seen as a vector $\mathbf{o}_{t,i}^{\text{dVAE}}$ containing log-probabilities over $|\mathcal{V}|$ vocabulary tokens. For each cell i, a discrete token is sampled by constructing a categorical distribution using these log-probabilities.

$$\mathbf{z}_{t,i} \sim \text{Categorical}(\mathbf{o}_{t,i}^{\text{dVAE}}).$$

In the discrete VAE decoder, the input is a discrete feature map of size $(H/4) \times (W/4) \times |\mathcal{V}|$ whose each cell is a one-hot vector $\mathbf{z}_i$ and the decoder tries to reconstruct the original image from this discrete feature map input. This mapping is implemented using a convolutional neural network as described below, finally outputting an image of size $H \times W \times 3$. Finally, the whole network is trained by minimizing a image reconstruction loss implemented using squared-error.

$$\mathcal{L}_{\text{dVAE}} = \sum_{t=1}^{T} \|\hat{\mathbf{x}}_t - \mathbf{x}_t\|_2^2, \qquad \mathbf{z}_{t,1}, \ldots \mathbf{z}_{t,L} = f_\phi^{\text{dVAE}}(\mathbf{x}_t), \qquad \hat{\mathbf{x}}_t = g_\theta^{\text{dVAE}}(\mathbf{z}_{t,1}, \ldots \mathbf{z}_{t,L}).$$

As there is a discrete sampling step required, we use the Gumbel-Softmax relaxation during training while we use hard sampling for generating discrete targets for the transformer, following Singh et al. (2022). The temperature parameter for the Gumbel-Softmax is decayed from 1.0 to 0.1 in the first 30K iterations of training as prescribed by Singh et al. (2022).

Transformer Decoder. We use the standard design of transformer as introduced by Vaswani et al. (2017) and adopted by Singh et al. (2022). Like SLATE, we use learned positional embeddings for each position $i \in \{1, \ldots, L\}$ in the sequence.

Layer	Kernel Size	Stride	Padding	Channels	Activation
Conv	1×1	1	0	64	ReLU
Conv	3×3	1	1	64	ReLU
Conv	1×1	1	0	64	ReLU
Conv	1×1	1	0	64	ReLU
Conv	1×1	1	0	256	ReLU
PixelShuffle (`upscale_factor=2`)	-	-	-	-	-
Conv	3×3	1	1	64	ReLU
Conv	1×1	1	0	64	ReLU
Conv	1×1	1	0	64	ReLU
Conv	1×1	1	0	256	ReLU
PixelShuffle (`upscale_factor=2`)	-	-	-	-	-
Conv	1×1	1	0	3	None

Table 4: Configuration of discrete VAE decoder used in our model.

During training, the transformer decoder takes the slots and the dVAE tokens as input and learns to predict the next token. We learn to do this next-token prediction for all tokens in parallel by applying causal masking in the style of GPT. In further detail, the dVAE tokens given as input to the transformer are obtained via argmax on the logits output by the dVAE encoder. That is, we take the dVAE logits $\mathbf{o}_{t,i}^{\text{dVAE}}$ and first perform argmax. For this argmax index, we retrieve a learned embedding from a codebook. To these retrieved embeddings, we add learned positional encodings and the resulting embeddings are then given as input to the transformer decoder. Similarly to SLATE, we do not let gradient be propagated across the argmax operation. The prediction targets are the argmax indices of the next dVAE token which we learn to predict using a cross-entropy loss.

Difference between SLATE and STEVE. The key difference between STEVE and SLATE is that STEVE temporally extends SLATE by adopting a recurrent slot encoder. In doing this, STEVE retains the same decoder architecture as that of SLATE.

Training. We trained all models upto a maximum of 200K training iterations (except for CATERTex for which we trained all models to 400K iterations). These amounted to training time of approximately 2 days. Similarly to SLATE, we applied a learning rate warmup from 0 to the peak learning rate in the first 30K training iterations. After that we decay the learning rate exponentially with a half-life of 250K training iterations.

The loss function is $\mathcal{L}_{\text{STEVE}} = \mathcal{L}_{\text{CE}} + \mathcal{L}_{\text{dVAE}}$ as described in Section 3. The gradients from the $\mathcal{L}_{\text{dVAE}}$ loss are only used to train the dVAE encoder and the dVAE decoder. The gradients from the $\mathcal{L}_{\text{CE}}$ cross-entropy loss are only used for training the slot encoder and the transformer decoder. This is due to the fact that we do not let the gradients be propagated accross the argmax operation used to obtain the discrete representation of the input frame. All 3 components i.e. the dVAE, the slot encoder, and the transformer are trained simultaneously in a single training run.

B.1 Computational Requirements

In Table 6, we empirically compare the computational requirements of STEVE and SAVi. We assess by how much the computational requirement change in going from the mixture-based to the transformer-based approach. We find that in videos with image size 64×64, STEVE requires similar resources as SAVi and about twice for image size 128×128. Given the quadratic memory complexity of transformers, the larger memory demand of STEVE is not surprising. However, one should also consider that the mixture-based baselines fail almost completely on datasets like CATERTex, MOVi-Tex, MOVi-D, and MOVi-E, unlike ours. Another feature of our method is that the memory demand of transformer decoder is approximately constant in the number of slots but linear for mixture-based decoders.

GPU Resources. We performed our experiments on Quadro RTX 8000 GPUs. We used 2 GPUs per each training seed, each of which took approximately 2-3 days to complete training.

		Image Size	
Module	Hyperparameter	64	128
General	Batch Size	24	24
	Episode Length	3	3
	Training Steps	200000	200000
Encoder	Corrector Iterations	2	2
	Slot Size	192	192
	MLP hidden size	192	192
	# Predictor Blocks	1	1
	# Predictor Heads	4	4
	Learning Rate	0.0001	0.0001
Transformer Decoder	# Decoder Blocks	4	8
	# Decoder Heads	4	4
	Hidden Size	192	192
	Dropout	0.1	0.1
	Learning Rate	0.0003	0.0003
DVAE	Learning Rate	0.0003	0.0003
	Patch Size	4×4 pixels	4×4 pixels
	Vocabulary Size	4096	4096
	Temperature Start	1.0	1.0
	Temperature End	0.1	0.1
	Temperature Decay Steps	30000	30000

Table 5: Hyperparameters of STEVE used in our experiments.

	Memory (GB)		Seconds per iteration	
Image Size	SAVi	STEVE	SAVi	STEVE
64	9.8	10.7	0.24	0.20
128	29.8	57.8	0.53	0.74

Table 6: Computational requirements of STEVE and SAVi. The configurations of STEVE are as described in Tables 5 and 2. The configurations of SAVi are matched with STEVE except for the its decoder which is a 4-layer spatial broadcast network (as described in the original SAVi paper). Here, all values are computed using same the batch size and episode length (taken to be 24 and 3, respectively) for an informative comparison.

Big-O Analysis. The memory complexity of the transformer decoder is dependent on the cost of self-attention between all the dVAE tokens and the cross-attention between the dVAE tokens and the slots. Let the number of dVAE tokens be L and the number of slots be N. Then the number of dVAE tokens is $L = HW/K^2$ where H and W are image height and width, respectively; and K is the dVAE patch size. To perform self-attention among all L tokens requires a memory complexity of $O(L^2)$. Furthermore, the L tokens also perform cross-attention on the N slots which leads to an attention matrix with memory cost $O(LN)$. Adding these attention costs leads to a total cost of $\approx O(L^2) + O(LN)$. But since it is common to have just a few slots N (e.g. $N = 11$ slots for CATER) and a much larger number of tokens (e.g. $L = 256$ tokens for CATER), the first term $O(L^2)$ is typically the dominant one in comparison to the other term $O(LN)$. Replacing $L = HW/K^2$ gives an approximate cost of $O((HW)^2/K^4)$. This can be contrasted with the memory cost of mixture decoder. Each object component can be decoded using a CNN with a cost of $O(HW)$ and there are N such components. Thus, the total cost is approximately $O(HWN)$. These suggest that as we continue of scale up the image size H and W (keeping everything else same), the memory needs of the transformer decoder would significantly surpass that of the mixture decoder.

B.2 Mixture-based Decoding without Spatial Broadcast

In Section 5.3, we performed an ablation of SAVi where we replaced its original Spatial Broadcast decoder with a more flexible CNN implemented entirely using transposed convolutions. In Tables 7 and 8, we describe these two CNN architectures in detail. We note that the decoder used in

Layer	Kernel Size	Stride	Channels	Activation
Spatial Broadcast	-	-	`slot_size`	-
Positional Encodings	-	-	`slot_size`	-
ConvTranspose	5×5	2	64	ReLU
ConvTranspose	5×5	2	64	ReLU
ConvTranspose	5×5	2	64	ReLU
ConvTranspose	5×5	1 (2)	64	ReLU

Table 7: Configuration of the CNN of the Spatial Broadcast decoder used in SAVi (Kipf et al., 2021). The values in parentheses are applicable for image size 128.

Layer	Kernel Size	Stride	Channels	Activation
ConvTranspose	5×5	2	64	ReLU
ConvTranspose	5×5	2	64	ReLU
ConvTranspose	5×5	2	64	ReLU
ConvTranspose	5×5	2	64	ReLU
ConvTranspose	5×5	2	64	ReLU
ConvTranspose	5×5	2	64	ReLU
ConvTranspose	5×5	1 (2)	64	ReLU

Table 8: Configuration of the CNN decoder implemented only using transposed convolutions which is used in our ablation of SAVi in Section 5.3. The values in parentheses are applicable for image size 128.

the ablation of SAVi has more layers and is therefore more expressive. It applies 7 transposed convolution layers while the original decoder had 4. Furthermore, unlike the original decoder that uses a Spatial Broadcast in the first layer which is intended to lower its flexibility, our ablation of SAVi uses transposed convolutions in all the layers of the CNN decoder.

C Additional Results.

In this section, we provide additional evaluation results for our model.

	STEVE	SAVi	OP3
CATER	**0.911** $\pm$ 0.015	0.818 $\pm$ 0.025	0.682 $\pm$ 0.045
CATERTex	**0.755** $\pm$ 0.024	0.224 $\pm$ 0.051	0.299 $\pm$ 0.055
MOVi-Solid	0.848 $\pm$ 0.034	**0.875** $\pm$ 0.005	0.568 $\pm$ 0.042
MOVi-Tex	**0.741** $\pm$ 0.011	-3.085 $\pm$ 0.236	0.040 $\pm$ 0.036
MOVi-D	**0.589** $\pm$ 0.008	0.391 $\pm$ 0.12	0.316 $\pm$ 0.028
MOVi-E	**0.567** $\pm$ 0.032	0.436 $\pm$ 0.113	0.216 $\pm$ 0.017

Table 9: **Comparison of property prediction performance with the baselines.** Given a frozen slot, we predict the 2D position coordinates of the object using linear regression. For training the regressors, we match the slots with the ground truth position labels via Hungarian matching using IoU score between the true and the predicted masks. We report the R^2 score, similarly to Kabra et al. (2021). We note that our model outperforms the baselines significantly, except in MOVi-Solid where our performance is comparable to that of SAVi.

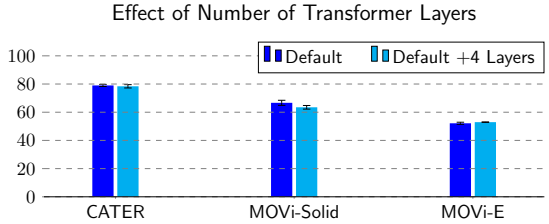


Figure 7: Effect of number of layers in the transformer decoder. The aim of this experiment is to test how susceptible is our performance to the flexibility of the transformer decoder. We compare our default transformer decoder configuration with a transformer decoder which has 4 additional layers in it. As this introduces more parameters to the decoder, it makes the decoder more flexible. We report the FG-ARI (in %) on videos of length 6. We find that the performances are comparable in both configurations. This shows that a careful tuning of decoder strength is not needed and this can be seen as a benefit of our model.

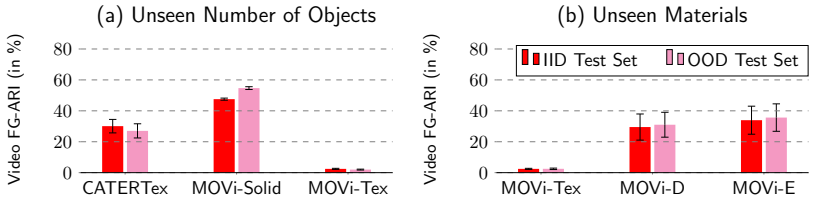


Figure 8: Out-of-distribution generalization to unseen number of objects and unseen materials in SAVi. We report the FG-ARI (in %) on videos of length 6.

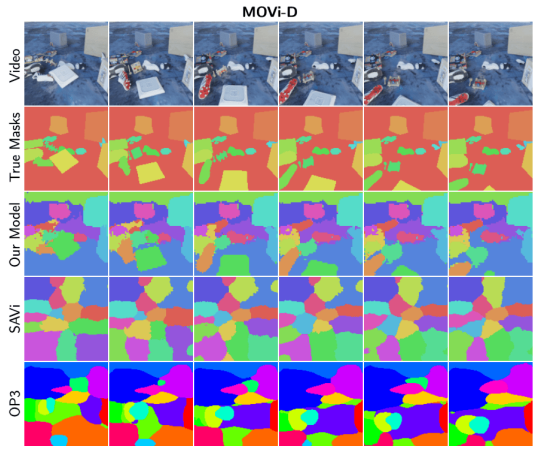


Figure 9: **Unsupervised Video Segmentation in MOVi-D.** We qualitatively compare video segmentation of STEVE with the baselines SAVi and OP3. The rows show the input video, the true segmentation, followed by the predicted segments of STEVE and the baselines.

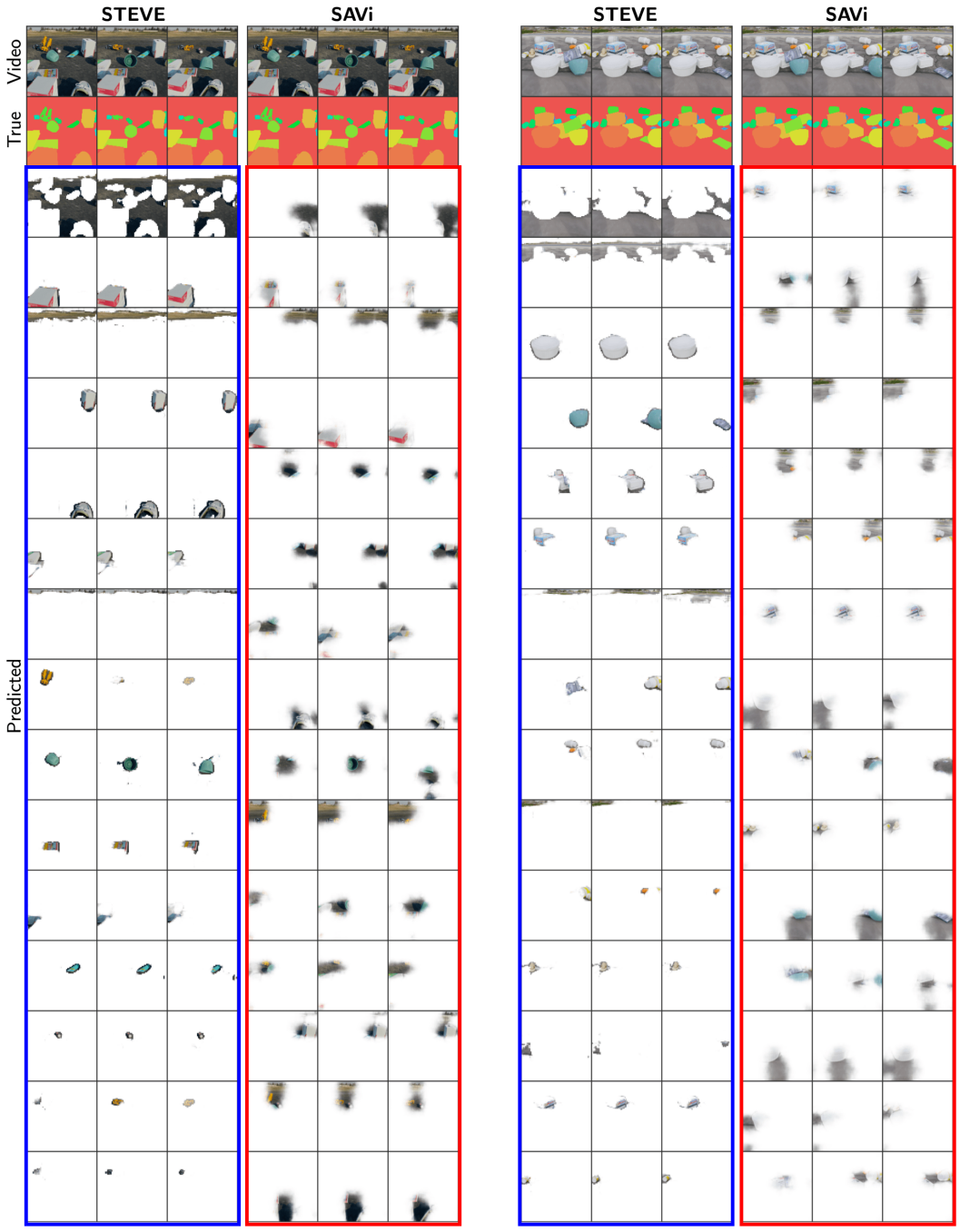


Figure 10: Additional Samples of Unsupervised Video Segmentation in MOVi-E.

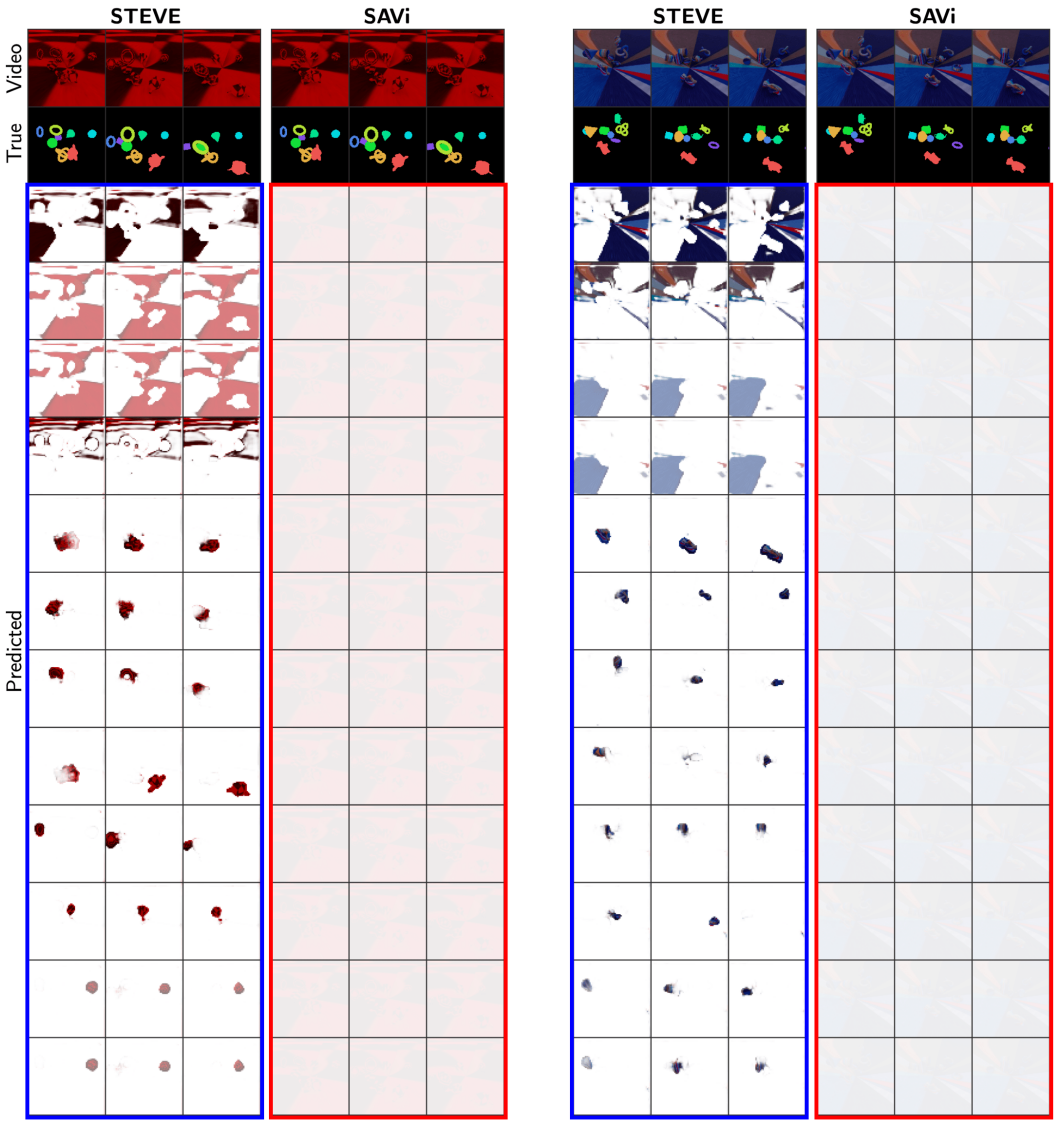


Figure 11: Additional Samples of Unsupervised Video Segmentation in MOVi-Tex.

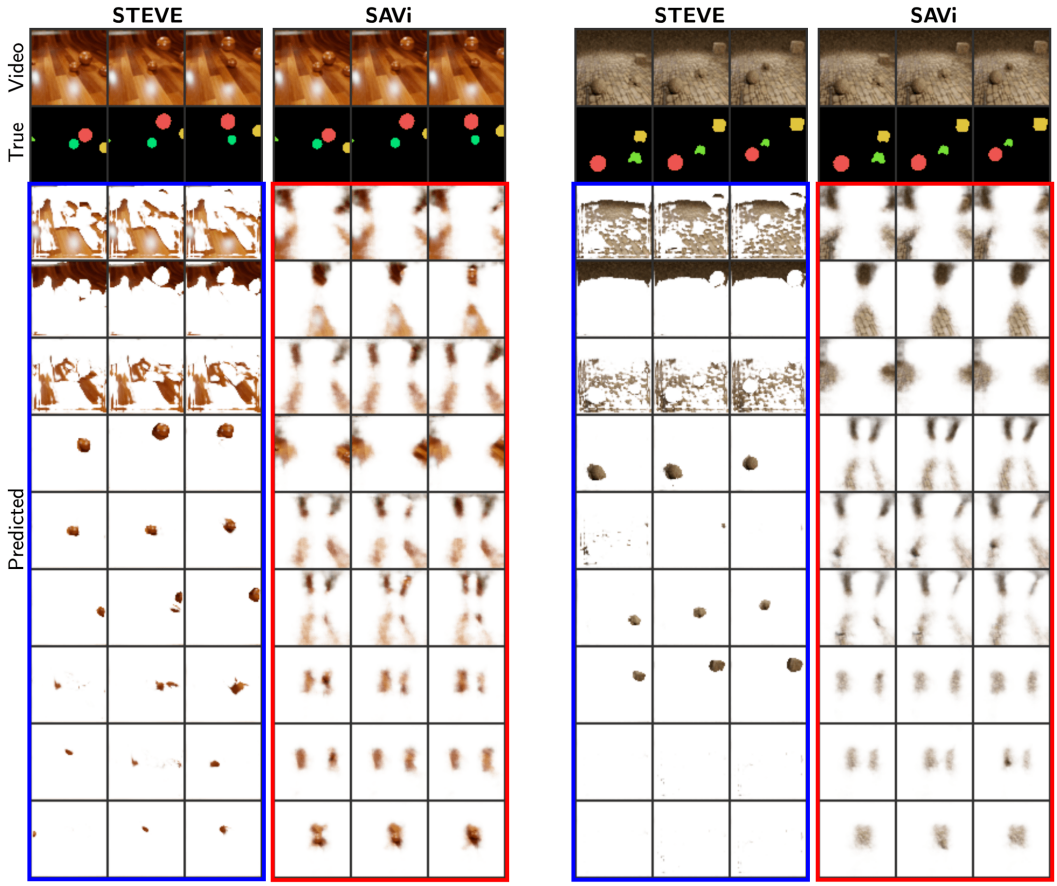


Figure 12: Additional Samples of Unsupervised Video Segmentation in CATERTex.

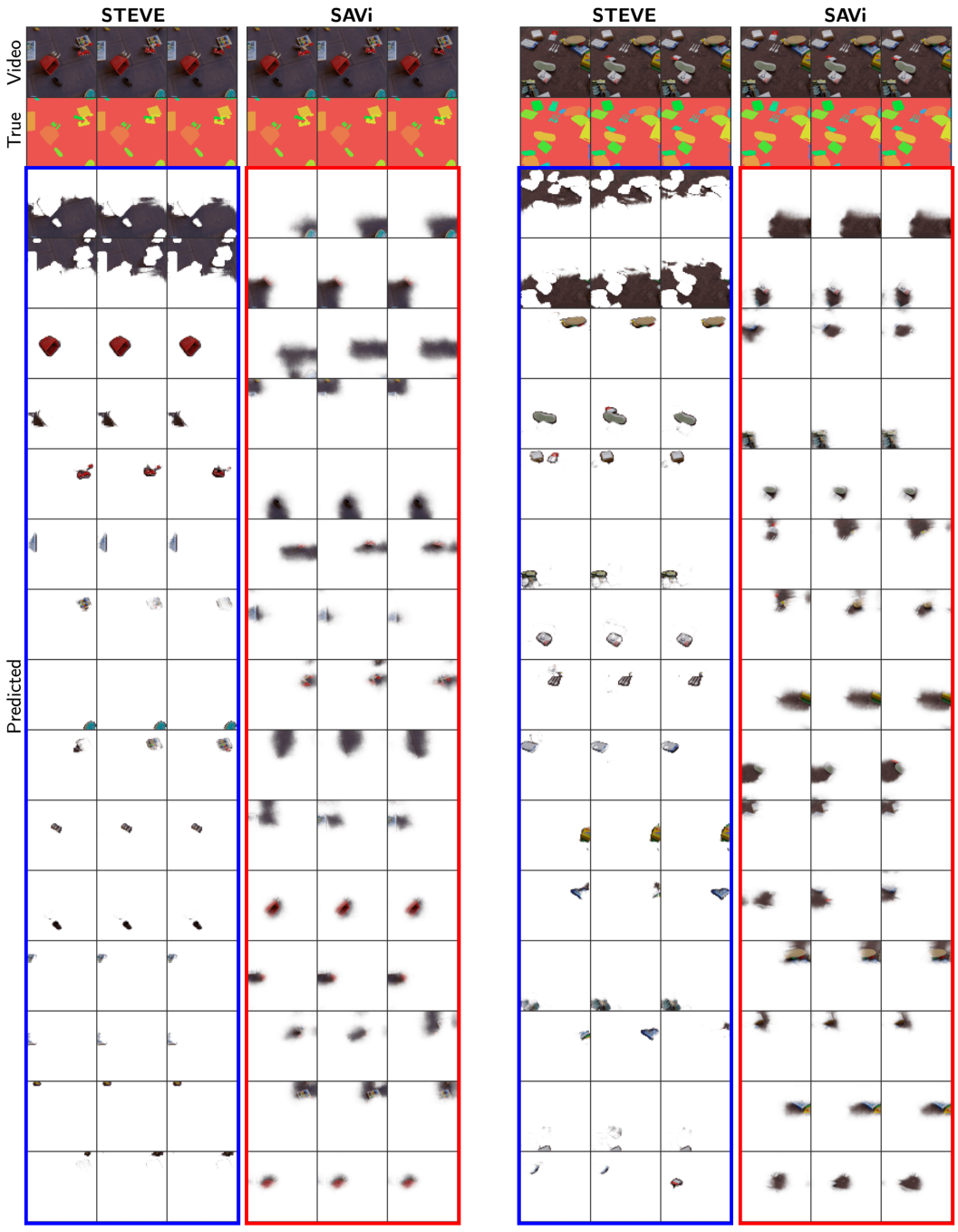


Figure 13: Additional Samples of Unsupervised Video Segmentation in MOVi-D.

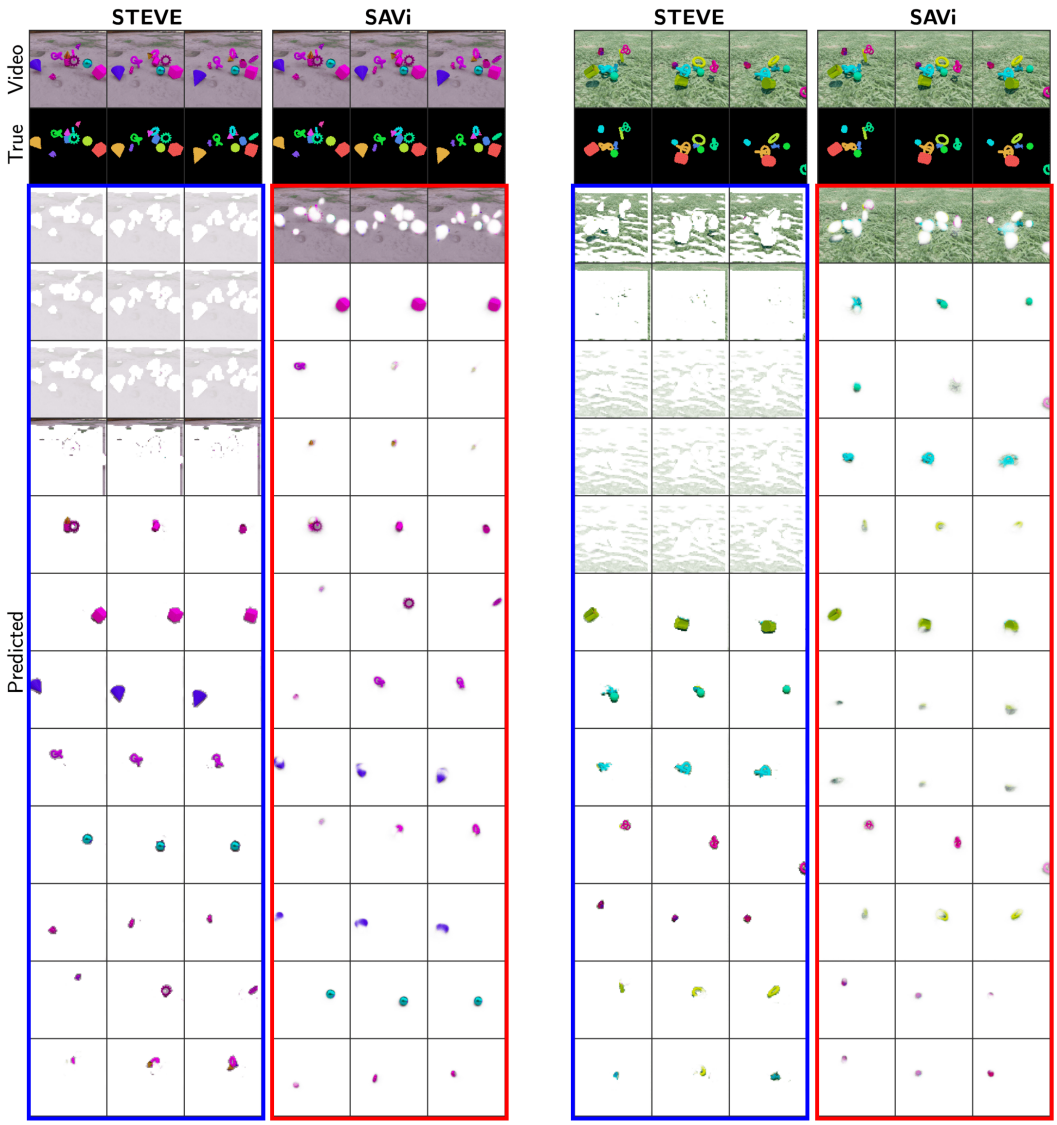


Figure 14: Additional Samples of Unsupervised Video Segmentation in MOVi-Solid.

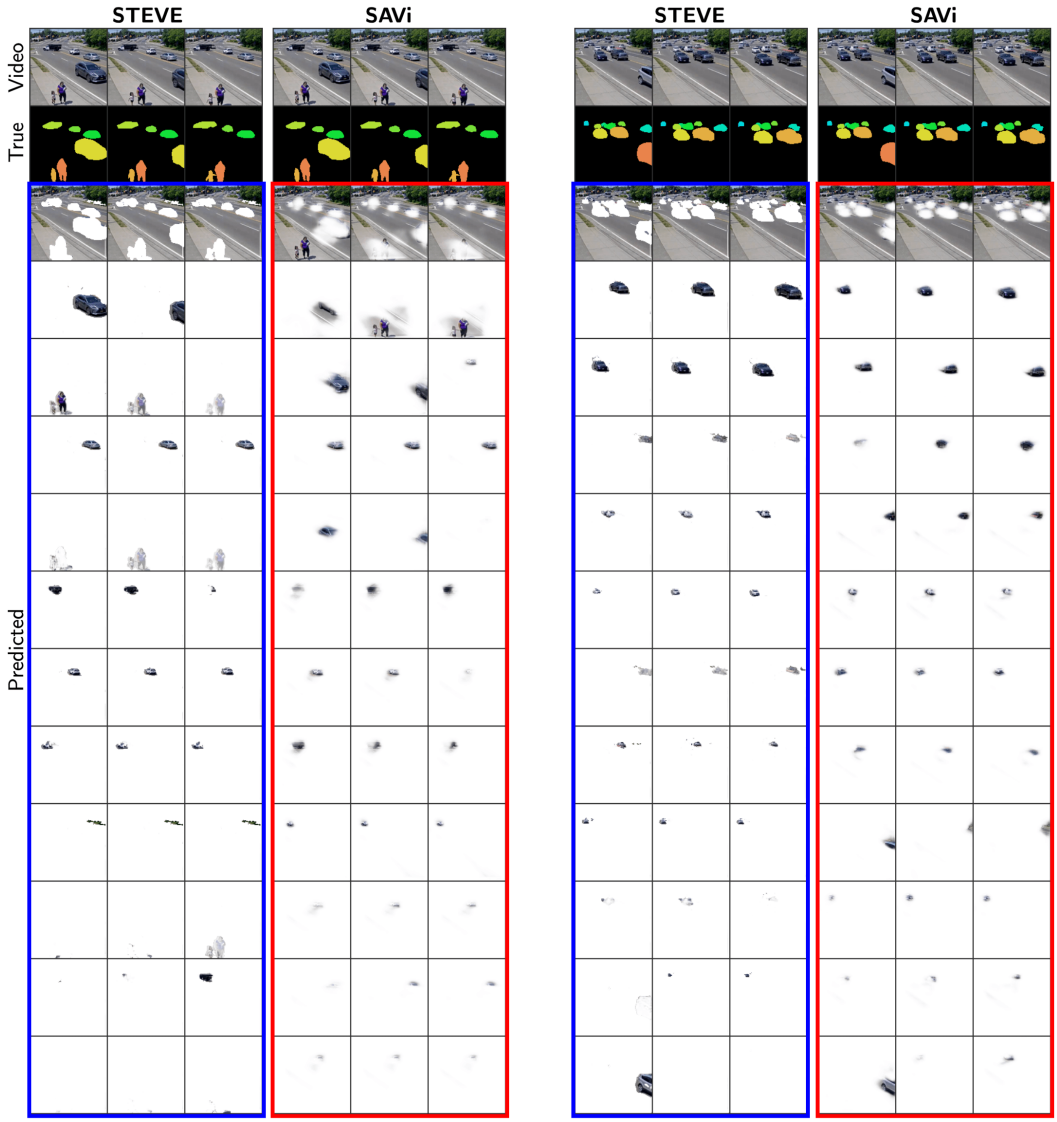


Figure 15: Additional Samples of Unsupervised Video Segmentation in Youtube Traffic.

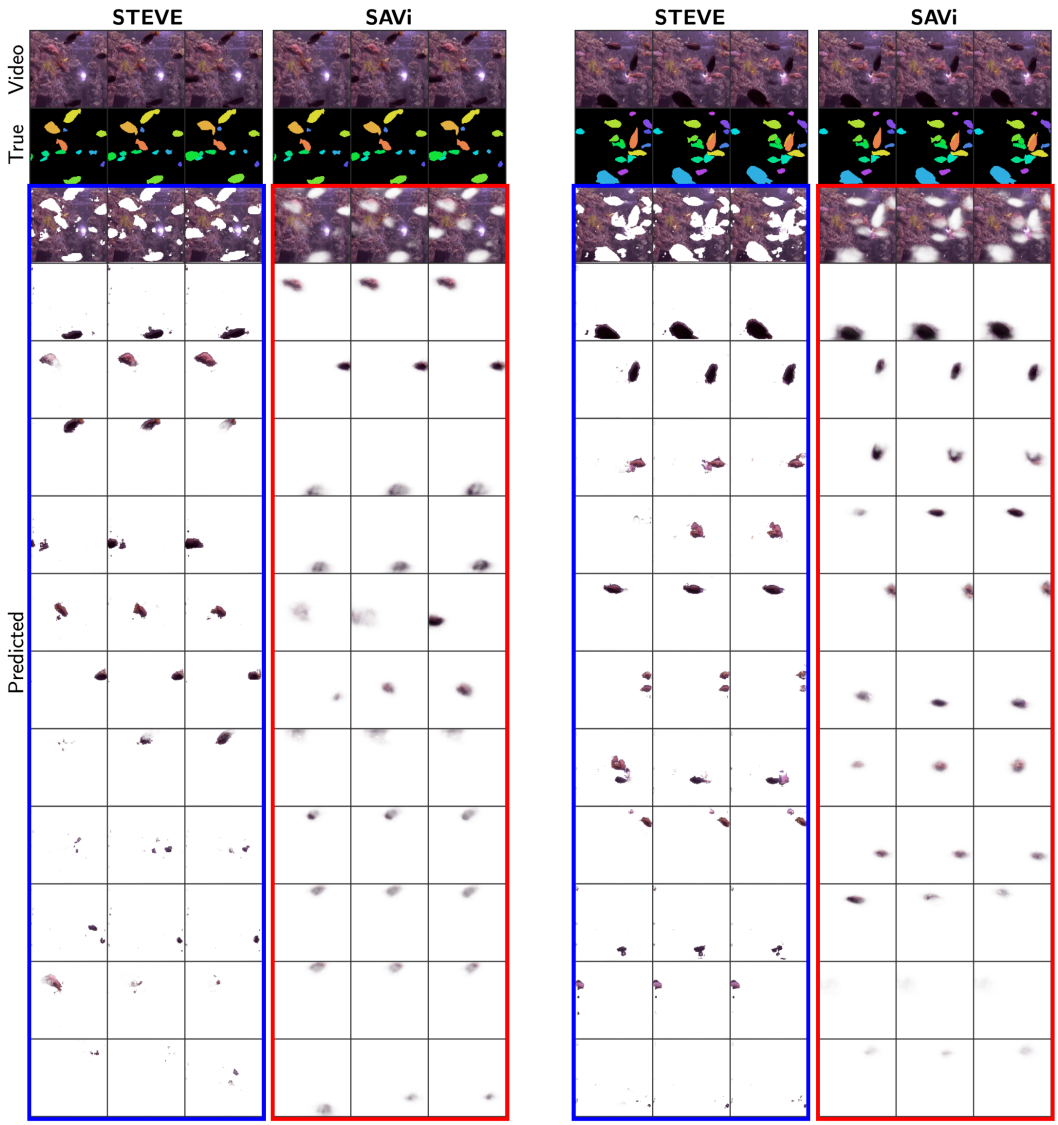


Figure 16: Additional Samples of Unsupervised Video Segmentation in Youtube Aquarium.

D Additional Related Work

Unsupervised Video Object Segmentation using Motion Cues. When objects in the video are moving independently, optical flow provides a strong signal of objectness as pixels with similar flow can be treated as an object. Therefore, several works pursue estimating optical flow in an fully unsupervised way (Ren et al., 2017; Janai et al., 2018; Wang et al., 2018; Jonschkowski et al., 2020; Stone et al., 2021). Leveraging flow, several methods have shown success in visually complex videos (Brox & Malik, 2010; Bideau & Learned-Miller, 2016; Yang et al., 2019; Tangemann et al., 2021; Yang et al., 2021). However, these can require access to optical flow even during deployment which can be challenging to obtain. In response, recent methods aim to infer objects via RGB-only input, but by using optical flow to supervise the training (Kipf et al., 2021; Bao et al., 2022). Even though these can handle naturalistic videos, their learning depends primarily on the flow information. If flow is absent e.g. in scenes with static objects, these can suffer (Greff et al., 2022). In comparison, our model is trained given only RGB video frames and we consider the use of motion information as orthogonal to our approach.

Unsupervised Segmentation Propagation. Several methods learn to propagate mask from one frame to another via unsupervised training and are shown to handle natural videos (Lai et al., 2020; Vondrick et al., 2018; Lai & Xie, 2019; Zhu et al., 2020; Oh et al., 2019; Wang et al., 2019a,b). However these methods only learn how to propagate the masks, and still require ground truth segmentation or bounding boxes in the first frame to actually perform tracking. Therefore, these methods are not fully unsupervised like ours.

E Additional Experiment Details

Evaluating the Effect of Number of Past Frames on Image Segmentation. We randomly take a video clip from the test set having T frames: $\mathbf{x}_1, \ldots, \mathbf{x}_T$. We would like to measure how the segmentation of the T-th frame $\mathbf{x}_T$ would change as a function of the number of past frames k shown to the model. For this, we predict the object masks $\mathcal{M}_T^k$ for the frame $\mathbf{x}_T$ given that the model has seen k frames $\mathbf{x}_{T-k}, \ldots, \mathbf{x}_{T-1}$ in the past. We compute the Image FG-ARI for the masks $\mathcal{M}_T^k$ by varying the value of $k \in \{0, \ldots, T-1\}$ and make the plot in Figure 2. We take $T = 7$.

Evaluation of Out-of-Distribution Generalization to More Objects. In this experiment, a model is tested on more objects than it was trained on. For this experiment, during testing, we increase the number of slots by an amount equal to the difference in maximum number of objects between the test and the training set.

F Limitations

We now describe the main limitations of STEVE and the future directions. First, there is some opportunity to improve the performance further within the proposed framework by adopting more advanced architectures than the minimal architecture we choose in this work. We would say that this is more of a future work than a limitation of this framework. Second, more computational efficiency is required in the future to handle images of higher resolution because the transformer decoder has a quadratic memory complexity and also because token-by-token generation can be slow. Third, the potential to scale the model to internet-scale datasets is discussed in this work but not investigated. Our current datasets typically contain about 200K video frames with an image resolution of 128. For scaling this up, we may also need to develop a new model in this framework. Fourth, although our focus in this paper is not on synthetic image generation, we found that the quality of image generation in naturalistic datasets has some room for further improvement. We believe that this could not only provide the ability to synthesize images and videos but also improve object decomposition as well. Fifth, even if our model outperforms previous works in general, we also observe some failure modes in MOVi-E such as splitting of large objects or merging of very small objects. To resolve this, we believe that it's worth investigating a much larger data on a bigger model or with a more sophisticated architecture. Lastly, the CNN backbone of our slot encoder is rather lightweight, having just 4-layers and a feature-size 64. For scaling this model to more complex natural images, it is likely that this lightweight CNN will not be enough.

G Impact Statement

There are no imminent negative societal consequences of our current work, however, future applications should be mindful to avoid malicious uses such as in surveillance and to minimize environmental impact when training larger transformers. At the same time, benign applications in scene understanding, robotics and autonomous navigation would benefit positively from our work.