Supplementary Material

RDIR: Capturing Temporally-Invariant Representations of Multiple Objects in Videos

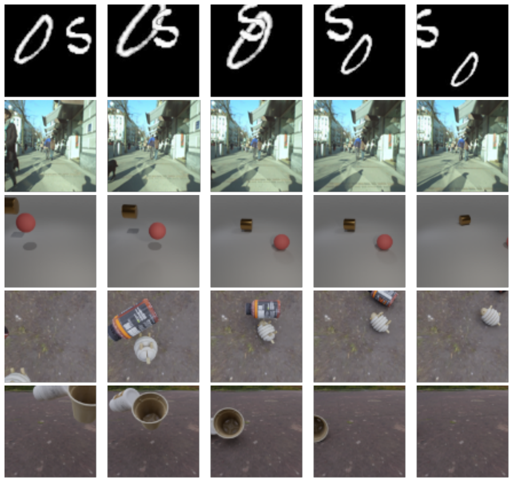
Figure 1. Sequences from datasets used in research: (1) multi-scale moving MNIST, (2) MOT15, (3) MOVi-A, (4) MOVi-C, (5) MOVi-E.

A. Dataset details

In this research, we used 6 datasets, which we describe in detail below. See Figure 1 for example sequence from each dataset (excluding COCO, which is an object detection dataset).

Multi-scale moving MNIST dataset is a benchmark dataset, generated to verify the robustness of the model to objects of varying sizes and locations in the image. It extends the idea of the multi-scale MNIST dataset proposed in SSDIR.

The dataset initializes a sequence with an image (sized 128×128) of multiple digits (their quantity randomly selected for each sequence, between n_{min} and n_{max}), scaled to one of the available sizes $\{s_1, s_2, ..., s_N\}$. Then, a sequence of T images is generated, where each digit moves around the canvas with the initial speed (v_x, v_y) (randomly selected between $(-1, 1)$). When hitting the border of the image, the digit bounces symmetrically. The size of each digit oscillates with a sinusoidal function, its size variation sv and oscillation period op selected from available: $\{sv_1, sv_2, ..., sv_N\}$, where $sv = 0.1$ means the size can vary by 10%; $\{op_1, op_2, ..., op_N\}$, where $op = 1.0$ means the oscillation period is equal to one second.

parameter	value
train sequences	8000
test sequences	2000
T	10
n_{min}	2
n_{max}	5
$\boldsymbol{s}$	$\{48, 72, 96\}$
$\boldsymbol{op}$	$\{1.0, 1.5, 2.0\}$
$\boldsymbol{sv}$	$\{0.0, 0.1, 0.2, 0.3\}$
FPS	10

Table 1. Parameters used for generating multi-scale moving MNIST dataset.

In this research, we used the main training dataset with settings presented in Table 1.

Then, for the evaluation we prepared separate datasets of the same size, without size oscillation and variance (each digit of the same size, with $sv = 0$) - **no scaling**, and without digit movement (the velocity was always 0) - **no translation**.

For the ablation study, we generated additional datasets with different numbers of objects (minimum 1 object, maximum between 1 and 7). These datasets contained only one subset with 1000 sequences.

MOT15 dataset was designed as a multi-object tracking benchmark, enables the evaluation of the representations in a more complex downstream task. The train subset contains 11 sequences of images, containing multiple walking pedestrians, as well as ground truth annotations, which include each object's bounding box and unique ID. The test subset does not provide ground truth IDs of each object, therefore in this research, we decided to extract the validation subset from the training subset, according to Table 2.

During each video-based model training, the sequences were split into 10-element subsequences, which could fit in the GPU memory. Then, for evaluation entire sequences

	sequence	resolution	frames
TRAIN	ADL-Rundle-8	1920×1080	654
	ETH-Bahnhof	640×480	1000
	ETH-Pedcross2	640×480	837
	KITTI-13	1242×375	340
	PETS09-S2L1	768×576	795
	TUD-Campus	640×480	71
	TUD-Stadtmitte	640×480	179
	Venice-2	1920×1080	600
			$\mathbf{\Sigma = 4467}$
VAL	ADL-Rundle-6	1920×1080	525
	ETH-Sunnyday	640×480	354
	KITTI-17	1224×370	145
			$\mathbf{\Sigma = 1024}$

Table 2. MOT15 dataset split details.

	resolution
SSDIR-YOLO	416×416
RDIR	416×416
SCALOR	64×64
SIMONe	128×128
PROVIDE	64×64

Table 3. Input sizes for each model training with MOT15 dataset.

were used, together with the provided annotations. Each sequence was resized to the maximum resolution for each model (bound by their design and the amount of GPU VRAM), presented in Table 3.

COCO dataset is a widely used benchmark dataset in computer vision research. The main aim is to enable the training and evaluation of algorithms for the task of object detection across real-life scenarios. The dataset consists of a diverse set of images, encompassing 80 annotated object classes, including people, animals, vehicles, and common household items. In this research, the dataset was used as the base for a pretrained object detection model, used later as a starting point to train a multi-object representation learning model on the MOT15 dataset.

MOVi datasets were used in the representation stability review in the original form provided by the authors. We decided to select MOVi-A, MOVi-C, and MOVi-E for the increasing level of complexity of the dataset. Each dataset contains sequences of 24 images (sized 256×256) and annotations. The dataset statistics are collected in Table 4.

For the entropy experiment, we generated additional smaller datasets with a single ($n_{min} = n_{max} = 1$) and two ($n_{min} = n_{max} = 2$) objects, referring to them as **single** and **dual** respectively. The dataset configuration was the same

	MOVi-A	**MOVi-C**	**MOVi-E**
train sequences	9703	9737	9749
test sequences	250	250	250

Table 4. Number of sequences in each MOVi dataset.

as the default dataset, except we generated only one subset with 1000 sequences.

B. Model architecture

In this section, we supply the description of the model architecture used by RDIR[1].

B.1. Encoder details

Algorithm 1 RDIR Encoder

```
1: input: image sequence $\boldsymbol{X}$
2: for $\boldsymbol{x}$ in $\boldsymbol{X}$ do
3:     $feats = BackboneNeck(\boldsymbol{x})$
4:     $[\boldsymbol{Z}_{where}, \boldsymbol{Z}_{where}] \leftarrow YOLOHead(feats)$
5:     $\boldsymbol{feats}_{mixed} \leftarrow Mixer(feats)$
6: end for
7: $\boldsymbol{feats}_{seq} = SeqEncoder(\boldsymbol{feats}_{mixed})$
8: for $feats_{seq}$ in $\boldsymbol{feats}_{seq}$ do
9:     $feats_{smixed} = SeqMixer(feats_{seq})$
10:    $\boldsymbol{Z}_{what} \leftarrow WhatEncoder(feats_{smixed})$
11:    $\boldsymbol{Z}_{depth} \leftarrow DepthEncoder(feats_{smixed})$
12: end for
13: output: latent representation for each image in sequence $\{\boldsymbol{Z}_{where}, \boldsymbol{Z}_{present}, \boldsymbol{Z}_{what}, \boldsymbol{Z}_{depth}\}$
```

The encoder pipeline is presented in Algorithm 1. We start by extracting the intermediate feature using the YOLOv4 backbone (CSPDarknet53) and neck (SPP+PANet), producing intermediate features. These features are then processed by the YOLO head to estimate objects' positions and class confidences; these are further processed to create $\boldsymbol{z}_{where}$ and $\boldsymbol{z}_{present}$ latent representation components. We modify the predicted bounding boxes by expanding them to squares, which allows for greater stability of the model (we found out that changing decoded objects' proportion in the spatial transformer module stops the model from converging).

The same intermediate features are forwarded to the Mixer component of RDIR, where each *Conv* block consists of a 2D convolutional layer, batch normalization, and Leaky ReLU activation function. We adjust the number and parameters of the convolutional layer to allow concatenation of the resulting feature maps, assuming the concate-

[1]The source code utilized for the research will be made publicly available for reproducibility upon acceptance of the paper.

nated feature map has a 2 times smaller number of channels than the original one. The output of the mixer contains the same number of feature maps (in the case of YOLOv4 backbone and neck this number is 3), with equal channel dimension (reduced to the smallest among all intermediate feature maps).

The output of the Mixer (a sequence of intermediate feature maps) is processed by the Sequence encoder. We include additional *Conv* blocks (similar to those in the Mixer), with a configurable number of convolutional layers (*seq CNN hidden*) and kernel size (*seq CNN kernel size*), controlling the level of sharing context among neighboring cells. Then, the output is flattened and forwarded as a sequence of multiple states to a recurrent layer of a configurable type (*RNN type*) and several recurrent cells (*RNN cells*). The output vector is transformed to restore the intermediate feature map shape, and processed by another block of convolutional layers (with the same config as those preceding the recurrent layer). Finally, the output is processed by another Mixer (with the same architecture as the previous one).

The resulting feature maps are used to generate latent representation in $\boldsymbol{z}_{what}$ and $\boldsymbol{z}_{depth}$ encoders. $\boldsymbol{z}_{what}$ encoder contains a configurable number of convolutional blocks with 3×3 kernels ($\boldsymbol{z}_{what}$ *hidden*); at the end there is another convolutional layer with 1-sized kernel, outputting feature maps with $\boldsymbol{z}_{what}$ *size* channels. These feature maps are used as grids of $\boldsymbol{z}_{what}$ latent representations. Similarly, $\boldsymbol{z}_{depth}$ encoder uses a single convolutional block and a 1-sized convolutional layer to produce feature maps for $\boldsymbol{z}_{depth}$ latent representations.

The representation learning part of the RDIR encoder can utilize a cloned backbone and neck to enable the model to learn better representations. During our research, we allowed cloning the neck and sharing the YOLO weights of the backbone for all latent representations. The cloned neck was trained along with the rest of the model, while the original backbone, neck, and YOLO head weights were frozen.

B.2. Decoder details

The flow of the decoder is presented in Algorithm 2. It is applied per image (it does not utilize the sequential nature of the data). First, each latent representation is filtered by sampling (training) or thresholding (inference) $\boldsymbol{z}_{present}$. During training, we found that introducing randomness allows the model to learn faster. To ensure the model learns the entire data distribution, we level the number of objects by adding negative examples to $\boldsymbol{z}_{present}$ for training. During inference, negative objects are not added, and instead, we apply Non-maximum Suppression with IoU threshold 0.45 to reduce the number of redundant objects predicted by YOLO Head.

Filtered $\boldsymbol{z}_{what}$ representations are then stacked and forwarded through the $\boldsymbol{z}_{what}$ decoder. It consists of a sequence of blocks containing 2D transposed convolutions, followed by batch normalization and Leaky ReLU activation function. The parameters of the convolutional layers are selected to provide a preset size of decoded images (*decoded size*) and the number of channels in the penultimate layer (*decoder channels*), starting from the $\boldsymbol{z}_{what}$ *size* channels. The final layer is a 2D transposed convolution with fixed 3 output channels, followed by a sigmoid activation function, outputting M individual objects' reconstructions images.

Algorithm 2 RDIR Decoder

```
1:  input: latent representation for each image in sequence
    $\{\boldsymbol{Z}_{where}, \boldsymbol{Z}_{present}, \boldsymbol{Z}_{what}, \boldsymbol{Z}_{depth}\}$
2:  for $\{\boldsymbol{z}_{where}, \boldsymbol{z}_{present}, \boldsymbol{z}_{what}, \boldsymbol{z}_{depth}\}$ in
    $\{\boldsymbol{Z}_{where}, \boldsymbol{Z}_{present}, \boldsymbol{Z}_{what}, \boldsymbol{Z}_{depth}\}$ do
3:      $\boldsymbol{z}_{what} = Filter\left(\boldsymbol{z}_{what}, \boldsymbol{z}_{present}\right)$
4:      $\boldsymbol{z}_{where} = Filter\left(\boldsymbol{z}_{where}, \boldsymbol{z}_{present}\right)$
5:      $\boldsymbol{z}_{depth} = Filter\left(\boldsymbol{z}_{depth}, \boldsymbol{z}_{present}\right)$
6:      for $[z_{what}, z_{where}]$ in $[\boldsymbol{z}_{what}, \boldsymbol{z}_{where}]$ do
7:          $o_{dec} = Decoder\left(z_{what}\right)$
8:          $\boldsymbol{o}_{transformed} \leftarrow STN\left(o_{dec}, z_{where}\right)$
9:      end for
10:     $\boldsymbol{Y} \leftarrow Merge\left(\boldsymbol{o}_{transformed}\right)$
11: end for
12: output: image sequence reconstruction $\boldsymbol{Y}$
```

Reconstructed objects are transformed to the original image size based on the $\boldsymbol{z}_{where}$ latent. The overall image reconstruction is merged by sorting them according to $\boldsymbol{z}_{depth}$ latent, and pasting sequentially to the output image. The result contains individual objects' reconstructions, without considering the image background.

C. Model training setup

RDIR was trained on a single NVIDIA A40 GPU in a staged training setup:

1. An object detection model is trained on an annotated dataset with supervision. We utilize YOLOv4, which is then parsed to reuse its backbone and prediction head in the encoder.

2. An image-based model is trained for learning multiple representations of objects on images. We remove the Sequence encoder and the final Mixer from the RDIR model, and train it as an image autoencoder without supervision, using MSE as the loss function. During this stage, the backbone and YOLO head weights are frozen. Finally, we re-use the checkpoint with the lowest validation loss for the following stage. In our research we discovered that this step allows us to reach better results than training the recurrent version di-

parameter	**MNIST**	**MOT15**	**MOVi-A**	**MOVi-C**	**MOVi-E**
batch size	256	4	64	64	64
image size	128	416	256	256	256
decoded size	32	32	32	32	32
decoder channels	16	128	256	256	256
z_{what} size	64	128	256	256	256
z_{what} hidden	3	5	5	5	5
α_{total}	5	5	5	5	5
α_{obj}	10	10	10	10	10
RNN type	BiGRU	GRU	BiGRU	BiGRU	BiGRU
RNN cells	2	2	2	2	2
seq CNN hidden	2	2	2	2	2
seq CNN kernel size	5	5	5	5	5

Table 5. Model parameters used for learning representations on each dataset.

rectly after the object detection model; the total training time is reduced as well.

3. Finally, we extend the model from the previous stage with the Sequence encoder and an additional Mixer to form the RDIR model. We then start with a short pre-training, where we find the initial set of parameters for the newly added modules (the rest of the model is frozen). After that we continue training the model in the same way as in the second stage (with the backbone and YOLO heads frozen), utilizing sequence MSE as the loss function.

In Table 5 we collected model hyperparameters used in the research.

D. Representation consistency

The analysis of representation consistency aims at verifying the ability of models to capture temporally- and spatially invariant representations of objects in the scene. To do that, we use models trained on the multi-scale moving MNIST dataset to produce representations of each sequence in the test datasets. Then, all representations are assigned to the appropriate class (by choosing the ground truth label according to the maximum intersection over union between the predicted object mask and the ground truth box), and we apply Kernel Density Estimation to find the probability density of per-class representations across the dataset. Then, we score each representation to calculate the average entropy of representations according to Eq. 1:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \qquad (1)$$

The final value is obtained by calculating the mean across all ground truth classes. The resulting value is low due to the high dimensionality of the representations. We compare the exact value of entropy - lower means less information, *i.e.* less variability of the representation.

entropy	MNIST (10^{-24})
SCALOR	1.961 ± 0.002
PROVIDE	1.951 ± 0.003
SIMONe	10.649 ± 2.665
SSDIR-YOLO	3.012 ± 0.195
RDIR	1.948 ± 0.001

Table 6. Representation stability on MNIST: average per-class representation entropy across the dataset. RDIR provides low entropy, which is enough to supply high quality of representations (as shown in the downstream task).

Table 6 shows a comparison of representation entropy among the compared models. RDIR achieves low representation entropy, which together with high performance in the downstream task shows, that RDIR can infer valuable representations, consistent across the varying sequences. Similar levels of entropy were achieved by SCALOR and PROVIDE, but these models performed significantly worse in the downstream task.

Both SSDIR-YOLO and SIMONe have higher entropy of representations. This shows that, although these representations can be applied in downstream tasks, they vary more significantly. It is worth noting, that the addition of the Sequence Encoder improves the representation stability.

In Table 7 we show a similar comparison for RDIR across additional, more complex MOVi datasets. Here as well, the model was trained on the train dataset and used for inference on the test dataset. To review the method's robustness to multiple objects in the scene, we also included additional datasets containing only one (*single*) and two (*dual*) objects in each sequence. We can see entropy of representations decreases with the increasing number of objects within the scene, as the entropy for *all* datasets is lower than in the

entropy		**RDIR** (10^{-100})
MOVi-A	all	1.832 ± 0.059
	single	2.126 ± 0.126
	dual	2.161 ± 0.178
MOVi-C	all	1.667 ± 0.006
	single	2.199 ± 0.079
	dual	1.945 ± 0.101
MOVi-E	all	1.791 ± 0.017
	single	2.255 ± 0.199
	dual	1.821 ± 0.035

Table 7. Representation stability on MOVi: average per-class representation entropy across the dataset. RDIR infers more stable representations when more objects are present within the image.

case of *dual* and *single*. This shows the method's robustness to the number of objects in the scene.

E. Computational expense

In this section, we present a comparative analysis of the computational costs associated with RDIR in contrast to the baseline methods. The computational expense of deep neural networks involves a range of considerations, including hardware, software, model design, and data. We assess this computational expense by evaluating each model convergence time and the count of trainable parameters (Table 8), as well as the iteration speed (Table 9). It is important to note that the training of the baseline methods was executed on hardware distinct from that used for our proposed method. Owing to constraints in computational resources, we couldn't execute all experiments on a single machine. Nevertheless, despite variations in GPU power and batch size used for training (which, in all instances, maximized the memory capacity of a single GPU card), it's evident that the staged training approach employed by RDIR incurs significantly lower computational costs compared to the other methods.

The inference speed benchmark was executed on the same GPU (NVIDIA A40). Each model was run on 1000 random sequences (generated and loaded in the memory before measuring execution time). For each model, we decided to use batch size 1. The input image size was 64×64; in the case of SSDIR-YOLO and RDIR images were upscaled to 128×128, following the approach from the paper. The results show a much higher inference speed of both SSDIR-YOLO and RDIR compared to other methods. This is caused by the extensive use of recurrent cells and iterative inference in the baseline methods; RDIR is highly optimized and allows for parallel inference at all stages.

F. Inference visualizations

In this section we supply qualitative results of RDIR processing and compare them with baseline methods: SSDIR-YOLO, SCALOR, PROVIDE, and SIMONe. The visualization consists of the input image ($input$), the reconstruction produced by the model ($reconstruction$), reconstruction with attention visualization ($attention$ - bounding boxes for spatial attention-based models and masks for scene-mixture models), and a few per-object reconstructions ($objects$) for a single element in the sequence, showing how the model interprets the scene and objects within it. We demonstrate visualizations for a few timesteps in the input sequence ($T = \{1, 4, 7, 10\}$), for two sample sequences from each dataset.

F.1. Multi-scale moving MNIST

Figures 2, 3, 4, 5, 6 show inference on the multi-scale moving MNIST dataset. Thanks to the pretraining phase, both RDIR and SSDIR can accurately attend to individual digits in the sequence. The added sequential context allows RDIR to perform slightly better in cases of high overlapping.

SCALOR succeeds in discovering the initial structure of the image, however, inference on subsequent frames diverges. In the case of scene-mixture models, both SIMONe and PROVIDE can accurately reconstruct the input image for each element in the sequence, however, the inferred masks span across multiple objects, which makes analyzing them individually difficult. It is worth mentioning how SIMONe can attend to objects only, without focusing on the background (which in this case does not contain any information).

F.2. MOT15

Figures 7, 8, 9, 10, 11 present inference of each model (RDIR, SSDIR-YOLO, SCALOR, PROVIDE and SIMONe respectively) trained on the MOT15 dataset.

It is easily visible, that RDIR and SSDIR-YOLO do not reconstruct the entire image, instead pasting per-object reconstructions, which results in inaccurate reconstructions of the input image. However, Per-object reconstructions show human-like silhouettes, proving these models can infer per-object representations, which can be used to generate a simplified reconstruction of their appearance. Due to the low resolution of object reconstructions, their quality is low; RDIR reconstructions seem to contain more details about each object's appearance (such as clothes color, and posture), whereas SSDIR-YOLO reconstructions are mainly alike silhouettes.

The reconstructions obtained from the other methods confirm, that the fully unsupervised setup cannot provide appropriate scene understanding in complex scenarios.

	SCALOR	PROVIDE	SIMONe	SSDIR-YOLO	**RDIR**
Training time ($10^3 s$)	435.54 ± 3.92	523.11 ± 3.22	238.71 ± 15.17	9.34 ± 0.68	22.34 ± 1.36
Batch size	16	4	16	256	32
GPU	RTX TITAN	RTX TITAN	RTX TITAN	A40	A40

Table 8. Model training expense comparison. SSDIR-YOLO and RDIR converge much faster than other methods, even considering the use of a more powerful GPU.

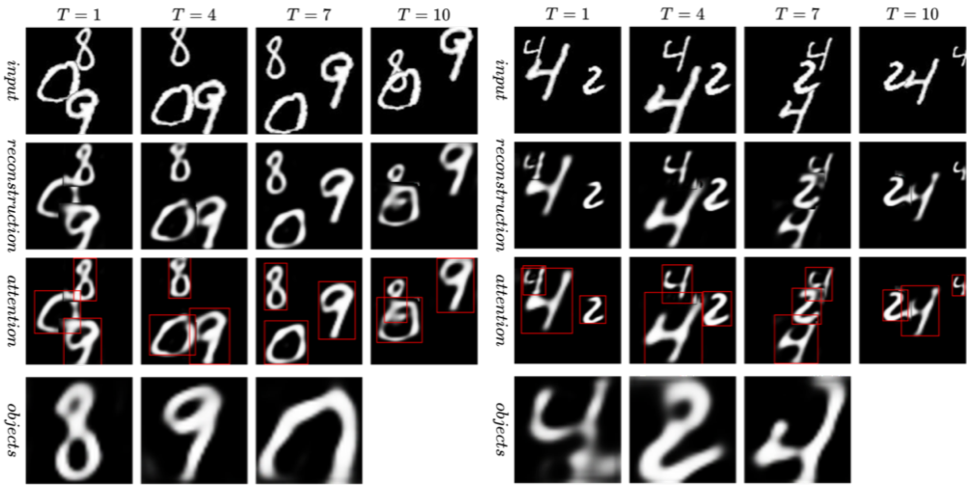

Figure 2. RDIR inference on multi-scale moving MNIST dataset.

	iterations per second
SCALOR	4.899
PROVIDE	2.758
SIMONe	8.938
SSDIR-YOLO	31.277
RDIR	27.645

Table 9. Model inference speed. SSDIR-YOLO and RDIR are much faster than other methods, despite using a higher input resolution.

Similarly to insights from previous studies, such models tend to split the scene into subparts, modeled by each attention mask, but these masks usually encapsulate multiple objects at once. Even though input images' reconstructions are more accurate than in the case of RDIR and SSDIR-YOLO, the analysis of attention masks shows that these models did not focus on objects in the scene (here: pedestrians), which explains the significantly lower metrics achieved by these models in the downstream multi-object tracking task.

F.3. MOVi datasets

Finally, Figures 12, 13, 14 present RDIR inference on the MOVi datasets. Once again, the staged training approach allows the model to accurately focus on the objects detected in the scene, however, the higher complexity of the data causes several false positive predictions. Nevertheless, the model can ignore them in the final reconstruction, as visible in the $reconstructions$ row. The model performs better on the simpler MOVi-A dataset, as it contains simple geometric objects. The quality of the individual objects is low, due to the low resolution used before transforming them to the original image position. Increasing the intermediate objects' resolution could potentially lead to a better quality of the latent representation, however, it comes at a significant increase in computational cost. Despite the reduced quality of per-object reconstructions, the final output of the model resembles the input in the area of detected objects.

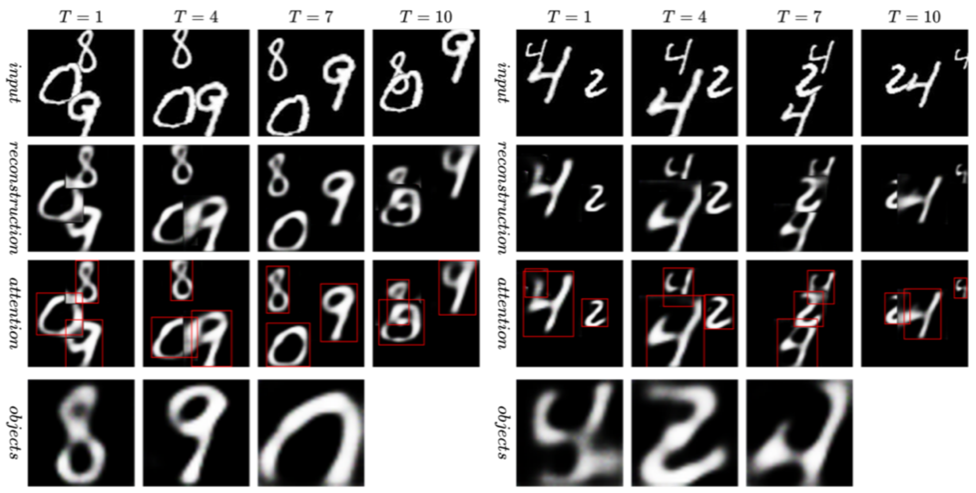

Figure 3. SSDIR-YOLO inference on multi-scale moving MNIST dataset.

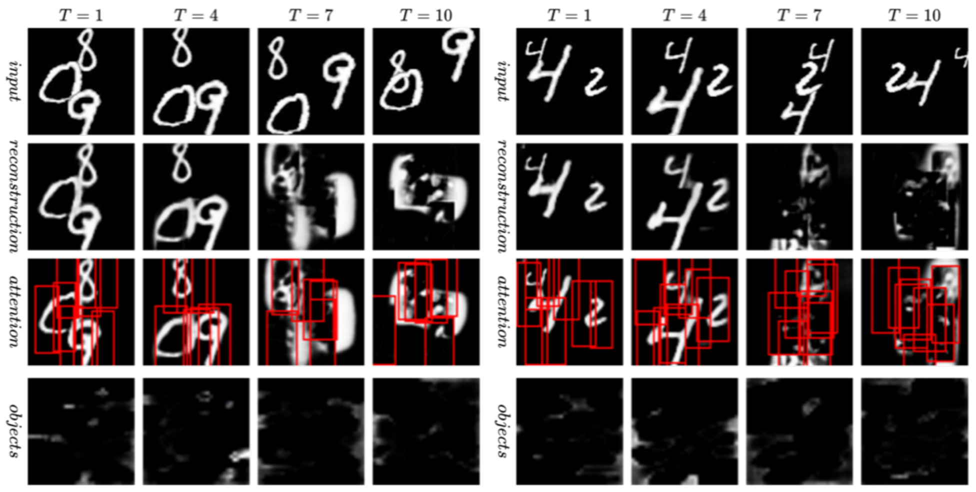

Figure 4. SCALOR inference on multi-scale moving MNIST dataset.

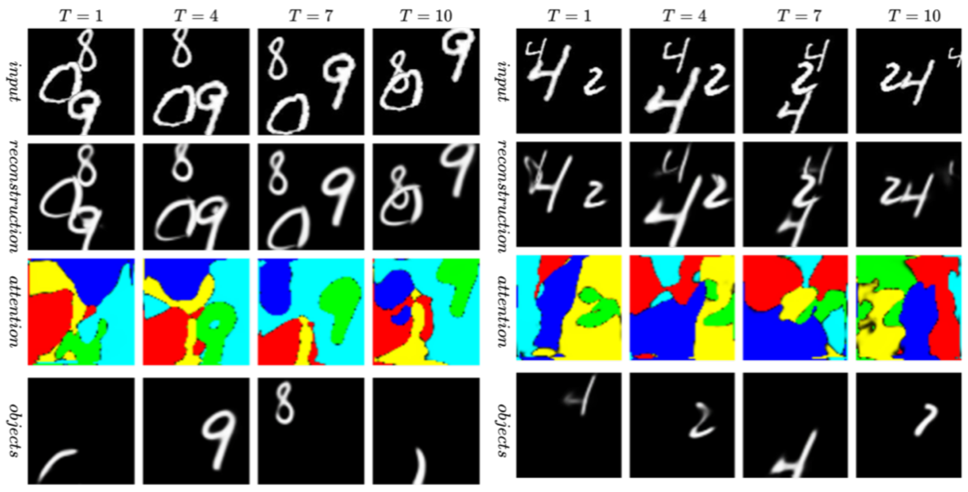

Figure 5. PROVIDE inference on multi-scale moving MNIST dataset.

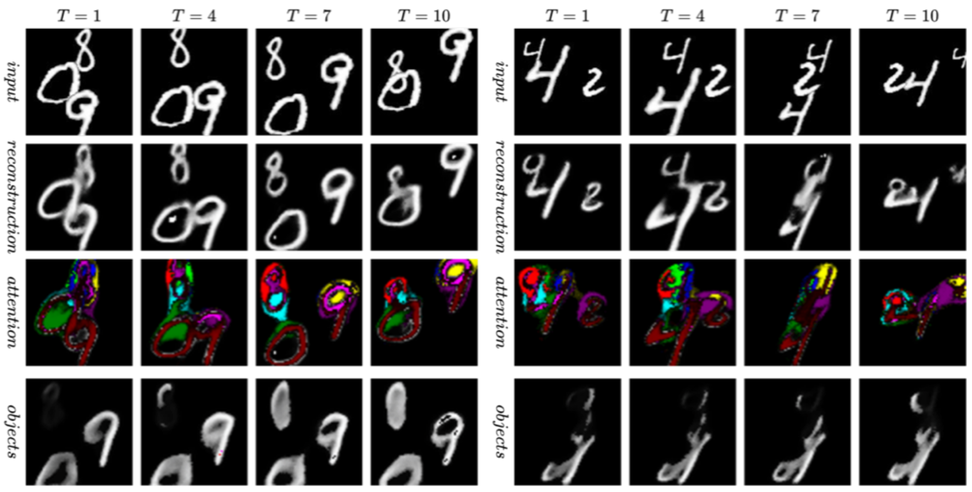

Figure 6. SIMONe inference on multi-scale moving MNIST dataset.

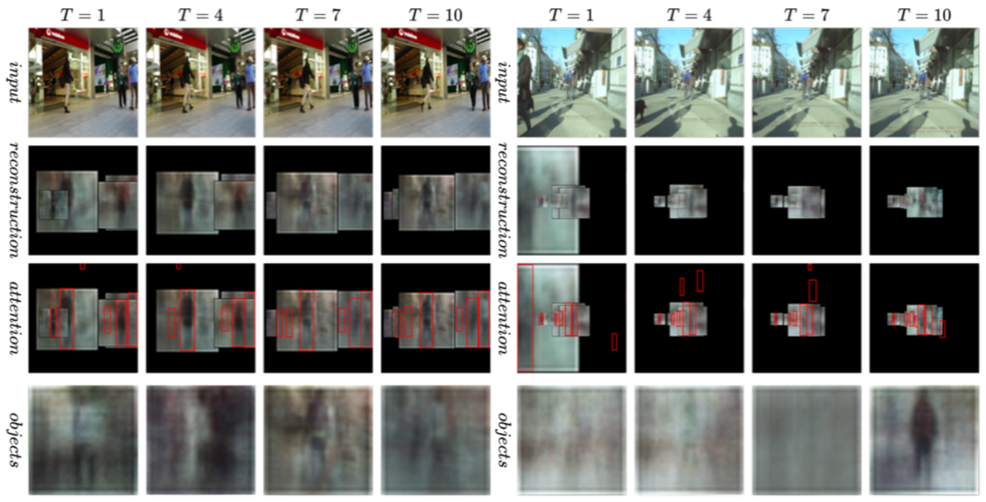

Figure 7. RDIR inference on MOT15 dataset.

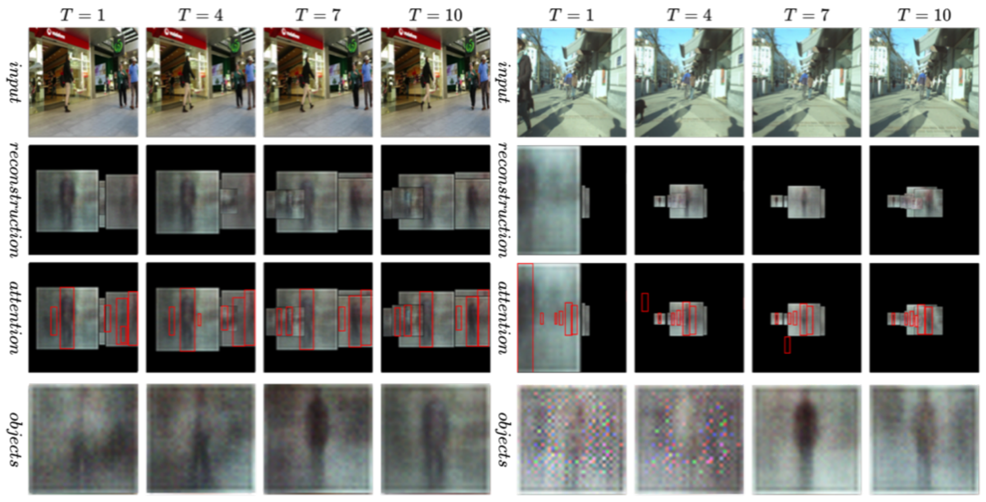

Figure 8. SSDIR-YOLO inference on MOT15 dataset.

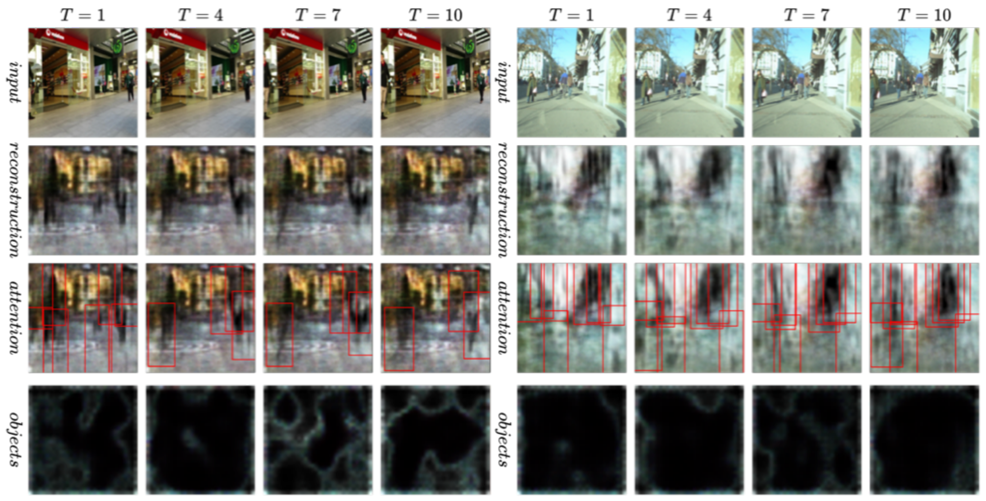

Figure 9. SCALOR inference on MOT15 dataset.

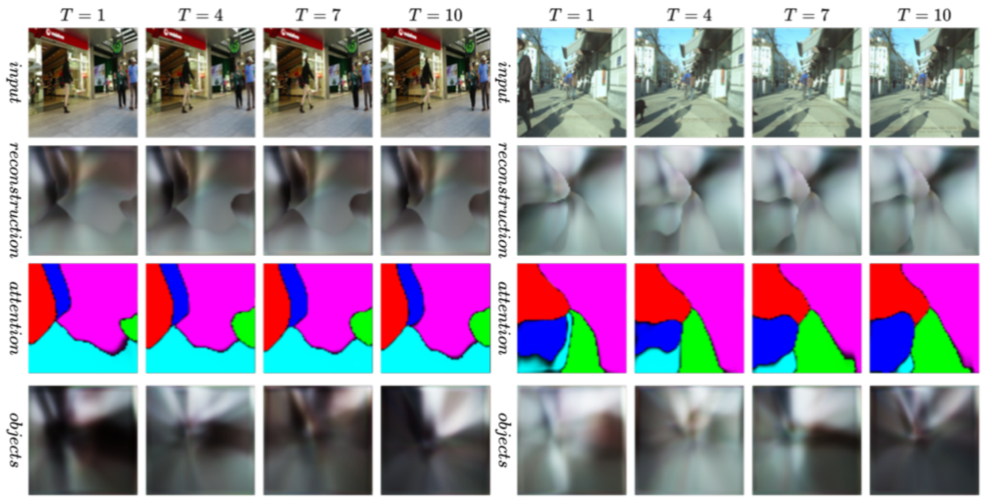

Figure 10. PROVIDE inference on MOT15 dataset.

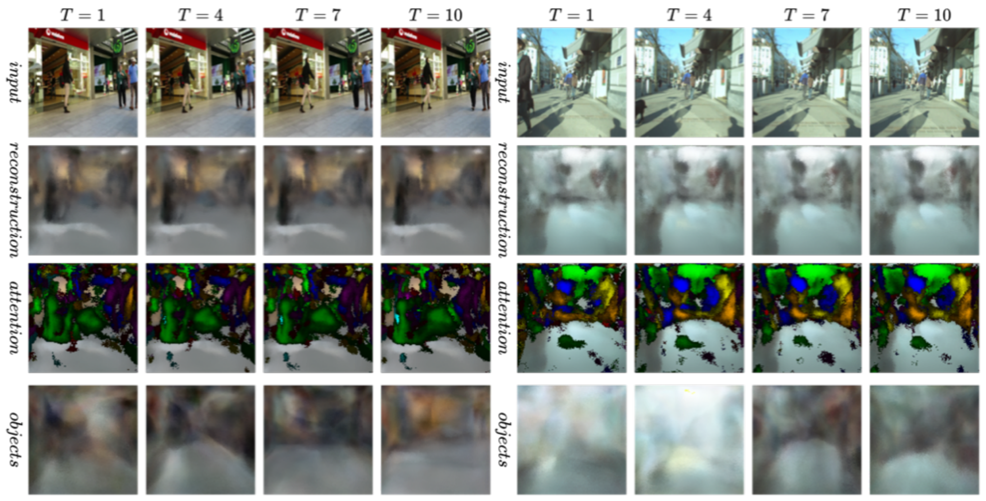

Figure 11. SIMONe inference on MOT15 dataset.

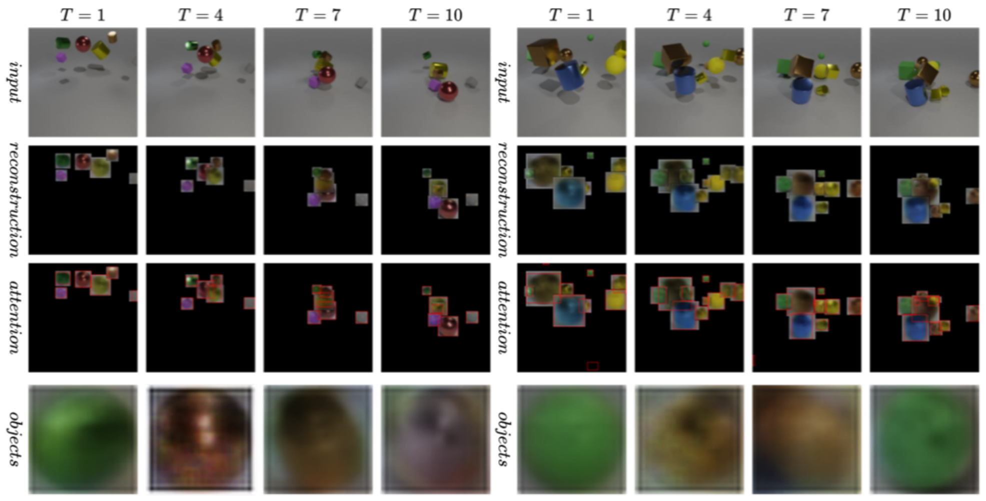

Figure 12. RDIR inference on MOVi-A dataset.

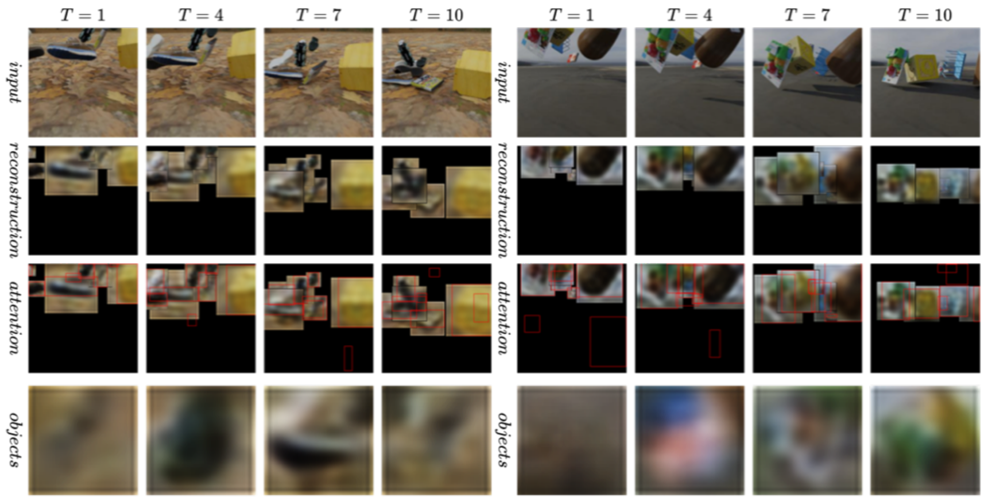

Figure 13. RDIR inference on MOVi-C dataset.

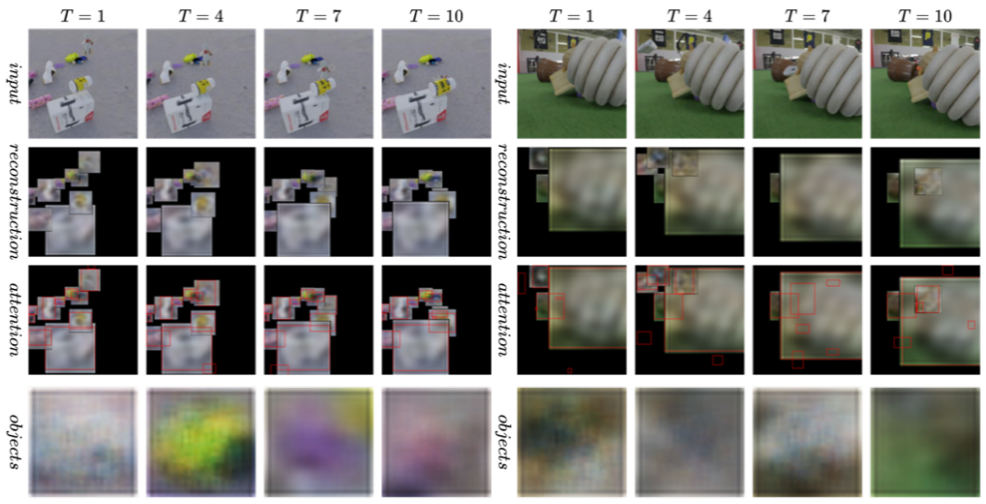

Figure 14. RDIR inference on MOVi-E dataset.