# A. Further Discussion

## A.1. Sequences

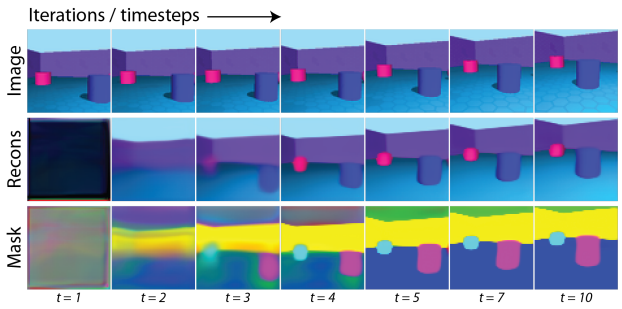


*Figure 12.* IODINE applied to Objects Room sequences by setting $N$, the number of refinement iterations, equal to the number of timesteps in the data.

The iterative nature of IODINE lends itself readily to sequential data, by, e.g., feeding a new frame at every iteration, instead of the same input image $\mathbf{x}$. This setup corresponds to one iteration per timestep, and using next-step-prediction instead of reconstruction as part of the training objective. An example of this can be seen in Figure 12 where we show a 16 timestep sequence along with reconstructions and masks. When using the model in this way, it automatically maintains the association of object to slot over time (i.e, displaying robust slot stability). Thus, object tracking comes almost for free as a by-product in IODINE. Notice though, that IODINE has to rely on the LSTM that is part of the inference network to model any dynamics. That means none of the dynamics of tracked objects (e.g. velocity) will be part of the object representation.

## A.2. Memory Limitations

It is worth pointing out that memory consumption presents an important limiting factor to scaling IODINE. To allow training by backpropagation, each slot and each refinement step require the storage of activations for an entire decoder and refinement network. Memory consumption during training thus scales linearly with both $K$ and $T$. This is particularly restrictive for sequential data, where the number of steps can grow very large. In our experiments from Appendix A.1, we found that 16 timesteps with a batch-size of 4 was the upper limit on GPUs with 12GB of RAM. Of course this also depends on the size of the input and the size of the network. Note also that at inference time there is no need to keep the activations of previous timesteps, so the dependence on $T$ can be eliminated there.

## A.3. Comparison with MONet

The Multi-Object NETwork (MONet; Burgess et al. 2019) is a complementary method for unsupervised object representation learning also developed recently. It learns to sequentially attend to individual objects using a masking network and a VAE. In each step the masking network segments out a yet unexplained part of the image (the next object) which is then fed to the VAE which has to reconstruct that object and the mask. Thus, in contrast to IODINE, MONet uses one iteration per object and doesn't adjust an object once it has been covered.

Both methods focus on the representation learning aspect and both ensure that all objects are encoded in the same format by sharing weights across objects. In our preliminary experiments MONet produced results very similar to IODINE on CLEVR both in terms of segmentation and regarding the quality of object representations, and also learns to inpaint occluded parts of objects.

Since MONet only visits each object once, it is a more lightweight method that requires less computation and memory to train and run. Recurrently iterating over objects also has the benefit that the model can dynamically vary the number of objects, whereas in IODINE the maximum number of objects is a hyperparameter that has to be fixed manually (though it can be changed at test time). The usage of a separate masking network which isn't directly subject to a representational bottleneck likely leads to less regularization for the segmentation mask. This could potentially allow MONet to better deal with complex segmentation shapes. But it also has to use that ability to directly produce masks that respect occlusion, whereas IODINE tends to produce masks for full unoccluded objects and leverages the softmax to resolve overlap. For more complex scenes, we also expect iterative refinement to be advantageous for resolving difficult cases. There, IODINE could start with a rough segmentation and then use the progressively better understanding of the constituent objects for refining the boundaries.

The segmentation process of MONet is deterministic which induces an order on the objects, which might be useful because it naturally prioritizes salient objects. We observed that it typically starts with the background, then processes large frontal objects, and finally smaller or farther away objects. But this approach does break symmetry between objects, and we prefer keeping such a bias out of the object segmentation learning as much as possible.

Another disadvantage of a deterministic segmentation is that it cannot directly deal with ambiguous cases like the one shown in Section 4.4 and Figure 10. The iterative message-passing-like approach of IODINE might also lend itself well for incorporating top-down feedback to bias the segmentation towards one that is useful for a given task. It is less clear how to do that in MONet, though adding a way for conditioning the masking network could potentially serve a similar purpose. Finally the iterative refinement of IODINE naturally extends to sequential data (see Appendix A.1)

which would be less straightforward for MONet.

In summary, it is not at all clear yet which approach will work better and under which circumstances. If the data is sequential or contains ambiguity, IODINE presents a better choice. For other data that is not visually more complex than CLEVR, both methods will likely produce similar results making MONet the simpler and less computationally intensive choice. For more complex data it is unclear yet which approach would be the better choice, and in fact a hybrid approach might be the most promising. Sequentially attending to objects and iterative refinement are not mutually exclusive and might support each other. We consider this a very attractive research direction and are excited to explore its possibilities.

## B. Dataset Details

### B.1. CLEVR

We regenerated the CLEVR dataset (Johnson et al., 2017) using the authors' open-source code, because we needed ground-truth segmentation masks for evaluation purposes. The dataset contains 70 000 images with a resolution of $240 \times 320$ pixels, from which we extract a square center crop of $192 \times 192$ and scale it to $128 \times 128$ pixels. Each scene contains between three and ten objects, characterized in terms of shape (cube, cylinder, or sphere), size (small or large), material (rubber or metal), color (8 different colors), position (continuous), and rotation (continuous).

The subset of images which contain 3-6 objects (inclusive) served as the training set for our experiments; we refer to it as CLEVR6. Unless noted otherwise, we evaluate models on the full CLEVR distribution, containing 3-10 objects. All references to CLEVR refer to the full distribution.

We do not make use of the question answering task. Figure 13 shows a few samples from the dataset.

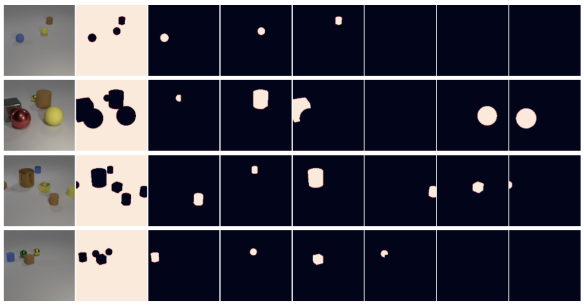

*Figure 13.* Samples from CLEVR6. The first column is the scene, the second column is the background mask and the following columns are the ground-truth object masks.

### B.2. Multi-dSprites

This dataset, based on the dSprites dataset (Matthey et al., 2017), consists of 60 000 images with a resolution of $64 \times 64$. Each image contains two to five random sprites, which vary in terms of shape (square, ellipse, or heart), color (uniform saturated colors), scale (continuous), position (continuous), and rotation (continuous). Furthermore the background color is varied in brightness but always remains grayscale. Figure 14 shows a few samples from the dataset.

We also used a binarized version of Multi-dSprites, where the sprites are always white, the background is always black, and each image contains two to three random sprites.

### B.3. Tetris

We generated this dataset of 60 000 images by placing three random Tetrominoes without overlap in an image of $35 \times 35$ pixels. Each Tetromino is composed of four blocks that are each $5 \times 5$ pixels. There are a total of 17 different Tetrominoes (counting rotations). We randomly color each Tetromino with one of 6 colors (red, green, blue, cyan, magenta, or yellow). Figure 15 shows a few samples from the dataset.

### B.4. Shapes

We use the same shapes dataset as in (Reichert & Serre, 2013). It contains 60 000 binary images of size $28 \times 28$ each with three random shapes from the set $\{\triangle, \triangledown, \square\}$.

### B.5. Objects Room

For the preliminary sequential experiments we used a sequential version of the *Objects Room* dataset (Burgess et al., 2019). This dataset consists of 64x64 RGB images of a cubic room, with randomly colored walls, floors and objects randomly scattered around the room. The camera is always positioned on a ring inside the room, always facing towards the centre and oriented vertically in the range $(-25°, 22°)$. There are 3 randomly shaped objects in the room with 1-3 objects visible in any given frame. This version contains sequences of camera-flights for 16 time steps, with the camera

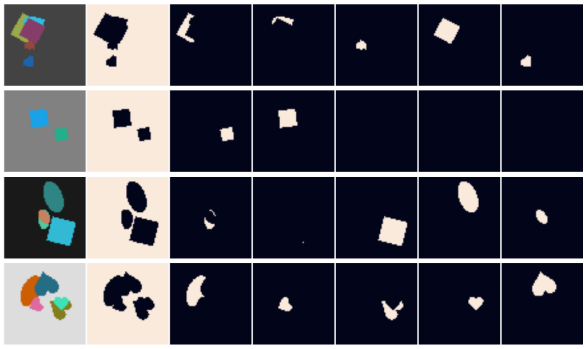

*Figure 14.* Samples from the Multi-dSprites dataset. The first column is the full image, the second column is the background mask and the following columns are the ground-truth object masks.

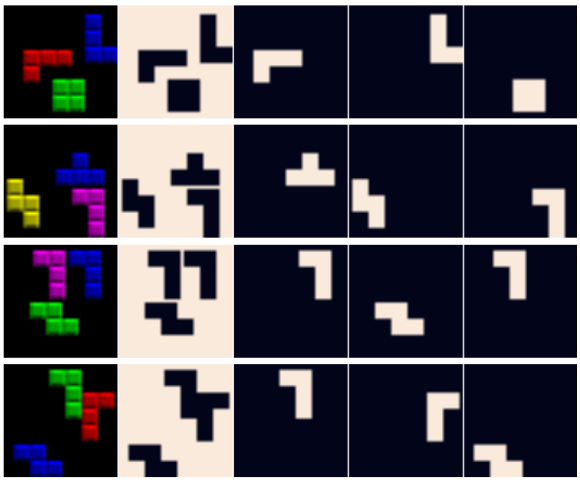

*Figure 15.* Samples from the Tetris dataset. The first column is the full image, the second column is the background mask and the following columns are the ground-truth object masks.

position and angle (within the above constraints) changing according to a fixed velocity for the entire sequence (with a random velocity sampled for each sequence).

## C. Model and Hyperparameter Details

**Training** Unless otherwise specified all the models are trained with the ADAM optimizer (Kingma & Ba, 2015), with default parameters and a learning rate of $0.0003$. We used gradient clipping as recommended by (Pascanu et al., 2012): if the norm of global gradient exceeds $5.0$ then the gradient is scaled down to that norm. Note that this is virtually always the case as the gradient norm is typically on the order of $10^5$, but we nonetheless found it useful to apply this strategy. We always use $\sigma = 0.1$ for the global scale of the output distribution $p(\mathbf{x}|\mathbf{z}^{(t)}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k^{(t)}, \sigma^2)$. Finally, batch size was 32 ($4 \times 8$GPUs).

**Initialization of Posterior** IODINE iteratively refines an initial posterior $\boldsymbol{\lambda}^{(1)}$ which is independent of the input data. Initially we set this initial value to match the prior (i.e. $q_{\boldsymbol{\lambda}}(\mathbf{z}_k^{(1)}) = \mathcal{N}(\mathbf{0}, \mathbf{1})$). But we found that this poses problems for the model, because of the competing requirements it poses for structuring the latent space w.r.t. the prior: On the one hand, samples from the prior need to be good starting values for iterative refinement. On the other hand, the prior should correspond to the accumulated posterior (KL term). For this reason we decided to simply make the parameters $\boldsymbol{\lambda}^{(1)}$ of the initialization distribution trainable parameters which are optimized alongside the weights of the decoder ($\boldsymbol{\theta}$) and of the refinement network ($\boldsymbol{\phi}$). This lead to faster training, and improved the visual quality of reconstructions from prior samples.

**Inputs** For all models, we use the following inputs to the refinement network, where LN means Layernorm and SG means stop gradients, and we omit the iteration index $.^{(t)}$ for brevity. The following image-sized inputs are concatenated and fed to the corresponding convolutional network:

| Description | Formula | LN | SG | Ch. |
|---|---|---|---|---|
| image | $\mathbf{x}$ | | | 3 |
| means | $\boldsymbol{\mu}$ | | | 3 |
| mask | $\boldsymbol{m}_k$ | | | 1 |
| mask-logits | $\hat{\boldsymbol{m}}_k$ | | | 1 |
| mask posterior | $p(\boldsymbol{m}_k\|\mathbf{x}, \boldsymbol{\mu})$ | | | 1 |
| gradient of means | $\nabla_{\boldsymbol{\mu}_k}\mathcal{L}$ | ✓ | ✓ | 3 |
| gradient of mask | $\nabla_{\boldsymbol{m}_k}\mathcal{L}$ | ✓ | ✓ | 1 |
| pixelwise likelihood | $p(\mathbf{x}\|\mathbf{z})$ | ✓ | ✓ | 1 |
| leave-one-out likelih. | $p(\mathbf{x}\|\mathbf{z}_{i\neq k})$ | ✓ | ✓ | 1 |
| coordinate channels | | | | 2 |
| | | | total: | 17 |

The posterior parameters $\boldsymbol{\lambda}$ and their gradients are flat vectors, and as such we concatenate them with the output of the convolutional part of the refinement network and use the result as input to the refinement LSTM:

| Description | Formula | LN | SG |
|---|---|---|---|
| gradient of posterior | $\nabla_{\boldsymbol{\lambda}_k}\mathcal{L}$ | ✓ | ✓ |
| posterior | $\boldsymbol{\lambda}_k$ | | |

**Architecture** All layers use the ELU (Clevert et al., 2015) activation function and the Convolutional layers use a stride equal to 1, unless mentioned otherwise. Architecture details for the individual datasets are summarized in the following subsections.

### C.1. CLEVR

All models were trained on scenes with 3-6 objects (CLEVR6) with $K = 7$ slots and $T = 5$ iterations. When evaluating on the full CLEVR dataset, we increased the number of slots to $K = 11$. For some of the analysis, we varied $T$ and $K$ as mentioned in the text.

The rest of the architecture and hyperparameters are described in the following.

**Decoder**

| Type | Size/*Ch.* | Act. Func. | Comment |
|---|---|---|---|
| Input: $\boldsymbol{\lambda}$ | 128 | | |
| Broadcast | *130* | | + coordinates |
| Conv $3 \times 3$ | *64* | ELU | |
| Conv $3 \times 3$ | *64* | ELU | |
| Conv $3 \times 3$ | *64* | ELU | |
| Conv $3 \times 3$ | *64* | ELU | |
| Conv $3 \times 3$ | *4* | Linear | RGB + Mask |

**Refinement Network**

| Type | Size/*Ch.* | Act. Func. | Comment |
|---|---|---|---|
| MLP | 128 | Linear | |
| LSTM | 256 | Tanh | |
| Concat $[\boldsymbol{\lambda}, \nabla_{\boldsymbol{\lambda}}\mathcal{L}]$ | 512 | | |
| MLP | 256 | ELU | |
| Avg. Pool | 64 | | |
| Conv $3 \times 3$ | *64* | ELU | |
| Conv $3 \times 3$ | *64* | ELU | |
| Conv $3 \times 3$ | *64* | ELU | |
| Conv $3 \times 3$ | *64* | ELU | |
| Inputs | *17* | | |

**Deconv Decoder** used in Section 4.3

| Type | Size/*Ch.* | Act. Func. | Comment |
|---|---|---|---|
| Input: $\boldsymbol{\lambda}$ | 128 | | |
| MLP | 512 | ELU | |
| MLP | 512 | ELU | |
| Reshape | *8* | | $8 \times 8 \times 8$ |
| Conv $5 \times 5$ | *64* | ELU | stride 2 |
| Conv $5 \times 5$ | *64* | ELU | stride 2 |
| Conv $5 \times 5$ | *64* | ELU | stride 2 |
| Conv $5 \times 5$ | *64* | ELU | stride 2 |
| Conv $5 \times 5$ | *64* | ELU | |
| Conv $5 \times 5$ | *4* | Linear | RGB + Mask |

### C.2. Multi-dSprites

Models were trained with $K = 6$ slots, and used $T = 5$ iterations.

**Decoder**

| Type | Size/*Ch.* | Act. Func. | Comment |
|---|---|---|---|
| Input: $\boldsymbol{\lambda}$ | 32 | | |
| Broadcast | *34* | | + coordinates |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *4* | Linear | RGB + Mask |

**Refinement Network**

| Type | Size/*Ch.* | Act. Func. | Comment |
|---|---|---|---|
| MLP | 32 | Linear | |
| LSTM | 128 | Tanh | |
| Concat $[\boldsymbol{\lambda}, \nabla_{\boldsymbol{\lambda}}\mathcal{L}]$ | 192 | | |
| MLP | 128 | ELU | |
| Avg. Pool | 32 | | |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *32* | ELU | |
| Inputs | *17* | | |

### C.3. Tetris

Models were trained with $K = 4$ slots, and used $T = 5$ iterations. For Tetris, in contrast to the other models, we did not use an LSTM in the refinement network.

**Decoder**

| Type | Size/*Ch.* | Act. Func. | Comment |
|---|---|---|---|
| Input: $\boldsymbol{\lambda}$ | 64 | | |
| Broadcast | *66* | | + coordinates |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *4* | Linear | RGB + Mask |

**Refinement Network**

| Type | Size/*Ch.* | Act. Func. | Comment |
|---|---|---|---|
| MLP | 64 | Linear | |
| Concat $[\boldsymbol{\lambda}, \nabla_{\boldsymbol{\lambda}}\mathcal{L}]$ | 256 | | |
| MLP | 128 | ELU | |
| Avg. Pool | 32 | | |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *32* | ELU | |
| Conv $5 \times 5$ | *32* | ELU | |
| Inputs | *17* | | |

## D. Additional Plots

**Decompositions** Figures 16–18 show additional decomposition samples on our datasets. Figure 19 shows a complete version of Figure 7, showing all individual masked reconstruction slots. Figures 20–22 show a comparison between the object reconstructions and the mask logits used for assigning decoded latents to pixels.

**Projections of Object Latents** Figures 24–26 demonstrate how object latents are clustered when projected onto the first two principal components of the latent distribution. Figures 27–29 show how object latents are clustered when projected onto a t-SNE (Maaten & Hinton, 2008) of the latent distribution.

**Traversals** Figures 30–32 show additional (randomly chosen) latent traversals for IODINE on CLEVR like on the right side of Figure 6.

**Input Ablations** Figures 33–40 give an overview of the impact of each of the inputs to the refinement network on the total loss, mean squared reconstruction error, KL divergence loss term, and the ARI segmentation performance (excluding the background pixels) on the CLEVR and Tetris datasets.

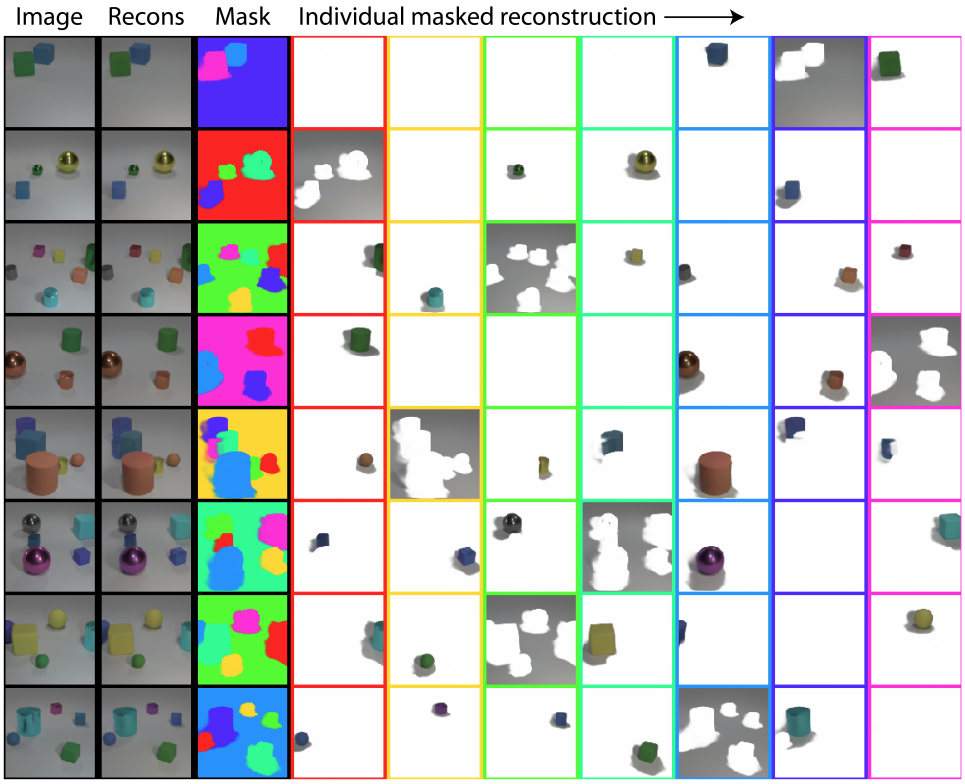


*Figure 16.* Additional segmentation and object reconstruction results on CLEVR6. Border colors are matched to the segmentation mask on the left.

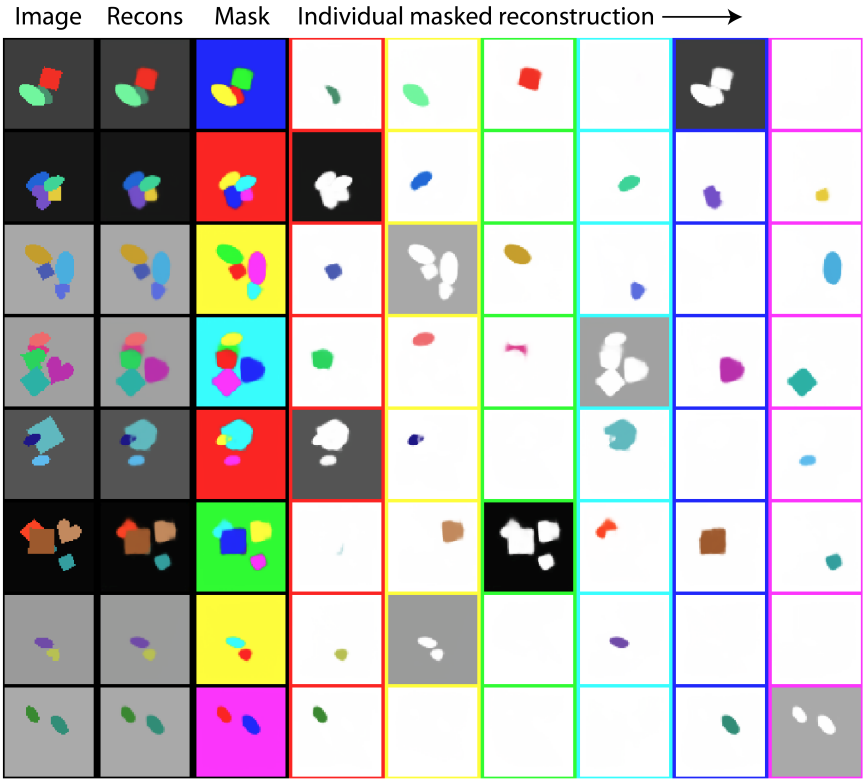


*Figure 17.* Additional segmentation and object reconstruction results on Multi-dSprites. Border colors are matched to the segmentation mask on the left.

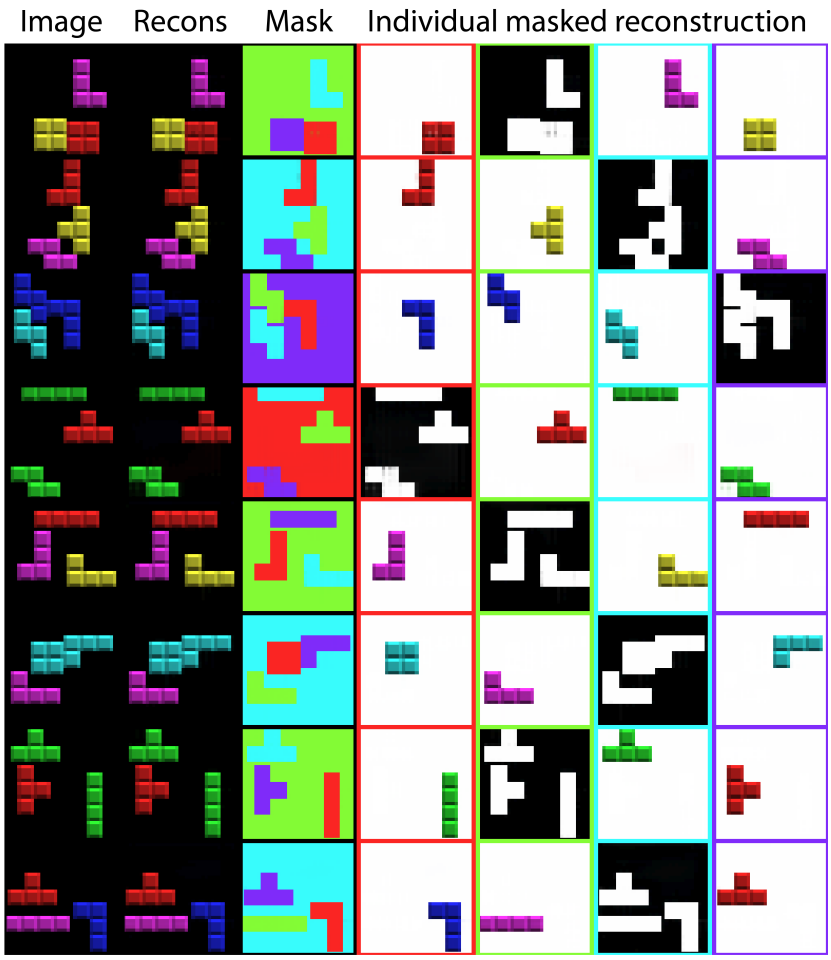


*Figure 18.* Additional segmentation and object reconstruction results on Tetris. Border colors are matched to the segmentation mask on the left.

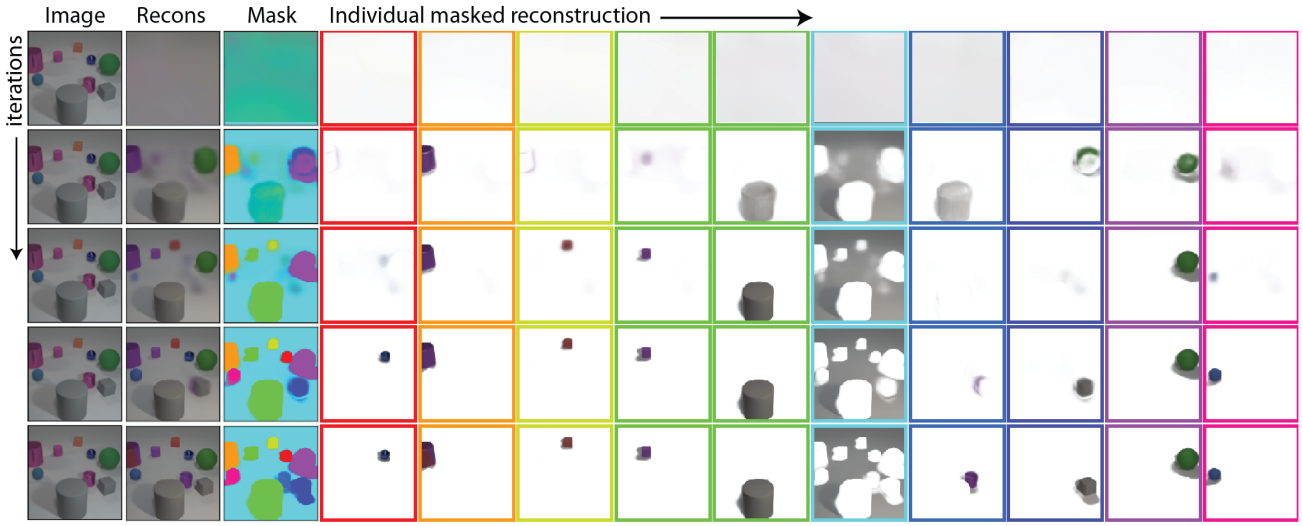


*Figure 19.* Full version of Figure 7, showcasing all slots.

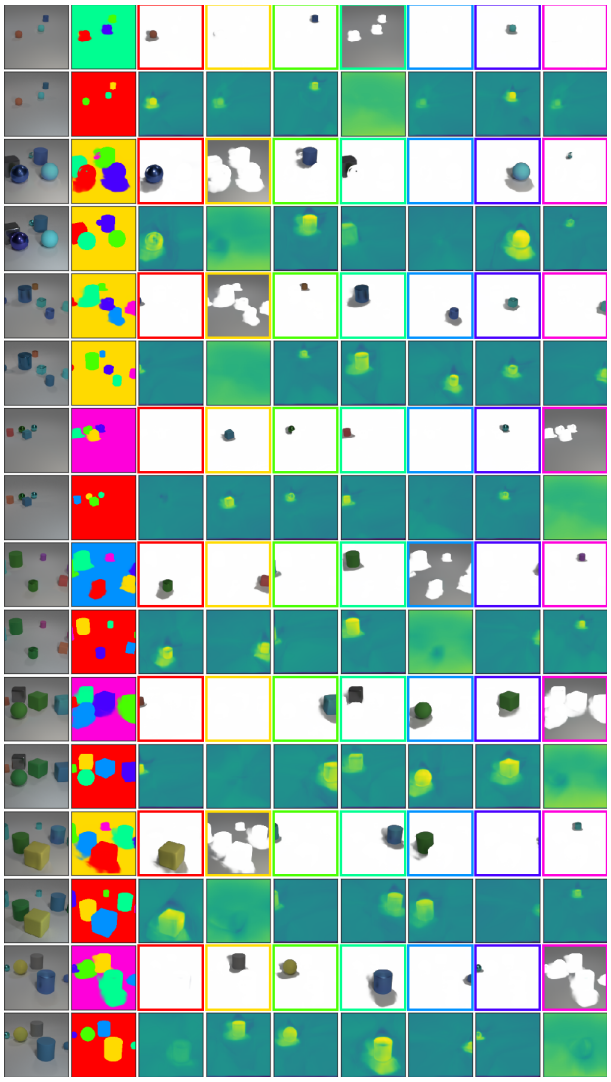

*Figure 20.* **CLEVR6** dataset. Odd rows: image and object masks as determined by the model. Even rows: first column is the input image, second one is the ground-truth masks and the following ones are mask logits produced by the model.

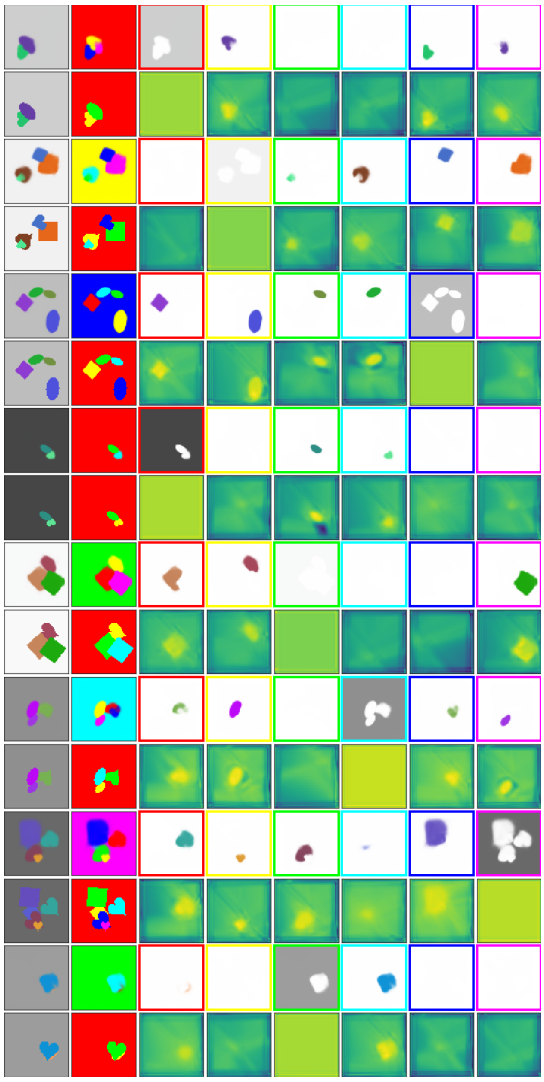

*Figure 21.* **Multi-dSprites** dataset. Odd rows: image and object masks as determined by the model. Even rows: first column is the input image, second one is the ground-truth masks and the following ones are mask logits produced by the model.

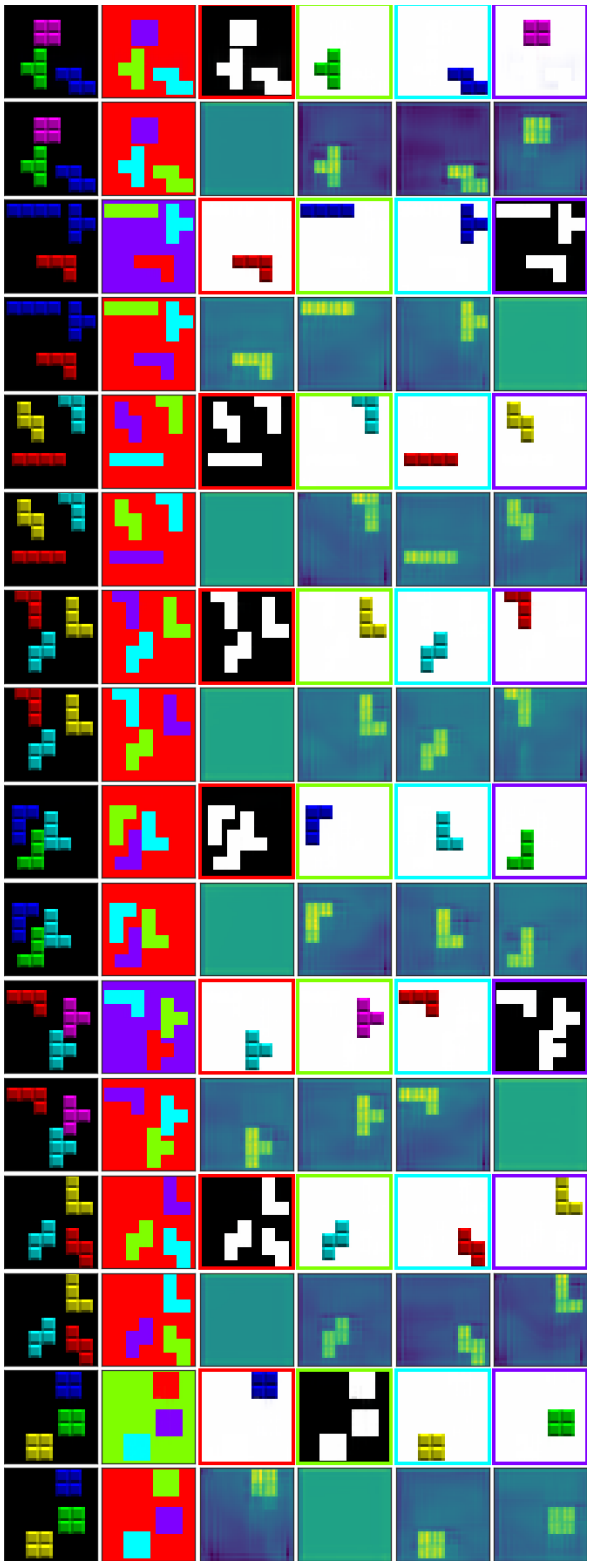

*Figure 22.* **Tetris** dataset. Odd rows: image and object masks as determined by the model. Even rows: first column is the input image, second one is the ground-truth masks and the following ones are mask logits produced by the model.

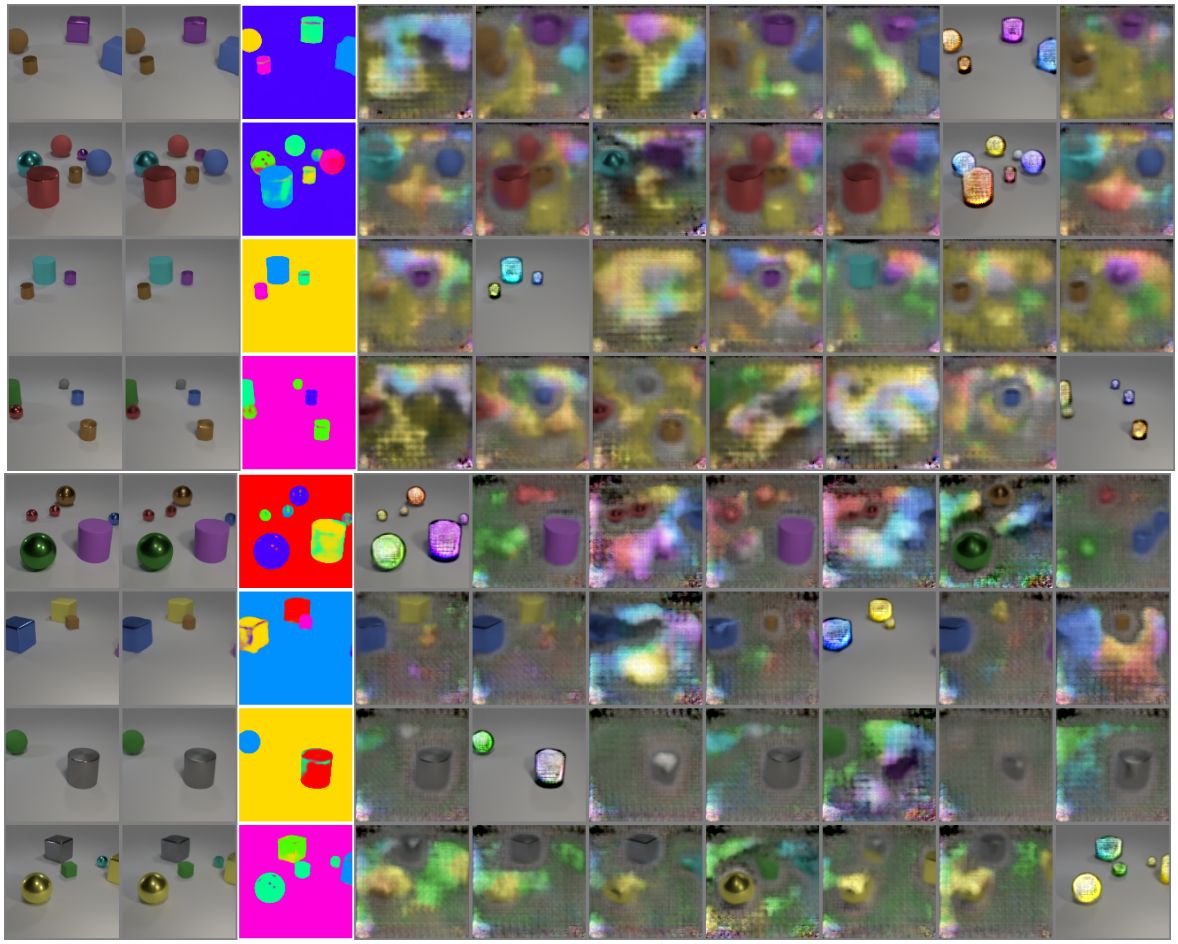

*Figure 23.* Segmentation and object reconstruction results on CLEVR6 using a deconvolution based decoder instead of the spatial broadcast decoder. Note that IODINE still cleanly segments objects from the background (now ignoring shadows), but specialization of the individual slots is much worse. Both, slots holding multiple objects, and objects replicated across multiple slots are much more frequent now. Slot reconstructions are also much less clean, containing much more noise and residue of other objects. (Note though, that in this figure we didn't mask the reconstructions as we have for Figure 16.)

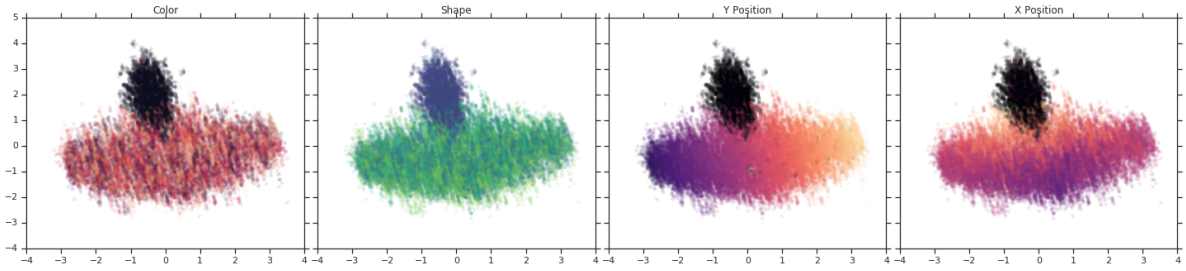


*Figure 24.* Projection on the first two principal components of the latent distribution for the **CLEVR6** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor.

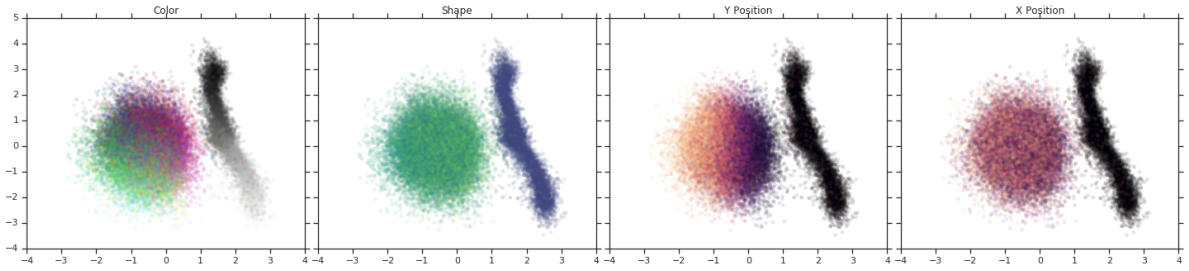


*Figure 25.* Projection on the first two principal components of the latent distribution for the **Multi-dSprites** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor.

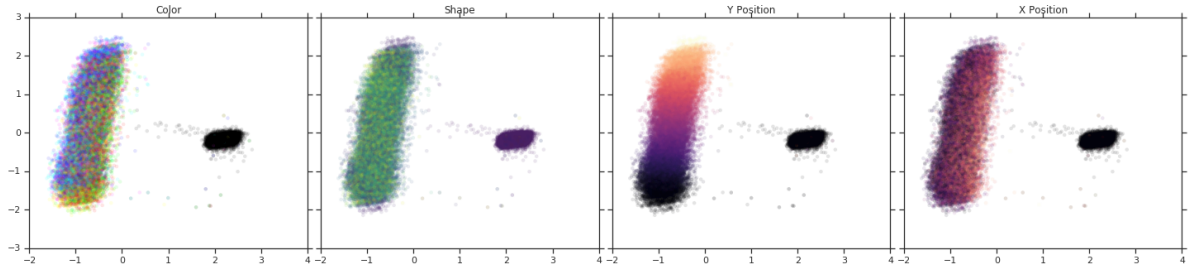


*Figure 26.* Projection on the first two principal components of the latent distribution for the **Tetris** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor.

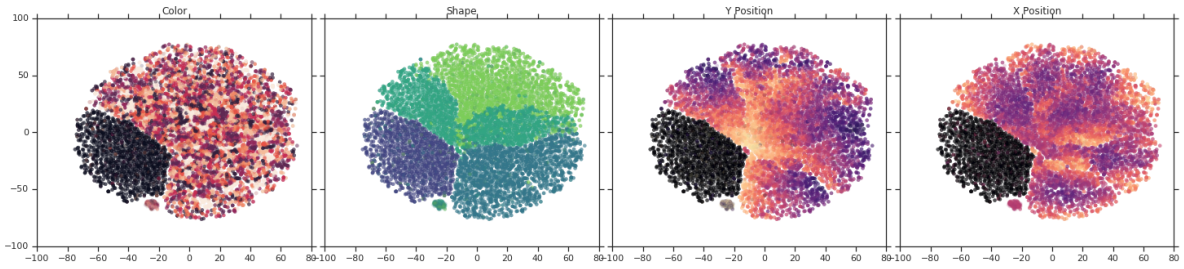


*Figure 27.* t-SNE of the latent distribution for the **CLEVR6** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor

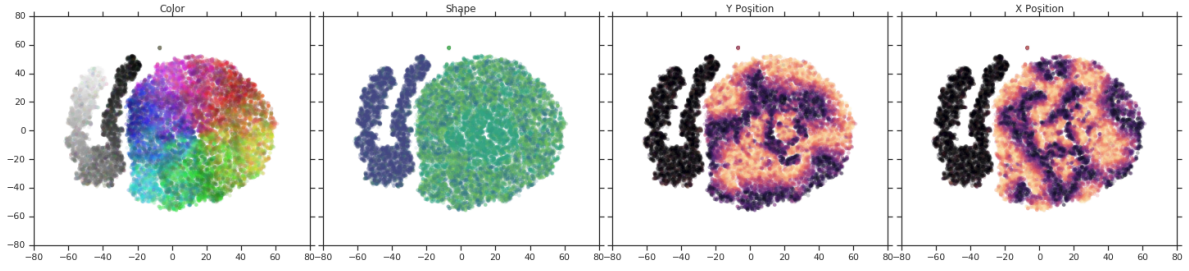


*Figure 28.* t-SNE of the latent distribution for the **Multi-dSprites** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor

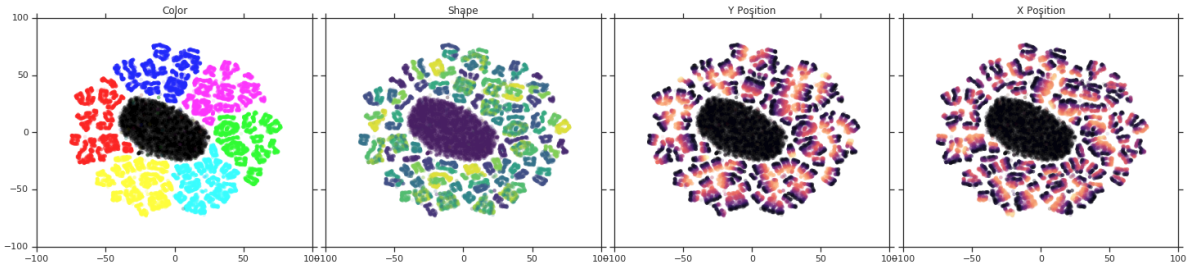


*Figure 29.* t-SNE of the latent distribution for the **Tetris** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor

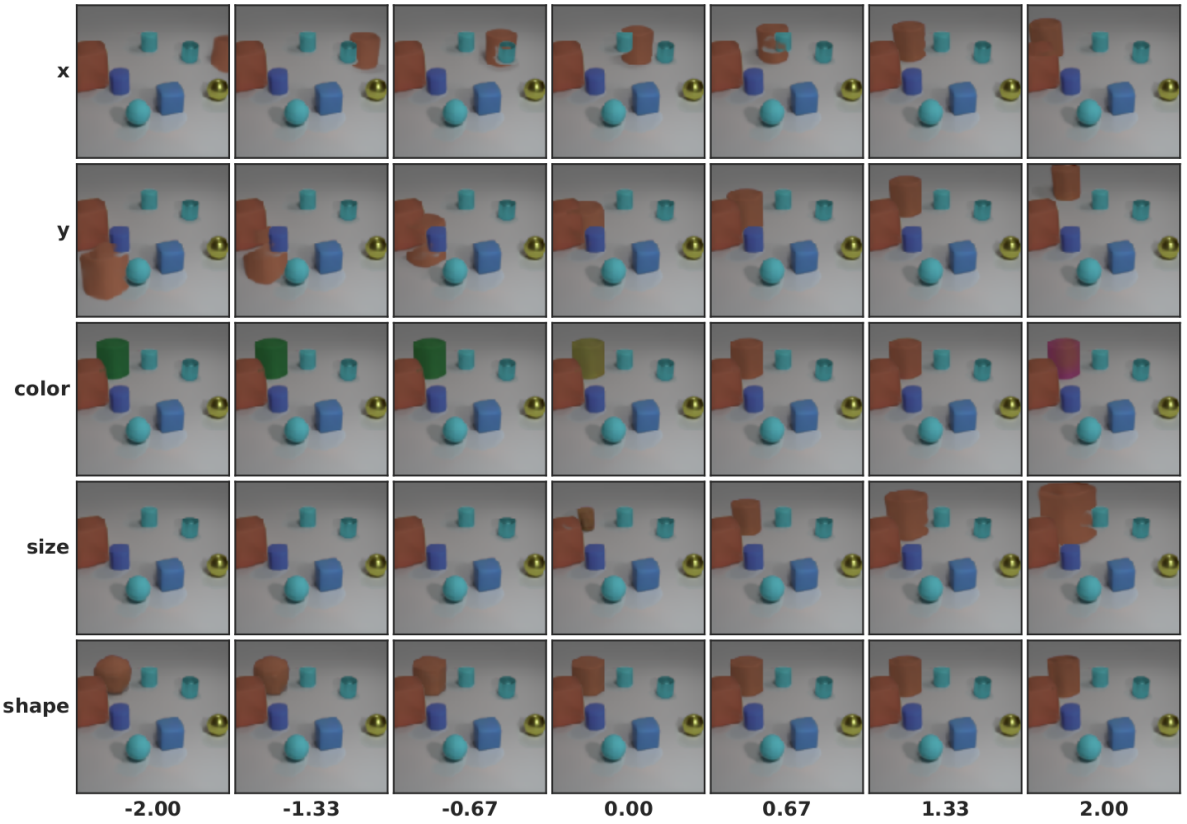


*Figure 30.* Latent traversal of IODINE on CLEVR (like right side of Figure 6), for a randomly chosen example and randomly chosen slot. Here the brown cylinder in the back is changing. Occlusion handling shows several flaws, that could be fixed by adjusting another latent (not shown) that encodes the depth ordering.

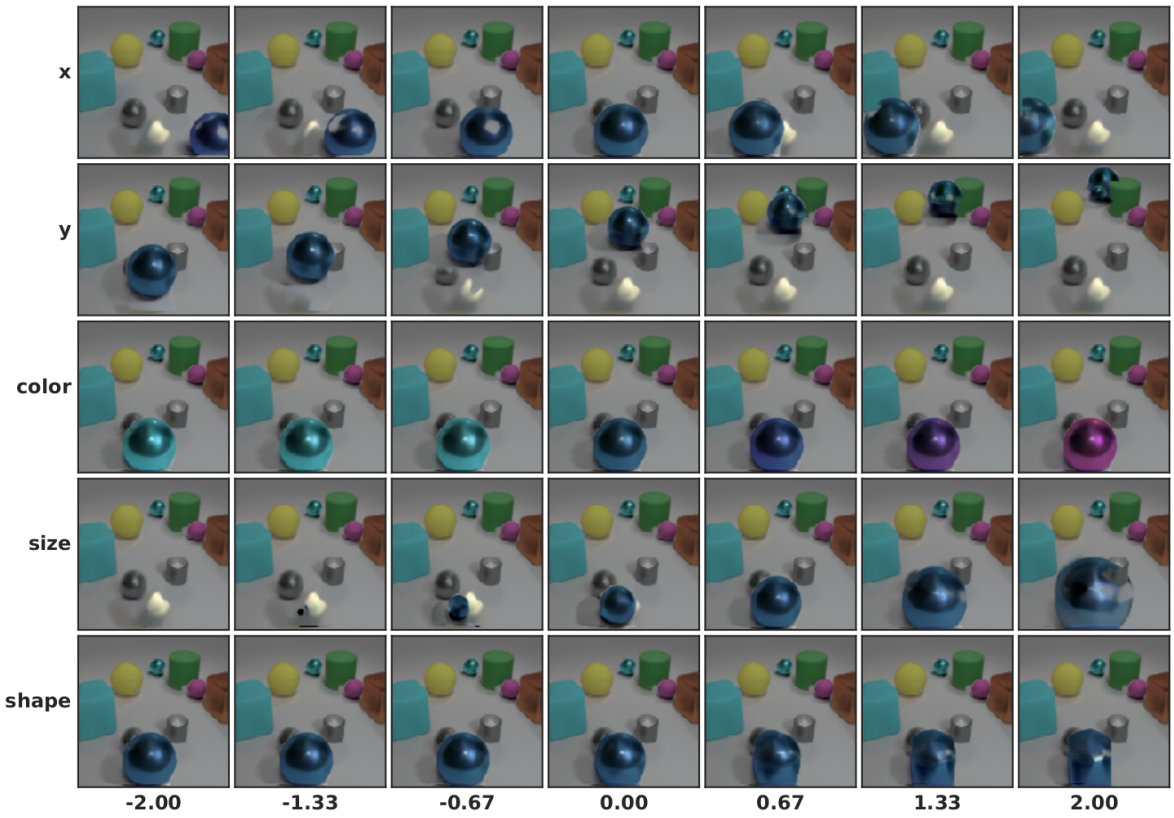


*Figure 31.* Latent traversal of IODINE on CLEVR (like right side of Figure 6), for a randomly chosen example and randomly chosen slot. Here the large blue sphere in the front is changing. Note that the background slot contains a bright spot behind the blue sphere that becomes visible when the sphere is moved away.


*Figure 32.* Latent traversal of IODINE on CLEVR (like right side of Figure 6), for a randomly chosen example and randomly chosen slot. Here the gray cylinder on the right is changing. Occlusion handling shows several flaws, that could be fixed by adjusting another latent (not shown) that encodes the depth ordering.

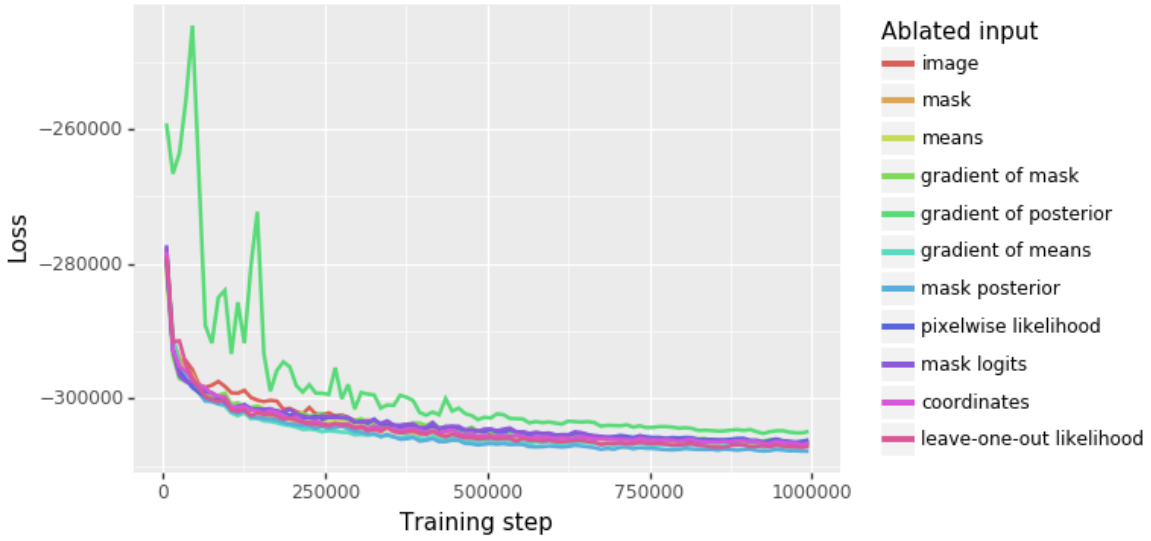


*Figure 33.* Ablation study for the model's total loss on CLEVR6. Each curve denotes the result of training the model without a particular input.

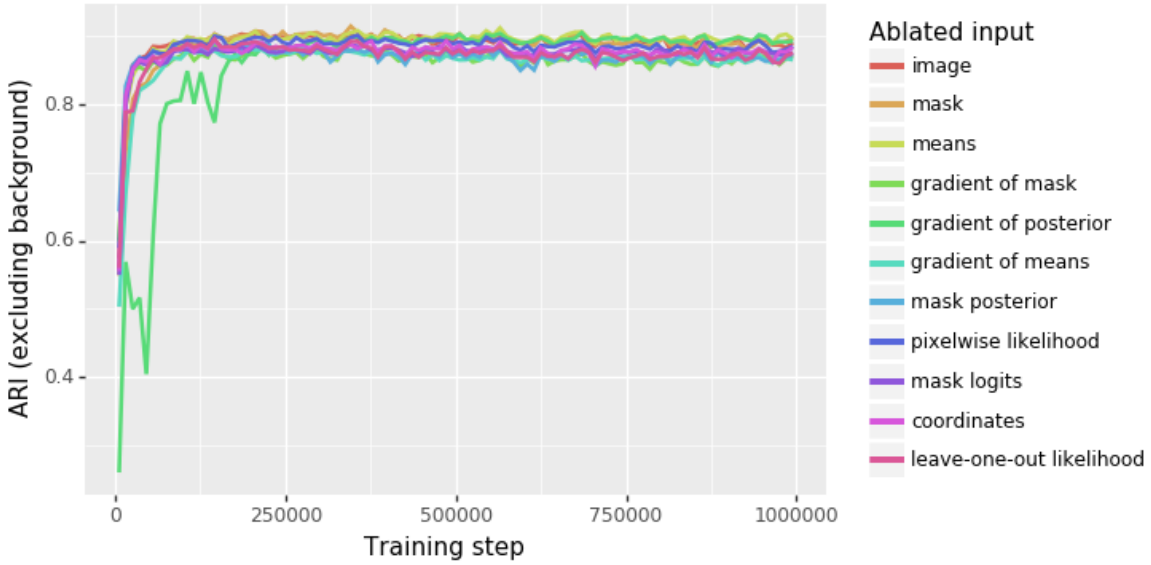


*Figure 34.* Ablation study for the model's segmentation performance in terms of ARI (excluding the background pixels) on CLEVR6. Each curve denotes the result of training the model without a particular input.

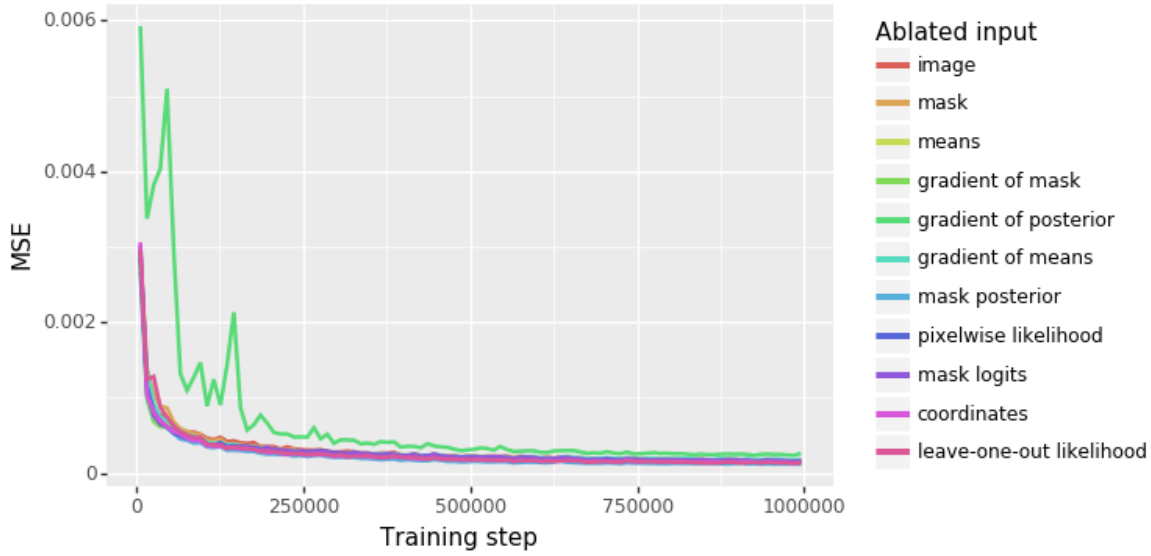


*Figure 35.* Ablation study for the model's reconstruction loss term on CLEVR6. Each curve denotes the result of training the model without a particular input. The y-axis shows the mean squared error between the target image and the output means (of the final iteration) as a proxy for the full reconstruction loss.

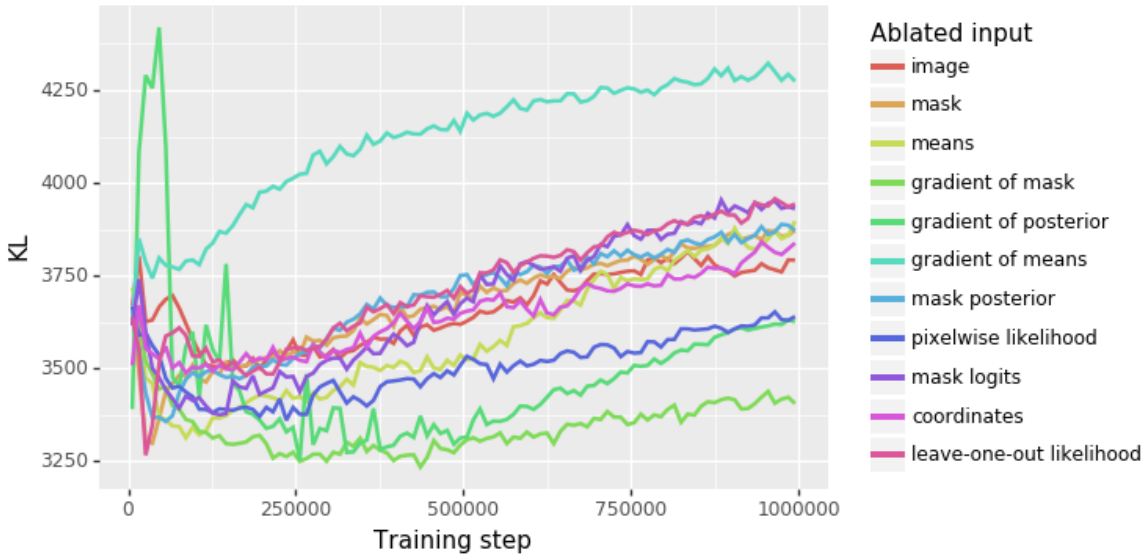


*Figure 36.* Ablation study for the model's KL divergence loss term on CLEVR6, summed over slots and iterations. Each curve denotes the result of training the model without a particular input.

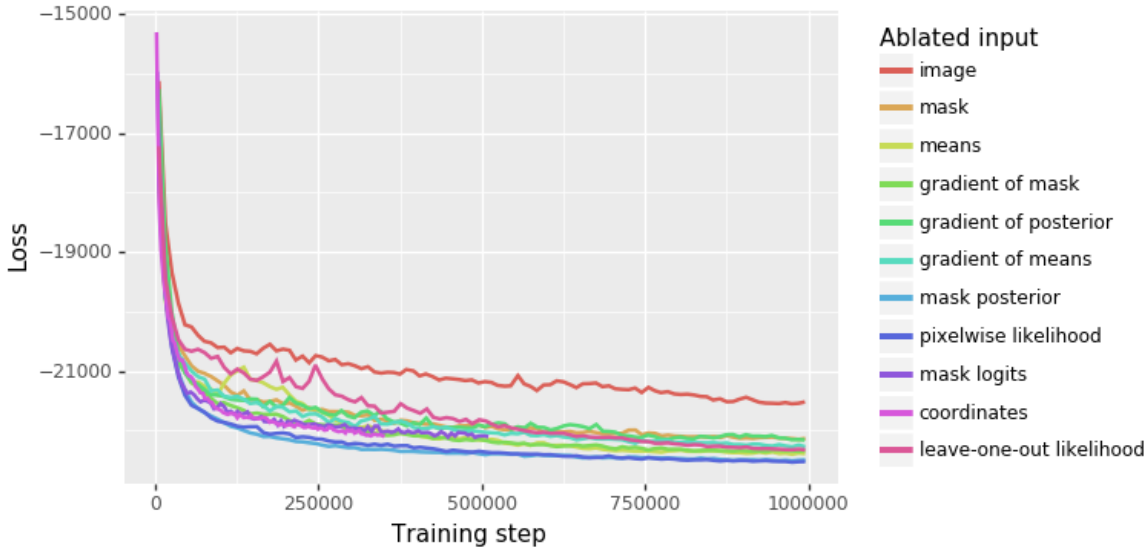


*Figure 37.* Ablation study for the model's total loss on Tetris. Each curve denotes the result of training the model without a particular input.

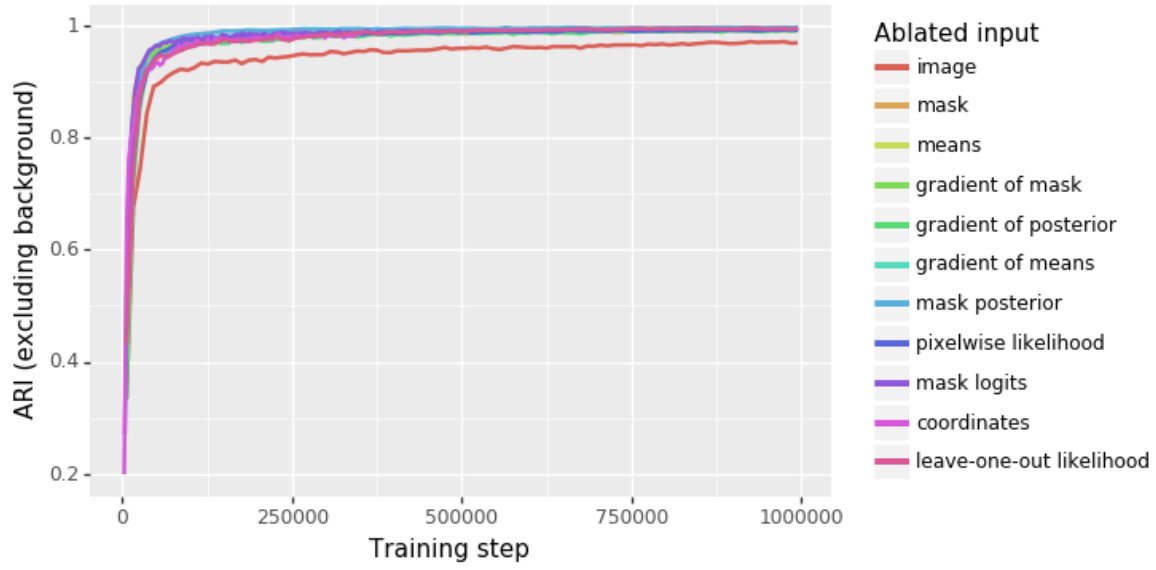


*Figure 38.* Ablation study for the model's segmentation performance in terms of ARI (excluding the background pixels) on Tetris. Each curve denotes the result of training the model without a particular input.

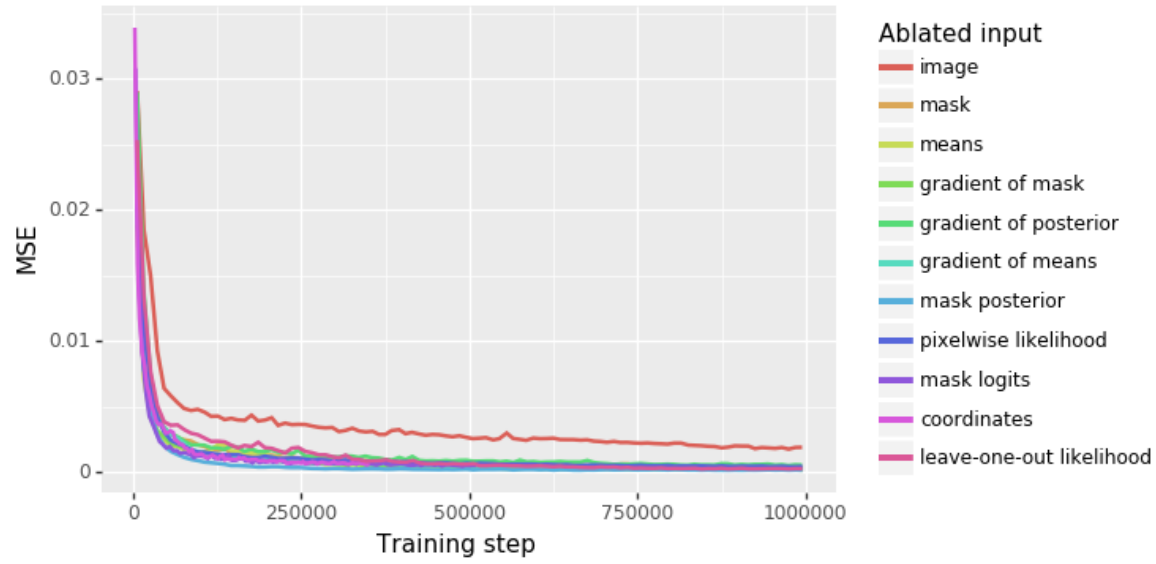


*Figure 39.* Ablation study for the model's reconstruction loss term on Tetris. Each curve denotes the result of training the model without a particular input. The y-axis shows the mean squared error between the target image and the output means (of the final iteration) as a proxy for the full reconstruction loss.

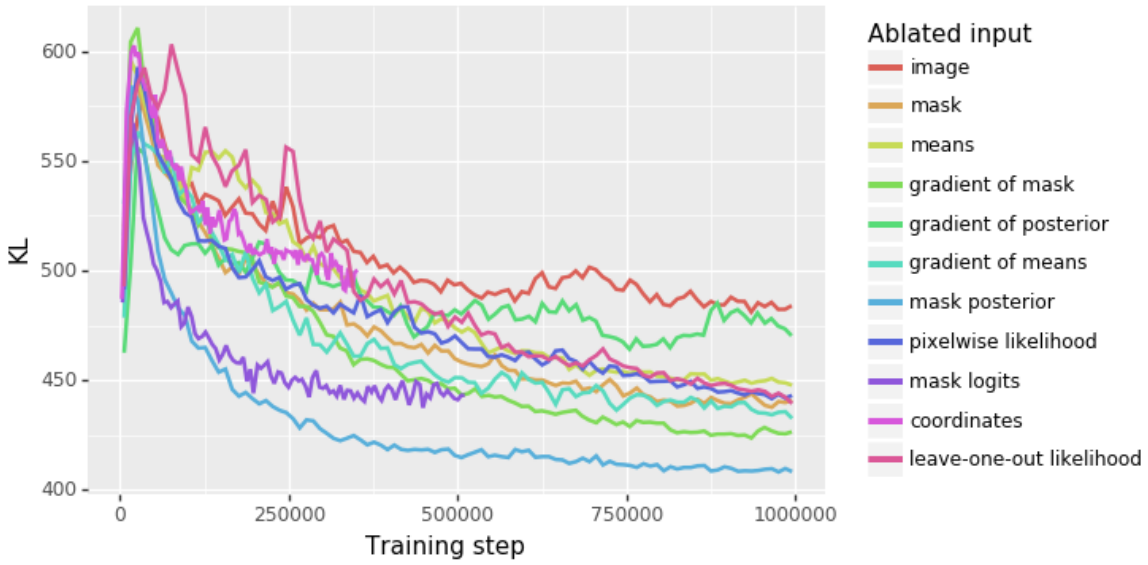


*Figure 40.* Ablation study for the model's KL divergence loss term on Tetris, summed over slots and iterations. Each curve denotes the result of training the model without a particular input.