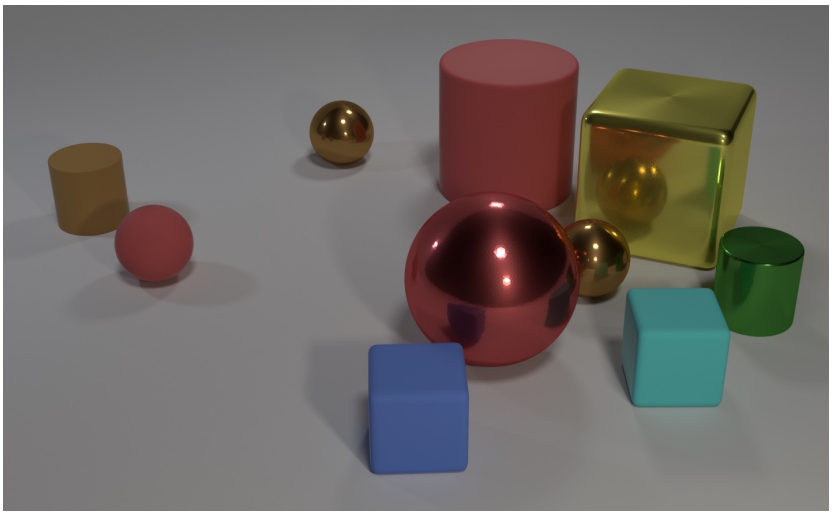
Unsupervised Object Discovery: Detection and Dynamics prediction

Marine Moutarlier
Robotics, EPFL, Lausanne, Switzerland
marine.moutarlier@epfl.ch

December 14, 2025



Abstract

Understanding how objects in visual scenes are detected and their movements predicted is essential in robotics for planning and decision-making, allowing systems to anticipate future states and act accordingly. Recent advances in unsupervised learning have enabled models to detect objects and their dynamics directly from RGB images or videos, without explicit annotations. The object dynamic prediction relies on the object decomposition. Slot Attention mechanisms have been used as a common tool for decomposing visual scenes into objects by allocating attention "slots" to similar features of the input. However, challenges persist in refining these slots for accurate object extraction particularly in an unsupervised manner where the model has no information on them. Such tasks can be limited by inaccurate slot assignments and insufficient generalization across varied object scenes. Here, we show that incorporating an invariant version of Slot Attention and reshaping the loss of the model might improve unsupervised object discovery and motion prediction. By enforcing robustness to object variations such as translation, scaling or rotation, it might be possible to improve the object decomposition. Our improvements show the way for more precise scene understanding in unsupervised settings. With improved object decomposition, the dynamics of objects can be better captured and analyzed. With continued advancements, such systems could transform applications requiring autonomous understanding of complex environments, such as robotics.

Contents

1 Introduction and motivation

1.1 General problem

Unsupervised scene decomposition and prediction are important in robotics particularly in areas like planning, decision-making, and autonomous navigation. For robots to operate in environments, understanding the scene around them is a requirement. Object prediction is important to allow systems to anticipate future states and act accordingly. It enables robots to anticipate obstacles, leading to smoother navigation. This understanding of dynamic environments enhances autonomy, making robots more adaptable.

Here we will dive more into approaches that leverage object recognition in RGB image or video datasets. If the dataset consists of videos, the first step is to extract individual frames. From these images, features are extracted using CNN and attention mechanisms computes similarity in those features, to identify the objects. These similar features corresponding to objects are then utilized to compute dynamics within the scene, including trajectories, interactions, and temporal consistency.

Multi-Object Network (MONet) [Burgess et al., 2019] is one of the early paper on unsupervised scene decomposition. This model trains a variational autoencoder (VAE) [Burgess et al., 2018] end-to-end, alongside a recurrent attention network, to generate attention masks for specific regions of images and reconstruct those regions. A VAE includes an encoder with a downsampling CNN followed by a multilayer perceptron (MLP), and a decoder that begins with an MLP and is followed by an upsampling CNN. However VAE strugles with spatial discontinuities. It has difficulties in reconstructing continuous spatial patterns, since the network doesn't have explicit spatial knowledge. It also has learning difficulties. A deconvolutional network has no spatial awareness and lacks specific information about where an object is located. As a result, it's difficult to ask it to render an object in a particular position.

The paper we will focus on for this work, Slot Former, [Wu et al., 2023] presents an enhancement of the VAE. It improves scene decomposition by replacing the VAE decoder with a Spatial Broadcast Decoder [Watters et al., 2019], incorporates attention mechanisms, and introduces an additional model for predicting object dynamics, which was absent in MONet [Burgess et al., 2019].

1.2 Background : Slot Former

Slot Former [Wu et al., 2023] is a Transformer-based model that learns to predict object dynamics from video clips. It breaks scenes into objects, and predicts future object dynamics. The input is a video frame, and the output is the next frame predicted, which is feedback to the Slot Former as a new input. It works recursively and can generate a video predicting the dynamics of the objects based on the first frame. It has two main models:

- Slot Attention for Videos (SAVi) [Kipf et al., 2022] which is the object-centric model responsible for object decomposition. SAVi is an adapted version of the Slot Attention model presented in [Locatello et al., 2020], made for video datasets. It uses Slot Attention to update the set of slot representations based on the input. The slot representations are similar features from the input image and are considered the "objects" See. 2.1. The input is an image and the output is slots supposedly representing the objects from the image. It is represented in green on Fig.1.

- Slot Former (SF) which is the model responsible for the object dynamic prediction. From the slots, a Transformer model processes these representations to predict the arrangement and state of the objects in the next frame. To make sure the Transformer understands the order of the frames, it adds positional encoding without changing how the slots are organized. This helps the model maintain consistency over time and makes better predictions. It learns by analyzing the positions of objects in each frame and how they change over a sequence of frames. It then creates a large vector in the latent space to represent this information. Finally, it uses a decoder to convert the predicted object representations back into a full image and makes sure that that the generated image has the characteristics of the objects. This is done through a VQ-GAN [Esser et al., 2021, Razavi et al., 2019]. It uses a generator to create new images from random noise based on these tokens, while a discriminator evaluates the realism of the generated images. During training, both components improve together, allowing the generator to create more realistic images. This is the part above the object-centric model in Fig.1.

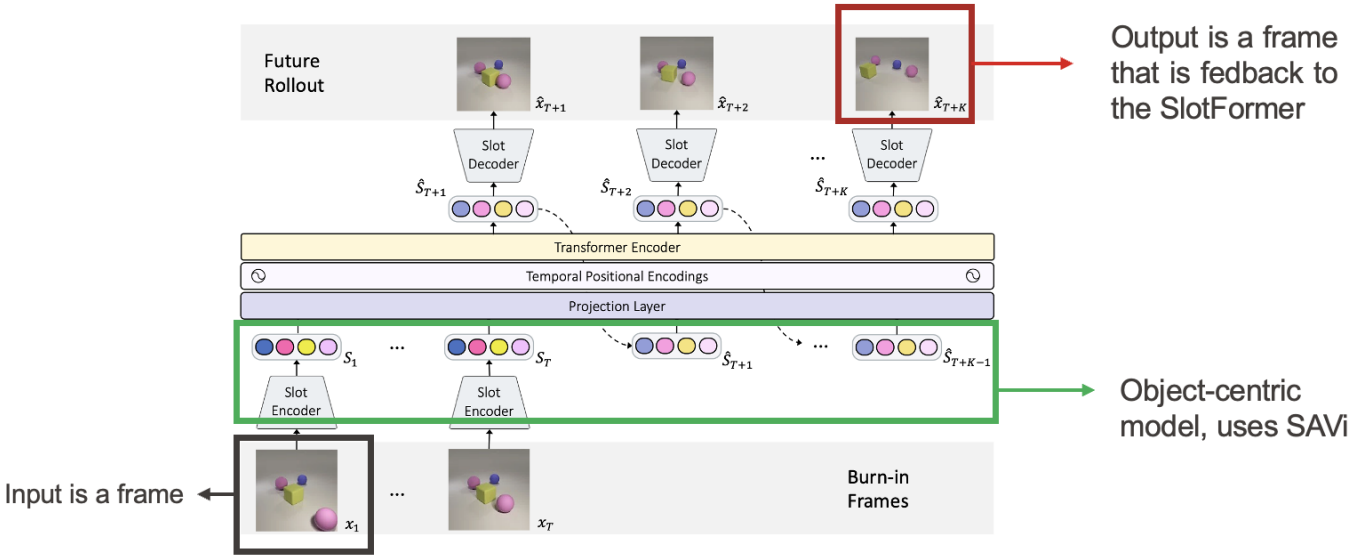


Figure 1: Slot Former architecture

1.3 Motivation: Inaccurate representation in the Slot Former

However, when trained locally, the Slot Former prediction results are not always optimal, as shown in Fig.3. In this figure, we can see that some of the object reconstruction is not existent. The yellow and the pink cubes are not a part of the predicted frame. Since the prediction relies entirely on the slot predictions from the SAVi model [Kipf et al., 2022], it is essential for these slots to accurately represent the objects in the frame. A poor object decomposition will lead to inaccurate predictions. The goal of this work is to investigate the efficiency of the Slot Attention decomposition and propose a new approach for improvement.

Figure 2: Original dataset frame

Figure 3: Frame predicted by Slot Former

2 Research and theorical side

2.1 Savi and its Slot Attention architecture

In this section, we present the architecture of the object-centric model used in the Slot Former computation.

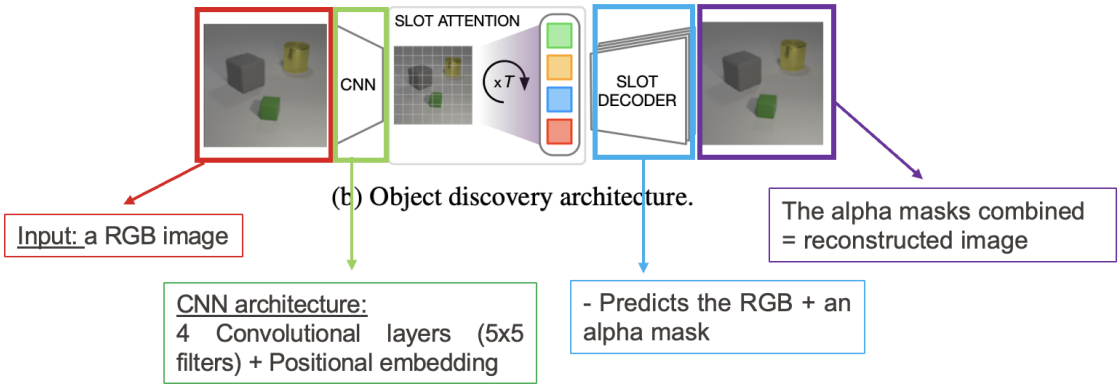


Figure 4: object-centric module based on Slot Attention

Slot Former relies heavily on the object-centric model, if the slot decomposition is bad, then the recontruction will also be bad. The object-centric model is SAVi, and is based on the Slot Attention module.[Locatello et al., 2020].

Slot Attention is a simple architectural component that can be placed on top of a CNN encoder to extract object representations from an image. It serves as the object extractor, not the entire object-centric model. The entire object-centric model is structured as follows:

- The object-centric model takes a RGB image as an input. This image then goes through a Conventional Neural Network (CNN) for image features extraction.
- The CNN has 4 layers and a 5x5 convolutional filter. We add a 4D position tensor that contains the distance from each pixel to each of the 4 borders of the image. Then flatten the features of the image augmented with the positional embedding. This allows Slot Attention to recognize where each feature is located in the image. Features near each other might belong to the same object, and positional embeddings help the model make that distinction. This is explained in the Fig.5.

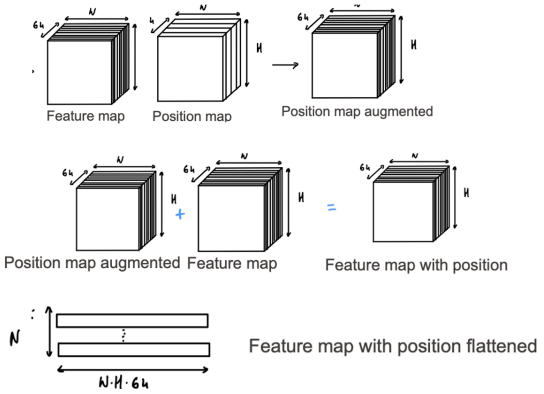


Figure 5: Encoder of the Slot Attention

- The Slot Attention module takes this set of augmented input features and maps them to a smaller set of output vectors called "slots," which can represent different objects in the input. The number of slots corresponds to number of objects plus the background. Slots are randomly initialized following a gaussian distribution accross pixels. The attention score is computed, which assigns the same weights to similar looking features, then are summed up with the previous initialization. This allows to group together, over couple iterations, similar features pixels. The number of iterations in the paper [Locatello et al., 2020] is set to 3. This will allow to consider

as the same slot, pixels with similar features and explain the whole image. However, there is no guarantee that they will really represent objects, as it groups similar features. Similar features might belong to different objects, and the Slot Attention module does not know that, as it has no idea of where the objects are in the image. The same applies to the background: sometimes it is fully recognized, while at other times, parts of the background are distributed across different slots. We consider the number of the slots to be K in the Fig.6. In the paper [Locatello et al., 2020], they decided to always assign the slot who has the most number of pixels from the image to the background. This is not always efficient as some of the slots representing object might have a bit of the background entangled to them. This is explained in the Fig.6.

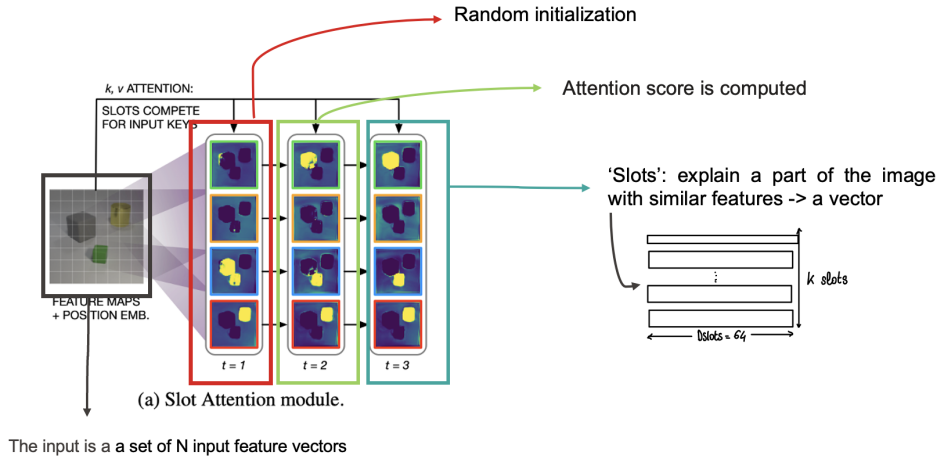


Figure 6: Attention module for the slots

- Once the slots are identified, we have K number of vectors for the slots, each of them has a Dslots dimension. They need to be decoded to be able to visualize the object decomposition. This is done individually with a spatial broadcast decoder [Watters et al., 2019]. Slot representations are broadcasted onto a 2D grid (per slot) and augmented with position embeddings. The latent vector is projected into a 2D grid and augmented by a positional embedding. The positional embedding is necessary for the reconstruction to replace in a logical spatial way the objects decomposed. Then it is passed through a deconvolutionnal CNN that decode an alpha mask and RGB channels for each slots. For the reconstruction, the alpha masks are normalized across slots using a Softmax and use them as mixture weights to combine the individual reconstructions into a single RGB image. This is explained in the Fig.7. The output is a recomposed image of the slots supposedly corresponding to objects.

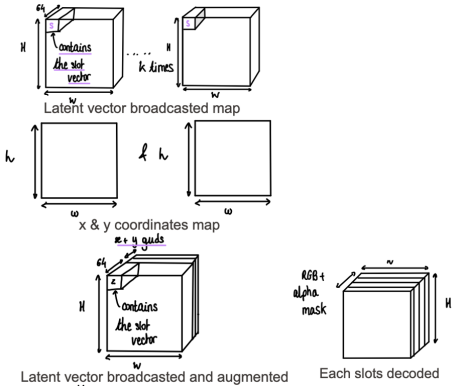


Figure 7: Decoder of the Slot Attention

2.2 Slot Attention and its limitations

Slot Attention has its limitations. Based on our local testing using the code provided in [Locatello et al., 2020], we analyzed the performance on the Tetrominoes dataset [Bozkurt et al., 2019] and the CLEVR dataset [Yi et al., 2019], and observed the following:

For Tetrominoes, as seen in Fig. 10, the object decomposition is not good. The object predicted masks do not represent the original tetris objects. The background is entangled in all of the representation. However, the green representation in Fig. 10 seems to start to resemble the red left object in Fig. 9. This can happen if the objects in the images are ambiguous or have similar visual properties. This is the case for this dataset. Every object is similar, (tetris like objects) and has dark features like the black background, making it hard for the model to distinguish them.

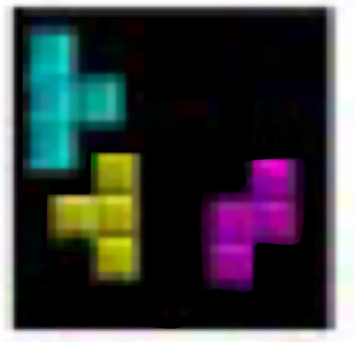


Figure 8: Original image from Tetraminoes

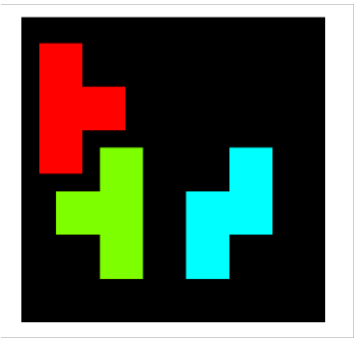


Figure 9: Ground truth mask

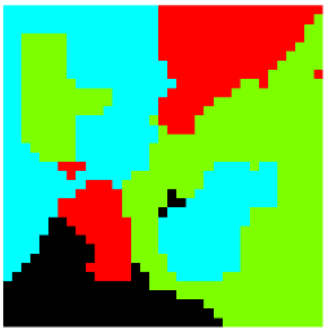


Figure 10: Predicted SA mask

The analysis for the CLEVR dataset is similar, though it shows better results. Distinct objects can be identified, and the background appears clearer. However, the shapes of the objects are still unrecognizable, with some objects either not being detected or mixed with the background, as observed with the orange cube in Fig. 12. Nevertheless, there is a noticeable improvement compared to the Tetraminoes dataset. The background is more distinguishable, and while some objects still include bits of the background, this way less the case than with Tetraminoes.

This improvement could be attributed to the differences in the dataset. The CLEVR dataset contains simple object, and it has better lighting and a higher contrast between objects and the background. Objects are in bright colors, where the background is always white. These factors contribute to the model's ability to better recognize object and achieve better decomposition.

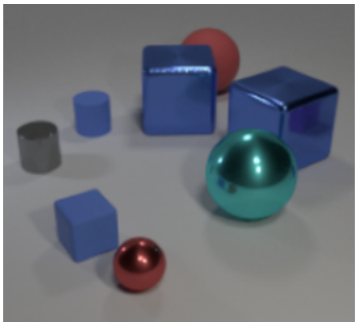


Figure 11: Original image from CLEVR

Figure 12: Ground truth mask

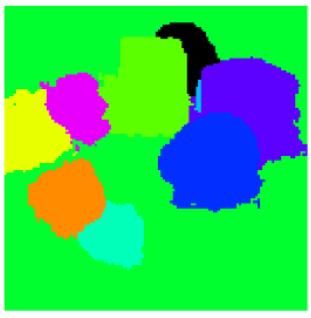


Figure 13: Predicted SA mask

These results show that Slot Attention has limitations. The background of a scene receives no special treatment as all slots use the same representation. Out of 5 tested seeds, only one learned the solution of placing the background into a separate slot (which is the one we visualize). The typical solution that a Slot Attention-based model finds is to distribute the background equally over all slots [Locatello et al., 2020]. On top of that, slots might not always represent objects.

Despite the claims in the paper, our testing revealed additional limitations. Specifically, the model has issues with consistent object-centric representations, particularly in datasets with visually similar objects. In some cases, slots did not correspond to distinct objects, as it is the case in Fig.10. These are extra issues that were no addressed in the original paper. It suggests that Slot Attention performance may vary depending on the dataset. This shows that there is space for improvement, to ensure that slots consistently represent individual objects.

2.3 Invariant Slot-Attention

2.3.1 Architecture of Invariant Slot Attention

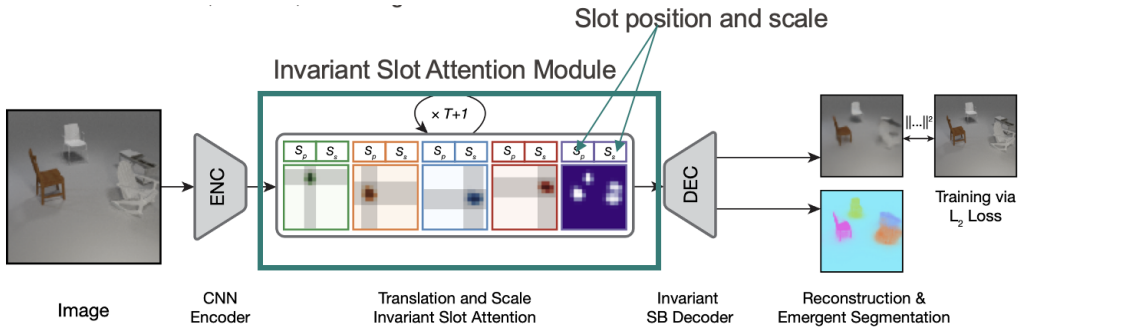


Figure 14: Architecture of Invariant Slot Attention

Invariant Slot Attention [Biza et al., 2023] is an alternative to Slot Attention module: It improves the ability to handle transformations like translation, rotation, or scaling, making it invariant to them. The model introduces equivariance principles to make sure that the representations are invariant to input transformations. This is will help in scene were objects might change their positions, orientations, or sizes.

A model is invariant to a transformation if the output does not change when this specified transformation is applied. A model is equivariant to a transformation if the output changes in a predictable way when the input is transformed.

The Invariant model has the same architecture than Slot Attention [Locatello et al., 2020], but with slot position and scaling added, see Fig.14. The attention weight calculation now depends on slot position and scaling. This makes the model more accurate in detecting the objects.

Translation and scaling invariance

To make the model invariant to translation and scaling, 2D slot positions (Sp) and scales (Ss) are defined. These can be randomly initialized or learned. The pair (Sp, Ss) is a reference frame for each slot. We then use this reference frame to adjust the input position encoding for each slot separately. By translating and scaling the positions using the inverse of the slot positions and scales, the estimated object pose is reversed. This writes as:

$$\text{Rel grid} = \frac{\text{Abs grid} - \mathbf{S}_p}{\mathbf{S}_s} \tag{1}$$

with: Abs Grid the grid of features as computed in the Slot Attention encoder.

Rotation invariance

The rotation is more difficult. The orientation is estimated by finding the axis with the highest variation in the attention mask. A rotation matrix S_r is created. The rotations are limited to $\pi/4$.

Afterward, we compute the relative position encoding by applying inverse translation, rotation, and scaling.

$$\text{Rel grid} = \frac{(S_r^k)^{-1}(\text{abs grid} - \mathbf{S}_p^k)}{\mathbf{S}_s^k} \tag{2}$$

2.3.2 Comparison of Slot Attention and Invariant Slot Attention

We visualize the comparison between Slot Attention and Invariant Slot Attention for different cases.

- Invariant Slot Attention for translation (ISA-T)
- Invariant Slot Attention for translation and scaling (ISA-TS)
- Invariant Slot Attention for translation, scaling and rotation (ISA-TSR)

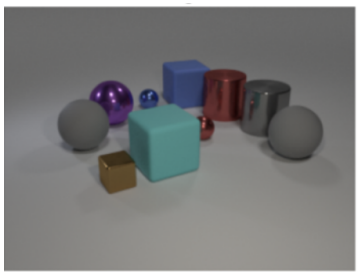


Figure 15: Original image

Figure 16: Ground truth mask

Figure 17: Predicted SA mask

Figure 18: Predicted ISA-T mask

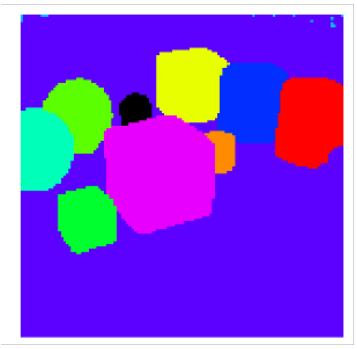


Figure 19: Predicted ISA-TS mask

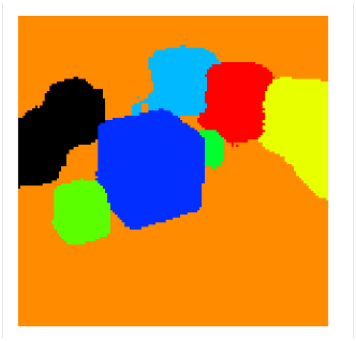


Figure 20: Predicted ISA-TSR mask

In the Slot Attention mask decomposition, the small black sphere at the back is not detected, as shown in Fig. 17. The shapes are also very blurred, and it is difficult to distinguish them. In contrast, the invariant versions show much better results. For ISA-T and ISA-TS, all objects are identified. While the objects appear blurry in ISA-T (Fig. 18), they are very recognizable in ISA-T (Fig. 19), which has almost perfect decomposition. However, ISA-TSR does not achieve as high results, as some objects are merged into one, such as the two spheres on the left in Fig. 20.

The improvement in the invariance is due to the fact that the model has learned about translation, scaling and rotation changes.

In the CLEVR dataset, the most important transformation to learn is how to handle translation and scaling in object. Spheres and cylinders are already invariant to rotation. Cubes won't rotate, they will always be on one side and in conclusion look the same. Rotation invariance is not necessary here. This could explain the less impressive results, as the model's rotation invariance could introduce some confusion as seen in Fig.20. However for translation and scaling, it is useful. For example, a sphere, cube, or cylinder may be placed at different locations or resized in the image. By making sure

that the model is invariant to those transformations, it always for better results in the decomposition. This is observed with Fig. 18 and 19 where the decomposition is much better than the baseline.

Learning those transformations makes the model better. It distinguishes better the objects even when they are close. On the other hand, Slot Attention does not performance as good, leading to blurred representations.

3 Methodology

3.1 Key Idea: Implemetation of Invariant Slot Attention in Slot Former and investigation

The Invariant Slot Attention mechanism was implemented in the Slot Former architecture and tested.

The prelimary results of the implementation can be found in section 4. These results show entanglement, the objects are not fully separated by the attention mechanism. To understand why this is, we performed a t-SNE computation to visualize and analyze the distribution of the learned representations. This is helpful to figure out what the issue is and to try to solve it. The loss was shaped according to better guide the model during training. This can help the model to learn more effectively the object representations, and to try to solve the entanglement issue.

3.2 t-SNE

To address the issue of entanglement, where objects with similar features are clustered together even though they should belong to different slots, we used t-SNE visualizations. T-distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two dimensions.

We computed the t-SNE throughout the training on the CNN features as well as on the slot features. T-SNE preserves local relationships between points in the data. In Slot Attention, objects that are more similar to each other in terms of their slot representations will be placed closer together in the lower-dimensional space. This makes it easy to visualize if close points in t-SNE are assigned to different slots. This can help evaluate the quality of object decomposition and whether Slot Attention is successfully learning meaningful object representations. If close points in t-SNE are assigned to different slots then the model struggles to distinguish different objects and the object decomposition is poor.

This revealed that the embeddings of such objects were poorly separated, having close points belonging to different slots and leading to classification errors.

3.2.1 Pseudocode for t-SNE computation on CNN and Slot Features

Algorithm 1 Compute t-SNE on CNN Features

1: Concatenate all CNN features: `all_features = torch.stack(cnn_features)`
2: Transpose features: `all_features_transposed = all_features.t()`
3: Flatten into 2D: `all_features_flat = all_features_transposed.view(...).numpy()`
4: Get number of samples + features: `num_samples, num_features = all_features_flat.shape`
5: Set perplexity: `perplexity = min(30, num_samples - 1)`
6: Perform t-SNE: `tsne_result = TSNE(...).fit_transform(all_features_flat)`
7: Create scatter plot using `tsne_result`

It begins by concatenating individual feature tensors, that are the output from the CNN into one tensor and transposing to have the good dimension for t-SNE. The features are then flattened into a 2D array, with each row representing a sample and each column representing a feature. The number of samples and features is determined, and the perplexity value is set based on the number of samples to optimize. After performing t-SNE on the flattened features, the resulting lower-dimensional data is visualized in a scatter plot. This was done every 50 samples, so every each video.

The scikit-learn implementation of t-SNE was used here.

They were computed for the entire batch size of 32, for a middle frame of the first evaluation video. The 2D array with shape [B * T *total_features, feature_size], where:

- B is the batch size, here 32
- T is the number of frames, here 1
- total_features is the number of features, here 4096
- feature_size is the dimensionality of each feature embedding, here 128

So the matrix for t-SNE has B * T * total_features points (rows), each represented by a vector of size feature_size (columns).

Algorithm 2 Compute t-SNE on Slot Features

```
Concatenate all slot features: all_slots = torch.cat(slots_features)
Flatten into 2D: all_slots_flat = all_slots.view(...).numpy()
Get number of samples: num_samples = all_slots_flat.shape[0]
Set perplexity: perplexity = min(30, num_samples - 1)
Perform t-SNE: tsne_result = TSNE(...).fit_transform(all_slots_flat)
Generate colors for slots
for slot i in range(num_slots) do
    Extract indices for slot i: slot_indices
    Scatter plot t-SNE result for slot i with color i
end for
```

It begins by concatenating the individual slot features, that are the output from the Slot Attention mechanism, into one tensor and flattening it into a 2D array. Each row in represents a sample, and each column corresponds to a feature. The number of samples is determined, and the perplexity value is set based on the number of samples to optimize. After performing t-SNE on the flattened features, the resulting lower-dimensional data is visualized in a scatter plot. The scikit-learn implementation of t-SNE was used here. This process is repeated every 50 samples (i.e., for each video). They were computed for the entire batch size of 32, for a middle frame of the first evaluation video. The a 2D array with shape [B * T * num_slots, slot_size], where each row represents a slot embedding in the high-dimensional space, where:

- B is the batch size, here 32
- T is the number of frames, here 1
- num_slots is the number of slots, here 6
- slot_size is the dimensionality of each slot embedding, here 128

So the matrix for t-SNE has B * T * num_slots points (rows), each represented by a vector of size slot_size (columns).

The visualization step involves generating unique colors for each slot, and for each slot, enabling the visualization of how each slot is distributed in the reduced 2D space.

3.2.2 Some visualizations of the t-SNE

We computed the t-SNE for the slot features on SA, ISA-T and ISA-TS. This visualization is performed at the end of a training epoch to analyze the final features distribution.

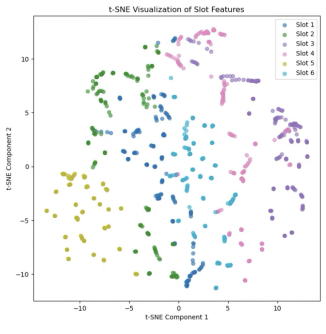


Figure 21: t-SNE SA

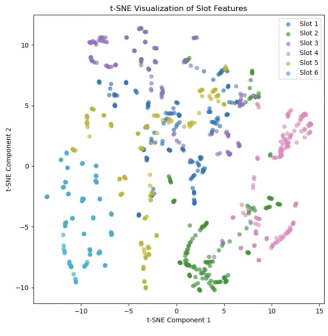


Figure 22: t-SNE ISA-T

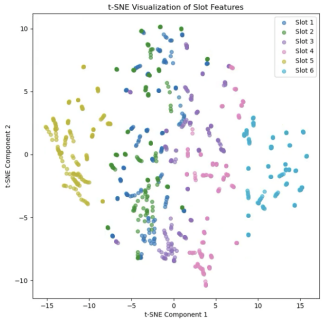


Figure 23: t-SNE ISA-TS

As seen in the obtained t-SNE results in Fig. 21, 22, 23, some close points belong to different slots. It is the case for the slots 1 and 2 in dark blue and green in the t-SNE SA Fig. 21 and this is even stronger for the slots 1,2,3 and 5 in the t-SNE ISA-T in Fig. 22. This indicates that the model has not fully learned to differentiate between features, and does not represent the objects in the scene correctly. Some slots are be overlapping in terms of data point of the features they represent. To address this issue, we are trying to improve the slot assignment by reshaping the loss function. By adjusting the loss, we can force the model to learn the importance of associating each slot correctly to one object solely, ensuring better separation of objects.

3.3 Proposed modified loss

To solve this, we introduced penalties:

1. Penalty on close t-SNE points for objects from different slots to encourage separation.
   Close points in the t-SNE embedding space are represented as different slots. It is hard for the model to distinguish them. Introducing a penalty for objects from different slots that are too close in the t-SNE space forces the model to learn to separate them. This ensures that each slot represents one object, enabling better predictions and generalizations.

2. Penalty on spatial components to prevent nearby objects from confusing the model.
   In some dataset, objects may be located close to each other. The model confuse objects that are spatially close, leading to poor object segmentation and incorrect slot assignments. Adding a penalty to spatial components forces the model learn to separate objects that would be close to each other. we are guiding the model to respect the spatial layout of the objects. This helps the model from grouping by mistake close objects together, even if they belong to different slots.

These penalties are incorporated into the final loss function. These penalty will increase the loss value, even though the model is trying to minimize it. This will force the model to try to not have a high penalty and will help it in learning better object separation.

3.3.1 Pseudocode for the proposed loss

Below is the pseudocode of the proposed loss. The choice of the hyperparameters was based on trial and error, and by looking at the value of the loss, as well as the value of the penalty to make sure that they were in similar range and that one was not overtaking the other one.

Algorithm 3 Enhanced Penalty Computation with Spatial Constraints

Initialize `iteration_counter` = 0
while True **do**
  if `iteration_counter` % 250 == 0 **then**
    Extract dimensions of `new_slots` as (X, Y, Z)
    Reshape `new_slots` to shape $(X \times Y, Z) \rightarrow$ `data`
    Apply t-SNE on `data` $\rightarrow$ `tsne_results`
    Compute pairwise Euclidean distances of `tsne_results` $\rightarrow$ `distances`
    Compute spatial positions from encoder output:
      Define H, W as visual resolution
      Create grid positions $\rightarrow$ `positions` of shape $[H \times W, 2]$
    Compute pairwise spatial distances $\rightarrow$ `spatial_distances`
    Set `penalty` = 0
    for each pair of points (i, j) where $i < j$ **do**
      if `distances[i, j]` < 2.0 AND $(i//Y) \neq (j//Y)$ **then**
        `penalty += (1.0 - distances[i, j])`^{2}
      end if
      if `spatial_distances[i % (H * W), j % (H * W)]` < 5.0 **then**
        `penalty += (1.0 - spatial_distances[i % (H * W), j % (H * W)])`^{2}
      end if
    end for
  end if
  Increment `iteration_counter` by 1
  Update `old_loss` $\rightarrow$ `old_loss + penalty` $\times$ `factor`
end while

At every 250th iteration, it reshapes the output of the Slot Attention module of data (`new_slots`) into a shape for dimensionality reduction via t-SNE. Euclidean distances between points are computed, if points in t-SNE feature space are close (distance < 2.0) but belong to different slots, a penalty is introduced. Simultaneously, spatial positions are derived from an encoder's output grid to calculate pairwise spatial distances. Points that have a close spatial proximity (distance < 5.0) also adds to the penalty. These penalties, computed as squared differences from a baseline, are used to update the overall loss function.

4 Results

4.1 Dataset Description

Two datasets have been used for this project: OBJ3D and CLEVRER.

4.1.1 OBJ3D

The OBJ3D dataset [Lin et al., 2020] consists of a total of 3,320 videos, including 2,920 training videos, 200 evaluation videos, and 200 test videos. The videos represent objects, which can be cubes, spheres or cylinders. Each video in the dataset has a resolution of 64x64 pixels and contains 50 frames. This dataset is commonly used for training and evaluating models in the field of video prediction and object detection. The ground truth of the objects'masks are not available for this dataset.

4.1.2 CLEVRER

The CLEVRER dataset [Yi et al., 2019] consists of a total of 20,000 videos, including 10,000 training videos, 5,000 evaluation videos, and 5,000 test videos. The videos represent objects and scenes in various configurations. The objects are like OBJ3D, they represent spheres, cubes or cylinders. Each video in the dataset has a resolution of 64x64 pixels and is in MP4 format. The dataset also provides ground truth data, which is available for use in tasks such as video prediction and object detection.

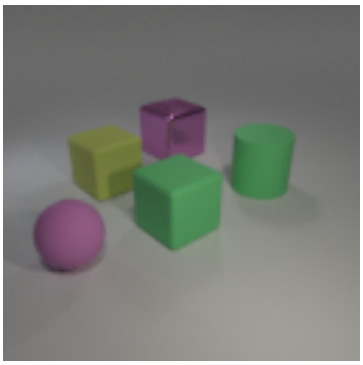


Figure 24: An example of a frame of OBJ3D

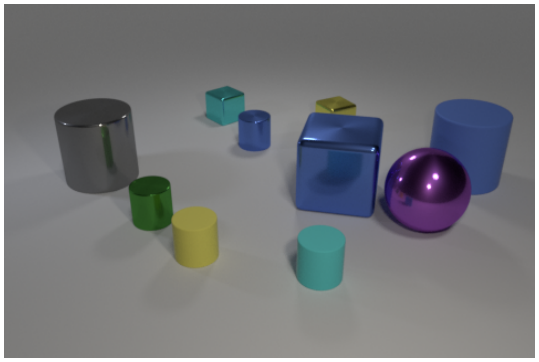


Figure 25: An example of a frame of CLEVRER

4.2 Grayscale results

We look at the effect of color during the training. For this, we have trained and tested the model on a grayscale dataset. This has been done to showcase the importance of color in the object decomposition. Two experiments have been done, training the whole model of SAVi with grayscale data and testing on the grayscale with a color trained model of SAVi. Studying the effect of color during training is useful for understanding the role of color as a feature in object decomposition tasks. By training and testing SAVi on grayscale data, we can isolate the contribution of color. Testing the model trained on color data with grayscale can prove how much the model relies on color for object segmentation.

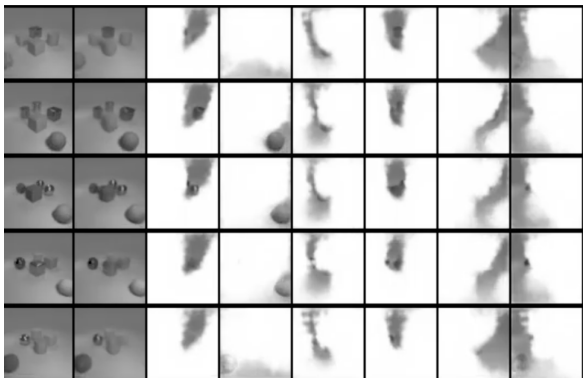


Figure 26: Baseline result on OBJ3D trained on grayscale

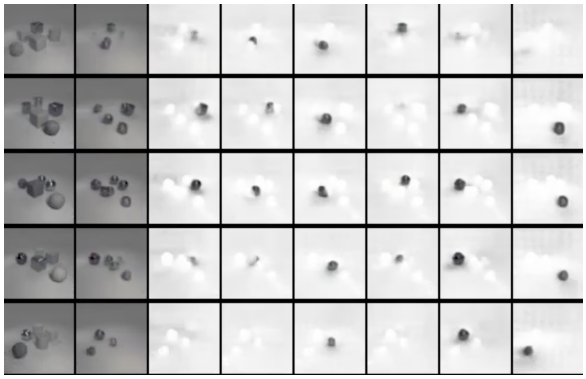


Figure 27: Baseline result on OBJ3D tested on grayscale

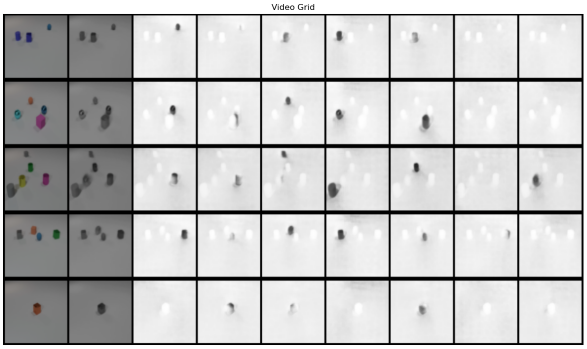


Figure 28: Baseline result on CLEVRER trained on grayscale

As we see in Fig.26, the decomposition is poor when a grayscale dataset is used for the training.

Color provides additional cues for distinguishing objects in a scene. Without this, models have to rely solely on texture, shape, and intensity gradients, which can make the separation of visually similar regions more difficult.

On the OBJ3D dataset, as the image resolution is smaller and the objects closer together as well as they have similar color to the background, the performance is poor. The objects are detected but mixed with the background as seen in Fig. 26. However when tested on a color trained model, the decomposition seems to be good. The model trained on grayscale, probably has not learned to capture relationships between objects. On the other hand, a model trained on color images learns more features as it contains more information. Even when the color is removed during testing, the model trained on color images could be using more features that improve its performance on grayscale data.

On the Clevrer dataset, the performance is better as seen in Fig. 28, as the objects are further away and the background is much lighter.

Color significantly enhances model performance in object decomposition. This is due to the fact that it has more robust features. The color has a great importance in the model training and is essential for good performance in these tasks.

We also visualize the t-SNE representation of the CNN features from the trained grayscale model. This visualization is performed at the end of a training epoch to analyze the final feature distribution.

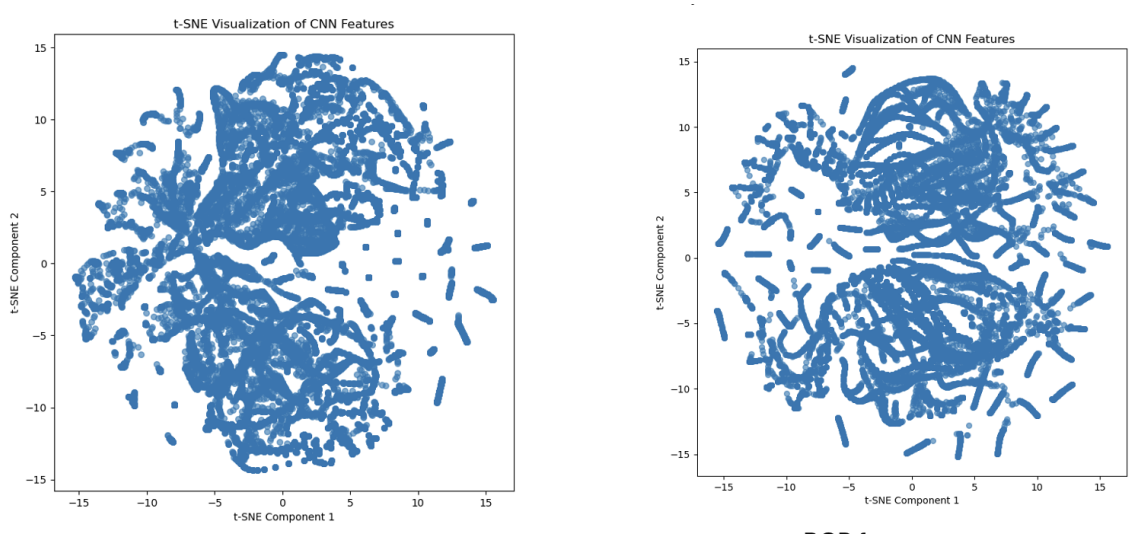


Figure 29: t-SNE on the CNN fetaures, right: on grayscale, left: on color

The two t-SNE visualizations show the visualization of CNN features from different image: grayscale (right) and color (left) in Fig.29. In the grayscale plot, the feature distribution appears is uniform and has tighter clusters. This might show that grayscale features may have less variance compared to color. On the other hand, the color t-SNE plot shows more separated clusters. Since can help in saying that color features provide additional details. This is not obvious as the CNN features are still very close spatially in this representation.

We also visualize the t-SNE representation of the slots features from the trained grayscale model. This visualization is performed at the end of a training epoch to analyze the final feature distribution.

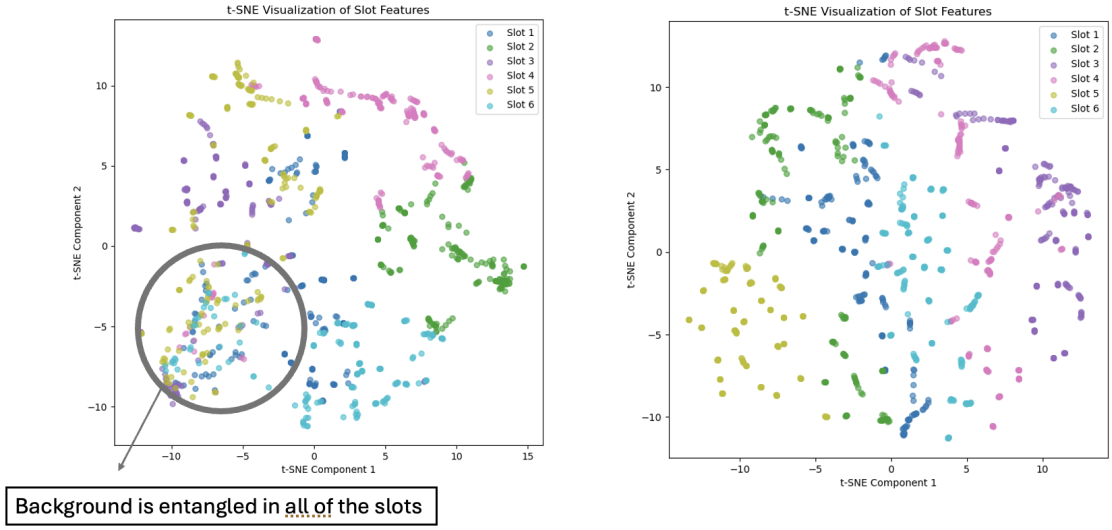


Figure 30: t-SNE on the slot features, right: on grayscale, left: on color

Similarly, in the grayscale plot, some very close points are assigned to almost all clusters. It seems like it might represent the background, as seen in Fig. 26 in part of every slot. On the contrary, the t-SNE color plot shows more dispersed clusters, suggesting that the color features provide additional details, as there is more separation between data points. This contrast illustrates how colors can influence the slots features representation.

4.3 Discarding Invariant Slot Attention rotation

To examine the behavior on ISA-TSR, we analyze the qualitative results of the trained model and visualize the t-SNE representation of the slot features to understand its performance.

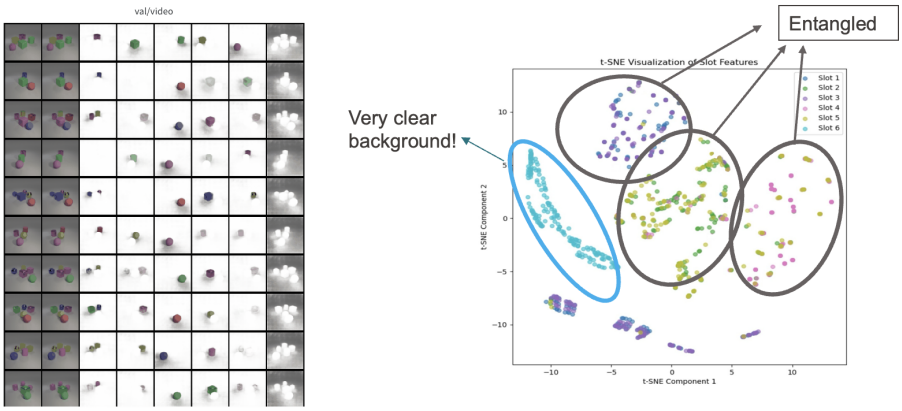


Figure 31: Evalutation of the object-centric ISA-TSR and its slot t-SNE on OBJ3D

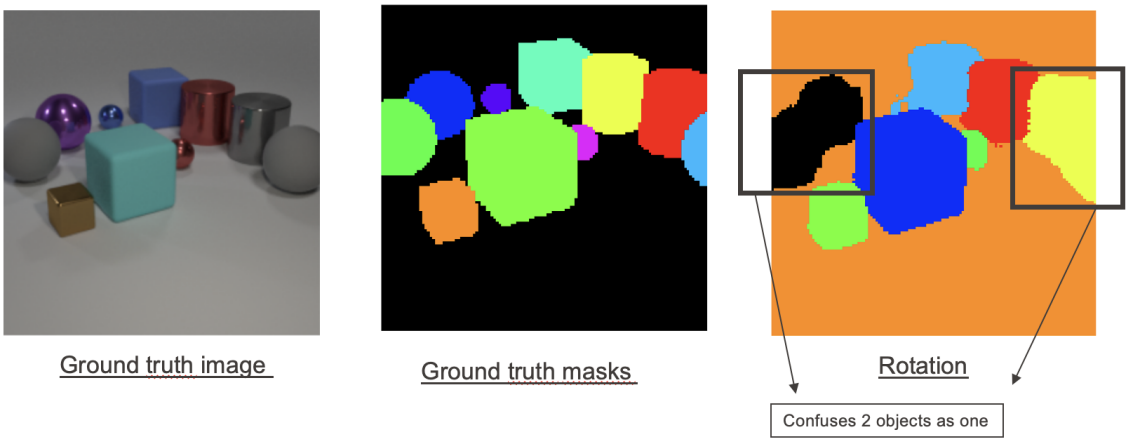


Figure 32: Recap of the original decomposition for ISA-TSR

We will dismiss further training on rotation tasks, as even with the optimized loss function, the model still struggles to differentiate closer objects that share similar features. This limitation is very clear in the t-SNE visualizations of slots features. The plots, Fig. 31 clearly illustrate that objects with similar characteristics are tightly clustered. Each slots show entanglement, more than one object are represented in one slot and other objects are represented over multiple slots. Even going back to the original paper performance from [Biza et al., 2023], two objects were already mistaken as one as seen in 32. As explained in 2.3.2., some objects like spheres and cylinders are already invariant to rotation meaning that rotation invariance may not be as crucial here. This could explain the less impressive results, as the model's rotation invariance may not be as useful in this context and could potentially introduce some confusion as seen in Fig.31.

Further computation on ISA-TSR will be discarded.

4.4 Qualitative Results

Visualizations of object slots show the effectiveness of the improved loss function.

4.4.1 OBJ3D

The first column represents the original data, the second the reconstructed image from the slots and the rest of the columns are the slots representations.

With baseline loss

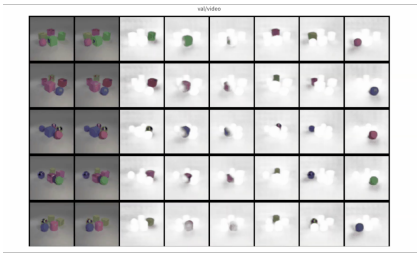


Figure 33: Slot-Attention

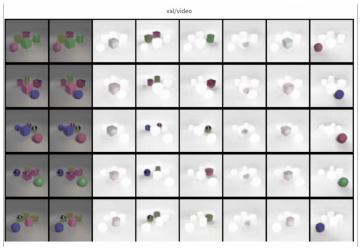


Figure 34: Invariant Slot Attention translation

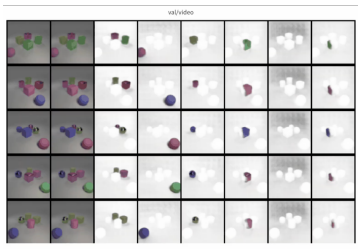


Figure 35: Invariant Slot Attention translation and scaling

The preliminary results of implementing Invariant Slot Attention reveal that, despite the model's improvements in attention mechanisms, some objects are still entangled together. This entanglement occurs because the model considers both the features (such as color or texture) and the spatial positions of the objects when forming the slot embeddings.

While the attention mechanism is designed to be invariant to certain transformations, the model still relies heavily on spatial proximity to separate objects. As a result, when objects with similar visual features are next to each other in the scene, they can become entangled in the embedding space. The model struggles to properly disentangle them because the spatial information introduces ambiguity in the attention process, leading to the objects being grouped together even when they belong to different slots. This phenomena is very clear on the invariant implementation the Fig. 34 and Fig. 35. The first slot on 35 and second slot on 34 show two objects on one slot. Similarly the first, fourth and fifth slot on 34 represent the same object. It is also the case for the second, for some objects on Fig. 33. This shows that there is still improvement.

With proposed loss

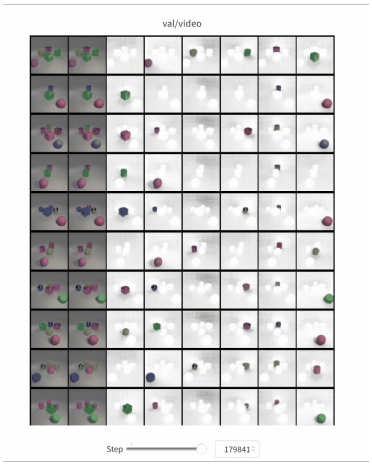


Figure 36: Slot-Attention

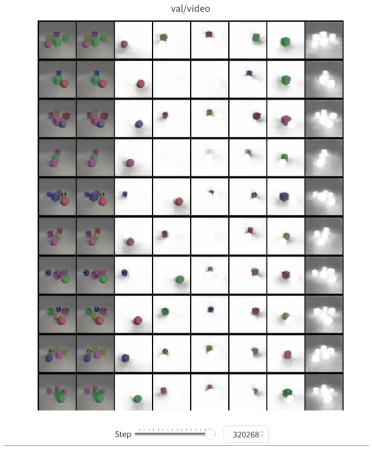


Figure 37: Invariant Slot Attention translation

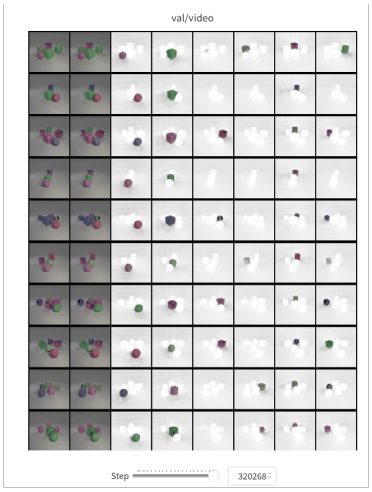


Figure 38: Invariant Slot Attention translation and scaling

With the optimized loss function, the results show improvement in the model's performance as seen in Fig.36, 37 and 38. The issue of entanglement has been resolved, and no instance of two objects are being assigned to the same slot. The background is even very clear on the invariant Fig.37. There still are objects that are represented in more than one slot for a few slots as seen in Fig.38. The optimized loss enables the model to better separate and assign individual objects to distinct slots.

Proposed loss t-SNE on slots

We also visualize the improvement of the proposed loss on t-SNE slot features.

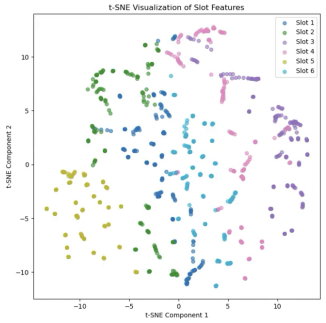


Figure 39: t-SNE SA baseline loss

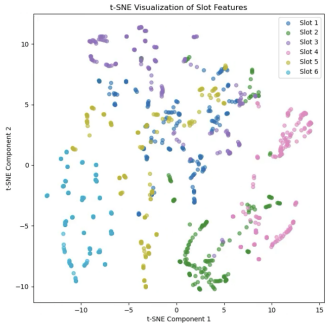


Figure 40: t-SNE ISA-T baseline loss

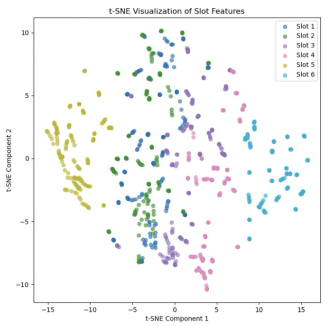


Figure 41: t-SNE ISA-TS baseline loss

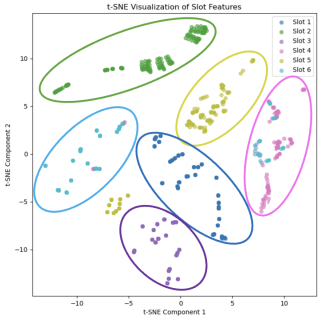


Figure 42: T-SNE SA proposed loss

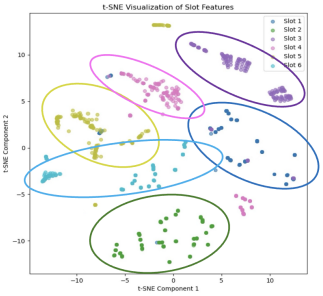


Figure 43: T-SNE ISA-T proposed loss

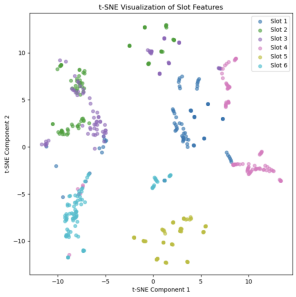


Figure 44: T-SNE ISA-TS proposed loss

This result is also verified through the t-SNE visualizations of the slot representations, which now shows clear clusters corresponding to individual objects. Unlike in the previous visualizations in Fig. 39, Fig. 40, Fig. 41, where closed points spatially belonged to different clusters, the optimized loss function shows that now each object forms a more distinct cluster. As they are circled corresponding to the correct color in Fig. 39 and 40, distinct cluster are observed. Few points are wrongly assigned. The clusters are also more spaced, indicating better separation and showing that the model has better learned differences.

The optimized loss significantly enhances the model's ability to disentangle objects.

4.4.2 CLEVRER

The same results can be observed through CLEVRER dataset. The first column represents the original data, the second the reconstructed image from the slots and the rest of the columns are the slots representations.

With baseline loss

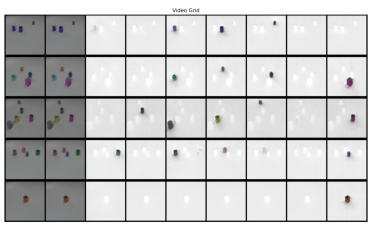


Figure 45: Slot-Attention

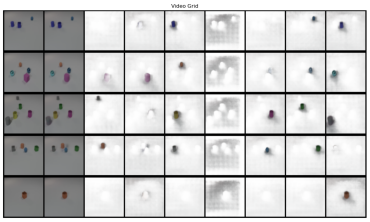


Figure 46: Invariant Slot Attention translation

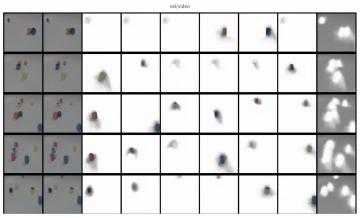


Figure 47: Invariant Slot Attention translation and scaling

The objects are more spaced out in the CLEVRER dataset, the issue of close objects spatial being assigned o the same cluster is not happening here. However as seen in Fig. 45 and 46 for the invariant implementation, the results are not the best. Some objects remain entangled with the background, and the decomposition is not as clean or distinct compared to the baseline results in Fig. 45. There is still room for improvement in achieving a clearer object decomposition.

With improved loss

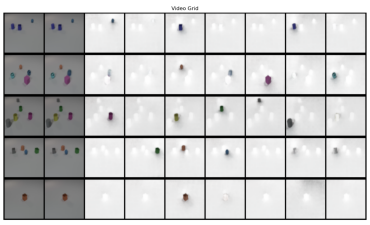


Figure 48: Slot-Attention

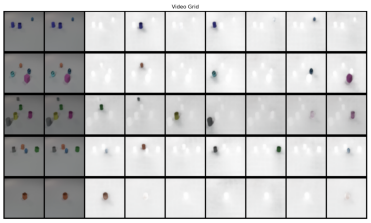


Figure 49: Invariant Slot Attention translation

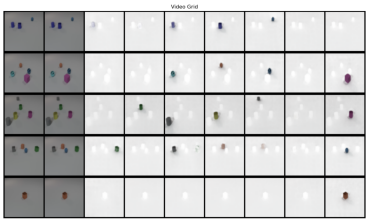


Figure 50: Invariant Slot Attention translation and scaling

With the new loss implementation, as shown in Fig. 49 and 50, the results demonstrate improvement. Objects are now clearly separated from the background, and the decomposition and distinct giving similar results to the baseline results in Fig. 45. This highlights the effectiveness of the new loss in achieving a clearer and more precise object decomposition.

4.5 Quantitative Results

To evaluate the quantitative performance of the new SAVi, we use two metrics: MSE and ARI.

Mean Squared Error (MSE): MSE measures prediction accuracy by computing the average squared difference between predicted pixel values $\hat{y}_i$ and true pixely_i values:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{3}$$

Lower MSE indicates better reconstruction.

Adjusted Rand Index (ARI): ARI evaluates clustering performance by comparing predicted clusters with ground truth labels, adjusting for random chance. It ranges from -1 (worse than random) to 1 (perfect clustering). It is computed as:

$$\text{ARI} = \frac{\text{RI} - \text{Expected RI}}{\text{Max RI} - \text{Expected RI}}, \tag{4}$$

where RI is the raw Rand Index, and the adjustment accounts for chance alignment.

The Rand Index is a measure used to evaluate the similarity between two data clusterings. It is used in clustering and segmentation tasks to assess the accuracy of predicted clusters compared to ground truth labels.

The Rand Index is calculated as:

$$RI = \frac{\text{Number of agreements}}{\text{Total number of pairs}} \tag{5}$$

Where the number of agreements includes both pairs of elements that are correctly assigned to the same cluster and pairs that are correctly assigned to different clusters and the total number of pairs are the total possible pairs of elements in the dataset.

A higher RI value (closer to 1) indicates better clustering performance.

Higher ARI reflects better object decomposition.

4.5.1 Quantitative results on OBJ3D

For the OBJ3D dataset, we do not have access to the ground truth maks. Only the MSE on the reconstructed image is computed.

Model	MSE baseline	MSE proposed baseline
SA	0.0002	7.59e-05
ISA-T	0.0017	0.0011
ISA-TS	0.012	0.0024

Table 1: MSE comparison of models for OBJ3D

For the SA baseline, the proposed loss shows an improvement, achieving a much lower value (7.59e-05 vs. 0.0002). For the ISA-T, both methods perform similarly, with a slight avdvantage to the proposed method (0.0011 vs. 0.0017). Similarly, in the ISA-TS configuration, the proposed method reports a smaller value (0.012 vs. 0.0024). The qualitative results from the proposed method are better, demonstrating improved decomposition capabilities. The improvement of the invariant method is questionable. It achieves rather good results but does surpass the baseline. Hovewer the baseline is pre-trained from the paper [Wu et al., 2023] and shows the absolute best scenario.

4.5.2 Quantitative results on CLEVRER

For the CLEVRER dataset, we have access to the ground truth masks. Both the MSE and ARI on the reconstructed image are computed.

Model	MSE baseline	MSE proposed loss
SA	0.00027	0.00015
ISA-T	0.0009	0.0005
ISA-TS	0.00026	0.00015

Table 2: MSE comparison of models for CLEVRER

For the SA baseline, the proposed method also a significant improvement, going fron 0.00027 to 0.00015 . For the ISA-T configuration, the improvement is also here going from 0.0009 to 0.0005. Similarly, in the ISA-TS configuration, the proposed method reports almost the same value as the baseline. The qualitative results from the proposed method are better, demonstrating improved decomposition and representation capabilities.

ARI Computation

To compute the ARI on the predicted maks, we proceed as follows:

1. Extract the ground truth masks from the JSON files.
2. Sum each mask to create a binary mask containing all the slots, as individual slots are not mapped to specific objects.
3. Extract the predicted masks from the model.
4. Set a threshold of 0.5 for background and foreground pixels to create a binary mask of the predictions.
5. Sum the predicted masks, similar to the ground truth masks.
6. Resize the predicted masks to match the shape of the ground truth masks.
7. Compute the ARI between the summed predicted and ground truth masks.

Below is an example with the visualization for one frame of a video.

Figure 51: Visualization of ground truth binary masks for CLEVRER

Figure 52: Visualization of predicted object-centric model masks for CLEVRER

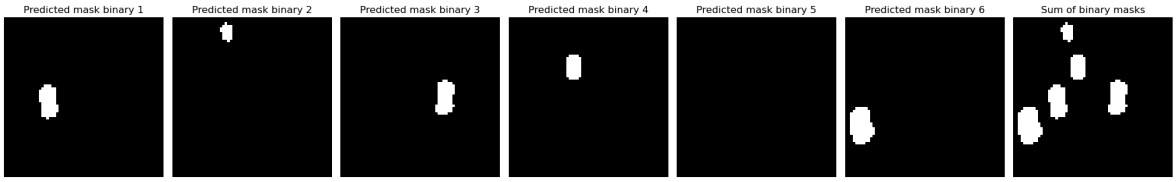


Figure 53: Visualization of predicted binary object-centric model masks for CLEVRER

An interpolation is performed during resizing, which can introduce small changes in the ARI computation. The value of the threshold also depends on the random seed, and objects may include slight amounts of background.

Model	ARI baseline	ARI proposed loss
SA	86.3%	64.76%
ISA-T	20.79%	72.61%
ISA-TS	1.44%	77.58%

Table 3: ARI comparison of models for CLEVRER

The results presented in Table 3 show a significant improvement in ARI scores when using the proposed loss function on the CLEVRER dataset. SA, achieves an ARI score of 86.37%, but the proposed loss function drops to 64.76%. On the other hand, both ISA-T and ISA-TS are improved with the proposed loss, with ARI scores going from 20.79% to 72.61% and from 1.44% to 77.58%, respectively. These results highlight the effectiveness of the proposed loss in enhancing the performance of the ISA models.

4.6 Results on Slot-Former

Here we show the results on Slot Former. The model of Slot Former has not been touched, the difference only relies on the object-centric model.

4.6.1 Qualitative results

Figure 54: An example of a frame on OBJ3D

Figure 55: Slotformer on OBJ3D for ISA-T

The qualitative results as seen in Fig.55 show the poor performance of the Slot Former when trained with a ISA-T model. This is anticipated, as the Slot Former does not utilize the disentangled position, unlike what is achieved with the proposed loss, as detailed in the pseudocode in 3.3.1. For an improved result, this has to be implemented on the Slot Former.

4.6.2 Evaluation Metrics

The following metrics are used to evaluate the performance of Slot-Former on the OBJ3D dataset:

1. **MSE (Mean Squared Error):** MSE calculates the average of the squared differences between the predicted and the ground truth pixel values.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{6}$$

   where y_i is the true value, $\hat{y}_i$ is the predicted value, and N is the total number of pixels.

   Lower MSE values indicate better reconstruction quality.

2. **PSNR (Peak Signal-to-Noise Ratio):** PSNR is a logarithmic metric that measures the ratio between the maximum possible pixel value of the image and the power of the noise.

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{I_{\text{max}}^2}{\text{MSE}}\right) \tag{7}$$

   where I_{max} is typically 255 for 8-bit images.

   Higher PSNR values indicate better quality as there is less noise in the reconstructed image.

3. **SSIM (Structural Similarity Index):** SSIM evaluates the perceptual similarity between two images by considering three factors: luminance, contrast, and structure.

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{8}$$

   where μ_x and μ_y are mean intensities, σ_x^2 and σ_y^2 are variances, and σ_{xy} is the covariance between the images. Constants C_1 and C_2 stabilize the division.

   Its value ranges from 0 to 1, where 1 indicates a perfect match between the two images.

4. **Perceptual Distance:** This metric measures the perceptual difference between the predicted and true images based on a perceptual feature space.A lower perceptual distance indicates better alignment between the predicted and ground truth images in terms of visual quality.

4.6.3 Quantitative results

	ISA-T	**ISA-TS**
Baseline	mse: 10.2269 psnr: 26.2649 ssim: 0.7762 percept_dist: 0.4015	mse: 9.3033 psnr: 26.7493 ssim: 0.7337 percept_dist: 0.4485
Proposed regularization	mse: 8.1599 psnr: 27.3066 ssim: 0.8003 percept_dist: 0.4071	mse: 7.6733 psnr: 27.6541 ssim: 0.8081 percept_dist: 0.3649

Table 4: Metrics obtained for Slot Former on OBJ3D

The metrics obtained from running the Slot Former model demonstrate an improvement with the proposed loss regularization, as shown in Table 4. For ISA-T, the MSE decreases from 10.2269 to 8.1599, while the PSNR increases from 26.2649 to 27.3066, reflecting better reconstruction quality. ISA-TS exhibits similar improvements. They also show that SlotFormer has the potential to deliver better predictions, but its design should be better aligned with the disentangled position information.

5 Discussion and Conclusion

5.1 Conclusion

In conclusion, Slot Former demonstrated improvements in object decomposition by providing more robust and consistent results on the object-centric model across different seeds. However, some challenges remain, particularly with the model's ability to fully leverage disentangled position information, which is important in improving performance. Although Slot Former has shown promise, the current design does not fully exploit the positional data, limiting its potential for generating more accurate predictions.

5.2 Next Steps

Future work could focus on addressing these limitations by refining SlotFormer's design to better fit the updated object-centric model ultimately improving its object dynamic prediction capabilities. In particular, efforts should be directed toward effectively utilizing the available positional data, fine-tuning the loss coefficients to optimize performance, and exploring additional techniques that can ensure proper background separation. By addressing these areas, SlotFormer has the potential to perform in a better way, capable of producing more accurate, reliable predictions. These improvements could be beneficial for its applicability across range of tasks and contribute to advancements in object decomposition models.

6 Appendix

6.1 The use of EPFL Cluster

USERNAME : **moutarli**
Global usage from 2024-09-01 to 2025-01-31

Account	Cluster	# jobs	GPU [h]	CPU [h]	eCO_2 [kg]	Costs [CHF]
master	izar	867	3,386.3	0.0	140.4	583.8
master	kuma	364	188.0	0.0	7.3	0.0

Walltime GPU	h	3,574.3
Walltime CPU	h	0.0
Number of jobs		1,231.0
Est. carbon footprint	kg	147.7
Costs	CHF	583.8

Figure 56: Total use of scitas

All training sessions were conducted on SCITAS clusters. Here is an overview of their usage. The account used was a master's account, and Python was the programming language employed.

Two different clusters were utilized for training: **KUMA** and **IZAR**. These are the only two GPU clusters available, and our training required GPUs. Initially, the work was carried out on the **KUMA** cluster, which had shorter queues and was more efficient. However,around the 15th of november, access to this cluster was restricted to PhD students, necessitating the relocation of the work to the **IZAR** cluster.

To connect to a Cluster run in a Terminal : `ssh <username>@<cluster>.hpc.epfl.ch`
Then enter your password and you are logged.

6.1.1 Storage management

Different things were stored in different places:

1. **Home** : Secure file system for the user's critical information. The code was stored in here. Data will be erased after two years of inactivity or 6 months after leaving EPFL, whichever occurs first. The contents of the /home file system are backed-up on a daily basis with a six month retention period

2. **Scratch** : Temporary storage needs and high-speed data processing. The scratch is automatically deleted every 30 days with no warning, so make sure that your work is back-uped somewhere else. The checkpoints for every training as well as the .npy files computed for the t-SNE were stored here.

3. **Work** : Team storage, ideal for users needing substantial storage capacity, particularly for research data retention over several years. The checkpoints for every training as well as the .npy files computed for the t-SNE when the storage of the scratch was full. All other important checkpoints were saved here. Data will be erased 6 months after the cessation of payment. The contents of the /work file system are not backed up by default.
The path used was : /work/scitas-share/moutarlier

Notes about the cluster :

1. Kuma has limited access and is not available to bachelor and master student since November 15th, whereas Izar is open to everyone.

2. Home and Work are shared amongst EPFL clusters but not the scratch.

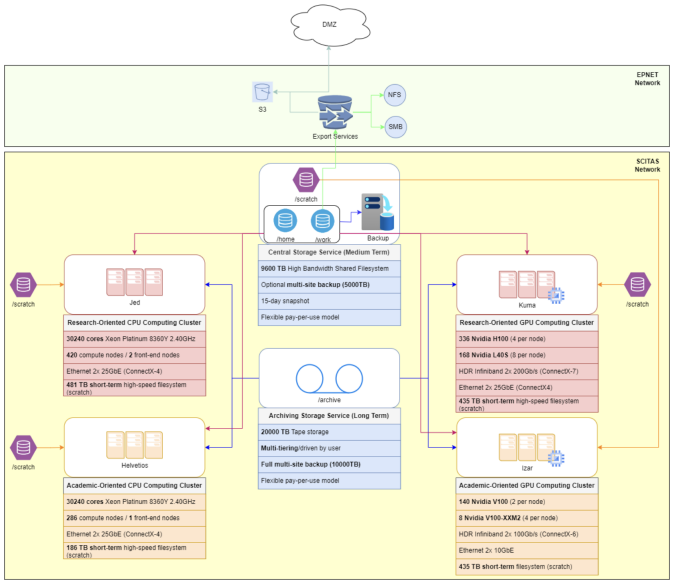


Figure 57: Architecture of EPFL clusters

6.1.2 Running a job

Seting an environnement

Before running a job, you need to create an environment containing the requirements for your run. Follow these steps:

1. **Load the correct Python module:**
Before creating an environment, load the appropriate Python version using the following command:

```
module load gcc python
```

You can replace `python` with the specific version you want to use. Note that:

- The default Python version is **Python 3.10.4**.
- **Python 2.7.18** is available for backward compatibility but is limited to specific environments.

2. **Create a virtual environment:**
   Use the following command to create your virtual environment:

   ```
   virtualenv --system-site-packages venvs/venv-for-demo
   ```

   The `--system-site-packages` option ensures that optimized packages already installed in the Python modules are used.

3. **Activate the virtual environment:**
   Activate your newly created environment with:

   ```
   source venvs/venv-for-demo/bin/activate
   ```

4. **Install additional packages (if needed):**
   Use `pip` to install any additional required packages:

   ```
   pip install <package>
   ```

After completing these steps, your environment is set up, and you are ready to run a job.

Note : The clusters do not share the environments, if you are changing clusters, you need to re-create an environment and re-download the packages.

6.1.3 Submitting a job

The clusters use the Slurm protocol to submit a job. You need to create a .sh file containing information on how you want your job to be run. Some spefications are proper to the cluster and other are general one.

Here is an example of a runned job :

Listing 1: SLURM example script for SAVi Training

```
#!/bin/bash
#SBATCH --job-name=savi
#SBATCH --nodes 1
#SBATCH --ntasks 1
#SBATCH --cpus-per-task 32
#SBATCH --time 40:00:00
#SBATCH --qos=gpu
#SBATCH --gres=gpu:2
#SBATCH --output embbedings_baseline.out


module load gcc/11.3.0
module load cuda/12.1.1
source venvs/venv-for-demo/bin/activate


python scripts/train.py --task base_slots \
    --params slotformer/base_slots/configs/savi_obj3d_params.py \
    --fp16 --ddp --cudnn
```

Below are the details of the SLURM directives included in the script:

SLURM Directives

The job script contains the following SLURM options:

1. –**job-name=savi**: The name of your job.
2. –**nodes 1**: Number of nodes to use. This is always set to one.
3. –**ntasks 1**: Maximum number of tasks (in an MPI sense) per job. This is always set to one.
4. –**cpus-per-task 32**: Number of CPU cores per task. This is set to 32 for the current runs.
5. –**time 1:00:00**: Maximum walltime required. Jobs exceeding this time limit will be terminated. Time is specified as HH:MM:SS. Since some jobs could take up to 30 hours, the limit is set to 40 hours to avoid premature termination. This limit has a maximum time
6. –**qos=gpu**: The partition for the cluster. Use `gpu` for Izar and to `h100` or `l40s` for Kuma.
7. –**gres=gpu:2**: Specifies the number of GPUs to allocate. In this case, 2 GPUs are requested. The max number depends on the number of GPUS of the cluster.
8. –**output=embbedings_baseline.out**: The name of the output file for logging and debugging.

Additionally, the script includes:

- Module loading and environment activation commands.
- The Python script to execute the desired task.

If something is wrong with the submission or if a limit is exceeded or if a SLURM option is wrong, you will have an output message :

```
sbatch: error: Batch job submission failed: Requested node configuration is not available
```

This message specifies what is wrong with your submission.

Submitting the Job

To submit the job, use the following command:

```
sbatch job.sh
```

This command places the job in the cluster's queue. The job's position in the queue depends on several factors.

Queueing Priorities

The waiting list for job execution is determined by the following priorities, in order:

1. **Job Age**: How long the job has been in the queue.
2. **Job Size**: The resources (e.g., CPUs, GPUs) reserved for the job.
3. **QOS**: The partition assigned to the job.
4. **User Fairshare**: A metric reflecting the user's resource usage relative to others.

Wait Times

During periods of high activity, jobs may experience significant wait times before execution. In practice, these wait times have ranged from a few hours to several days, depending on the cluster's load and your job's priority.

6.1.4 Accessing job characteristics

View Your Submitted Jobs

To visualize your submitted jobs, use the `Squeue` command. This will display:

```
$ Squeue
     JOBID          NAME   ACCOUNT        USER NODE   GPUS      ST        REASON            START_TIME
    123456          run1    scitas         bob     6     1         R          None  2023-02-03T04:18:37
    123457          run2    scitas         bob     6     2        PD    Dependency                   N/A
```

The output provides information about:

- **JOBID**: The ID assigned to the job.
- **NAME**: The name of the job, as specified in the SLURM script.
- **ACCOUNT**: The account under which the job is running.
- **USER**: The username of the job submitter.
- **NODE**: The number of nodes requested for the job.
- **CPUS**: The number of CPUs requested for the job.
- **MIN_MEMORY**: The minimum memory requested.
- **ST**: The state of the job (`R` for running, `PD` for pending, etc.).
- **REASON**: The reason why the job is pending or any other status details.
- **START_TIME**: The expected start time for pending jobs or the actual start time for running jobs.
- **NODELIST**: The nodes allocated to the job.

Access the CPU and RAM information

To access the CPU and RAM usage of your submitted job run `htop`. This will display the CPU and RAM usage on the cluster for submitting jobs and will help you for tuning your parameters.

Accessing GPU information

To monitor the GPU usage of a submitted job, follow these steps:

1. Use the `Squeue` command to view your submitted jobs. Identify the **node number** associated with the job you want to investigate.
2. Connect to the specific node using the command: `ssh <node_number>`.
3. Once connected, run the command `nvidia-smi`. This will display detailed GPU usage information, including:
    - The GPUs being utilized by your job.
    - The percentage of GPU usage for each GPU.

To exit the node do `CTRL + D`
This process is particularly useful for optimizing your job submissions and ensuring efficient resource usage.

Cancelling a job

To cancel a specific job run `scancel JOBID`

Other important information

- If you submit a job, you don't submit the code with it. If you change the code while the submission is pending, the job will be submitted with the updated code.
- However if you submit a job and it is running, you can modify it and make another submission and the new submission will run on the updated code and the old submission on the old code.
- There are no limitations of the number of jobs, other than you might slow down the whole cluster for other users.
- When you submit a job, the estimated price will appear. It depends on the number of GPUS, CPUS and wall-time (the more, the more expensive).

6.1.5 View results in tensorboard

How to visualize in tensorboard?

1. Enable the virtual environment and add the Python path.
2. Navigate to the `/logs` folder in your project directory.
3. Run the following command to start TensorBoard on a specific port (choose a random port number):

```
tensorboard --logdir ./ --port=8677 --bind_all
```

4. Open a local terminal on your personal machine.
5. Run the following SSH command to create a tunnel to the remote cluster, replacing the placeholders with the correct details and ensuring the port numbers match:

```
ssh -L 8677:kuma1.kuma.cluster:8677 -l user kuma.hpc.epfl.ch -f -N
```

6. Open your web browser and navigate to the following URL, ensuring you use the correct port number:

```
http://localhost:8677
```

6.1.6 KUMA cluster

The first part of the training was done on KUMA cluster. Kuma is a GPU cluster. There are 2 partitions on Kuma, to differentiate nodes based on GPU type:

1. **h100**, to use the Nvidia H100 GPU nodes (with FP64 capabilities)
2. **l40s**, to use the Nvidia L40s GPU nodes (with FP32 capabilities)

Here are their specifications:

Node Type	Count	Processor	Memory	Storage	Hostnames	GPUs
H100	84	AMD EPYC 9334 @ 2.7 GHz	384 GB	6.4 TB	kh[001-084]	4NVIDIA 94GB
L40s	20	AMD EPYC 9334 @ 2.7 GHz	384 GB	7.6 TB x3	kl[001-020]	8NVIDIA 48GB

Table 5: Specifications of Compute Nodes

Some specifications of KUMA cluster:

- Our work uses the –fp16 option. This option is better paired with the L40s partition.
- Using a fraction of a GPU, such as 0.5, is not possible. GPUs cannot be split by time or memory allocation. Each job must request a whole GPU.
- A GPU refers to the entire physical GPU, not a virtual instance. For example, requesting 1 GPU means you are allocated one full physical GPU, not a virtualized portion of it.
- The default QOS limits jobs to 3 days.

6.1.7 IZAR cluster

The rest was done on IZAR. IZAR has different specifications. There are 5 partitions on Izar, only the default one was used.

1. **gpu**: For jobs using up to 8 nodes, with a time limit of 3 days. This is the default.
2. **week**: For jobs using up to 8 nodes, with a time limit of 7 days.
3. **gpu_free**: For jobs using up to 1 node, with a time limit of 12 hours.
4. **debug**: For testing your job on up to 1 node, with a time limit of 1 hour and high priority.
5. **build**: For jobs using up to 20 cores and 1 GPU, with a time limit of 8 hours.

6.1.8 Other problems faced

1. Be careful about the memory of the GPUS, IZAR GPUS have less memory than KUMA ones so if there is a memory leak, the GPUS can be full and then the job won't use the GPU anymore. This will eventually stops the job.
2. Sometimes there are updates of CUDA made by SCITAS. Make sure that your imported version of CUDA is the most recent one, or your job might not run.
3. If you are running jobs with multi-gpu, you need a different version of python to be loaded otherwise the submission will only use one GPU.

References

[Biza et al., 2023] Biza, O., Steenkiste, S., Sajjadi, M. S. M., Elsayed, G., Mahendran, A., and Kipf, T. (2023). Invariant slot attention: Object discovery with slot-centric reference frames. *Accepted at ICML 2023*.

[Bozkurt et al., 2019] Bozkurt, A., Esmaeili, B., Dy, J., Brooks, D., and van de Meent, J.-W. (2019). Tetrominoes dataset. https://github.com/neu-pml/tetrominoes/.

[Burgess et al., 2018] Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., and Lerchner, A. (2018). Understanding disentangling in β-vae. *Presented at the 2017 NIPS Workshop on Learning Disentangled Representations*.

[Burgess et al., 2019] Burgess, C. P., Matthey, L., Watters, N., Kabra, R., Higgins, I., Botvinick, M., and Lerchner, A. (2019). Monet: Unsupervised scene decomposition and representation.

[Esser et al., 2021] Esser, P., Rombach, R., and Ommer, B. (2021). Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12873–12883.

[Kipf et al., 2022] Kipf, T., Elsayed, G. F., Mahendran, A., Stone, A., Sabour, S., Heigold, G., Jonschkowski, R., Dosovitskiy, A., and Greff, K. (2022). Conditional object-centric learning from video. *LLD 2019*.

[Lin et al., 2020] Lin, Z., Wu, Y.-F., Peri, S., Fu, B., Jiang, J., and Ahn, S. (2020). Improving generative imagination in object-centric world models. *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

[Locatello et al., 2020] Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. (2020). Object-centric learning with slot attention. *NeurIPS 2020*.

[Razavi et al., 2019] Razavi, A., Van den Oord, A., and Vinyals, O. (2019). Generating diverse high-fidelity images with vq-vae-2. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32.

[Watters et al., 2019] Watters, N., Matthey, L., Burgess, C. P., and Lerchner, A. (2019). Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes. *LLD 2019*.

[Wu et al., 2023] Wu, Z., Dvornik, N., Greff, K., Kipf, T., and Garg, A. (2023). Slotformer: Unsupervised visual dynamics simulation with object-centric models. *Accepted by ICLR 2023*.

[Yi et al., 2019] Yi, K., Gan, C., Li, Y., Kohli, P., Wu, J., Torralba, A., and Tenenbaum, J. B. (2019). Clevrer: Collision events for video representation and reasoning. In *International Conference on Learning Representations (ICLR)*.