Supplementary Material

A Overview

Here we provide technical details in support of our main paper. Below is a summary of the contents.

Sec. B: Qualitative examples of relation classification on CLEVR-CoGenT;
Sec. C: Qualitative examples of relative direction regression on Leonardo;
Sec. D: Visualizations of the attention learned by SORNet;
Sec. E: More details on the predicates and tasks in the Leonardo dataset;
Sec. F: Additional Details on model architecture and training.

B Qualitative Results on CLEVR-CoGenT

Fig. 2 shows qualitative results of spatial relation classification on CLEVR-CoGenT (Sec. 5.1 in main paper). These examples demonstrate that SORNet is able to identify objects not only using color cues, but also shape (e.g. blue sphere vs blue cylinder in the topmost example), size (e.g. small cyan cube vs big cyan cube in the second from top example) and material (e.g. small purple metal cube vs small purple rubber cube in the third from top example). We also visualize the relevant canonical object views provided to the model. As we can see, the canonical object views can have very different appearance from the corresponding objects in the input image. It is more appropriate to consider these canonical views as a visual replacement for natural language, rather than the result of object detection or segmentation.

C Qualitative Results on Relative Direction Predition

In Fig. 3 we visualize the results of relative direction prediction (Sec. 5.5 in main paper). Specifically, we train regressors on top of frozen SORNet embeddings to predict the relative direction (a 3D unit vector) and distance (a 1D scalar) between each pair of objects, as well as between the end effector and each object. We visualize the predicted direction as arrows, scaled by the predicted distance. Trained only on a thousand examples, the regressor is able to predict continuous spatial relations accurate enough to guide robot execution (see our supplementary video), thanks to the spatial information encoded in the SORNet embeddings. Note that SORNet is never trained on explicit poses of objects, but it understands relative locations just like humans. As a result, the object-centric embedding can be quickly finetuned for downstream tasks that require accurate spatial information.

D Attention Visualization

Fig. 4 visualizes the attention weights learned by the visual transformer model in order to obtain the object-centric embeddings. More specifically, we take the normalized attention weights from the tokens corresponding to the canonical object views to the tokens corresponding to the context patches and convert the weights into a colormap, where the intensity of the colormap corresponds to the magnitude of the attention weight over that patch. We then overlay the colormap with the input image. We can see that while the model puts the highest attention to the patch containing the object of interest, it also learns to pay attention to the robot arm and other objects, while ignoring irrelevant background. We also visualize the canonical object patches given to the model, which can look wildly different from the same objects in the image. The model needs to associate the canonical view with the input view of the object under different lighting conditions and occlusions.

E The Leonardo Dataset

Here we include additional details for the Leonardo dataset.

E.1 List of Predicates

Table **??** shows the 52 predicates for 4-object test scenes.

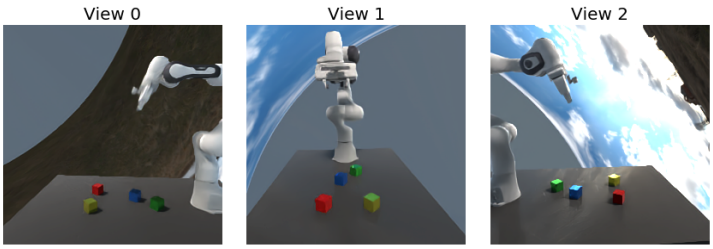


Figure 1: 3 camera views used to train the multi-view models.

E.2 Training and test tasks

Fig. 5 shows initial frame, final frame and goal conditions for a sample from each task in the Leonardo dataset.

E.3 Viewpoints

Fig. 1 shows the 3 camera views used to train the multi-view models. Each camera view is also slightly perturbed around the base camera pose during training.

F Model Architecture and Training

F.1 ResNet Baseline

The ResNet baseline uses ResNet18 backbone implemented in torchvision. The feature map before average pooling is flattened into a 512-dimensional vector and passed through 4 2-layer MLPs with 512 hidden units and outputs predicates relevant for each of the 4 objects (19 for each objects). For binary predicates (e.g. `stacked(red_block, blue_block)` during inference, we add the logits from MLPs responsible for both objects to make the final prediction.

F.2 ViT Baseline

The ViT baseline uses ViT-B/32 backbone, with 12 layers of 12-head self-attention layers. The width of the model (dimension of token embeddings) is set to 768. This ViT model passes the embedding from a single trainable classification token through a 2-layer MLP with 512 hidden units to predict all 52 predicates.

F.3 ViT Multihead Baseline

The ViT multihead baseline also uses ViT-B/32 backbone with the same architecture as the ViT baseline, except that it has 4 trainable classification tokens which gives 4 embedding vectors. Then, the same predicate classifier for SORNet is used to predict unary and binary predicate values.

F.4 SORNet

SORNet uses the same backbone architecture as the two baselines above. The canonical object views are linearly-projected and flattened just like the context patches from the input image. The predicate classifier takes the token vectors corresponding to the canonical object views from the topmost layer of the transformer and outputs predicate values. Each MLP in the predicate classifier has 512 hidden units and outputs a single scalar.

F.5 Training Hyperparameters

All models are trained using binary cross-entropy loss with SGD optimizer (momentum set to 0.9) on 4 GPUs with 32G memory. The batch size for a single GPU is 512. The ResNet baseline uses learning rate 0.01 while the ViT baslines and SORNet uses learning rate 0.0001. All models are trained for 80 epochs.

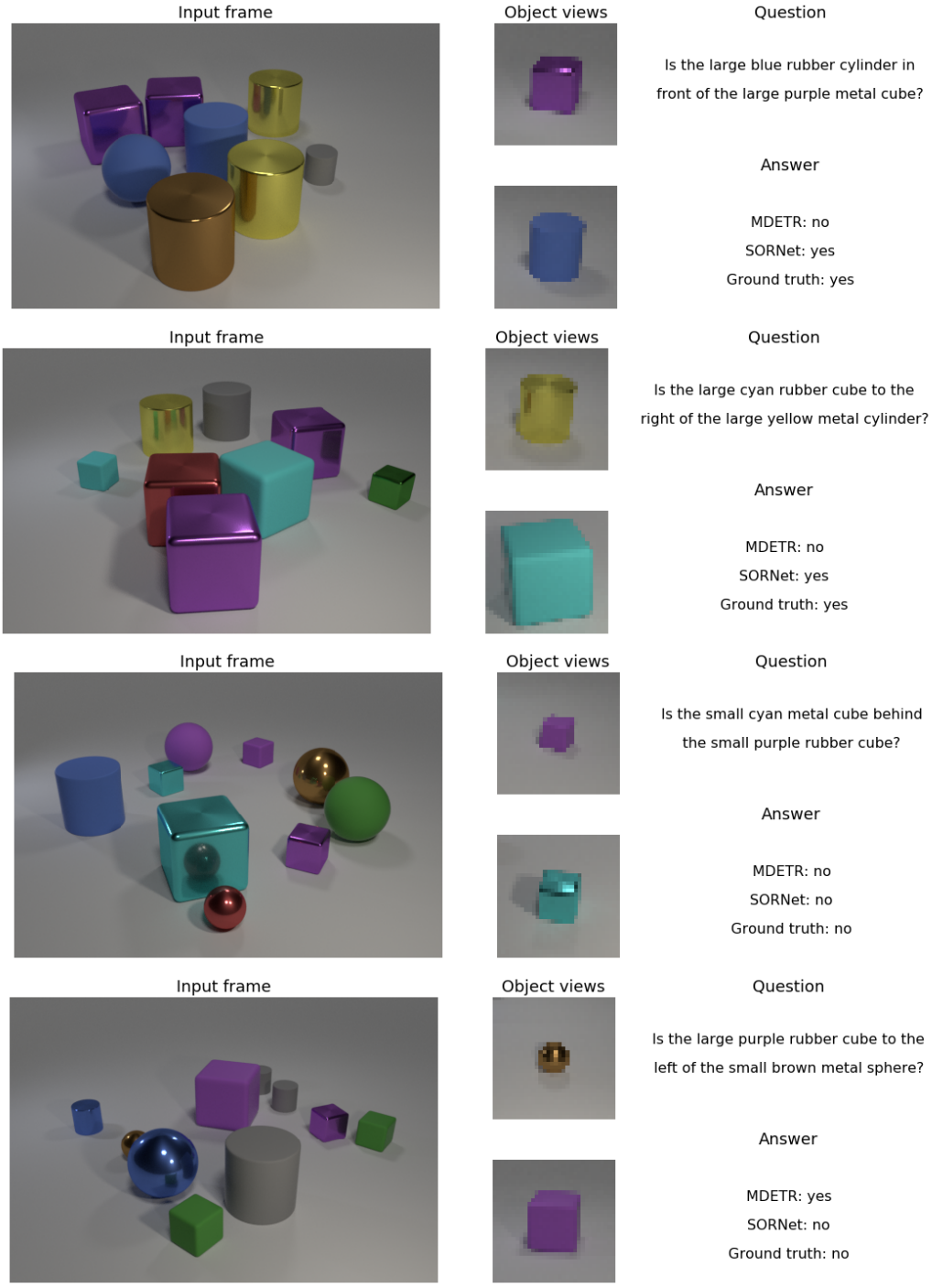


Figure 2: relation classification in CLEVR-CoGenT. In addition to color, SORNet is able to disguish objects based on shape, size and material. It is also able to deal with heavy occlusion. Note that the object patches provided to SORNet can have very different appearance than the corresponding objects in the input frame.

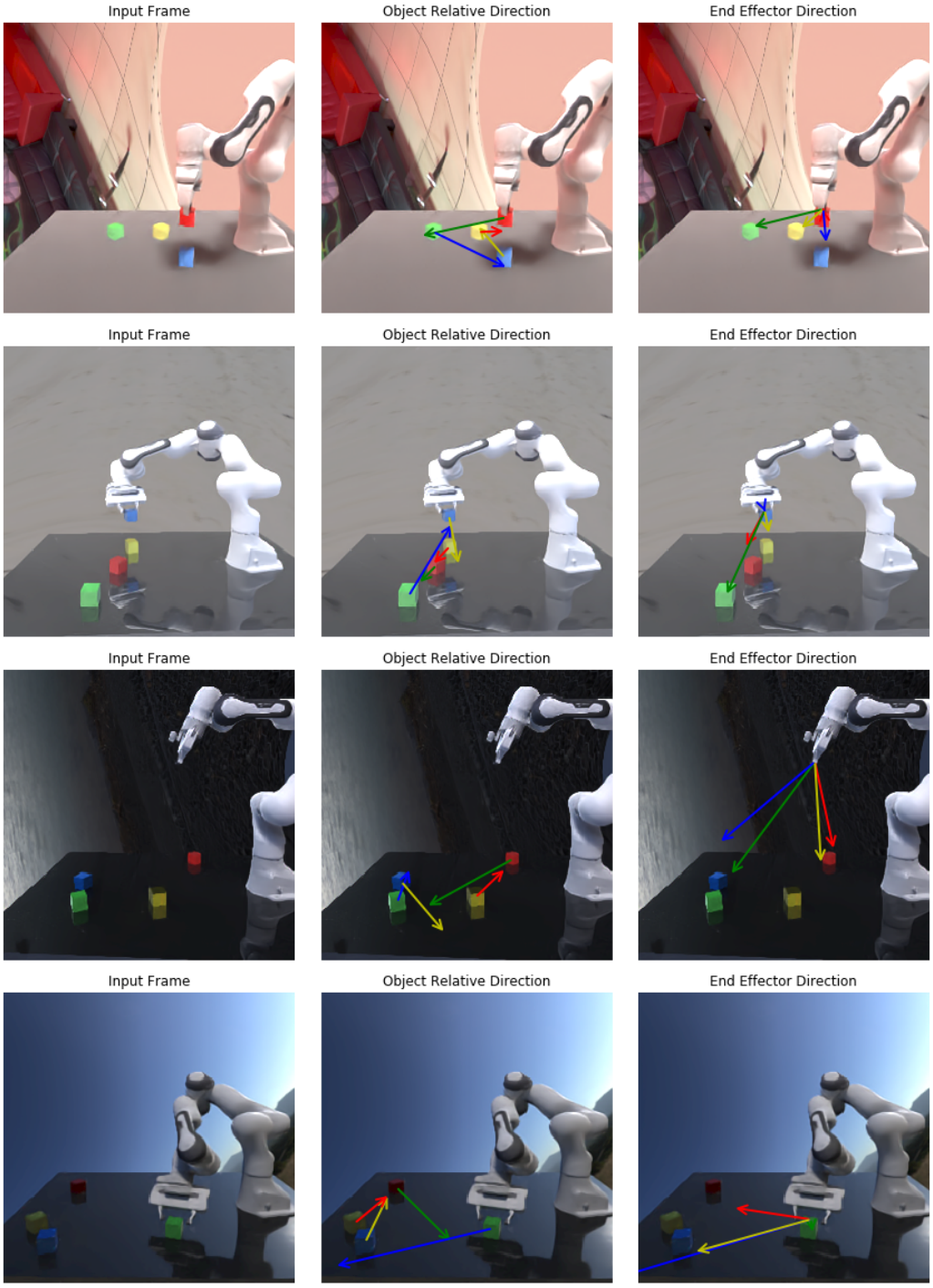


Figure 3: Relative Direction Prediction using pretrained SORNet embeddings. The color of the arrow corresponds to the target and the origin of the arrow correspond to the source. The length of the arrow corresponds to the predicted distance. The second row shows relative direction between object pairs (e.g. red block to green block). Not all predictions are visualized to keep the plot clean. The third row shows the relative direction from the end effector to the objects. The model's prediction is accurate most of the time but be less accurate on challenging cases such as the highly reflective tabletop in the bottom example.

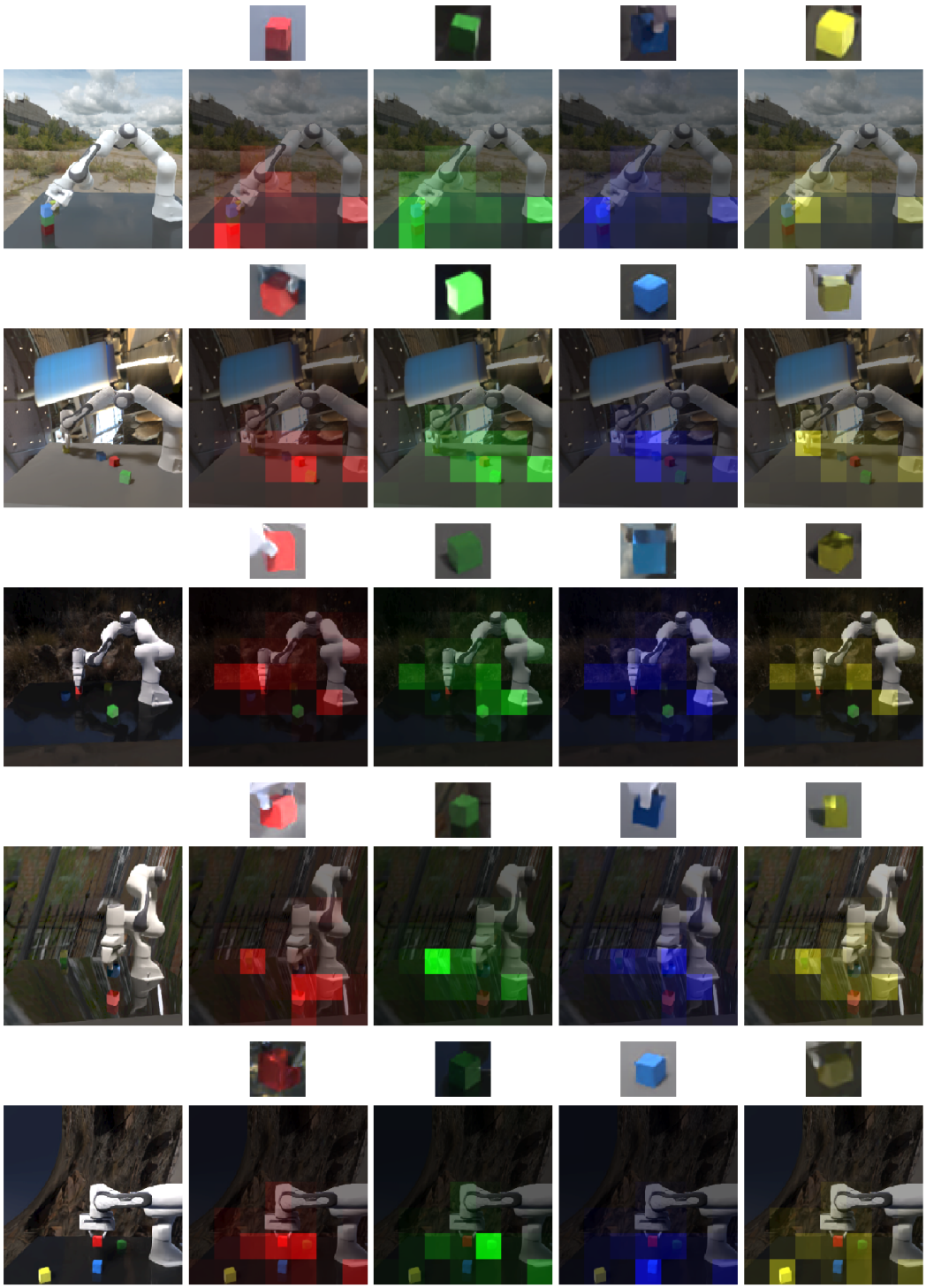


Figure 4: Visualization of the attention learnt by SORNet. Leftmost column is the input frame and remaining columns visualize the attention weights of the object tokens over the context patches. The corresponding canonical object view is shown on top of each attention visualization. We can see that SORNet learns to attend to not only the object of interest but also the robot and other objects as well, while ignoring irrelevant background.

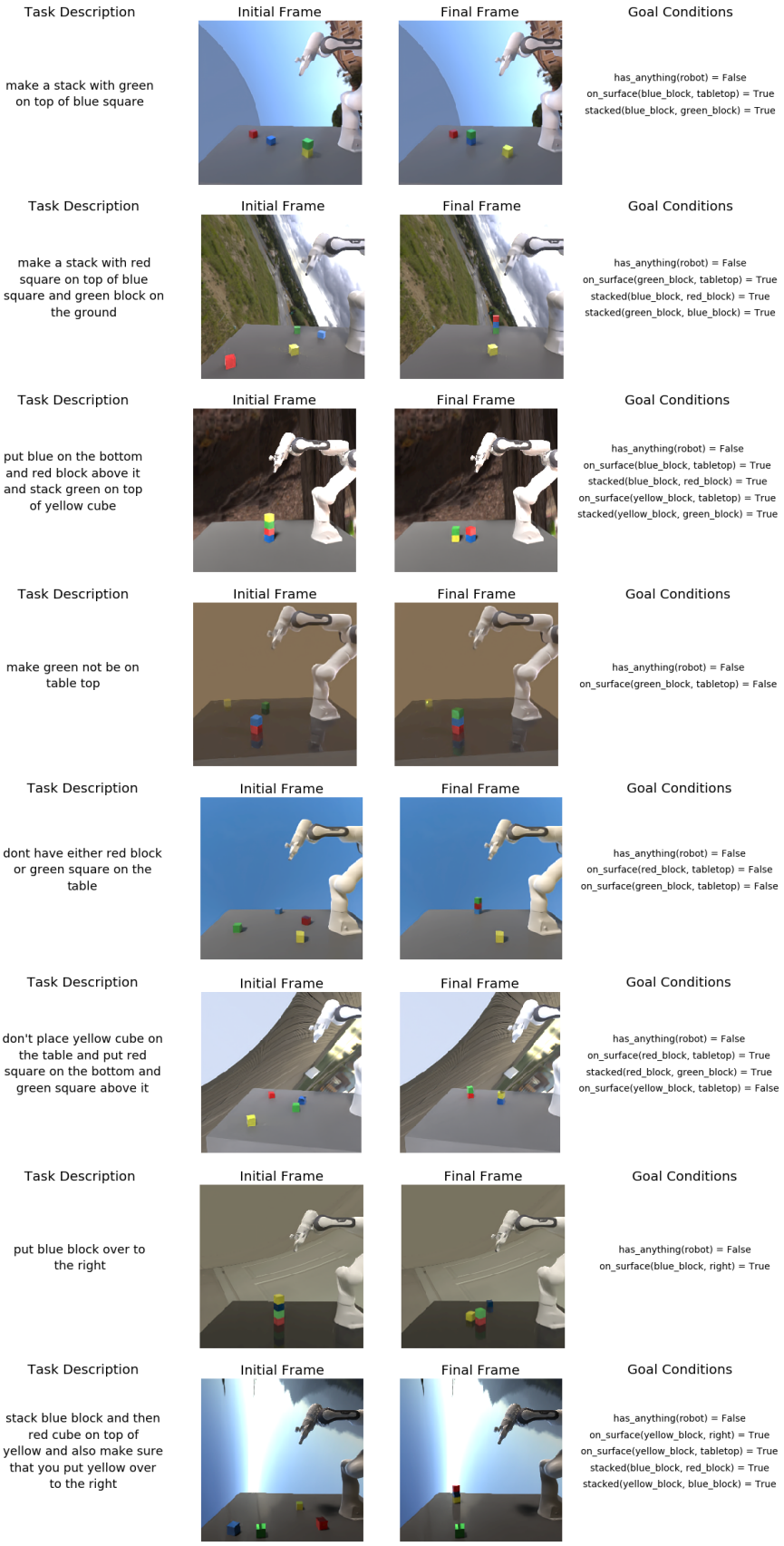
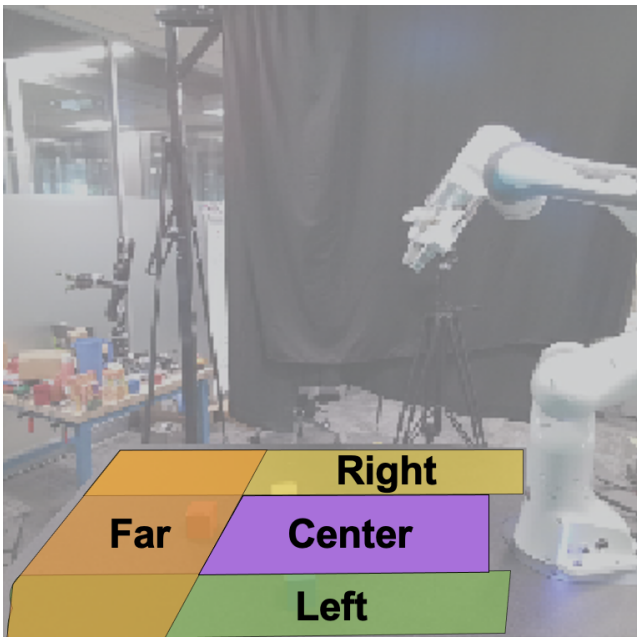


Figure 5: Visualization of the different tasks in the Leonardo test set.



(a) Table regions

on_surface(red_block, left)	has_obj(robot, red_block)
on_surface(red_block, right)	has_obj(robot, green_block)
on_surface(red_block, far)	has_obj(robot, blue_block)
on_surface(red_block, center)	has_obj(robot, yellow_block)
on_surface(green_block, left)	
on_surface(green_block, right)	top_is_clear(red_block)
on_surface(green_block, far)	top_is_clear(green_block)
on_surface(green_block, center)	top_is_clear(blue_block)
on_surface(blue_block, left)	top_is_clear(yellow_block)
on_surface(blue_block, right)	
on_surface(blue_block, far)	in_approach_region(robot, red_block)
on_surface(blue_block, center)	in_approach_region(robot, green_block)
on_surface(yellow_block, left)	in_approach_region(robot, blue_block)
on_surface(yellow_block, right)	in_approach_region(robot, yellow_block)
on_surface(yellow_block, far)	
on_surface(yellow_block, center)	

stacked(red_block, green_block)	aligned_with(red_block, green_block)
stacked(red_block, yellow_block)	aligned_with(red_block, yellow_block)
stacked(red_block, blue_block)	aligned_with(red_block, blue_block)
stacked(green_block, yellow_block)	aligned_with(green_block, yellow_block)
stacked(green_block, blue_block)	aligned_with(green_block, blue_block)
stacked(green_block, red_block)	aligned_with(green_block, red_block)
stacked(blue_block, green_block)	aligned_with(blue_block, green_block)
stacked(blue_block, yellow_block)	aligned_with(blue_block, yellow_block)
stacked(blue_block, red_block)	aligned_with(blue_block, red_block)
stacked(yellow_block, green_block)	aligned_with(yellow_block, green_block)
stacked(yellow_block, blue_block)	aligned_with(yellow_block, blue_block)
stacked(yellow_block, red_block)	aligned_with(yellow_block, red_block)

(b) Predicatets in a 4-object scene

Table 1: List of 52 predicates that capture spatial relationships and skill preconditions for rearrangement tasks with 4 objects (red block, green block, blue block, yellow block), on a table divided into 4 regions (left, right, far and center) with respect to the robot