SIR: Structured Image Representations for Explainable Robot Learning

Paul Mattes, Jan Schwab, Jens Oliver Bosch, Maximilian Xiling Li,
Nils Blank, Minh-Trung Tang, Moritz Haberland and Rudolf Lioutikov
Intuitive Robots Lab, Karlsruhe Institute of Technology, Germany
paul.mattes@kit.edu

Abstract

*Existing robot policies based on learned visual embeddings lack explicit structure and are sensitive to visual distractions. Thus, the representations that drive their behaviour are often opaque, making their decision-making process difficult to interpret. To address this, we introduce **S**tructured **I**mage **R**epresentations (SIR), a method that leverages Scene Graphs (SGs) as an intermediate representation for robot policy learning. Our approach first constructs a fully connected graph, using 2D or 3D image-derived features as initial node representations. Then, a module learns to sparsify this graph end-to-end, creating a minimal, task-relevant sub-graph that is passed to the action generation model. This process makes our model intrinsically explainable. Evaluations on RoboCasa show that our sparse graph policies outperform image-based baselines on average with 19.5% vs 14.81% success rate. We also demonstrate that our graph-based representations are significantly more robust to distractor objects, showing almost no performance degradation, as opposed to image representations. Most importantly, we show that the learned sparse graphs are a powerful tool for model analysis. By analysing when the model's sub-graph deviates from human expectation, such as by including distractor nodes or omitting key objects, we successfully uncover dataset biases, including spurious correlations and positional biases.*

1. Introduction

Imitation Learning (IL) [1, 19] has witnessed significant advancements in robotics in recent years, primarily driven by the emergence of attention-based [34] and diffusion-based [32] methods. Goal-Conditioned Imitation Learning (GCIL) [4] particularly benefits from these enhanced methods, enabling it to perform a wide range of tasks based on language goals [15, 20, 26, 27, 29]. Concurrently, real-world robot agents need to be able to act in more complex environments with a greater number of observed objects [2, 12, 18, 44].

These developments create a growing need for structured and expressive scene representations in robot learning [7]. Most existing approaches rely on learned visual embeddings, often extracted from convolutional backbones or vision foundation models [18, 27, 28, 44]. While such embeddings provide a compact encoding of visual information, they remain opaque and lack explicit structure. This makes them difficult to interpret, offering no clear explainability with respect to a generated decision. We propose **Structured Image Representations (SIR) for Explainable Robot Learning**, which addresses this challenge by leveraging Scene Graphs (SGs) to introduce a structured image representation usable in Goal-Conditioned Imitation Learning (GCIL). SGs provide a unifying representation that can incorporate diverse modalities extracted from perception, including symbolic information (e.g., object labels), geometric cues (e.g., bounding boxes and point clouds), and high-level image features. By capturing the environment in this structured, relational form, SGs present a data structure that make robot behaviour more interpretable than image-based approaches. SIR further enhances the explainability of SGs by operating on a learned sparsified graph. This sparsification process creates a minimal, task-relevant sub-graph, offering a clear insight into exactly which objects and interactions the model considers critical for its decision. SGs generated this way thus offer a compact, expressive, and highly interpretable intermediate state representation.

Our GCIL model, SIR, first generates a fully connected SG of the scene. A trainable module then sparsifies this graph into a sub-graph by keeping only the nodes the model considers important for the task. Finally, a two-layer Graph Neural Network (GNN) embeds this sparse sub-graph to produce the final scene representation, which the action generation model uses as input. To showcase the effectiveness of SIRs representations, we evaluate two action generators: Multimodal Diffusion Transformer (MDT) [27], which provides SOTA performance and adaptability, and a Behavior Cloning (BC)-Transformer as a baseline. The overall method can be seen in Figure 1. The contributions

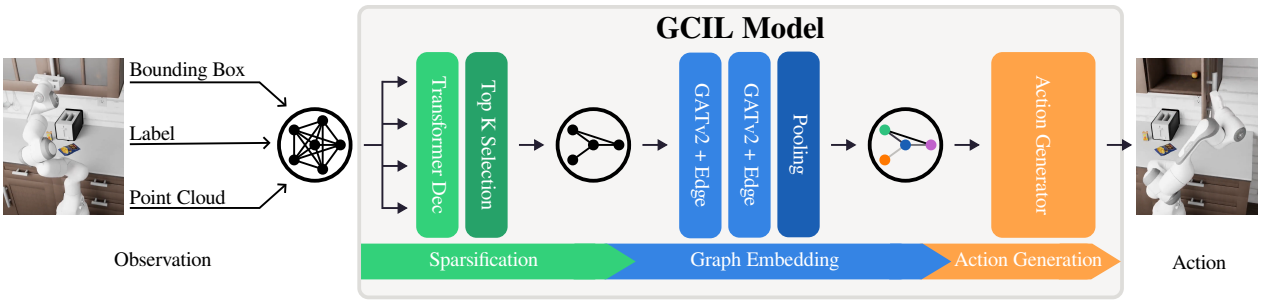


Figure 1. First, SIR extracts image features to generate an initial, fully connected SG, using these features as node representations. This Fully-Connected-Graph (FC-Graph) is then passed to the Goal-Conditioned Imitation Learning (GCIL) model. Within the model, a learnable sparsification module first prunes the Fully-Connected-Graph (FC-Graph) into a minimal, task-relevant sub-graph, retaining only the nodes the model deems important for action generation. This sparse sub-graph is then processed by a two-layer Graph Neural Network (GNN) that employs global average pooling to create a final graph embedding. This embedding serves as the state representation for the downstream action generation model.

of this paper are twofold:

- We conduct a thorough investigation into how SGs can be used as an effective scene representation for robot learning. As part of this, we systematically analyse which image modalities serve as effective initial node representations.
- We propose a method for learnable SG sparsification and analyse the resulting sub-graphs to provide insights into the model's decision-making process. This analysis allows us to investigate how the model perceives the scene by examining whether it correctly selects task-relevant objects and whether it includes irrelevant ones.

Based on these observations, we formulate four research questions:

RQ1: How do SG embeddings compare to image embeddings as model input in GCIL?

RQ2: How does the choice of the initial node representation (e.g., symbolic labels, geometric cues, or visual features) affect the performance of SG-based models?

RQ3: How robust are models that use SG embeddings compared to image embeddings in regard to distractor objects?

RQ4: Do sparse SGs facilitate the interpretability and analysis of the decision-making process of the respective behaviour model?

We evaluate SIR in RoboCasa [17, 42] and CALVIN [16], a language conditioned imitation learning benchmark for long-horizon manipulation. Our results demonstrate that policies using Fully-Connected-Graphs (FC-Graphs) as input already achieve performance comparable or superior to image-based models. Furthermore, our learnable sparsification method not only enhances explainability but also yields an additional improvement in overall performance. We also find that graph-based representations are significantly more robust to distractor objects introduced during inference. Finally, the resulting sparse graphs provide a direct mechanism for the interpretability of the model behaviour, enabling analysis based on the specific nodes the model includes or excludes. Critically, SIR is not a post-hoc explainability method, but intrinsically explainable because the sparse graph serves as a learned intermediate representation during action generation. This characteristic offers significantly greater potential for interpretability and analysis of the model's decision-making process.

2. Related Work

Current graph-based IL methods are often plan-based [3, 13, 33, 43] or treat graphs as auxiliary input [6, 31]. We instead investigate how structured scene representations, in the form of SGs, can serve as the direct state for robot learning. To our knowledge, SIR is the first approach to enable GCIL to solve complex, everyday kitchen tasks using SGs as the direct observation representation, without relying on planning. This contrasts with recent methods, like Instant-Policy [35], which uses graphs within a diffusion framework but is restricted to point-cloud embeddings. *Compose by Focus* [23] uses SGs as direct input, but only for simple 3-4 node graphs, relies solely on point cloud values, and is evaluated on simple manipulation tasks. In contrast, SIR works on larger, fully-connected or reduced SGs, integrates diverse image modalities as initial node representations, and is validated on complex everyday kitchen tasks.

2.1. Graph-Based Imitation Learning through Planning

Graphs have been used for 3D object prediction, e.g. the robot has to push a block into a desired position [33]. More challenging tasks include real world pick and place, where a combination of symbolic and geometric image-based graphs are used [43]. The symbolic graph is grounded through the geometric graph. Furthermore, graphs can include object and goal nodes to predict which object interaction is needed to achieve a specific goal [3, 13]. An-

other approach, called ConceptGraphs [7], builds 3D SGs to input into an Large Language Model (LLM) for downstream plan-based task execution. It appears that many graph-based IL approaches in robotics involve some sort of planning algorithm, like Task and Motion Planning (TAMP) or something similar [3, 7, 13, 33, 43]. The plan-based approaches use high-level information graphs, that need specific key-points to update the graphs during execution. Otherwise, their internal information would not change. These key-points are much easier to identify using planning algorithms. SIR demonstrates that graphs work for step-based GCIL, using low-level information graphs without the need of plan-based algorithms or specific key-points.

2.2. Graph-Based Step-Wise Imitation Learning

Step-wise IL using graphs is an under-researched field. One method uses graphs for visual imitation, where hand movement is mapped to a graph for grasping and reaching improvements [30]. Another approach uses graphs for swarm movement to learn the underlying interaction mechanism [11]. A third approach, Instant Policy, uses graphs for In-Context Imitation Learning (ICIL) in everyday robotic tasks [35], modelling it as a graph generation problem with a learned diffusion process. The fourth approach, called *Compose by Focus*, uses SGs as a direct scene representation and input to a diffusion model, focusing on simple graphs (3-4 nodes) with point cloud features as node embeddings [23]. In contrast, SIR does not use the graph as a direct part of behaviour learning. Instead, we use the graph as an intermediate representation to build a more structured scene understanding. Our work is the first to systematically investigate how SGs perform when using different image-derived modalities as initial node features. Furthermore, SIR is trained to predict the important sub-graph, thereby learning an explainable and sparse representation, which can be used for further downstream analysis.

2.3. Learning Graph Sparsification

Large graphs are reduced for various reasons, including storage [36], runtime [40], out-of-distribution handling [38], or interpretability [41]. These *graph reduction* techniques are categorized as *sparsification*, *condensation* or *coarsening* [8]. Condensation and coarsening are not suitable for interpretability, as the resulting graph's relation to the original is unclear. Sparsification methods typically focus on removing edges, not nodes [25, 37, 41]. This is inadequate for interpretability in graph classification/regression tasks. Removing only edges, especially in fully-connected graphs, does not significantly impact GNN performance compared to removing nodes [25]. While some GNN pooling layers (e.g., Differentiable Pooling (DiffPool) [39], Graph Explicit Pooling (GrePool) [14], Self-Attention Graph Pooling (SAGPool) [10]) select nodes, they do so after or between message passing layers. This is not truly interpretable, as information from unselected nodes may have already propagated into the final graph embedding. Therefore, we cannot be certain which nodes contributed most significantly. To the best of our knowledge, this work is the first to learn end-to-end node removal before message passing, ensuring that removed nodes have no impact on the final graph embedding.

3. Method

The architecture of SIR is illustrated in Figure 1. It includes four core components: (1) SG extraction from a given image, (2) graph sparsification, (3) graph embedding generation and (4) action generation. The resulting graph embedding serves as the state representation for a downstream action generation model, which, in this work, is instantiated as either MDT [27] or a BC-Transformer [15]. The three modules inside the GCIL model are trained end-to-end, while the scene graph extraction step is frozen.

3.1. Scene Graph Generation

We construct the initial SGs by extracting all objects in a scene using either ground-truth or predicted segmentation masks. These objects are then constituted as the nodes in a FC-Graph, where the initial node representations are derived from the given RGB or RGB-D image. We investigate four primary feature modalities:

- **Label:** The object label as a one-hot encoding.
- **Cropped-Image-Feature:** A visual embedding from a pre-trained network, generated from the BB crop.
- **BB-Coordinates:** The 2D bounding box corner and centre coordinates, normalized in pixel space.
- **Point-Cloud-Feature:** An embedding from a pre-trained network, applied to the object's associated point cloud.

A key aspect is that these feature modalities can be easily concatenated as the initial node embedding in the graph itself by design. When including either bounding box or point cloud information, the edge features between nodes are initialised with geometric distance. Otherwise, the edge features are initialised with 1, in order to aid the message passing in the later graph embedding stage.

3.2. Scene Graph Sparsification

Although graphs provide the scene information in a more structured way than images, a FC-Graph contains all nodes available in the scene, reducing its interpretability. We therefore aim to extract the most relevant sub-graph for a specific task. Since the information in this extracted sub-graph is the only information about the scene available to the action generation model, this sub-graph serves as the explanation for the generated actions. To extract task-relevant SGs, SIR calculates a score for every node and uses the

highest scored nodes of a graph. To predict these node scores (NSs), we use a two-layer Transformer-Decoder [34] architecture with four heads per layer. Every layer employs Adaptive Layer Normalization (AdaLN) [21], which conditions the node embeddings on the language goal. We will refer to this module as *FiLMDecoder*. We define the resulting node weight (NW) as

$$\text{NW}(n) = \begin{cases} \text{NS}(n), & \text{if n is selected in the sub-graph.} \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

for a node n based on the score $\text{NS}(n)$ predicted by the FiLMDecoder. To prevent a collapse where all node scores converge to a similar value, we introduce a *soft histogram loss*. This loss encourages the predicted scores to be uniformly distributed in the [0,1] range. Instead of hard binning, our method employs a Gaussian kernel to softly assign each score to multiple histogram bins. These assignments are summed to form a differentiable soft histogram, which is then normalized. Finally, we compute the Mean Squared Error (MSE) between this soft histogram and a uniform target distribution. During training, we apply a weight of 0.1 to this loss term. The nodes with the highest scores are then chosen for the sub-graph using instruction-grounded node selection. Specifically, we select the top-k highest-scoring nodes from the graph, where k is a task-specific parameter chosen according to the number of task-relevant objects. Additionally, the model is guided with an additional L1 loss applied to the node weights, encouraging NW(n) to be high for instruction-relevant nodes and low for irrelevant ones.

3.3. Scene Graph Embedding Generation

We generate the SG embedding using a GNN composed of two Graph Attention v2 (GATv2) residual layers, followed by a global average pooling layer. The GATv2 layers update node features by weighting information from neighbours using learned attention scores. The final graph-level representation is obtained by averaging the features of all nodes after the propagation steps. Consequently, this global average pooling aggregates features from all nodes into the final graph embedding, including those that may have received low attention weights during the graph propagation.

Differentiability Further, we adapt GATv2 to include edge weights. The message passed along an edge is not only multiplied by the attention score but also by the edge weight. To enable learning the FiLMDecoder end-to-end, we (1) keep the gradient of $\text{NS}(n)$ as the gradient of $\text{NW}(n)$, (2) include the node weights in the edge weights $\text{EdgeWeight}(u, v) = \text{NW}(u) \cdot \text{NW}(v)$, and (3) include the node scores in the pooling step. Point (2) ensures that during message passing, no information leaves the "removed" nodes. Point (3) ensures that during the pooling step, no information is passed into the graph embedding, while explicitly including $\text{NS}(n)$ improves gradient flow and ensures improved learning of the FiLMDecoder:

$$GraphEmbedding = \frac{\sum_{n \in V} \text{NW}(n) * X_n}{\sum_{n \in V} 1_{\text{NW}(n)>0}}, \tag{2}$$

where X_n describes the final node feature of node n. Regarding the pooled features, this is equal to mean-pooling over the kept nodes with $\text{NW}(n) > 0$.

3.4. Action Generation

Actions are generated using the given downstream action generation model, which gets as input the embedded SG and an embedded language goal using CLIP [24]. Both action generator models, MDT and BC-Transformer, use one observation to generate 10 future actions.

4. Experiments

We conduct experiments on the RoboCasa [17] and CALVIN [16] benchmarks with two distinct action generation models: MDT [27] and a BC-Transformer [15]. For each benchmark, we trained each model configuration twice with different seeds and perform evaluation over 100 rollouts. The 24 atomic tasks of RoboCasa [17] are evaluated using the standard groups: *Pick and Place*, *Drawers*, *Doors*, *Buttons*, *Levers*, *Knobs*, and *Insertion*, as well as the overall 24-task average. For the CALVIN benchmark, we evaluate on the D $\rightarrow$ D setting. To ensure a fair and controlled comparison between graph-based and baseline methods, we do not use the full sensory information described in RoboCasa or CALVIN. Instead, we focus on a subset of information provided by the static cameras, to avoid biasing results with too many different input features.

Baselines We evaluate one main baseline method, which incorporates images as observation input. Four additional baselines used in ablations leverage images, point clouds or both as observation features. All five methods are directly comparable to the graph-based representation of using either image or point cloud features. Baseline models using images as observation input use a pre-trained ResNet18 [9] to generate embeddings, which is fine-tuned during training. The model denoted with "Own pre-trained Image" uses the image encoder, which was trained for embedding the cropped bounding boxes for the graph node features and is also fine-tuned during training. Baseline models using point cloud observations input it patch-wise into the action generation model as described in FPV-Net [5]. Evaluations using the same embedding network for point clouds as we did for the initial node features of our graphs resulted in worse performance. Therefore, we kept the better results of the baseline. More detailed information can be found in Section 7.1.

Table 1. The results represent success rate in percent on RoboCasa [17] over 100 rollouts with 2 model versions trained on different seeds. The numbers behind the task category names indicate the number of tasks per category. Graph Methods use Cropped-Image-Features and BB-Coordinates as initial node representations.

Observation	Pick/Place (8)	Doors (4)	Drawers (2)	Knobs (2)	Levers (3)	Buttons (3)	Insert (2)	**Avg (24)**
Image (Baseline)	**1.19 ± 0.44**	25.13 ± 0.88	**49.75 ± 0.75**	7.25 ± 3.75	23.67 ± 0.34	17.00 ± 0.33	**4.75 ± 0.25**	14.81 ± 0.02
Fully-Connected-Graph	0.06 ± 0.03	28.62 ± 4.25	39.25 ± 1.75	14.00 ± 0.50	40.00 ± 2.67	18.83 ± 0.17	**4.75 ± 0.75**	16.98 ± 0.85
SIR (Ours)	0.13 ± 0.00	**30.25 ± 0.25**	46.25 ± 1.75	**16.50 ± 0.00**	**48.50 ± 2.17**	**21.83 ± 1.84**	**4.75 ± 2.25**	**19.50 ± 0.33**

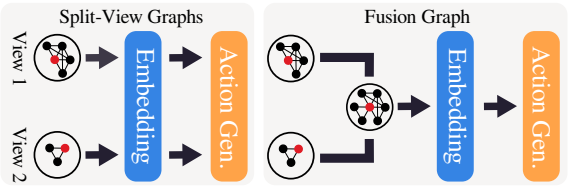


Figure 2. Multiple views of a scene observation can be handled through single embeddings using the Split-View approach or merging node features in a single graph to get one embedding for all views.

Graph-based observations For RoboCasa-based experiments, graphs are generated from the static left and right camera observations. We explore two processing strategies: embedding them separately using two GNNs (Split-View Graph), or concatenating node features from both views into a single "Fusion Graph" processed by one GNN. Both approaches are shown in Figure 2 and can be adapted to any multi-view benchmark using static cameras. In contrast, the CALVIN environment only includes one static camera, thus resulting in a single graph. SIR can either use the extracted FC-Graph of the image or the sparsified sub-graph to generate the graph embeddings. Initial node features for the SGs are generated using the following methods: The Cropped-Image-Feature is generated by first cropping the object's bounding box from the image. This crop is then encoded using a ResNet18 backbone, which was pre-trained on an BB-image reconstruction task. Similarly, the Point-Cloud-Feature is embedded using a pre-trained PointNet [22] architecture. This network was trained on an object point cloud reconstruction task, using a Chamfer distance loss. Label information is one-hot encoded and depends on the maximum number of objects in the scene. BB-Coordinates are a normalized vector based on pixel-space including all four corner coordinates and the centre point. Initial node features for all sparse graph methods are Cropped-Image-Feature and BB-Coordinates, as they achieve the highest performance, while maintaining fast inference speed.

Explainability Evaluation We evaluate our explanation subgraphs qualitatively. During the rollout, we monitor the subgraphs $G_{sub,i}$ extracted by our sparsification method at each time step i. We then construct the explanation $G_{expl} = (V_{G_{expl}}, E_{G_{expl}})$ for a rollout (i.e., "rollout-explanation") as the mean of the extracted subgraphs. Therefore, we calculate the percentage $p_{p,n}$ that a node (edge) n was present in the subgraph as

$$p_{p,n} = \frac{\text{Number of subgraphs } n \text{ was present in}}{\text{Number of scene graphs } n \text{ was present in}}. \quad (3)$$

The more all $p_{p,n}$ (for $n \in V_{G_{expl}} \cup E_{G_{expl}}$) converge to $\{0, 1\}$, the more consistent is the explanation throughout the rollout. The same way, we can construct an explanation subgraph for a task (i.e., "task-explanation"), by considering all time steps of all rollouts of the task.

5. Results and Discussion

Selected results for the evaluation of the baselines and graph-based models on RoboCasa using MDT are displayed in Table 1. All graph-based models use Cropped-Image-Features and BB-Coordinates as initial node embeddings, as well as distance based edge features. Ablations with other initial node embeddings are shown in Table 3. The full list of single tasks per category can be found in Section 8.5. Results for the BC model, using generated segmentation masks and CALVIN can be found in Sec. 8.2, Sec. 8.1 and Sec. 8.3.

5.1. Scene Graphs as Observations

In RoboCasa, image-based models achieve a 14.81% average success rate, as shown in Table 1. FC-Graph-based models, without sparsification, reach nearly 17%, and SIR with instruction-grounded sparsification widens this gap, achieving 19.5%. Additional results in Table 3 confirm this performance gap, even when using different observation types, such as point clouds or their combination with images. However, the performance gains are not uniform across all task categories. While graph-based models show significant improvements in most settings (e.g., *Doors*, *Levers*, *Knobs*, and *Buttons*), they do not surpass the image-based baseline in the *Drawers* or *Pick and Place* task. We assume this is due to heavy dataset biases present for some tasks, as investigated in Section 5.4. These observations, together with results from Sec. 8.2, answer **RQ1**: Graph-embeddings outperform image-embeddings across diverse task settings.

Table 2. Ablation results represent success rate in percent on RoboCasa [17] over 100 rollouts with 2 model versions trained on different seeds. Ablations consider different sparsification approaches.

Sparsification Method	Avg (24)
Random Node Removal	5.48 ± 0.19
Naive NR (**no** soft histogram loss)	9.60 ± 1.73
Threshold	17.17 ± 0.38
TopK	18.44 ± 0.77
SIR	$\mathbf{19.50 \pm 0.33}$

5.2. Ablations

Ablation results of initial graph node representations is detailed in Table 3. We compare the graph-based models to baselines using corresponding input features, where applicable. Our results show that the best-performing node representations are Cropped-Image-Features, either alone (16.65%) or in combination with BB-Coordinates (15.90% or 16.98% for the Fusion Graph). These methods outperform the standard image baseline (14.81%) and the FiLM-conditioned image baseline on the language goals (15.85%). A significant gap is evident when using point cloud data. The baseline model using only point clouds achieves just 4.13%, and 13.25% when combined with images. In contrast, the graph-based counterparts are far more effective, reaching 11.08% for Point-Cloud-Features alone and 15.04% for the combination. These observations indicate that GNNs are a more efficient architecture for integrating point cloud information as node features compared to inputting it directly into the action generation model [5]. Therefore, graph-based models can effectively integrate diverse node representations and, in doing so, outperform their corresponding baselines, answering **RQ2**.

We further compare our instruction-guided sparsification method to simpler sparsification methods in Table 2. For random node removal, we simply remove random nodes from the graph. In Naive NR, node removal is learned without the soft histogram loss. We further compare SIR to Threshold node removal, where all nodes with a score over a specified threshold are retained. In TopK, we do not employ a task-specific k and do not guide the node weights with an L1 loss. Overall, SIR with instruction-grounded sparsification outperforms other sparsification methods, whereas the soft-histogram loss has the highest impact on performance.

5.3. Distractor Objects

We further evaluate the robustness of SGs compared to images with respect to multiple distractor objects present in the scene. In particular, we place between 3 and 9 additional objects in the environment. The results are displayed in Figure 3, with more detailed results in Table 7 in the Appendix. Including distractor objects results in a clear performance decrease for the image baseline, which drops by 3.3% on average. A similar drop of 2.9% is seen for our model using TopK node removal. In contrast, SIR, the FC-Graph-based model and the Threshold model show almost no performance degradation on average when distractor objects are introduced. In fact, these models even show slight performance increases in some task settings, such as *Drawers*, *Knobs*, and *Levers*. However, all models, including the graph-based ones, decrease in performance on the *Doors* task. This provides a clear answer to **RQ3**: Image-based models are sensitive to novel distractor objects, whereas SIR, FC-Graph models and Threshold models are highly robust, achieving similar performance as before.

Table 3. Ablation results represent success rate in percent on RoboCasa [17] over 100 rollouts with 2 model versions trained on different seeds. Ablations consider different image features used as input for the action generation model or as initial node features for the SGs.

Observation	Avg (24)
Baselines	
Image	14.81 ± 0.02
Image + FiLM	15.85 ± 0.29
Own Pretrained Image	10.11 ± 0.76
Point Clouds	4.13 ± 0.13
Image + Point Clouds	13.25 ± 0.15
Split-View Graph - Fully-Connected	
Cropped-Img	16.65 ± 0.23
BB-Coordinates + Label	10.98 ± 0.44
BB-Coordinates + Cropped-Img	15.90 ± 0.32
Point Clouds	11.08 ± 0.29
Label + Point Clouds	12.50 ± 1.0
Cropped-Img + Point Clouds	15.04 ± 0.09
Fusion Graph - Fully-Connected	
BB-Coordinates + Label	11.06 ± 0.57
BB-Coordinates + Cropped-Img	16.98 ± 0.85

5.4. Explainable Model Behaviour

Learning the sub-graph as an intermediate representation during training enables an analysis of the model's understanding of the current observed scene. We differentiate three main sub-graph types, displayed in Figure 4: (1) Human expected sub-graph, (2) sub-graph with distractor nodes and (3) sub-graph with missing nodes. Our learnable approaches do not consistently produce the human expected sub-graph, because the model itself learns the graph, which can result in deviations. These deviations, which fall into categories (2) and (3), are the primary source of insight, as they allow us to analyse the model's actual decision-making

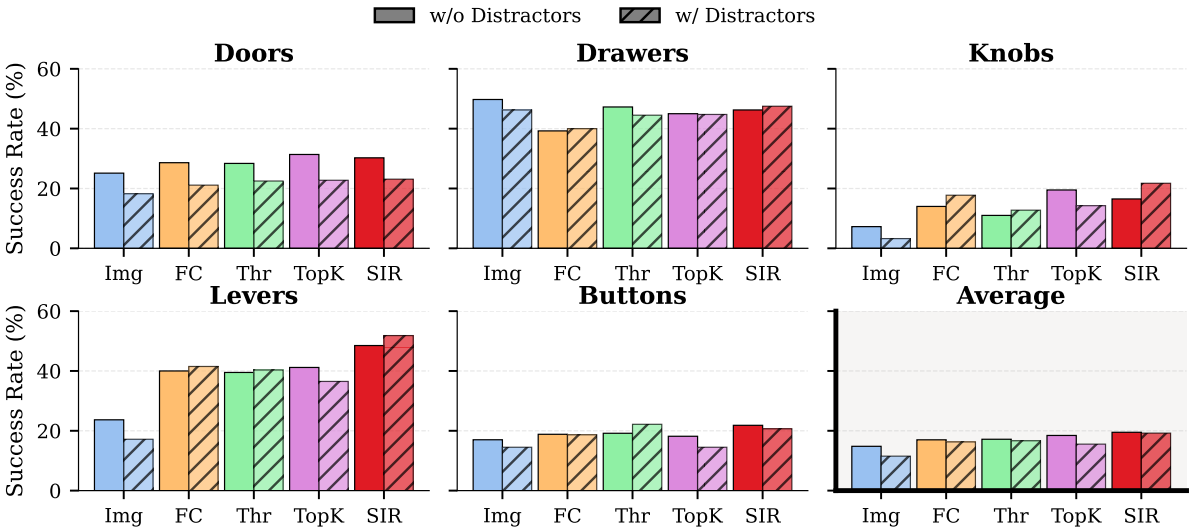


Figure 3. When novel distractor objects, not seen during training, are introduced at inference, the image-based baseline's (Img) performance drops significantly. A similar decrease is seen in our TopK sparsified model. In sharp contrast, the fully-connected (FC), Threshold (Thr) and the SIR graph models demonstrate high robustness, maintaining their average performance. These graph-based models even show performance increases in specific categories, such as Knobs and Levers. More details in Table 7 in the Appendix.

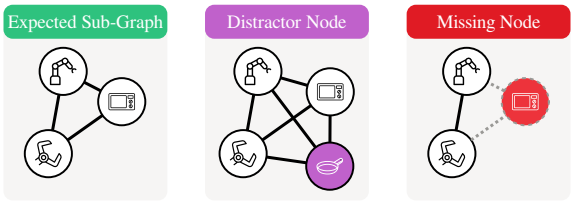


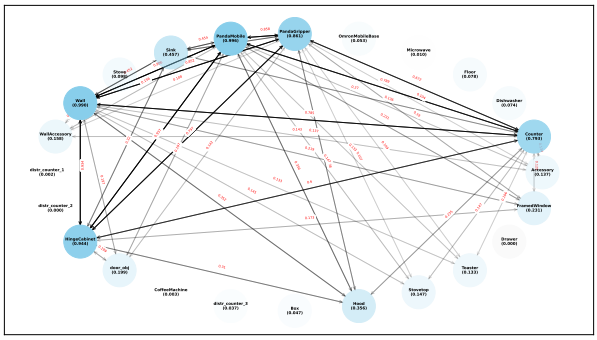
Figure 4. Possible sub-graph generation, when using our explanation graph consistency metric.

process. Figure 5 displays four task-explanation sub-graphs generated using Equation (3). The graphs are based on the 100 rollouts of the specific model on the given task. For the *CloseSingleDoor* task either the door of the cabinet or the microwave has to be closed. In case of the *OpenDoubleDoor*, always two cabinet doors have to be opened. We mainly consider explanation graphs from the TopK approach, as this does not introduce prior biases and truly showcases the models intent. SIR also delivers valuable insights, because even with human biases, the model still produces sub-graphs that are not expected.

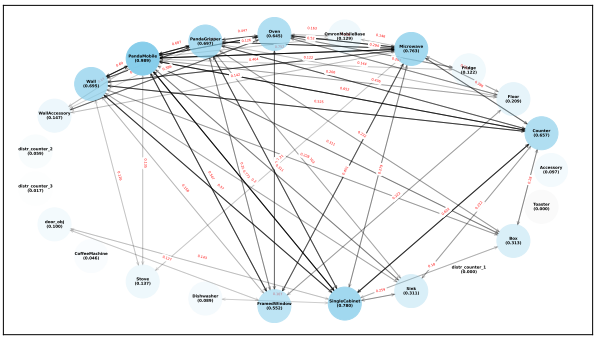
Sub-Graph with Distractor Nodes As seen in Figure 5, the learned sub-graphs often include distractor objects (all sub-figures except Fig. 5d). For instance, Fig. 5a displays the sub-graph for the *OpenDoubleDoor* task, one of the poor-performing task settings. In this case, the overall sub-graph is relatively consistent but includes objects like *Wall* and *Counter*, which are not directly related to opening doors. In Figure 5c, the sub-graph for the *CloseDrawer* task is less consistent, yet the model's performance is high (81% success rate). This model consistently includes unrelated objects, such as the *Oven*, *FramedWindow* and *Microwave*. These observations lead to the assumption that the model exploits **spurious correlations** in the given training data, where seemingly unimportant objects provide enough information to solve the given task.

Sub-Graph with Missing Nodes The sub-graphs for the *CloseSingleDoor* task reveal a compelling insight when comparing the TopK Fig. 5b and SIR Fig. 5d. The TopK model Fig. 5b includes the relevant objects, *SingleCabinet* and *Microwave*, with a high frequency (over 70%), although it also selects various irrelevant nodes. SIR Fig. 5d, however, learns an entirely different sub-graph: it almost exclusively selects the *PandaMobile* and *PandaGripper* nodes. It consistently excludes the primary task objects *SingleCabinet* and *Microwave*. Despite this complete omission of key objects, SIR outperforms the TopK model by over 5%. This result strongly indicates that the dataset contains significant **positional biases**. The model has learned it can succeed by executing a fixed trajectory based only on its own gripper's state, rendering the actual position of the target door obsolete. This also holds for the *CloseDrawer* task in Fig. 5c, where *Drawer* is only included 11% of the time.

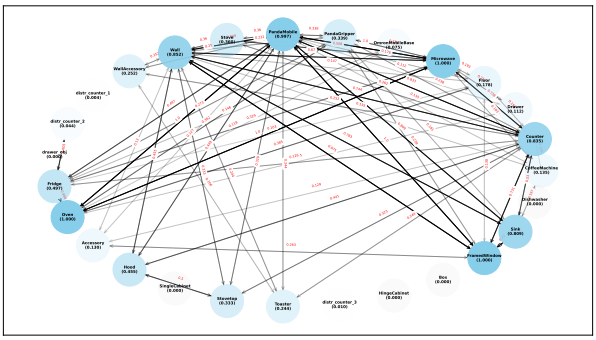
Interpretable Behaviour SIR provides insight into action generation models by learning an intermediate sub-graph end-to-end. The key insights do not come from "correct" sub-graphs, but from their deviations from a human-



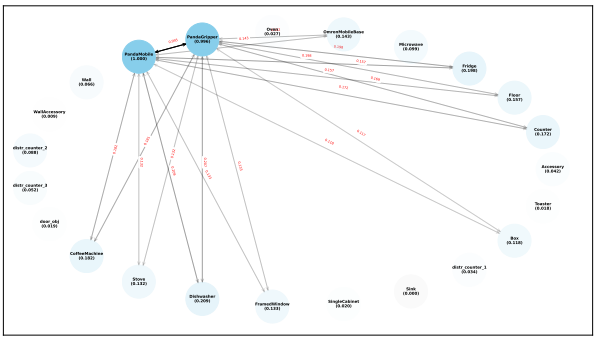
(a) Task-explanation graph for model *TopK Node Removal*, task *OpenDoubleDoor*.



(b) Task-explanation graph for model *TopK Node Removal*, task *CloseSingleDoor*.



(c) Task-explanation graph for model *TopK Node Removal*, task *CloseDrawer*.



(d) Task-explanation graph for model *Instruction-Grounded Node Removal*, task *CloseSingleDoor*.

Figure 5. Comparing Fig. 5a (a hard task) and Fig. 5b (an easier task) shows that the explanation for the harder task is more consistent, suggesting the model learns to identify more informative nodes as task complexity increases. These learned sub-graphs are effective at revealing significant dataset biases. For example, in Fig. 5c, the model includes many unimportant nodes, which indicates that it may be exploiting spurious correlations in the dataset for that task. Similarly, in Fig. 5d, the instruction-grounded model only includes the robot nodes and omits the target objects, which strongly suggests a positional bias in the data.

expected graph. This allows us to evaluate the model's behaviour: Is it succeeding for the right reasons, or is it focusing on unimportant features? Furthermore, when a model succeeds despite a deviating sub-graph, it reveals critical insights into the dataset and how the model has learned to exploit it. These observations are only possible due to SIR's end-to-end nature. This contrasts with methods like a VLM or LLM used to pre-filter objects [23]. A VLM would always include logically important objects and exclude unimportant ones, creating a "clean" sub-graph. This, however, would completely mask the underlying dataset or model biases that SIR's end-to-end learned sub-graphs successfully expose. All these points can be used to answer **RQ4**, clearly demonstrating the effectiveness of our proposed method to understand model and even dataset intrinsic.

6. Conclusion

In this paper, we introduced SIR, a method to generate and use learned, sparsified SGs as an intermediate representation for robot policy learning in GCIL. Our investigation shows that graph-based representations achieve higher average success rates than image-based baselines and are a highly effective architecture for integrating diverse modalities like point clouds. Furthermore, graph-based policies are significantly more robust to distractor objects, showing almost no performance degradation where image-based policies fail. Our most critical finding is that the learned, sparsified sub-graphs serve as a powerful tool for model and dataset debugging. By analysing when the model's graph deviates from human intuition, such as by including distractor nodes or excluding key task-relevant nodes, we successfully identified significant spurious correlations and positional biases in the dataset. This demonstrates that an end-to-end learned, explainable representation like SIR can uncover flaws in training data. Such insights would be completely masked by non-end-to-end methods, like Vision Language Model (VLM) pre-filtering, which would always select the "correct" objects and hide these biases. For future work, we plan to extend our node selection method to allow the model to learn how many nodes are important, rather than relying on heuristics. We also aim to further leverage our explanation-based analysis to evaluate and improve other models and datasets.

References

[1] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009. 1

[2] Open X-Embodiment Collaboration. Open X-Embodiment: Robotic learning datasets and RT-X models. https://arxiv.org/abs/2310.08864, 2023. 1

[3] Francesco Di Felice, Salvatore D'Avella, Alberto Remus, Paolo Tripicchio, and Carlo Alberto Avizzano. One-shot imitation learning with graph neural networks for pick-and-place manipulation tasks. *IEEE Robotics and Automation Letters*, 2023. 2, 3

[4] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. 1

[5] Atalay Donat, Xiaogang Jia, Xi Huang, Aleksandar Taranovic, Denis Blessing, Ge Li, Hongyi Zhou, Hanyi Zhang, Rudolf Lioutikov, and Gerhard Neumann. Towards fusing point cloud and visual representations for imitation learning, 2025. 4, 6

[6] Yassine El Manyari, Patrick Le Callet, and Laurent Dollé. Imitation from observation using rl and graph-based representation of demonstrations. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1258–1265. IEEE, 2022. 2

[7] Qiao Gu, Ali Kuwajerwala, Sacha Morin, Krishna Murthy Jatavallabhula, Bipasha Sen, Aditya Agarwal, Corban Rivera, William Paul, Kirsty Ellis, Rama Chellappa, et al. Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5021–5028. IEEE, 2024. 1, 3

[8] Mohammad Hashemi, Shengbo Gong, Juntong Ni, Wenqi Fan, B. Aditya Prakash, and Wei Jin. A Comprehensive Survey on Graph Reduction: Sparsification, Coarsening, and Condensation. pages 8058–8066, 2024. ISSN: 1045-0823. 3

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4, 1

[10] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-Attention Graph Pooling. In *Proceedings of the 36th International Conference on Machine Learning*, pages 3734–3743. PMLR, 2019. ISSN: 2640-3498. 3

[11] Kai Li, Zhao Ma, Liang Li, and Shiyu Zhao. Collective behavior clone with visual attention via neural interaction graph prediction. *arXiv preprint arXiv:2503.06869*, 2025. 3

[12] Maximilian Xiling Li, Paul Mattes, Nils Blank, Korbinian Franz Rudolf, Paul Werner Lödige, and Rudolf Lioutikov. Multi-objective photoreal simulation (mops) dataset for computer vision in robotic manipulation. In *Structured World Models for Robotic Manipulation*, 2025. 1

[13] Yixin Lin, Austin S Wang, Eric Undersander, and Akshara Rai. Efficient and interpretable robot manipulation with graph neural networks. *IEEE Robotics and Automation Letters*, 7(2):2740–2747, 2022. 2, 3

[14] Chuang Liu, Wenhang Yu, Kuang Gao, Xueqi Ma, Yibing Zhan, Jia Wu, Bo Du, and Wenbin Hu. Careful Selection and Thoughtful Discarding: Graph Explicit Pooling Utilizing Discarded Nodes, 2023. arXiv:2311.12644 [cs]. 3

[15] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *5th Annual Conference on Robot Learning*, 2021. 1, 3, 4

[16] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3): 7327–7334, 2022. 2, 4, 1

[17] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems (RSS)*, 2024. 2, 4, 5, 6, 1

[18] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024. 1

[19] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, Jan Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018. 1

[20] Tim Pearce, Tabish Rashid, Anssi Kanervisto, Dave Bignell, Mingfei Sun, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Ida Momennejad, Katja Hofmann, and Sam Devlin. Imitating human behaviour with diffusion models. In *The Eleventh International Conference on Learning Representations*, 2023. 1

[21] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual Reasoning with a General Conditioning Layer. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018. 4

[22] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 5

[23] Han Qi, Changhe Chen, and Heng Yang. Compose by focus: Scene graph-based atomic skills, 2025. 2, 3, 8

[24] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021. 4

[25] Mandeep Rathee, Zijian Zhang, Thorben Funke, Megha Khosla, and Avishek Anand. Learnt Sparsification for Interpretable Graph Neural Networks, 2021. arXiv:2106.12920 [cs]. 3

[26] Moritz Reuss, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. Goal conditioned imitation learning using score-based diffusion policies. In *Robotics: Science and Systems*, 2023. 1

[27] Moritz Reuss, Ömer Erdinç Yağmurlu, Fabian Wenzel, and Rudolf Lioutikov. Multimodal diffusion transformer: Learning versatile behavior from multimodal goals. In *Robotics: Science and Systems*, 2024. 1, 3, 4

[28] Moritz Reuss, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Otto, and Rudolf Lioutikov. FLOWER: Democratizing generalist robot policies with efficient vision-language-flow models. In *9th Annual Conference on Robot Learning*, 2025. 1

[29] Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022. 1

[30] Maximilian Sieb, Zhou Xian, Audrey Huang, Oliver Kroemer, and Katerina Fragkiadaki. Graph-structured visual imitation. In *Conference on Robot Learning*, pages 979–989. PMLR, 2020. 3

[31] Kunal Pratap Singh, Jordi Salvador, Luca Weihs, and Aniruddha Kembhavi. Scene graph contrastive learning for embodied navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10884–10894, 2023. 2

[32] Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum likelihood training of score-based diffusion models. *Advances in neural information processing systems*, 34: 1415–1428, 2021. 1

[33] Hsiao-Yu Fish Tung, Zhou Xian, Mihir Prabhudesai, Shamit Lal, and Katerina Fragkiadaki. 3d-oes: Viewpoint-invariant object-factorized environment simulators, 2020. 2, 3

[34] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017. 1, 4

[35] Vitalis Vosylius and Edward Johns. Instant policy: In-context imitation learning via graph diffusion. In *The Thirteenth International Conference on Learning Representations*, 2025. 2, 3

[36] Hongjia Xu, Liangliang Zhang, Yao Ma, Sheng Zhou, Zhuonan Zheng, and Jiajun Bu. Learning to Reduce the Scale of Large Graphs: A Comprehensive Survey. *ACM Trans. Knowl. Discov. Data*, 19(5):101:1–101:25, 2025. 3

[37] Yang Ye and Shihao Ji. Sparse Graph Attention Networks. *IEEE Transactions on Knowledge and Data Engineering*, 35 (1):905–916, 2023. 3, 1

[38] Yujia Yin, Tianyi Qu, Zihao Wang, and Yifan Chen. A Recipe for Causal Graph Regression: Confounding Effects Revisited. 2025. 3, 1

[39] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018. 3

[40] Guibin Zhang, Xiangguo Sun, Yanwei Yue, Chonghe Jiang, Kun Wang, Tianlong Chen, and Shirui Pan. Graph Sparsification via Mixture of Graphs, 2024. arXiv:2405.14260 [cs]. 3

[41] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. Robust Graph Representation Learning via Neural Sparsification. In *Proceedings of the 37th International Conference on Machine Learning*, pages 11458–11468. PMLR, 2020. ISSN: 2640-3498. 3

[42] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020. 2

[43] Yifeng Zhu, Jonathan Tremblay, Stan Birchfield, and Yuke Zhu. Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6541–6548. IEEE, 2021. 2, 3

[44] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023. 1

SIR: Structured Image Representations for Explainable Robot Learning

Supplementary Material

7. Additional information

Source code will be made publicly available upon acceptance. Our GNN implementation employs residual connections and normalization to mitigate oversmoothing, a phenomenon to which small, fully-connected graphs are particularly susceptible.

7.1. Pre-Trained Models

The pre-trained models for generating Cropped-Image-Features and Point-Cloud-Features were trained using human demonstration data from RoboCasa [17]. We extracted all objects from the scene at every 10th time step to train these vision networks using a reconstruction loss. For Cropped-Image-Features alone, we employ a ResNet18 [9] backbone with an embedding dimension of 256. However, when combining Cropped-Image-Features with BB-Coordinates, we use a smaller ResNet8 backbone with an embedding dimension of 37. This lower dimension was chosen specifically to match the size of the object label vectors (one-hot encoded length of 37), allowing the visual features to serve as a direct replacement for symbolic labels.

7.2. Graph Sparsification

Usually, graph sparsification methods are based on the Information Bottleneck (IB) principle [38] or an L0-regularization (or L1/L2) on the edge weights [37] (in our case: node weights). However, we empirically find that such methods are not applicable in our case, as they either keep all or zero nodes in the sub-graph. Regarding our finding that the dataset is biased, we attribute this phenomenon to the lack of discriminability of the information content of the node features. I.e., since there are strong correlations between objects' positions in the scene (or the position is not relevant at all), there is no extreme advantage of choosing one node over another, which makes it difficult to select one node over another to keep in the sub-graph. This is further complicated by the necessity for hard masks, which can hinder differentiability and optimizations, but are required in interpretability-seeking settings. This demonstrates the necessity for our soft histogram loss, which encourages an explicit ranking for the nodes, and the TopK-selection mechanism to ensure that there are neither all nor zero nodes in the sub-graphs.

8. Additional Evaluation Results

The following sections present results for SIR using BC, generated graphs and the CALVIN benchmark [16]. Additionally, we provide fine-grained performance metrics for all 24 RoboCasa tasks and tabular results for the distractor experiments. Table 15 displays the average results of all models using MDT as the action generator.

Table 4. Ablation results on RoboCasa using generated graphs, which are predicted using a fine-tuned DETR model. Models are not trained on these generated graphs, these are only swapped for ground truth information during rollout.

Task	Cropped-Img	BB + Crop-Img
Pick/Place (8)	0.06	0.06
Doors (4)	11.75	18.13
Drawers (2)	34.00	28.75
Knobs (2)	9.00	12.00
Levers (3)	39.67	35.83
Buttons (3)	12.33	10.50
Insert (2)	3.25	3.25
Avg (24)	12.33	12.50

8.1. Generated Graphs Results

This approach is called *generated graphs* and relies on a fine-tuned DETR for graph generation. The DETR model is used during rollout to predict the objects in a given scene and their segmentation masks, which the bounding boxes can be derived from. We evaluated using generated graphs on FC-Graph on RoboCasa using the Split-View approach. Models were not trained on the generated graphs, but on the ground truth information. This introduces a distribution shift, because the generated graphs will be imperfect, which is also visible in the performance drop, seen in Table 4. Compared to using ground truth graphs, the models drop over 4 percent for only using Cropped-Image-Features and 3.4 percent for using BB-Coordinates and Cropped-Image-Features. These results could have two causes, either the model can not handle the distribution shift from ground truth to generated graphs or generated graphs in general will lead to a worse performance. In future experiments, we also want to **train** networks on generated graphs to have a clear answer to this observation.

8.2. RoboCasa - Behaviour Cloning Results

The results for the BC-Transformer as action generation model in RoboCasa can be seen in Table 5. Using images as observation input only decreases the average result slightly compared to using MDT as the action generation model. In comparison, using graphs and BC-Transformer, performance drops heavily, for all approaches. But using SIR still results in the highest average success rate of 16.27

Table 5. Results for using BC-Transformer as the action generator on RoboCasa across all 24 tasks.

Feature Input	Pick/Place (8)	Doors (4)	Drawers (2)	Knobs (2)	Levers (3)	Buttons (3)	Insert (2)	**Avg (24)**
Baselines								
Image	**0.94**	25.63	36.50	5.75	22.33	**24.00**	**7.00**	14.48
Split-View Graph - Fully-Connected								
Cropped-Img	0.13	12.63	31.50	11.75	34.50	6.00	5.75	11.25
BB-Coordinates + Label	0.00	11.88	23.00	**16.25**	15.00	0.00	0.25	7.33
BB-Coordinates + Cropped-Img	0.06	9.00	24.75	6.50	36.33	5.33	3.00	9.58
Fusion Graph - Fully-Connected								
BB-Coordinates + Cropped-Img	0.25	18.13	34.5	11.25	31.83	5.0	0.3	11.77
Sparse Graph Methods								
TopK	0.31	17.25	36.75	10.0	37.83	6.67	2.25	12.63
SIR (Ours)	0.38	**29.63**	**42.25**	14.5	**43.33**	6.83	2.5	**16.27**

Table 6. Completion scores on CALVIN using only the static camera as observation D $\rightarrow$ D for 100 Rollouts using two seeds for each model. The maximum completion number is 5.

Model	**Task Completion (Max 5)**
Baseline	
Image	$\mathbf{1.5 \pm 0.2}$
Fully-Connected Graphs	
Cropped-Img	1.3 ± 0.02
BB-Coordinates + Cropped-Img	1.2 ± 0.03
Sparse Graph Methods	
TopK	1.4 ± 0.02

percent. These observation lead to the conclusion that graph observation can be better utilized by diffusion-based methods compared to BC-based methods, but overall achieve a higher result compared to image-based models regardles of training objective.

8.3. CALVIN - MDT Results

CALVIN [16] serves as our second evaluation benchmark. The results, presented in Table 6, report the average number of tasks completed out of a possible five per rollout. Image-based models achieve an average of 1.5 tasks, slightly outperforming the sparse TopK graph method, which completes 1.4 tasks. It is important to note that our experiments on the CALVIN environment are still preliminary. We hypothesize that further fine-tuning of the sparsification methods could yield performance gains similar to those observed in RoboCasa.

8.4. Distractor Objects

The introduction of distractor objects in RoboCasa leads to a clear decline in performance for both the image baseline and the TopK sparsification approach, displayed in Table 7. While success rates for the *Pick and Place* task appear to increase for all models except the image baseline, the absolute scores are too low to be considered conclusive. In the *Insert* task, performance decreases across the board, though graph-based models exhibit significantly smaller degradation.

8.5. Per Task Results

High average reward for the grouped tasks from the RoboCasa [17] paper does not mean that the model performs well in each subtask. Therefore, we included all single task results in the following tables: *Pick and Place* in Table 8, *Doors* in Table 9, *Drawers* in Table 10, *Knobs* in Table 11, *Levers* in Table 12, *Buttons* in Table 13 and *Insert* in Table 14. Each task grouping (except *Pick and Place*) includes tasks which are easier solvable and harder tasks. SIR only performs best on 4 single atomic tasks, which indicates that the higher average performance is distributed across all single tasks.

9. Additional Explainability Results

We further present additional explainability results for SIR on 6 tasks in RoboCasa, not present in the main paper, as well as a Grad-CAM visualization of the image baseline models.

9.1. Sub-Graph Visualizations

The observed explanations fall into two distinct categories based on the model's adherence to pre-defined important objects. The first group (Figs. 6 to 8) demonstrates consistent reliance on the designated important nodes. In contrast, the second group (Figs. 9 to 11) exhibits significant variance across training seeds. While *PandaGripper* and *PandaMobile* remain constant, other object selections appear arbitrary. This suggests that in tasks where the model diverges from the expected nodes yet maintains high performance, it is exploiting underlying dataset biases to solve the task. Crucially, this inference is valid only for tasks

Table 7. Distractor objects: Success rate over 100 rollouts. Higher values indicate better performance.

Feature Input	Pick/Place (8)	Doors (4)	Drawers (2)	Knobs (2)	Levers (3)	Buttons (3)	Insert (2)	**Avg (24)**
Baseline								
Image	**1.19**	25.13	**49.75**	7.25	23.67	17.00	**4.75**	14.81
Image w/ Distractors	0.56	18.25	46.25	3.25	17.17	14.50	2.50	11.52
Fully Connected Graph								
FC-Graph	0.06	28.62	39.25	14.00	40.00	18.83	**4.75**	16.98
FC-Graph w/ Distractors	0.38	21.12	40.00	17.75	41.50	18.67	3.75	16.29
Sparse Graph Methods								
Threshold	0.00	28.38	47.25	11.00	39.50	19.17	3.00	17.17
Threshold w/ Distractors	0.40	22.50	44.50	12.75	40.33	22.17	2.50	16.67
TopK	0.12	**31.37**	45.00	19.50	41.17	18.17	4.50	18.44
TopK w/ Distractors	0.38	22.75	44.75	14.25	36.50	14.50	4.00	15.54
SIR	0.12	30.25	46.25	16.50	48.50	**21.83**	**4.75**	**19.50**
SIR w/ Distractors	0.56	23.12	47.50	**21.75**	**51.83**	20.67	4.25	19.23

Table 8. The results represent success rate over 100 rollouts with 2 models trained on different seeds for Pick and Place tasks.

Feature Input	PnPCab ToCounter	PnPCounter ToCab	PnPMicrowave ToCounter	PnPCounter ToMicrowave	PnPSink ToCounter	PnPCounter ToSink	PnPStove ToCounter	PnPCounter ToStove	**Average**
Baselines									
Image	3.0	1.5	2.5	0.0	0.0	0.5	0.5	1.5	1.19
Image + FiLM	1.0	0.0	0.0	0.5	0.0	0.25	0.5	0.5	0.34
Own Pretrained Image	0.5	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.13
Point Clouds	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Image + Point Clouds	1.5	0.0	0.0	0.5	0.5	1.0	0.5	0.5	0.56
Split-View Graph - Fully-Connected									
Cropped-Img	1.0	0.0	0.5	0.0	0.0	0.0	0.5	0.0	0.25
BB-Coordinates + Label	0.5	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.13
BB-Coordinates + Cropped-Img	1.5	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.25
Point Clouds	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.06
Label + Point Clouds	1.0	0.5	0.0	0.0	0.5	0.67	0.5	0.0	0.40
Cropped-Img + Point Clouds	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.25
Fusion Graph - Fully-Connected									
BB-Coordinates + Label	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.06
BB-Coordinates + Cropped-Img	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.06
Sparsification Methods									
Random Node Removal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Naive NR (no soft histogram loss)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Threshold	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
TopK	0.5	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.12
SIR (Ours)	0.5	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.12

where the model outperforms baselines, as arbitrary focus in low-performing models likely indicates a failure to learn the task.

9.2. Grad-CAM Visualizations

We visualize four steps of a rollout using the image baseline on the *CloseDrawer* task in Figure 12. The model's focus shifts between the robot arm, gripper, and counter/drawer. Over the whole rollout the focus includes nearly the entire image. This lack of precision renders such visualizations ineffective for interpreting model behaviour or gaining insights into the dataset.

Table 9. Door tasks: success rate over 100 rollouts with 2 models trained on different seeds.

Feature Input	Close Single Door	Open Single Door	Close Double Door	Open Double Door	**Average**
Baselines					
Image	63.0	6.0	30.0	1.5	25.13
Image + FiLM	64.5	7.5	36.0	3.0	27.75
Own Pretrained Image	55.5	4.0	8.0	0.5	17.00
Point Clouds	0.0	0.0	0.0	0.0	0.00
Image + Point Clouds	59.0	6.0	33.5	2.5	25.25
Split-View Graph - Fully-Connected					
Cropped-Img	48.5	16.0	17.5	1.5	20.88
BB-Coordinates + Label	54.0	3.5	18.5	0.5	19.13
BB-Coordinates + Cropped-Img	59.0	14.5	18.0	1.0	23.13
Point Clouds	21.5	0.0	24.0	0.0	11.38
Label + Point Clouds	24.5	6.5	31.5	1.0	15.88
Cropped-Img + Point Clouds	39.5	10.0	26.0	2.0	19.38
Fusion Graph - Fully-Connected					
BB-Coordinates + Label	47.5	3.0	6.0	0.0	14.12
BB-Coordinates + Cropped-Img	64.0	18.5	29.0	3.0	28.62
Sparsification Methods					
Random Node Removal	9.0	7.0	6.5	0.0	5.62
Naive NR (no soft histogram loss)	33.5	0.5	0.0	0.0	8.50
Threshold	61.5	17.5	32.0	2.5	28.38
TopK	60.0	25.0	35.0	5.5	31.37
SIR (Ours)	65.5	22.0	33.0	0.5	30.25

Table 10. Drawer tasks: success rate over 100 rollouts with 2 models trained on different seeds.

Feature Input	Close Drawer	Open Drawer	**Average**
Baselines			
Image	85.5	14.0	49.75
Image + FiLM	80.5	10.0	45.25
Own Pretrained Image	68.5	10.0	39.25
Point Clouds	2.0	0.0	1.0
Image + Point Clouds	82.0	7.5	44.75
Split-View Graph - Fully-Connected			
Cropped-Img	67.5	7.0	37.25
BB-Coordinates + Label	61.0	2.0	31.5
BB-Coordinates + Cropped-Img	63.0	7.5	35.25
Point Clouds	51.0	7.5	29.25
Label + Point Clouds	57.5	7.0	32.25
Cropped-Img + Point Clouds	70.0	11.5	40.75
Fusion Graph - Fully-Connected			
BB-Coordinates + Label	55.0	8.0	31.5
BB-Coordinates + Cropped-Img	75.0	3.5	39.25
Sparsification Methods			
Random Node Removal	18.5	0.5	9.5
Naive NR (no soft histogram loss)	46.0	1.0	23.5
Threshold	87.5	7.0	47.25
TopK	81.5	8.5	45.0
SIR (Ours)	80.5	12.0	46.25

Table 11. Stove tasks: success rate over 100 rollouts with 2 models trained on different seeds.

Feature Input	TurnOn Stove	TurnOff Stove	**Average**
Baselines			
Image	12.5	2.0	7.25
Image + FiLM	17.0	5.0	11.00
Own Pretrained Image	9.5	5.0	7.25
Point Clouds	1.0	1.5	1.25
Image + Point Clouds	9.0	2.0	5.50
Split-View Graph - Fully-Connected			
Cropped-Img	25.0	5.0	15.00
BB-Coordinates + Label	21.5	5.5	13.50
BB-Coordinates + Cropped-Img	21.5	7.0	14.25
Point Clouds	28.0	9.0	18.50
Label + Point Clouds	23.5	6.5	15.00
Cropped-Img + Point Clouds	12.0	5.0	8.50
Fusion Graph - Fully-Connected			
BB-Coordinates + Label	17.0	9.0	13.00
BB-Coordinates + Cropped-Img	22.5	5.5	14.00
Sparsification Methods			
Random Node Removal	8.0	2.5	5.25
Naive NR (no soft histogram loss)	8.5	5.0	6.75
Threshold	15.5	6.5	11.0
TopK	26.0	13.0	19.50
SIR (Ours)	26.5	6.5	16.50

Table 12. Sink tasks: success rate over 100 rollouts with 2 models trained on different seeds.

Feature Input	TurnOn Sink Faucet	TurnOff Sink Faucet	Turn Sink Spout	**Average**
Baselines				
Image	22.0	24.0	25.0	23.67
Image + FiLM	32.0	32.5	34.0	32.83
Own Pretrained Image	11.5	25.5	19.5	18.83
Point Clouds	27.5	2.5	29.5	19.83
Image + Point Clouds	21.5	18.5	22.0	20.67
Split-View Graph - Fully-Connected				
Cropped-Img	36.0	64.5	44.5	48.33
BB-Coordinates + Label	12.0	17.5	34.0	21.17
BB-Coordinates + Cropped-Img	30.5	51.5	39.0	40.33
Point Clouds	15.5	24.0	51.5	30.33
Label + Point Clouds	26.0	31.5	53.0	36.83
Cropped-Img + Point Clouds	24.5	55.0	43.0	40.83
Fusion Graph - Fully-Connected				
BB-Coordinates + Label	11.5	25.5	44.5	27.17
BB-Coordinates + Cropped-Img	23.0	54.5	42.5	40.00
Sparsification Methods				
Random Node Removal	5.0	21.5	24.5	17.00
Naive NR (no soft histogram loss)	15.0	33.0	39.5	29.17
Threshold	20.5	54.5	43.5	39.5
TopK	22.0	55.0	46.5	41.17
SIR (Ours)	31.5	69.0	45.0	48.50

Table 13. Button tasks: success rate over 100 rollouts with 2 models trained on different seeds.

Feature Input	TurnOff Microwave	TurnOn Microwave	Coffee PressButton	**Average**
Baselines				
Image	22.0	15.0	14.0	17.00
Image + FiLM	27.5	14.5	8.5	16.83
Own Pretrained Image	9.5	9.5	11.5	10.17
Point Clouds	24.0	5.5	3.0	10.83
Image + Point Clouds	17.5	15.0	13.0	15.17
Split-View Graph - Fully-Connected				
Cropped-Img	22.5	13.5	15.5	17.17
BB-Coordinates + Label	12.0	8.5	2.5	7.67
BB-Coordinates + Cropped-Img	25.5	14.5	13.0	17.67
Point Clouds	12.0	13.5	3.5	9.67
Label + Point Clouds	11.5	10.5	3.0	8.33
Cropped-Img + Point Clouds	15.0	15.0	18.0	16.00
Fusion Graph - Fully-Connected				
BB-Coordinates + Label	12.0	15.5	5.5	11.00
BB-Coordinates + Cropped-Img	30.5	17.5	8.5	18.83
Sparsification Methods				
Random Node Removal	13.5	7.5	4.0	8.33
Naive NR (no soft histogram loss)	21.5	9.5	10.5	13.83
Threshold	31.5	15.0	11.0	19.17
TopK	25.0	18.5	11.0	18.17
SIR (Ours)	38.5	14.0	13.0	21.83

Table 14. Coffee tasks: success rate over 100 rollouts with 2 models trained on different seeds.

Feature Input	Coffee Serve Mug	Coffee Setup Mug	**Average**
Baselines			
Image	9.5	0.0	4.75
Image + FiLM	3.0	0.0	1.50
Own Pretrained Image	4.0	0.0	2.00
Point Clouds	2.5	0.0	1.25
Image + Point Clouds	4.0	0.5	2.25
Split-View Graph - Fully-Connected			
Cropped-Img	12.0	1.0	6.50
BB-Coordinates + Label	9.0	0.5	4.75
BB-Coordinates + Cropped-Img	12.5	1.5	7.00
Point Clouds	4.5	0.0	2.25
Label + Point Clouds	3.0	0.5	1.75
Cropped-Img + Point Clouds	11.0	1.5	6.25
Fusion Graph - Fully-Connected			
BB-Coordinates + Label	4.5	0.5	2.50
BB-Coordinates + Cropped-Img	8.5	1.0	4.75
Sparsification Methods			
Random Node Removal	3.5	0.0	1.75
Naive NR (no soft histogram loss)	7.0	0.0	3.50
Threshold	6.0	0.0	3.0
TopK	8.5	0.5	4.50
SIR (Ours)	9.0	0.5	4.75

Table 15. Success rate on RoboCasa across all 24 tasks over 100 rollouts with 2 models trained on different seeds.

Feature Input	Pick/Place (8)	Doors (4)	Drawers (2)	Knobs (2)	Levers (3)	Buttons (3)	Insert (2)	**Avg (24)**
Baselines								
Image	**1.19**	25.13	**49.75**	7.25	23.67	17.00	4.75	14.81
Image + FiLM	0.34	27.75	45.25	11.00	32.83	16.83	1.50	15.85
Own Pretrained Image	0.13	17.00	39.25	7.25	18.83	10.17	2.00	10.11
Point Clouds	0.00	0.00	1.00	1.25	19.83	10.83	1.25	4.13
Image + Point Clouds	0.56	25.25	44.75	5.50	20.67	15.17	2.25	13.25
Split-View Graph - Fully-Connected								
Cropped-Img	0.25	20.88	37.25	15.00	48.33	17.17	6.50	16.65
BB-Coordinates + Label	0.13	19.13	31.50	13.50	21.17	7.67	4.75	10.98
BB-Coordinates + Cropped-Img	0.25	23.13	35.25	14.25	40.33	17.67	**7.00**	15.90
Point Clouds	0.06	11.38	29.25	18.50	30.33	9.67	2.25	11.08
Label + Point Clouds	0.40	15.88	32.25	15.00	36.83	8.33	1.75	12.50
Cropped-Img + Point Clouds	0.25	19.38	40.75	8.50	40.83	16.00	6.25	15.04
Fusion Graph - Fully-Connected								
BB-Coordinates + Label	0.06	14.12	31.50	13.00	27.17	11.00	2.50	11.06
BB-Coordinates + Cropped-Img	0.06	28.62	39.25	14.00	40.00	18.83	4.75	16.98
Sparsification Methods								
Random Node Removal	0.00	5.62	9.50	5.25	17.00	8.33	1.75	5.48
Naive NR (no soft histogram loss)	0.00	8.50	23.50	6.75	29.17	13.83	3.50	9.60
Threshold	0.0	28.38	47.25	11.0	39.5	19.17	3.0	17.17
TopK	0.12	**31.37**	45.00	**19.50**	41.17	18.17	4.50	18.44
SIR (Ours)	0.12	30.25	46.25	16.50	**48.50**	**21.83**	4.75	**19.50**

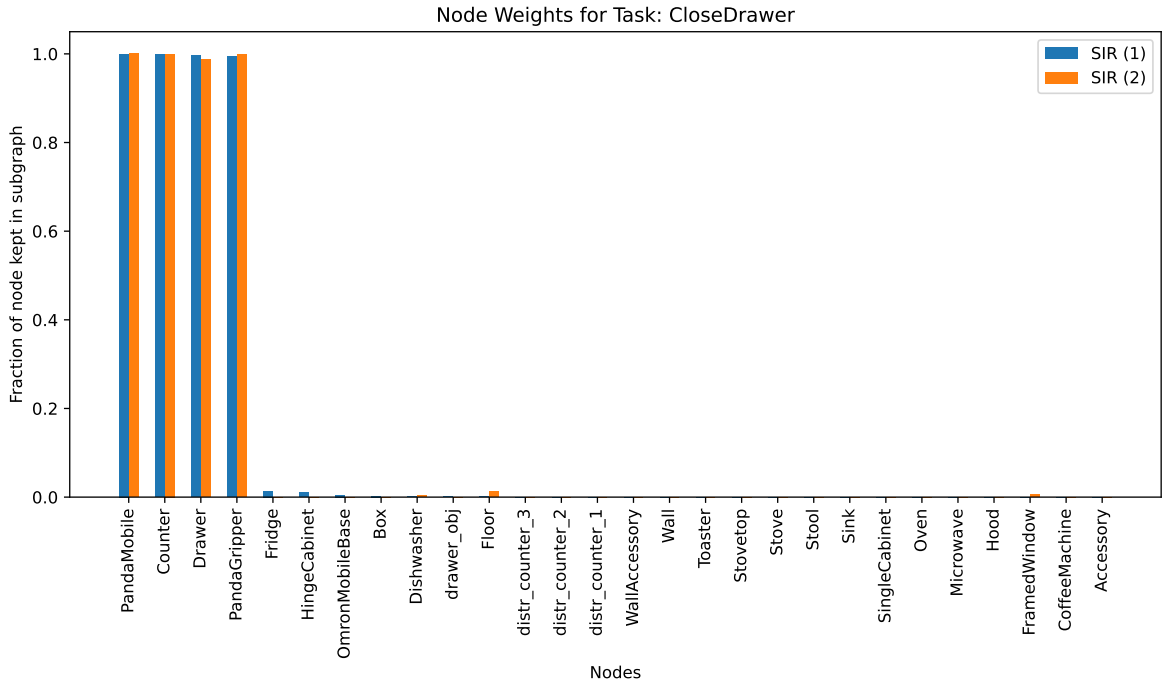


Figure 6. Percentage of nodes to be kept in the sub-graph per seeded model *CloseDrawer*. In contrast to Figure 5c, SIR is able to include the drawer in the graph. This stands in no conflict with our previous claim that the drawer is not important, as the instruction-grounded method is explicitly *encouraged* by the additional loss to include the drawer. This further stands in no conflict with our results, where even in the instruction-grounded approach, the relevant nodes are not included, as the instruction-grounded approach is not *forced* to use the important nodes.

Table 16. MDT Policy Success Rate (%) for all 24 Atomic Tasks (100 Rollouts) using the In-hand camera as additional modality.

Atomic Task	Image + In-hand	FC-Graph + In-hand	SIR + In-hand
Pick and Place (8)			
PnPCab → Ctr	8.5	6.5	5.5
PnP Ctr → Cab	11.5	9.0	6.0
PnPMW → Ctr	4.0	2.0	0.5
PnP Ctr → MW	12.0	5.0	1.0
PnPSink → Ctr	6.5	5.0	7.5
PnP Ctr → Sink	6.0	9.5	8.0
PnP Stove → Ctr	1.5	8.0	6.5
PnP Ctr → Stove	1.0	2.0	1.0
Doors (4)			
Close Single Door	67.5	75.0	76.0
Open Single Door	29.5	38.5	49.0
Close DoubleDoor	39.0	33.5	18.0
Open DoubleDoor	15.5	10.5	2.0
Drawers (2)			
Close Drawer	93.5	84.0	89.5
Open Drawer	25.5	25.5	25.0
Knobs (Stove) (2)			
TurnOn Stove	9.0	35.5	31.5
TurnOff Stove	4.5	14.5	12.0
Levers (Sink) (3)			
TurnOn Sink Faucet	34.5	41.0	36.0
TurnOff Sink Faucet	25.0	36.0	40.0
Turn Sink Spout	31.0	26.0	29.5
Buttons (3)			
Turn Off Microwave	47.0	55.5	58.5
Turn On Microwave	40.0	52.0	60.5
Coffee Press Button	49.0	62.0	67.5
Insertion (Coffee) (2)			
Coffee Serve Mug	22.5	29.0	27.5
Coffee Setup Mug	4.0	3.5	4.0
Average	24.5	30.5	27.6

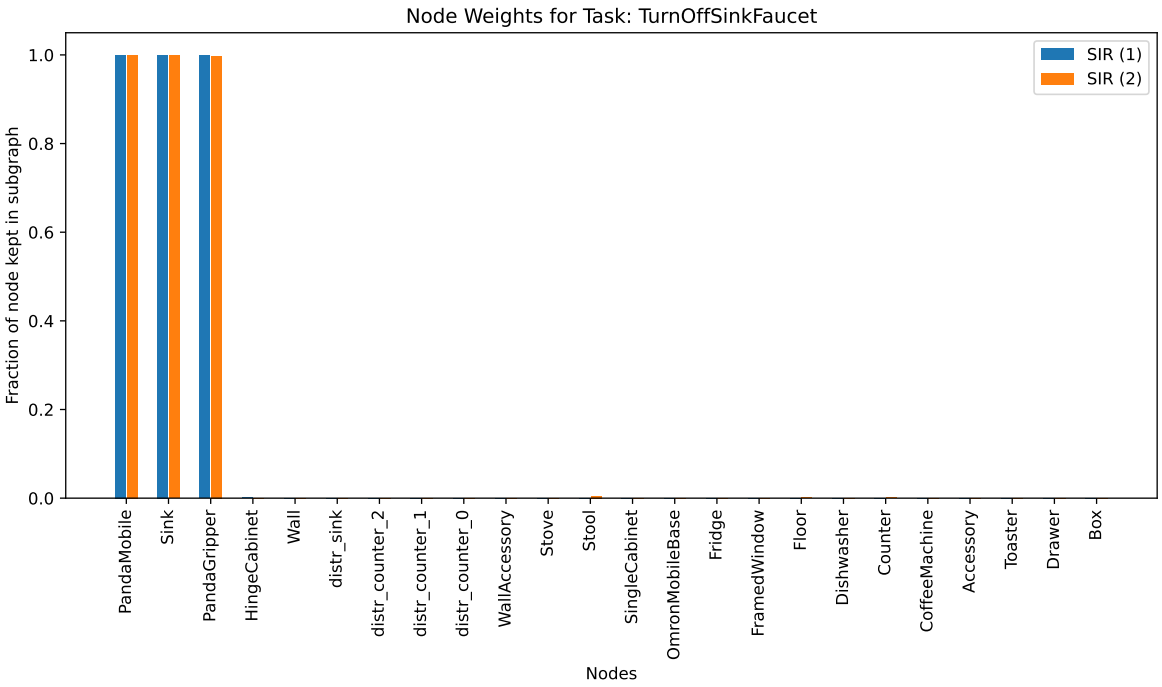


Figure 7. Percentage of nodes to be kept in the sub-graph per seeded model *TurnOffSinkFaucet*.

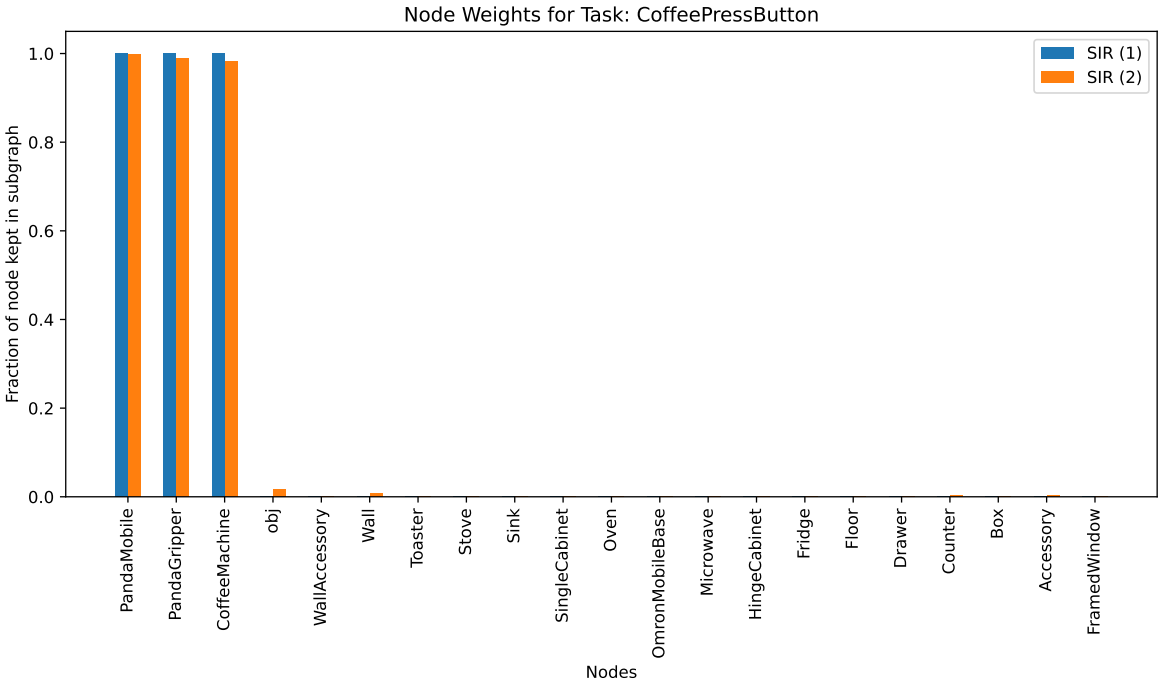


Figure 8. Percentage of nodes to be kept in the sub-graph per seeded model *CoffeePressButton*.

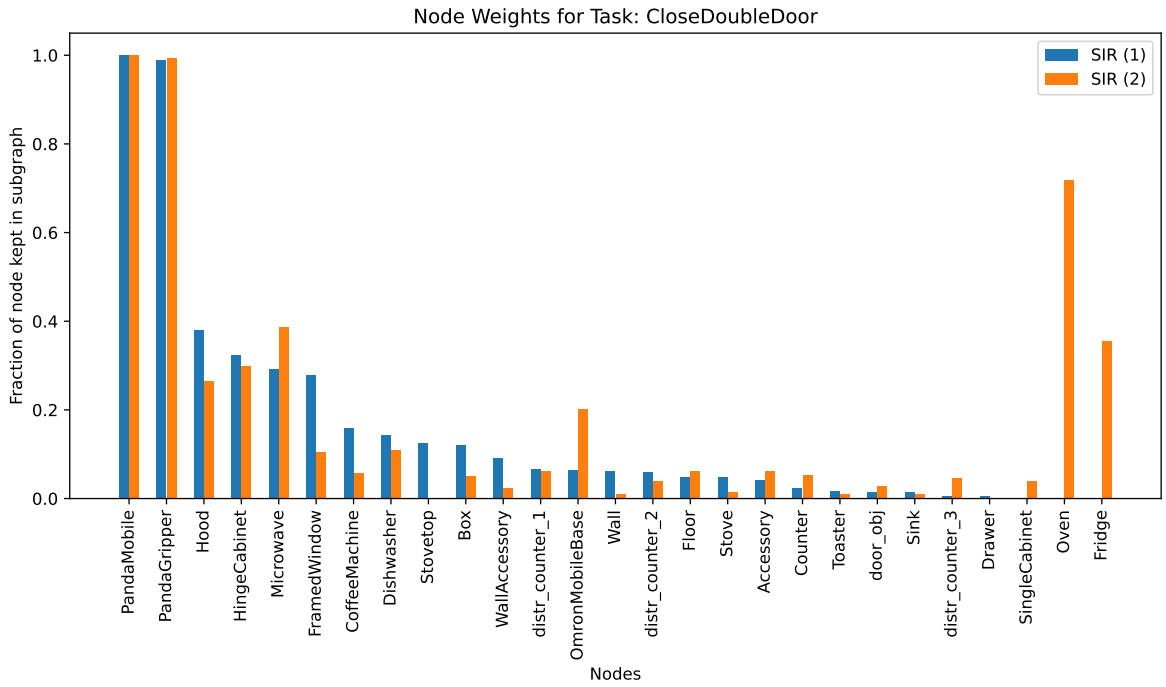


Figure 9. Percentage of nodes to be kept in the sub-graph per seeded model *CloseDoubleDoor*. Similar to Figure 5d, here also the object of which the door has to be close (HingeCabinet) is not always selected to be in the graph.

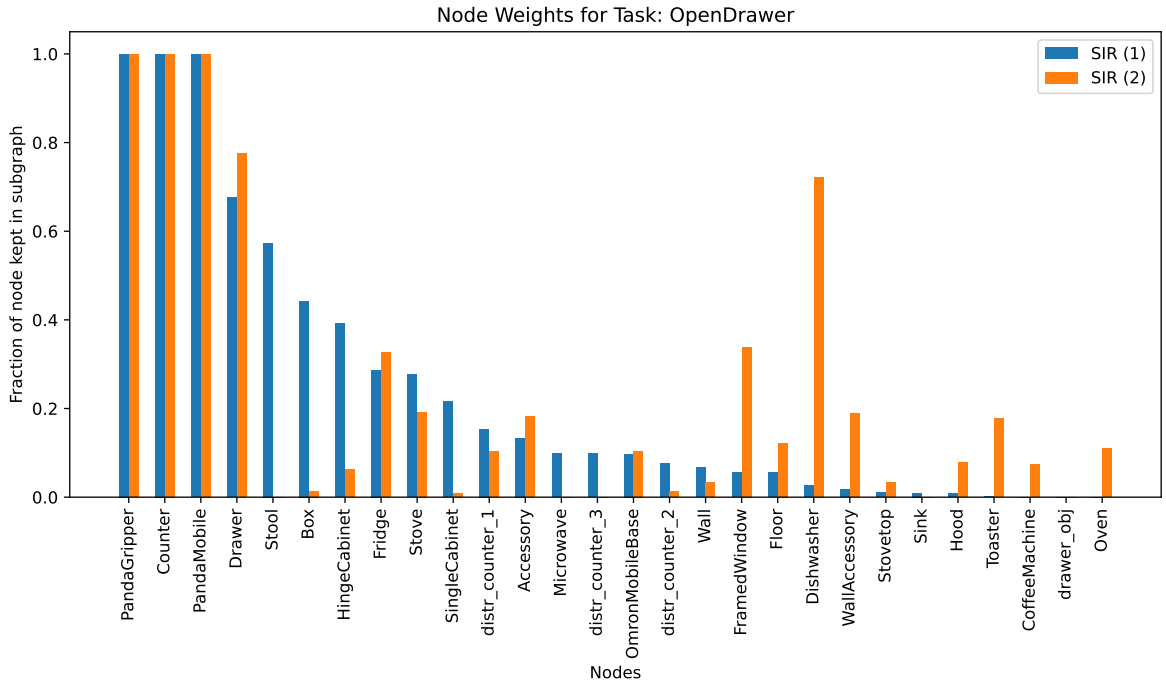


Figure 10. Percentage of nodes to be kept in the sub-graph per seeded model *OpenDrawer*. Although the important nodes are included, other nodes are not consistently removed.

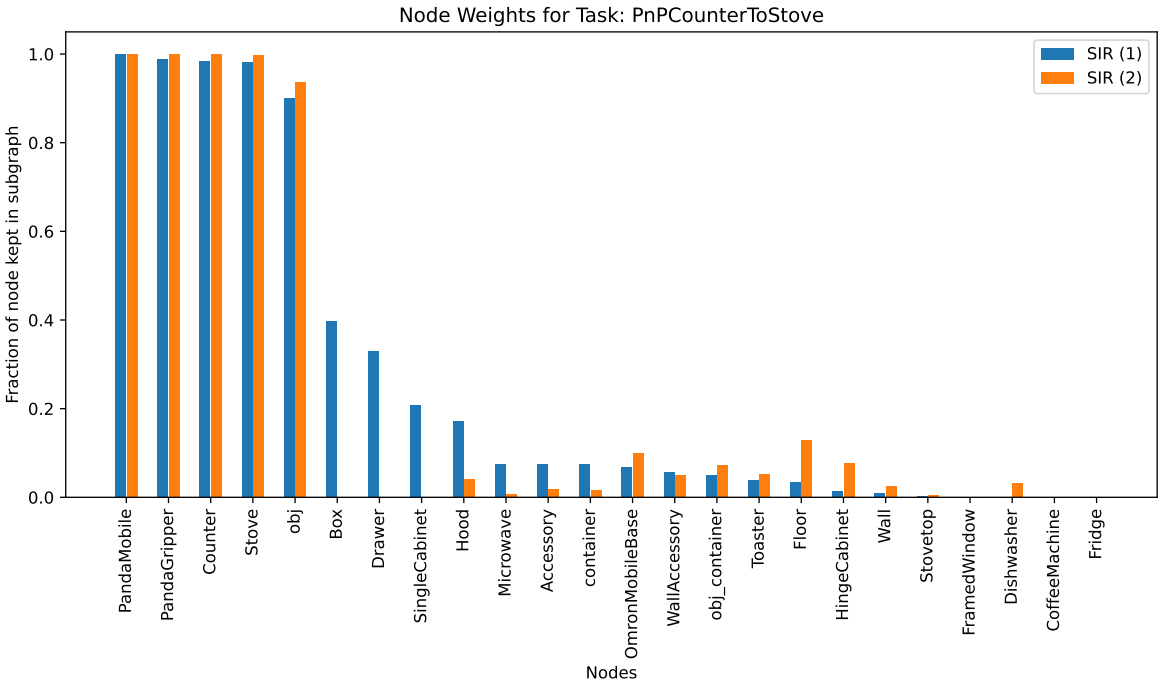


Figure 11. Percentage of nodes to be kept in the sub-graph per seeded model in *PnPCounterToStove*.

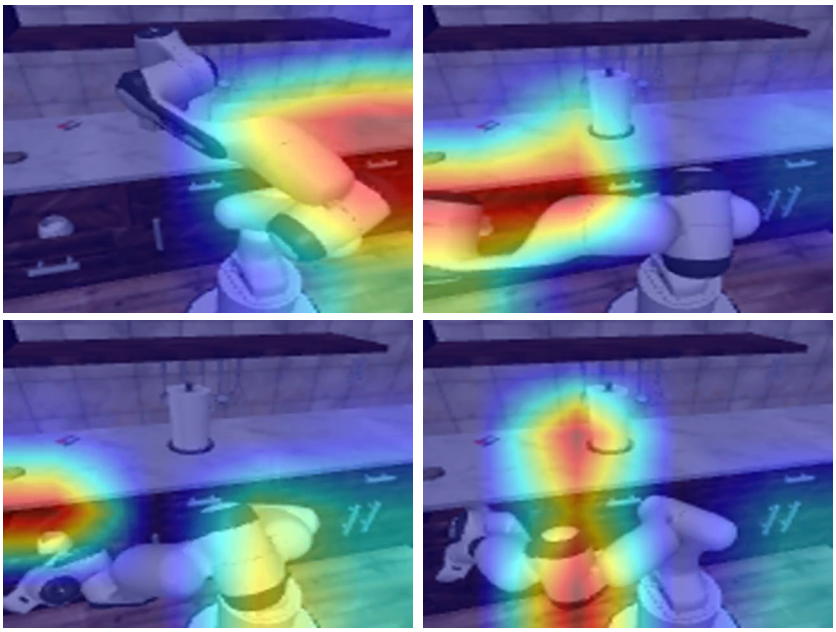


Figure 12. Grad-CAM explanations over time for a rollout of the *CloseDrawer* task (frames 20, 70, 120, 170). We find that the explanations produced by Grad-CAM are neither consistent, as the activated regions are jumping around a lot in the image, nor expressive, as the highlighted regions are not bound to objects.