Image Manipulation via Multi-Hop Instructions - A New Dataset and Weakly-Supervised Neuro-Symbolic Approach

Harman Singh[1] **Poorva Garg**[1*] **Mohit Gupta**[1*]
Kevin Shah[1†] **Ashish Goswami**[1†] **Satyam Modi**[1†]
Arnab Kumar Mondal[1] **Dinesh Khandelwal**[2] **Dinesh Garg**[2] **Parag Singla**[1]
[1]Indian Institute of Technology Delhi [2]IBM Research AI

Abstract

We are interested in *image manipulation via natural language text* – a task that is useful for multiple AI applications but requires complex reasoning over multi-modal spaces. We extend recently proposed Neuro Symbolic Concept Learning (NSCL) (Mao et al., 2019), which has been quite effective for the task of Visual Question Answering (VQA), for the task of image manipulation. Our system referred to as NEUROSIM can perform *complex multi-hop reasoning* over *multi-object scenes* and only requires *weak supervision* in the form of annotated data for VQA. NEUROSIM parses an instruction into a symbolic program, based on a Domain Specific Language (DSL) comprising of object attributes and manipulation operations, that guides its execution. We create a new dataset for the task, and extensive experiments demonstrate that NEUROSIM is highly competitive with or beats SOTA baselines that make use of supervised data for manipulation.

1 Introduction

The last decade has seen significant growth in the application of *neural models* to a variety of tasks including those in computer vision (Chen et al., 2017; Krizhevsky et al., 2012), NLP (Wu et al., 2016), robotics and speech (Yu and Deng, 2016). It has been observed that these models often lack interpretability (Fan et al., 2021), and may not always be well suited to handle complex reasoning tasks (Dai et al., 2019). On the other hand, *classical AI systems* can seamlessly perform complex reasoning in an interpretable manner due to their *symbolic representation* (Pham et al., 2007; Cai and Su, 2012). But these models often lack in their ability to handle low-level representations and be robust to noise. *Neuro-Symbolic models* (Dong et al., 2019; Mao et al., 2019; Han et al., 2019) overcome these limitations by combining the power of (purely) neural with (purely) symbolic representations. Studies (Andreas et al., 2016; Hu et al., 2017; Johnson et al., 2017a; Mao et al., 2019) have shown that neuro-symbolic models have several desirable properties such as *modularity, interpretability,* and *improved generalizability*.

Our aim in this work is to build neuro-symbolic models for the task of *weakly supervised manipulation of images comprising multiple objects, via complex multi-hop natural language instructions.* Specifically, we are interested in weak supervision that only uses the data annotated for VQA tasks, avoiding the high cost of getting supervised annotations in the form of target manipulated images. Our key intuition here is that this task can be solved simply by querying the manipulated representation without ever explicitly looking at the target image. The prior work includes weakly supervised approaches (Nam et al., 2018; Li et al., 2020) that require textual descriptions of images during training and are limited to very simple scenes (or instructions). (See Section 2 for a survey).

Our solution builds on Neuro-Symbolic Concept Learner (NSCL) proposed by (Mao et al., 2019) for solving VQA. We extend this work to incorporate the notion of manipulation operations such as *change*, *add,* and *remove* objects in a given image. As one of our main contributions, we design novel neural modules and a training strategy that just uses VQA annotations as weakly supervised data for the task of image manipulation. The neural modules are trained with the help of *novel loss functions* that measure the faithfulness of the manipulated scene and object representations by accessing a separate set of *query networks*, interchangeably referred to as *quantization networks*, trained just using VQA data. The manipulation takes place through interpretable programs created using primitive neural and symbolic operations from a Domain Specific

[†] Equal Contribution, [*]Equal Contribution.
First four authors did this work while at IIT Delhi.
Correspondence: `harmansingh.iitd@gmail.com`, `parags@iitd.ac.in`

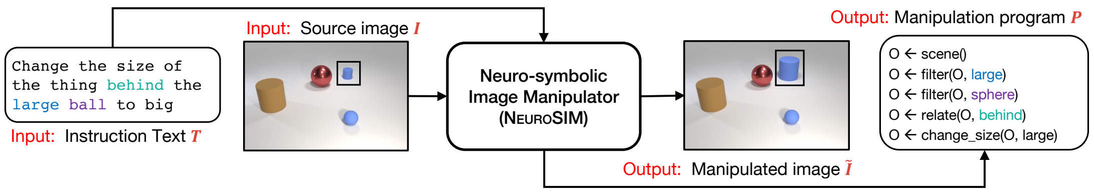

Input: Source image I
Change the size of the thing behind the large ball to big
Input: Instruction Text T
Neuro-symbolic Image Manipulator (NeuroSIM)
Output: Manipulation program P
O ← scene()
O ← filter(O, large)
O ← filter(O, sphere)
O ← relate(O, behind)
O ← change_size(O, large)
Output: Manipulated image $\tilde{I}$

Figure 1: The problem setup. See Section 1 for more details.

Language (DSL). Separately, a network is trained to render the image from a scene graph representation using a combination of L_1 and adversarial losses as done by (Johnson et al., 2018). The entire pipeline is trained without any intermediate supervision. We refer to our system as Neuro-Symbolic Image Manipulator (NEUROSIM). Figure 1 shows an example of I/O pair for our approach. Contributions of our work are as follows:

1. We create NEUROSIM, the first neuro-symbolic, weakly supervised, and interpretable model for the task of text-guided image manipulation, that does not require output images for training.
2. We extend CLEVR (Johnson et al., 2017b), a benchmark dataset for VQA, to incorporate manipulation instructions and create a new dataset called as *Complex Image Manipulation via Natural Language Instructions* (CIM-NLI). We also create CIM-NLI-LARGE dataset to test zero-shot generalization.
3. We provide extensive quantitative experiments on newly created CIM-NLI, CIM-NLI-LARGE datasets along with qualitative experiments on Minecraft (Yi et al., 2018). Despite being weakly supervised, NEUROSIM is highly competitive to supervised SOTA approaches including a recently proposed diffusion based model (Brooks et al., 2023). NEUROSIM also performs well on instructions requiring multi-hop reasoning, all while being interpretable. We publicly release our code and data [1].

2 Related Work

Table 1 categorizes the related work across three broad dimensions - *problem setting*, *task complexity*, and *approach*. The problem setting comprises two sub-dimensions: i) supervision type - *self, direct,* or *weak*, ii) instruction format - *text or UI-based*. The task complexity comprises of following sub-dimensions: ii) scene complexity – *single* or *multiple objects*, ii) instruction complexity - *zero or multi-hop instructions*, iii) kinds of manipulations allowed - *add, remove,* or *change*. Finally, the approach consists of the following sub-dimensions: i) model – *neural* or *neuro-symbolic* and ii) whether a symbolic program is generated on the way or not. Dong et al. (2017), TAGAN (Nam et al., 2018), and ManiGAN (Li et al., 2020) are close to us in terms of the problem setting. These manipulate the source image using a GAN-based encoder-decoder architecture. Their weak supervision differs from ours – We need VQA annotation, they need captions or textual descriptions. The complexity of their natural language instructions is restricted to 0-hop. Most of their experimentation is limited to single (salient) object scenes.

In terms of task complexity, the closest to us are approaches such as TIM-GAN (Zhang et al., 2021), GeNeVA (El-Nouby et al., 2019), which build an encoder-decoder architecture and work with a latent representation of the image as well as the manipulation instruction. They require a large number of manipulated images as explicit annotations for training.

In terms of technique, the closest to our work are neuro-symbolic approaches for VQA such as NSVQA (Yi et al., 2018), NSCL (Mao et al., 2019), Neural Module Networks (Andreas et al., 2016) and its extensions (Hu et al., 2017; Johnson et al., 2017a). Clearly, while the modeling approach is similar and consists of constructing latent programs, the desired tasks are different in the two cases. Our work extends the NSCL approach for the task of automated image manipulation.

Jiang et al. (2021),Shi et al. (2021) deal with editing global features, such as brightness, contrast, etc., instead of object-level manipulations like in our case. Recent models such as InstructPix2Pix (Brooks et al., 2023), DALL-E (Ramesh et al., 2022) and Imagen (Saharia et al., 2022) on text-to-image generation using diffusion models are capable of editing images but require captions for input images; preliminary studies (Marcus et al., 2022) highlight their shortcomings in composi-

[1]https://github.com/dair-iitd/NeuroSIM

Prior Work	Problem Setting		Task Complexity			Approach	
	Supervision Type	Instruction Format	SC	IC	Operations	Model	Program
SIMSG	Self Supervision	UI	MO	N/A	change, remove, add	N	✗
PGIM	Direct Supervision	N/A	MO[*]	N/A	change (image level)	NS	✓
GeNeVA	Direct Supervision	Text	MO	Multi-Hop	add	N	✗
TIM-GAN	Direct Supervision	Text	MO	Zero-Hop	change, remove, add	N	✗
Dong et. al	Weak Supervision	Text	SO	Zero-Hop	change	N	✗
TAGAN	Weak Supervision	Text	SO	Zero-Hop	change	N	✗
ManiGAN	Weak Supervision	Text	SO	Zero-Hop	change	N	✗
InstructPix2Pix	Pre-training + Supervision	Text	MO	Multi-Hop	change, remove, add	N	✗
NEUROSIM (ours)	Weak Supervision	Text	MO	Multi-Hop	change, remove, add	NS	✓

Table 1: Comparison of Prior Work. Abbreviations (column titles) SC:= Scene Complexity, IC:=Instruction Complexity. Abbreviations (column values) MO:= Multiple Objects, MO*:= Multiple Objects with Regular Patterns, SO:= Single Object, N:= Neural, NS:= Neuro-Symbolic, N/A:= Not applicable, ✓:= Yes, ✗:= No. See Section 2 for more details.

tional reasoning and handling relations.

3 Neuro-Symbolic Image Manipulator

3.1 Motivation and Architecture Overview

The key motivation behind our approach comes from the following hypothesis: consider a learner L (e.g., a neural network or the student in Fig 2) with sufficient capacity trying to achieve the task of manipulation over Images I. Further, let each image be represented in terms of its properties, or properties of its constituents (e.g. objects like apple, leaf, tree, etc. in Fig 2), where each property comes from a finite set S e.g, attributes of objects in an image. Let the learner be provided with the prior knowledge (for e.g. through Question Answering as in Fig 2) about properties (e.g., color) and their possible values (e.g., red). Then, in order to learn the task of manipulation, it suffices to provide the learner with a *query network*, which given a manipulated image $\tilde{I}$ constructed by the learner via command C, can correctly answer questions (i.e. query) about the desired state of various properties of the constituents of the image $\tilde{I}$. The query network can be internal to the learner (e.g., the student in Fig 2 can query himself for checking the color of apples in the manipulated image). The learner can query repeatedly until it learns to perform the manipulation task correctly. Note, the learner does not have access to the supervised data corresponding to triplets of the form (I_s, C, I_f), where I_s is the starting image, C is the manipulation command, and I_f is the target manipulated image. Inspired by this, we set out to test this hypothesis by building a model capable of manipulating images, without target images as supervision.

Figure 3 captures a high-level architecture of the proposed NEUROSIM pipeline. NEUROSIM allows manipulating images containing multiple objects, via complex natural language instructions. Similar to Mao et al. (2019), NEUROSIM assumes the availability of a *domain-specific language* (DSL) for parsing the instruction text T into an executable program P. NEUROSIM is capable of handling *addition, removal,* and *change* operations over image objects. It reasons over the image for locating where the manipulation needs to take place followed by carrying out the manipulation operation. The first three modules, namely *i) visual representation network, ii) semantic parser,* and *iii) concept quantization network* are suitably customized from the NSCL and trained as required for our purpose. In what follows, we describe the design and training mechanism of NEUROSIM.

3.2 Modules Inherited from NSCL

1] Visual Representation Network: Given input image I, this network converts it into a scene graph $G_I = (N, E)$. The nodes N of this scene graph are object embeddings and the edges E are embeddings capturing the relationship between pair of objects (nodes). Node embeddings are obtained by passing the bounding box of each object (along with the full image) through a ResNet-34 (He et al., 2016). Edge embeddings are obtained by concatenating the corresponding object embeddings.

2] Semantic Parsing Module: The input to this module is a manipulation instruction text T in natural language. Output is a *symbolic program* P generated by parsing the input text. The symbolic programs are made of operators, that are part of

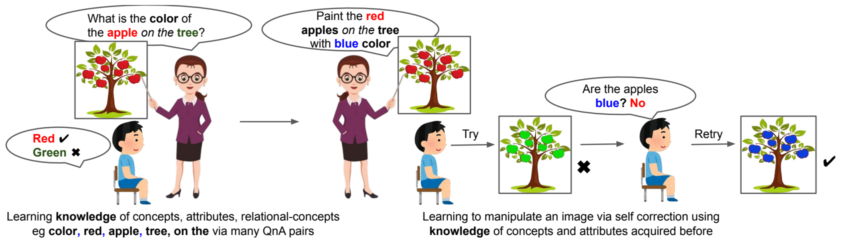

What is the color of the apple on the tree?
Red ✔
Green ✖
Paint the red apples on the tree with blue color
Try
Are the apples blue? No
Retry
Learning knowledge of concepts, attributes, relational-concepts eg color, red, apple, tree, on the via many QnA pairs
Learning to manipulate an image via self correction using knowledge of concepts and attributes acquired before

Figure 2: Motivating example NEUROSIM. Best viewed under magnification. See Section 3.1 for more details

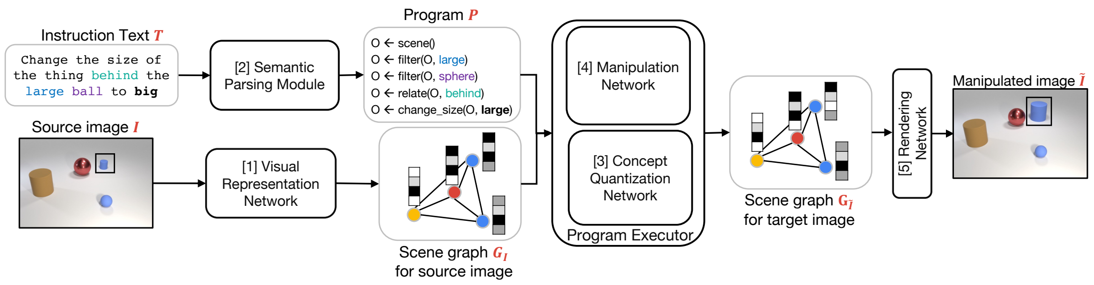

Instruction Text T
Change the size of the thing behind the large ball to big
[2] Semantic Parsing Module
Program P
O ← scene()
O ← filter(O, large)
O ← filter(O, sphere)
O ← relate(O, behind)
O ← change_size(O, large)
Source image I
[1] Visual Representation Network
Scene graph G_I for source image
[4] Manipulation Network
[3] Concept Quantization Network
Program Executor
Scene graph G_Ĩ for target image
[5] Rendering Network
Manipulated image Ĩ

Figure 3: High level architecture of NEUROSIM. See Section 3 for more details.

our DSL (Specified in Appendix Section A).

3] Concept Quantization Network: Any object in an image is defined by the set of *visual attributes* (A), and set of symbolic values (S_a) for each attribute $a \in A$. E.g., attributes can be *shape, size,* etc. Different symbolic values allowed for an attribute are also known as *concepts*. E.g., $S_{\text{color}} = \{\text{red, blue, green}, \ldots\}$. Each visual attribute $a \in A$ is implemented via a separate neural network $f_a(\cdot)$ which takes the object embedding as input and outputs the attribute value for the object in a *continuous (not symbolic)* space. Let $f_{\text{color}} : \mathbb{R}^{d_{\text{obj}}} \rightarrow \mathbb{R}^{d_{\text{attr}}}$ represent a neural network for the *color* attribute and consider $o \in \mathbb{R}^{d_{\text{obj}}}$ as the object embedding. Then, $v_{\text{color}} = f_{\text{color}}(o) \in \mathbb{R}^{d_{\text{attr}}}$ is the embedding for the object o pertaining to the color attribute. Each symbolic concept $s \in S_a$ for a particular attribute a (e.g., different colors) is also assigned a respective embedding in the same continuous space $\mathbb{R}^{d_{\text{attr}}}$. Such an embedding is denoted by c_s. These concept embeddings are initialized at random, and later on, fine-tuned during training. An attribute embedding (e.g. v_{color}) can be compared with the embeddings of all the concepts (e.g., $c_{\text{red}}, c_{\text{blue}}$, etc.) using cosine similarity, for the purpose of concept quantization of objects.

Training for VQA: As a first step, we train the above three modules via a curriculum learning process (Mao et al., 2019). The semantic parser is trained jointly with the concept quantization networks for generating programs for the question texts coming from the VQA dataset. The corresponding output programs are composed of primitive operations coming from the DSL (e.g. *filter*, *count*, etc.) and do not include constructs related to manipulation operations. This trains the first three modules with high accuracy on the VQA task.

3.3 Novel Modules and Training NEUROSIM

NEUROSIM training starts with three sub-modules trained on the VQA task as described in Section 3.2. Next, we extend the original DSL to include three additional functional sub-modules within the semantic parsing module, namely *add*, *remove*, and *change*. Refer to appendix section A for details on the DSL. We now reset the semantic parsing module and train it again from scratch for generating programs corresponding to *image manipulation instruction text* T. Such a program is subsequently used by the downstream pipeline to reason over the scene graph G_I and manipulate the image. In this step, the semantic parser is trained using an off-policy program search based `REINFORCE` (Williams, 1992) algorithm. Unlike the training of semantic parser for the VQA task, in this step, we *do not* have any final *answer like* reward supervision for training. Hence, we resort to a weaker form of supervision. In particular, consider an input instruction text T and set of all possible manipulation program templates $\mathbb{P}_t$ from which one can create any actual program P that is executable over the scene graph of the input image. For a program $P \in \mathbb{P}_t$, our reward is positive if this program P selects any object (or part of the scene graph) to be sent to the manipulation networks (change/add/remove). See Appendix C for more details. Once the semantic parser is retrained, we clamp the first three modules and continue using them for the purpose of parsing instructions and converting images into their scene graph representations. Scene graphs are manipulated using our novel module called *manipulation network* which is described next.

4] Manipulation Network: This is our key module responsible for carrying out the manipulation operations. We allow three kinds of manipulation operations – *add, remove,* and *change*. Each of these operations is a composition of a quasi-symbolic and symbolic operation. A symbolic operation corresponds to a function that performs the required structural changes (i.e. addition/deletion of a node or an edge) in the scene graph G_I against a given instruction. A quasi-symbolic operation is a dedicated neural network that takes the relevant part of G_I as input and computes new representations of nodes and edges that are compatible with the changes described in the parsed instruction.

(a) Change Network: For each visual attribute $a \in A$ (e.g. *shape, size,* …), we have a separate *change neural network* that takes the pair of *(object embedding, embedding of the changed concept)* as input and outputs the embedding of the *changed* object. This is the quasi-symbolic part of the change function, while the symbolic part is identity mapping. For e.g., let $g_{\text{color}} : \mathbb{R}^{d_{\text{obj}}+d_{\text{attr}}} \rightarrow \mathbb{R}^{d_{\text{obj}}}$ represent the neural network that changes the *color* of an object. Consider $o \in \mathbb{R}^{d_{\text{obj}}}$ as the object embedding and $c_{\text{red}} \in \mathbb{R}^{d_{\text{attr}}}$ as the concept embedding for the *red* color, then $\widetilde{o} = g_{\text{color}}(o; c_{\text{red}}) \in \mathbb{R}^{d_{\text{obj}}}$ represents the changed object embedding, whose color would be *red*. After applying the change neural network, we obtain the changed representation of the object $\widetilde{o} = g_a(o; c_{s_a^*})$, where s_a^* is the desired changed value for the attribute a. This network is trained using the following losses.

$$\ell_a = -\sum_{\forall s \in S_a} \mathbb{I}_{s=s_a^*} \log\left[p(h_a(\widetilde{o}) = s)\right] \tag{1}$$

$$\ell_{\overline{a}} = -\sum_{\forall a' \in A, a' \neq a} \sum_{\forall s \in S_{a'}} p(h_{a'}(o) = s) * \log[p(h_{a'}(\widetilde{o}) = s)] \tag{2}$$

where, $h_a(x)$ gives the concept value of the attribute a (in symbolic form $s \in S_a$) for the object x. The quantity $p\left(h_a(x) = s\right)$ denotes the probability that the concept value of the attribute a for the object x is equal to s and is given as follows $p\left(h_a(x) = s\right) = \exp^{dist(f_a(x), c_s)} / \sum_{\widetilde{s} \in S_a} \exp^{dist(f_a(x), c_{\widetilde{s}})}$ where, $dist(a, b) = (a^\top b - t_2)/t_1$ is the shifted and scaled cosine similarity, t_1, t_2 being constants. The first loss term ℓ_a penalizes the model if the (symbolic) value of the attribute a for the manipulated object is different from the desired value s_a^* in terms of probabilities. The second term $\ell_{\overline{a}}$, on the other hand, penalizes the model if the values of any of the other attributes a', deviate from their original values. Apart from these losses, we also include following additional losses.

$$\ell_{\text{cycle}} = \|o - g_a(\widetilde{o}; c_{\text{old}})\|_2; \tag{3}$$

$$\ell_{\text{consistency}} = \|o - g_a(o; c_{\text{old}})\|_2 \tag{4}$$

$$\ell_{\text{objGAN}} = -\sum_{o' \in O}[\log D((o') + \log(1 - D\left(g_a(o'; c)\right))] \tag{5}$$

where c_{old} is the original value of the attribute a of object o, before undergoing change. Intuitively the first loss term ℓ_{cycle} says that, changing an object and then changing it back should result in the same object. The second loss term $\ell_{\text{consistency}}$ intuitively means that changing an object o that has value c_{old} for attribute a, into a new object with the same value c_{old}, should not result in any change. These additional losses prevent the change network from changing attributes which are not explicitly taken care of in earlier losses (1) and (2). For e.g., rotation or location attributes of the objects that are not part of our DSL. We also impose an adversarial loss ℓ_{objGAN} to ensure that the new object embedding $\widetilde{o}$ is from the same distribution as real object embeddings. See Appendix C for more details.

(b) Remove Network: This network takes the scene graph G_I of the input image and removes the subgraph from G_I that contains the nodes (and incident edges) corresponding to the object(s) that need to be removed, and returns a new scene graph $G_{\widetilde{I}}$ which is reduced in size. The quasi-symbolic function for the remove network is identity.

(c) Add Network: For adding a new object into the scene, *add network* requires the symbolic values of different attributes, say $\{s_{a_1}, s_{a_2}, \ldots, s_{a_k}\}$, for the new object, e.g., {red, cylinder, …}. It also requires the spatial relation r (e.g. RightOf) of the new object with respect to an existing object in the scene. The add function first predicts the object (node) embedding $\widetilde{o}_{\text{new}}$ for the object to be added, followed by predicting edge embeddings for new edges incident on the new node. New object embedding is obtained as follows: $\widetilde{o}_{\text{new}} = g_{\text{addObj}}(\{c_{s_{a_1}}, c_{s_{a_2}}, \cdots, c_{s_{a_k}}\}, o_{rel}, c_r)$ where, o_{rel} is the object embedding of an existing object, relative to which the new object's position r is specified. For each existing objects o_i in the scene, an edge $\widetilde{e}_{\text{new},i}$ is predicted between the newly added object $\widetilde{o}_{\text{new}}$ and existing object o_i in following manner: $\widetilde{e}_{\text{new},i} = g_{\text{addEdge}}(\widetilde{o}_{\text{new}}, o_i)$. Functions $g_{\text{addObj}}(\cdot)$ and $g_{\text{addEdge}}(\cdot)$ are quasi-symbolic operations. Symbolic operations in *add network* comprise adding the above node and the incident edges into the scene graph.

The *add network* is trained in a *self-supervised* manner. For this, we pick a training image and create its scene graph. Next, we randomly select an object o from this image and quantize its concepts, along with a relation with any other object o_i in the same image. We then use our *remove* network to remove this object o from the scene. Finally, we use the quantized concepts and the relation that were gathered above and add this object o back into the scene graph using $g_{\text{addObj}}(\cdot)$ and $g_{\text{addEdge}}(\cdot)$. Let the embedding of the object after adding it back is $\widetilde{o}_{\text{new}}$. The training losses are as follows:

$$\ell_{\text{concepts}} = -\sum_{j=1}^{k} \log\left(p(h_{a_j}(\widetilde{o}_{\text{new}}) = s_{a_j})\right) \quad (6)$$

$$\ell_{\text{relation}} = -\log(p(h_{\text{r}}(\widetilde{o}_{\text{new}}, o_i) = r)) \quad (7)$$

$$\ell_{\text{objSup}} = \|o - \widetilde{o}_{\text{new}}\|_2 \quad (8)$$

$$\ell_{\text{edgeSup}} = \sum\nolimits_{i \in O} \|e_{\text{old},i} - \widetilde{e}_{\text{new},i}\|_2 \quad (9)$$

$$\ell_{\text{edgeGAN}} = -\sum_{\forall i \in O} [\log D(\{o; e_{\text{old},i}; o_i\}) + \log(1 - D(\{\widetilde{o}_{\text{new}}; \widetilde{e}_{\text{new},i}; o_i\}))] \quad (10)$$

where s_{a_j} is the required (symbolic) value of the attribute a_j for the original object o, and r is the required relational concept. O is the set of the objects in the image, $e_{\text{old},i}$ is the edge embedding for the edge between original object o and its neighboring object o_i. Similarly, $\widetilde{e}_{\text{new},i}$ is the corresponding embedding of the same edge but after when we have (removed + added back) the original object. The loss terms ℓ_{concepts} and ℓ_{relation} ensure that the added object comprises desired values of attributes and relation, respectively. Since we had first removed and then added the object back, we already have the original edge and object representation, and hence we use them in loss terms given in equation 9. We use adversarial loss equation 10 for generating real (object, edge, object) triples and also a loss similar to equation 5 for generating real objects.

3.4 Image Rendering from Scene Graph

5] Rendering Network: Following Johnson et al. (2018), the scene graph for an image is first generated using the *visual representation network*, which is the processed by a GCN and passed through a mask regression network followed by a box regression network to generate a coarse 2-dimensional structure (scene layout). A Cascaded Refinement Network (Chen and Koltun, 2017) is then employed to generate an image from the scene layout. A *min-max adversarial training procedure* is used to generate realistic images, using a patch-based and object-based discriminator.

4 Experiments

Datasets: Among the existing datasets, CSS (Vo et al., 2019) contains simple 0-hop instructions and is primarily designed for the text-guided image retrieval task. Other datasets such as i-CLEVR (El-Nouby et al., 2019) and CoDraw are designed for iterative image editing. i-CLEVR contains only "add" instructions and CoDraw doesn't contain multi-hop instructions. Hence we created our own *multi-object multi-hop instruction* based image manipulation dataset, referred to as CIM-NLI. This dataset was generated with the help of CLEVR toolkit (Johnson et al., 2017b). CIM-NLI consists of (Source image I, Instruction text T, Target image $\widetilde{I}^*$) triplets. The dataset contains a total of $18K$, $5K$, $5K$ unique images and $54K$, $14K$, $14K$ instructions in the *train, validation and test* splits respectively. Refer to Appendix B for more details about the dataset generation and dataset splits.

Baselines: We compare our model with purely supervised approaches such as TIM-GAN (Zhang et al., 2021), GeNeVA (El-Nouby et al., 2019) and InstructPix2Pix (Brooks et al., 2023). In order to make a fair and meaningful comparison between the two kinds (supervised and our, weakly-supervised) approaches, we carve out the following set-up. Assume the cost required to create one single annotated example for image manipulation task be α_m while the corresponding cost for the VQA task be α_v. Let $\alpha = \alpha_m/\alpha_v$. Let β_m be the number of annotated examples required by a supervised baseline for reaching a performance level of η_m on the image manipulation task. Similarly, let β_v be the number of annotated VQA examples required to train NEUROSIM to reach the performance level of η_v. Let $\beta = \beta_m/\beta_v$. We are interested in figuring out the range of β for which performance of our system (η_v) is at least as good as the baseline (η_m). Correspondingly we can compute the ratio of the labeling effort required, i.e., $\alpha * \beta$, to reach these performance levels.

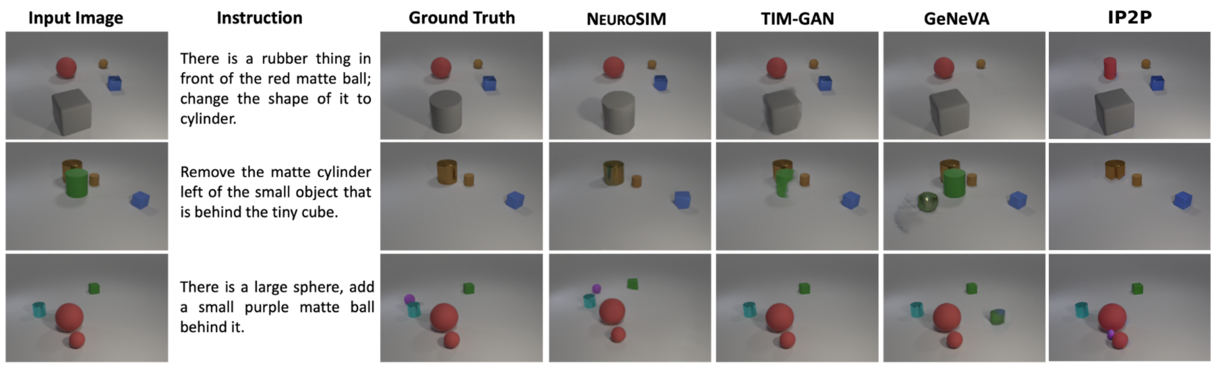

Input Image
Instruction
Ground Truth
NeuroSIM
TIM-GAN
GeNeVA
IP2P
There is a rubber thing in front of the red matte ball; change the shape of it to cylinder.
Remove the matte cylinder left of the small object that is behind the tiny cube.
There is a large sphere, add a small purple matte ball behind it.

Figure 4: Visual comparison of NeuroSIM with various baselines. See Section 4.4 for more details.

If $\alpha * \beta > 1$, our system achieves the same or better performance, with lower annotation cost. Weakly supervised models (Li et al., 2020; Nam et al., 2018) are designed for a problem setting different from ours – single salient object scenes, simple 0-hop instructions (Refer Section 2 for details). Further, they require paired images and their textual descriptions as annotations. We, therefore, do not compare with them in our experiments. See Appendix G, H for computational resources and hyperparameters respectively.

Evaluation Metrics: For evaluation on image manipulation task, we use three metrics - i) *FID*, ii) Recall@k, and iii) Relational-similarity (rsim). FID (Heusel et al., 2017) measures the realism of the generated images. We use the implementation proposed in Parmar et al. (2022) to compute FID. Recall@k measures the semantic similarity of gold manipulated image $\widetilde{I}^*$ and system produced manipulated image $\widetilde{I}$. For computing Recall@k, we follow Zhang et al. (2021), i.e. we use $\widetilde{I}$ as a query and retrieve images from a corpus comprising the entire test set. *rsim* measures how many of the ground truth relations between the objects are present in the generated image. We follow (El-Nouby et al., 2019) to implement rsim metric that uses predictions from a trained object-detector (Faster-RCNN) to perform relation matching between the scene-graphs of ground-truth and generated images.

4.1 Performance with varying Dataset Size

Table 2 compares the performance of NeuroSIM other SoTA methods two level of β 0.054 and 0.54 representing use of 10% and 100% samples from CIM-NLI. Despite being weakly supervised, NeuroSIM performs significantly better than the baselines with just 10k data samples (especially TIM-GAN) and not too far from diffusion model

Method	$\beta = 0.054$ FID	$R1$	$R3$	rsim	$\beta = 0.54$ FID	$R1$	$R3$	rsim
GeNeVA	42.6	6.6	58.7	80.1	28.5	4.6	64.4	84.9
TIM-GAN	24.2	31.9	74.2	88.4	22.7	58.1	90.2	94.0
IP2P	3.4	40.6	77.0	88.8	2.2	49.2	84.8	94.5
NeuroSIM	35.0	45.3	65.5	91.3	35.1	45.5	66.7	91.5

Table 2: Performance comparison of NeuroSIM with TIM-GAN and GeNeVA, and InstructPix2Pix (IP2P) with 10% data ($\beta = 0.054$) and full data ($\beta = 0.54$). We always use $100K$ VQA examples (5K Images, 20 questions per image) for our weakly supervised training.$R1$, $R3$ correspond to Recall@1,3 respectively. FID: lower is better; Recall/rsim: higher is better. See Section 4.1 for more details.

based IP2P in full data setting, using the R@1 performance metric. This clearly demonstrates the strength of our approach in learning to manipulate while only making use of VQA annotations. We hypothesize that, in most cases, NeuroSIM will be preferable since we expect the cost of annotating an output image for manipulation to be significantly higher than the cost of annotating a VQA example. To reach the performance of the NeuroSIM in a low data regime, TIM-GAN requires a larger number of expensive annotated examples (ref. Table 13 in Appendix). The FID metric shows similar trend across dataset sizes and across models. The FID scores for NeuroSIM could potentially be improved by jointly training VQA module along with image decoder and is a future direction.

We evaluate InstructPix2Pix (IP2P) (Brooks et al., 2023), a state-of-the-art pre-trained diffusion model for image editing, in a zero-shot manner on the CIM-NLI dataset. Considering its extensive pre-training, we expect IP2P to have learned the concepts present in the CIM-NLI dataset. In this setting IP2P achieves a FID score of 33.07 and R@1 score of 7.48 illustrating the limitations of large-scale models in effectively executing complex instruction-based editing tasks without full dataset fine-tuning. Table 2 contains the results obtained by IP2P after fine-tuning for 16k iterations on CIM-NLI dataset.

Method	Larger Scenes		Hops		
	$R1$	$R3$	ZH	MH	$\triangle$
GeNeVA $54K$	5.0	65.8	6.3	6.4	(+0.1)
GeNeVA $5.4K$	8.2	64.6	8.5	9.9	(+1.4)
TIM-GAN $54K$	66.3	92.4	84.0	76.2	(-7.8)
TIM-GAN $5.4K$	30.2	80.7	56.4	41.6	(-14.8)
IP2P $54K$	69.2	99.8	72.5	67.4	(-5.1)
IP2P $5.4K$	64.9	99.4	69.3	54.8	(-14.5)
NEUROSIM $5.4K$	63.7	89.1	64.5	63.0	(-1.5)

Table 3: (Left) Performance on generalization to Larger Scenes. (Right) $R1$ results for 0-hop (ZH) vs multi-hop (MH) instruction-guided image manipulation. See Sections 4.2 and 4.3 for more details.

4.2 Performance versus Reasoning Hops

Table 3 (right) compares baselines with NEUROSIM for performance over instructions requiring zero-hop (ZH) versus multi-hop ($1-3$ hops) (MH) reasoning. Since there are no *Add* instructions with ZH, we exclude them from this experiment for the comparison to be meaningful. GeNeVA performs abysmally on both ZH as well as MH. We see a significant drop in the performance of both TIM-GAN and IP2P when going from ZH to MH instructions, both for training on $5.4K$, as well as, $54K$ datapoints. In contrast, NEUROSIM trained on 10% data, sees a performance drop of only 1.5 points showing its robustness for complex reasoning tasks.

4.3 Zero-shot Generalization to Larger Scenes

We developed another dataset called CIM-NLI-LARGE, consisting of scenes having $10-13$ objects (See Appendix B for details). We study the combinatorial generalization ability of NEUROSIM and the baselines when the models are trained on CIM-NLI containing scenes with $3-8$ objects only and evaluated on CIM-NLI-LARGE. Table 3 captures such a comparison. NEUROSIM does significantly better, i.e., 33 pts (R1) than TIM-GAN and is competitive with IP2P when trained on 10% ($5.4K$data points) of CIM-NLI. We do see a drop in performance relative to baselines when they are trained on full (54K) data, but this is expected as effect of supervision takes over, and ours is a weakly supervised model. Nevertheless, this experiments demonstrates the effectiveness of our model for zero-shot generalization, despite being weakly sueprvised.

4.4 Qualitative Analysis and Interpretability

Figure 4 shows anecdotal examples for visually comparing NEUROSIM with baselines. Note, GeNeVA either performs the wrong operation on the image (row #1, 2, 3) or simply copies the input image to output without any modifications. TIM-GAN often makes semantic errors which show its lack of reasoning (row #3) or make partial edits (row #1). IP2P also suffers from this where it edits incorrect object (row #1,2). Compared to baselines, NEUROSIM produces semantically more meaningful image manipulation. NEUROSIM can also easily recover occluded objects (row #4). For more results, see Appendix I, J. NEUROSIM produces interpretable output programs, showing the steps taken by the model to edit the images, which also helps in detecting errors (ref. Appendix L).

4.5 Evaluating Manipulated Scene Graph

We strongly believe *image rendering module* of NEUROSIM pipeline and *encoder* modules used for computing Recall@k add some amount of inefficiencies resulting in lower $R1$ and $R3$ scores for us. Therefore, we decide to assess the quality of manipulated scene graph $G_{\tilde{I}}$. For this, we consider the *text guided image retrieval* task proposed by (Vo et al., 2019). In this task, an image from the database has to be retrieved which would be the closest match to the desired manipulated image. Therefore, we use our manipulated scene graph $G_{\tilde{I}}$ as the latent representation of the input instruction and image for image retrieval. We retrieve images from the database based on a novel *graph edit distance* between NEUROSIM generated $G_{\tilde{I}}$ of the desired manipulated images, and scene graphs of the images in the database. This distance is defined using the *Hungarian algorithm* (Kuhn, 1955) with a simple cost defined between any 2 nodes of the graph (ref. Appendix D for details). Table 4 captures the performance of NEUROSIM and other popular baselines for the image retrieval task. NEUROSIM significantly outperforms supervised learning baselines by a margin of $\sim 50\%$ without using output image supervision, demonstrating that NEUROSIM

Method	$R1$	$R3$
Text-Only	0.2	0.4
Image-Only	34.1	83.6
Concat	39.5	86.9
TIRG	34.8	84.6
NEUROSIM	85.8	92.9

Table 4: $G_{\tilde{I}}$ Quality via image retrieval.

meaningfully edits the scene graph. Refer to Section 4.7 for human evaluation results and Appendix Section D-E, K, for more results including results on Minecraft dataset and ablations.

4.6 A Hybrid Approach using NEUROSIM

From Table 3, we observe that both TIM-GAN and IP2P suffer a significant drop in performance when moving from ZH to MH instructions, whereas NEUROSIM is fairly robust to this change. Further, we note that the manipulation instructions in our dataset are multi-hop in terms of reasoning, but once an object of interest is identified, the actual manipulation operation can be seen as single hop. We use this observation to design a hybrid supervised baseline that utilizes the superior reasoning capability of NEUROSIM and high quality editing and generation capabilities of IP2P.

We take the CIM-NLI test set and parse the text-instructions through our trained semantic-parser to obtain the object embeddings over which the manipulation operation is to be performed. We utilize our trained query networks to obtain the symbolic attributes such as color, shape, size and material of the identified object. Using these attributes we simplify a complex multi-hop instruction into a simple instruction with 0 or 1 hops using a simple template based approach (see Appendix Section N for details). These simplified instructions are fed to the fine-tuned IP2P model to generate the edited images. We refer to our hybrid approach as IP2P-NS where NS refers to Neuro-Symbolic. Table 5 presents the results. We find that there is a clear advantage of using a hybrid neuro-symbolic model integrating NEUROSIM with IP2P. We see a significant gain on FID, recall, rsim when we use the hybrid approach, especially in the low resource setting ($\beta = 0.054$). Compared to IP2P, the hybrid neuro-symbolic approach results in better FID, recall and rsim scores, except a small drop in R1 for $\beta = 0.54$ setting. This opens up the possibility of further exploring such hybrid models in future for improved performance (in the supervised setting).

	$\beta = 0.054$				$\beta = 0.54$			
Method	FID	$R1$	$R3$	rsim	FID	$R1$	$R3$	rsim
IP2P	3.4	40.6	77.0	88.8	2.2	49.2	84.8	94.5
NEUROSIM	35.0	45.3	65.5	91.3	35.1	45.5	66.7	91.5
IP2P-NS	1.96	45.5	83.2	94.0	1.8	48.0	85.5	95.6

Table 5: Comparison between IP2P-NS and IP2P.

Qn.	NEUROSIM 5.4K	TIM-GAN 54K	IP2P 54K
Q1	0.41	0.27	0.25
Q2	0.33	0.84	0.78

Table 6: Human evaluation comparing various models.

4.7 Human Evaluation

For the human evaluation study, we presented 10 evaluators with a set of five images each, including: The input image, the ground-truth image and manipulated images generated by NEUROSIM 5.4K, TIM-GAN 54K, and IP2P 54K. Images generated by the candidate models were randomly shuffled to prevent any bias. Evaluators were asked two binary questions, each requiring a 'yes' (1) or 'no' (0) response, to assess the models: (Q1) Does the model perform the desired change mentioned in the input instruction?, (Q2) Does the model not introduce any undesired change elsewhere in the image? Refer to Appendix Section M for more details about exact questions and the human evaluation process.

The average scores from the evaluators across different questions can be found in Table 6. The study achieved a high average Fleiss' kappa score (Fleiss et al., 2013) of 0.646, indicating strong inter-evaluator agreement. Notably, NEUROSIM (5.4K) outperforms TIM-GAN and IP2P (54K) in Q1 suggesting its superior ability to do reasoning, and identify the relevant object as well as affect the desired change. In contrast, TIM-GAN and IP2P score significantly better in Q2, demonstrating their ability not to introduce unwanted changes elsewhere in the image, possibly due to better generation quality compared to NEUROSIM.

5 Conclusion

We present a neuro-symbolic, interpretable approach NEUROSIM to solve image manipulation task using weak supervision in the form of VQA annotations. Our approach can handle multi-object scenes with complex instructions requiring multi-hop reasoning, and solve the task without any output image supervision. We also curate a dataset of image manipulation and demonstrate the potential of our approach compared to supervised baselines. Future work includes understanding the nature of errors made by NEUROSIM, having a human in the loop to provide feedback to the system for correction, and experimenting with real image datasets.

6 Ethics Statement

All the datasets used in this paper were synthetically generated and do not contain any personally identifiable information or offensive content. The ideas and techniques proposed in this paper are useful in designing interpretable natural language-guided tools for image editing, computer-aided design, and video games. One of the possible adverse impacts of AI-based image manipulation is the creation of *deepfakes* (Vaccari and Chadwick, 2020) (using deep learning to create fake images). To counter deepfakes, several researchers (Dolhansky et al., 2020; Mirsky and Lee, 2021) have also looked into the problem of detecting real vs. fake images.

7 Limitations

A limitation of our approach is that when transferring to a new domain, having different visual concepts requires not only learning new visual concepts but also the DSL needs to be redefined. Automatic learning of DSL from data has been explored in some prior works (Ellis et al., 2021, 2018), and improving our model using these techniques are future work for us. We can also use more powerful graph decoders for image generation, for improved image quality, which would naturally result in stronger results on image manipulation.

Acknowledgements

We thank anonymous reviewers for their insightful suggestions that helped in greatly improving our paper. We also thank Rushil Gupta and other members of IIT Delhi DAIR group for their helpful comments and suggestions on this work. This work was supported by an IBM AI Horizons Network (AIHN) grant. We thank IIT Delhi HPC facility[2], IBM cloud facility, and IBM Cognitive Computing Cluster (CCC) for computational resources. Ashish Goswami is a PhD student at Yardi-ScAI@IIT-Delhi and is supported by Yardi School of AI Publication Grant. Parag Singla was supported by the DARPA Explainable Artificial Intelligence (XAI) Program with number N66001-17-2-4032, IBM AI Horizon Networks (AIHN) grant and IBM SUR awards. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views or official policies, either expressed or implied, of the funding agencies.

[2] *http://supercomputing.iitd.ac.in*

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Learning to compose neural networks for question answering. In *Proceedings of NAACL-HLT*, pages 1545–1554.

Tim Brooks, Aleksander Holynski, and Alexei A. Efros. 2023. Instructpix2pix: Learning to follow image editing instructions. In *Proceedings of CVPR*, pages 18392–18402.

Shaowei Cai and Kaile Su. 2012. Configuration checking with aspiration in local search for SAT. In *Proceedings of AAAI*, pages 434–440.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. 2017. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848.

Lichang Chen, Guosheng Lin, Shijie Wang, and Qingyao Wu. 2020. Graph edit distance reward: Learning to edit scene graph. In *Proceedings of ECCV*, pages 539–554.

Qifeng Chen and Vladlen Koltun. 2017. Photographic image synthesis with cascaded refinement networks. In *Proceedings of ICCV*, pages 1520–1529.

Blender Online Community. 2018. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam.

Wang-Zhou Dai, Qiu-Ling Xu, Yang Yu, and Zhi-Hua Zhou. 2019. Bridging machine learning and logical reasoning by abductive learning. In *Proceedings of NeurIPS*, pages 2811–2822.

Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. 2020. The deepfake detection challenge (dfdc) dataset. *ArXiv preprint*, abs/2006.07397.

Hao Dong, Simiao Yu, Chao Wu, and Yike Guo. 2017. Semantic image synthesis via adversarial learning. In *Proceedings of ICCV*, pages 5707–5715.

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. 2019. Neural logic machines. In *Proceedings of ICLR*.

Alaaeldin El-Nouby, Shikhar Sharma, Hannes Schulz, R. Devon Hjelm, Layla El Asri, Samira Ebrahimi Kahou, Yoshua Bengio, and Graham W. Taylor. 2019. Tell, draw, and repeat: Generating and modifying images based on continual linguistic instruction. In *Proceedings of ICCV*, pages 10303–10311.

Kevin Ellis, Lucas Morales, Mathias Sablé-Meyer, Armando Solar-Lezama, and Josh Tenenbaum. 2018. Learning libraries of subroutines for neurally–guided bayesian program induction. In *Proceedings of NeurIPS*.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. 2021. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of PLDI*, pages 835–850.

Feng-Lei Fan, Jinjun Xiong, Mengzhou Li, and Ge Wang. 2021. On interpretability of artificial neural networks: A survey. *IEEE Transactions on Radiation and Plasma Medical Sciences*, 5(6):741–760.

Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. 2013. *Statistical methods for rates and proportions*. john wiley & sons.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Proceedings of NeurIPS*, pages 2672–2680.

Chi Han, Jiayuan Mao, Chuang Gan, Josh Tenenbaum, and Jiajun Wu. 2019. Visual concept-metaconcept learning. In *Proceedings of NeurIPS*, pages 5002–5013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of CVPR*, pages 770–778.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of NeurIPS*, pages 6626–6637.

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. 2017. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of ICCV*.

Wentao Jiang, Ning Xu, Jiayun Wang, Chen Gao, Jing Shi, Zhe Lin, and Si Liu. 2021. Language-guided global image editing via cross-modal cyclic mechanism. In *Proceedings of ICCV*, pages 2115–2124.

Justin Johnson, Agrim Gupta, and Li Fei-Fei. 2018. Image generation from scene graphs. In *Proceedings of CVPR*, pages 1219–1228.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. 2017a. Inferring and executing programs for visual reasoning. In *Proceedings of ICCV*, pages 3008–3017.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. 2017b. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of CVPR*, pages 1988–1997.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of ICLR*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of NeurIPS*, pages 1106–1114.

H. W. Kuhn. 1955. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97.

Bowen Li, Xiaojuan Qi, Thomas Lukasiewicz, and Philip H. S. Torr. 2020. Manigan: Text-guided image manipulation. In *Proceedings of CVPR*, pages 7877–7886. IEEE.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proceedings of ICLR*.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. 2019. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *Proceedings of ICLR*.

Gary Marcus, Ernest Davis, and Scott Aaronson. 2022. A very preliminary analysis of dall-e 2. *ArXiv preprint*, abs/2204.13807.

Yisroel Mirsky and Wenke Lee. 2021. The creation and detection of deepfakes: A survey. *ACM Computing Surveys (CSUR)*, 54(1):1–41.

Seonghyeon Nam, Yunji Kim, and Seon Joo Kim. 2018. Text-adaptive generative adversarial networks: Manipulating images with natural language. In *Proceedings of NeurIPS*, pages 42–51.

Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. 2022. On aliased resizing and surprising subtleties in gan evaluation. In *Proceedings of CVPR*, pages 11400–11410.

Duc Nghia Pham, John Thornton, and Abdul Sattar. 2007. Building structure into local search for sat. In *Proceedings of IJCAI*, pages 2359–2364.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with clip latents. *ArXiv preprint*, abs/2204.06125.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. 2022. Photorealistic text-to-image diffusion models with deep language understanding. *ArXiv preprint*, abs/2205.11487.

Jing Shi, Ning Xu, Yihang Xu, Trung Bui, Franck Dernoncourt, and Chenliang Xu. 2021. Learning by planning: Language-guided global image editing. In *Proceedings of CVPR*, pages 13590–13599.

Cristian Vaccari and Andrew Chadwick. 2020. Deepfakes and disinformation: Exploring the impact of synthetic political video on deception, uncertainty, and trust in news. *Social Media+ Society*, 6(1).

Nam Vo, Lu Jiang, Chen Sun, Kevin Murphy, Li-Jia Li, Li Fei-Fei, and James Hays. 2019. Composing text and image for image retrieval - an empirical odyssey. In *Proceedings of CVPR*, pages 6439–6448.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv preprint*, abs/1609.08144.

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. 2018. Neural-symbolic VQA: disentangling reasoning from vision and language understanding. In *Proceedings of NeurIPS*, pages 1039–1050.

Dong Yu and Li Deng. 2016. *Automatic speech recognition*, volume 1. Springer.

Tianhao Zhang, Hung-Yu Tseng, Lu Jiang, Weilong Yang, Honglak Lee, and Irfan Essa. 2021. Text as neural operator: Image manipulation by text instruction. In *Proceedings of ACM MM*, pages 1893–1902.

Appendix

A Domain Specific Language (DSL)

Table 7 captures the DSL used by our NeuroSIM pipeline. The first 5 constructs in this table are common with the DSL used in Mao et al. (2019). The last 3 operations (`Change`, `Add`, and `Remove`) were added by us to allow for the manipulation operations. Table 8 shows the type system used by the DSL in this work. The first 5 types are inherited from (Mao et al., 2019) while the last one is an extension of the type system for handling the inputs to the `Add` operator.

B Dataset Details

We use CLEVR dataset and CLEVR toolkit (code to generate the dataset). These are public and are under CC and BSD licenses respectively, and are used by many works, including ours, for research purposes. We now give details of the datasets we create, building upon CLEVR.

B.1 CIM-NLI Dataset

This dataset was generated with the help of CLEVR toolkit (Johnson et al., 2017b) by using following recipe.

1. First, we create a source image I and the corresponding scene data by using *Blender* (Community, 2018) software.

2. For each source image I created above, we generate multiple instruction texts T's using its scene data. These are generated using templates, similar to question templates proposed by (Johnson et al., 2017b).

3. For each such (I, T) pair, we attach a corresponding symbolic program P (not used by NeuroSIM though) as well as scene data for the corresponding changed image.

4. Finally, for each (I, T) pair, we generate the target gold image $\widetilde{I}^*$ using Blender software and its scene data from the previous step.

Below are some of the important characteristics of the CIM-NLI dataset.

- Each source image I comprises several objects and each object comprises four visual attributes - *color, shape, size,* and *material.*

- Each instructions text T comprises one of the following three kinds of manipulation operations - *add, remove,* and *change.*

- An *add* instruction specifies *color, shape, size,* and *material* of the object that needs to be added. It also specifies a direct (or indirect) relation with one or more existing objects (called reference object(s)). The number of relations that are required to traverse for nailing down the target object is referred to as *# of reasoning hops* and we have allowed instructions with up to 3-*hops reasoning*. We do not generate any 0-hop instruction for *add* due to ambiguity of where to place the object inside the scene.

- A *change* instruction first specifies zero or more attributes to uniquely identify the object that needs to be changed. It may also specify a direct (or indirect) relation with one or more existing reference objects. Lastly, it specifies the target values of an attribute for the identified object which needs to be changed.

- A *remove* instruction specifies zero or more attributes of the object(s) to be removed. Additionally, it may specify a direct (or indirect) relation with one or more existing reference objects.

Table 9 captures the fine grained statistics about the CIM-NLI dataset. Specifically, it further splits each of the *train, validation,* and *test* set across the instruction types - *add, remove,* and *change*.

B.2 CIM-NLI-LARGE Dataset

We created another dataset called CIM-NLI-LARGE to test the generalization ability of NeuroSIM on images containing more number of objects than training images. CIM-NLI-LARGE tests the *zero-shot transfer* ability of both NeuroSIM and baselines on scenes containing more objects.

Each image in CIM-NLI-LARGE dataset comprises of $10-13$ objects as opposed to $3-8$ objects in CIM-NLI dataset which was used to train NeuroSIM. The CIM-NLI-LARGE dataset consists of $1K$ unique input images. We have created 3 instructions for each image resulting in a total of $3K$ instructions. The number of *add* instructions is significantly less since there is very little free space available in the scene to add new objects. To create scenes with 12 and 13 objects, we made all objects as *small size* and the minimum distance between

Operation	Signature [*Output* ← *Input*])	Semantics
`Scene`	ObjSet ← ()	Returns all objects in the scene.
`Filter`	ObjSet ← (ObjSet, ObjConcept)	Filter out a set of objects from ObjSet that have a concept (e.g. red) specified in ObjConcept.
`Relate`	ObjSet ← (ObjSet, RelConcept, Obj)	Filter out a set of objects from ObjSet that have concept specified relation concept (e.g. RightOf) with object Obj.
`Query`	ObjConcept ← (Obj, Attribute)	Returns the Attribute value for the object Obj.
`Exist`	Bool ← (ObjSet)	Checks if the set ObjSet is empty.
`Change`	Obj ← (Obj, Concept)	Changes the attribute value of the input object (Obj), corresponding to the input concept, to Concept
`Add`	Graph ← (Graph, RelConcept, Obj, ConceptSet)	Adds an object to the input graph, generating a new graph having the object with attribute values as ConceptSet, and present in relation RelConcept of the input Obj
`Remove`	Graph ← (Graph, ObjSet)	Removes the input objects and their edges from the input graph to output a new graph

Table 7: Extended Domain Specific Language (DSL) used by NEUROSIM.

objects was reduced so that all objects could fit in the scene. Table 10 captures the statistics about this dataset.

B.3 Multi-hop Instructions

In what follows, we have given examples of the instructions that require multi-hop reasoning to nail down the location/object to be manipulated in the image.

- *Remove the tiny green rubber ball.* (0-hop)
- *There is a block right of the tiny green rubber ball, remove it.* (1-hop)
- *Remove the shiny cube left of the block in front of the gray thing.* (2-hop)
- *Remove the small thing that is left of the brown matte object behind the tiny cylinder that is behind the big yellow metal block.* (3-hop)

C Model Details

C.1 Semantic Parser

C.1.1 Details on Parsing

We begin by extending the type system of (Mao et al., 2019) and add `ConceptSet` because our *add* operation takes as input a set of concepts depicting attribute values of the new object being added (refer Table 8 for the details). Next, in a manner similar to (Mao et al., 2019), we use a rule based system for extracting concept words from the input text. We, however, add an extra rule for extracting `ConceptSet` from the input sentence. Rest of the semantic parsing methodology remains the same as given in (Mao et al., 2019), with the difference being that our training is weakly supervised (refer Section 3.3 of the main paper).

C.1.2 Training

As explained in Section 3.3 of the main paper, for training with weaker form of supervision, we use an off-policy program search based `REINFORCE` (Williams, 1992) algorithm for calculating the exact gradient. For this, we define a set of all possible program templates $\mathbb{P}_t$. For a given input instruc-

Type	Remarks
`ObjConcept`	Concepts for any given object, such as *blue, cylinder,* etc.
`Attribute`	Attributes for any given object, such as *color, shape,* etc.
`RelConcept`	Relational concepts for any given object pair, such as *RightOf, LeftOf,* etc.
`Object`	Depicts a single object
`ObjectSet`	Depicts multiple objects
`ConceptSet`	A set of elements of ObjConcept type

Table 8: Extended type system for the DSL used by NEUROSIM.

Operation	Split	# $(I, T, \widetilde{I}^*)$	# reasoning hops			# objects		
			min	mean	max	min	mean	max
Add	train	17827	1	2.00	3	3	5.51	8
	valid	4459	1	2.00	3	3	5.50	8
	test	4464	1	2.00	3	3	5.45	8
Remove	train	15999	0	1.50	3	3	5.50	8
	valid	5000	0	1.50	3	3	5.50	8
	test	5000	0	1.50	3	3	5.48	8
Change	train	19990	0	1.50	3	3	5.45	8
	valid	4996	0	1.50	3	3	5.56	8
	test	4998	0	1.50	3	3	5.52	8

Table 9: Statistics of CIM-NLI dataset introduced in this paper.

tion text T, we create a set of all possible programs $\{P_T\}$ from $\mathbb{P}_t$. For e.g. given a template $\{remove(relate(\cdot, filter(\cdot, scene())))\}$, this is filled in all possible ways, with concepts, conceptSet, attributes and relational concepts extracted from the input sentence to get programs for this particular template. All such programs created using all templates form the set P_T. All P_T are executed over the scene graph of the input image. A typical program structure in our work is of the form *manip_op(reasoning())*, where *manip_op* represents the manipulation operator, for example *change, add,* or *remove*; and *reasoning()* either selects objects for *change* or *remove*, or it selects a reference object for adding another object in relation to it. After a hyperparameter search for the reward (refer Section H of the appendix), we assign a reward of +8 if the *reasoning()* part of the program leads to an object being selected for *change/remove* instruction or a related object being selected for *add* instruction. If no such object is selected, we give a reward of +2. Reward values were decided on the basis of validation set accuracy. We find that with this training strategy, we achieve the validation set accuracy of 95.64%, where this accuracy is calculated based on whether a program lead to an object being selected or not. Note, this is a proxy to the actual accuracy. For finding the actual accuracy, we would need a validation set of (instruction, ground truth output program) pairs, but we do not use this supervised data for training or validation.

Operation	# $(I, T, \widetilde{I}^*)$	# reasoning hops			# objects		
		min	mean	max	min	mean	max
Add	393	1	2.0	3	10	11.53	13
Remove	524	0	1.50	3	10	11.48	13
Change	2083	0	1.51	3	10	11.50	13

Table 10: Statistics of CIM-NLI-LARGE dataset.

C.2 Manipulation Network

In what follows, we provide finer details of manipulation network components.

Change Network: As described in Section 3.3 of the main paper, we have a *change neural network* for each attribute. For changing the current attribute value of a given object o, we use the following neural network: $\widetilde{o} = g_a(o; c_{s_a^*})$, where s_a^* is the desired changed value for the attribute a. $\widetilde{o}$ is the new representation of the object. We model $g_a(\cdot)$ by a single layer neural network without having any non-linearity. The input dimension of this neural network is $(256 + 64)$ because we concatenate the object representation $o \in \mathbb{R}^{256}$ with the desired concept representation $d \in \mathbb{R}^{64}$. We pass this concatenated vector through $g_a(\cdot)$ to get the revised representation of the object: $\widetilde{o} \in \mathbb{R}^{256}$.

The loss used to train the weights of the change network is a weighted sum of losses equation 1 to equation 5 given in the main paper. This leads to the overall loss function given below.

$$L_{\text{overall\_change}} = \lambda_1\, \ell_a + \lambda_2\, \ell_{\overline{a}} + \lambda_3\, \ell_{\text{cycle}} + \lambda_4\, \ell_{\text{consistency}} + \lambda_5\, \ell_{\text{objGAN}}$$

where, ℓ_{objGAN} above is the modified GAN loss (Goodfellow et al., 2014). Here $\lambda_1 = 1$, $\lambda_2 = 1/((\text{num\_attrs}-1)*(\text{num\_concepts}))$, $\lambda_3 = \lambda_4 = 10^3$, and $\lambda_5 = 1/(\text{num\_objects})$. Here, (num_objects) is the number of objects in input image, (num_attrs) is the total number of attributes for each object, and (num_concepts) are the total number of concepts in the NSCL (Mao et al., 2019) framework.

The object discriminator is a neural network with input dimension 256 and a single 300 dimensional hidden layer with ReLU activation function. This discriminator is trained using standard GAN objective ℓ_{objGAN}. See Fig 5a for an overview of the change operator

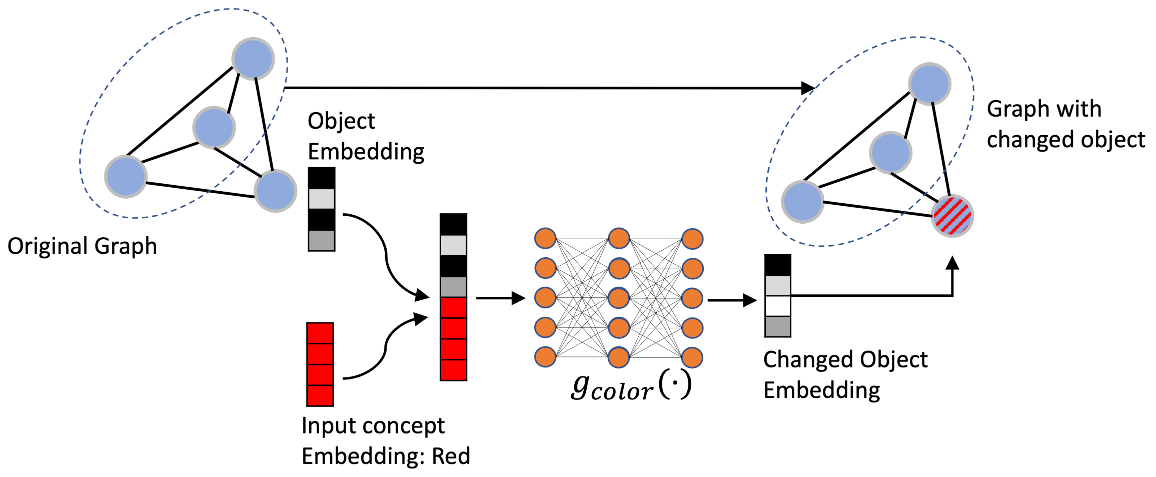
Remove Network: The remove network is a symbolic operation as described in Section 3.3 of the main paper. That is, given an input set of objects, the remove operation deletes the subgraph of the scene graph that contains the nodes corresponding to removed objects and the edges incident on those nodes. See Fig 5c for an overview of the remove operator.

Add Network: The neural operation in the add operator comprises of predicting the object representation for the newly added object using a function $g_{\text{addObj}}(\cdot)$. This function is modeled as a single layer neural network without any activation. The input to this network is a concatenated vector $[[c_{s_{a_1}}, c_{s_{a_2}}, \cdots, c_{s_{a_k}}], o_{rel}, c_r]$, where $[c_{s_{a_1}}, c_{s_{a_2}}, \cdots, c_{s_{a_k}}]$ represents the concatenation of all the concept vectors of the desired new objects. The vector o_{rel} is the representation of the object with whom the relation (i.e. position) of the new object has been specified and c_r is the concept vector for that relationship. The input dimension of $g_{\text{addObj}}(\cdot)$ is $(k * 64 + 256 + 64)$ and the output dimension is 256. For predicting representation of newly added edges in the scene graph, we use edge predictor $g_{\text{addEdge}}(\cdot)$. The input to this edge predictor function is the concatenated representation of the objects which are linked by the edge. The input dimension of $g_{\text{addEdge}}(\cdot)$ is $(256 + 256)$ and the output dimension is 256.

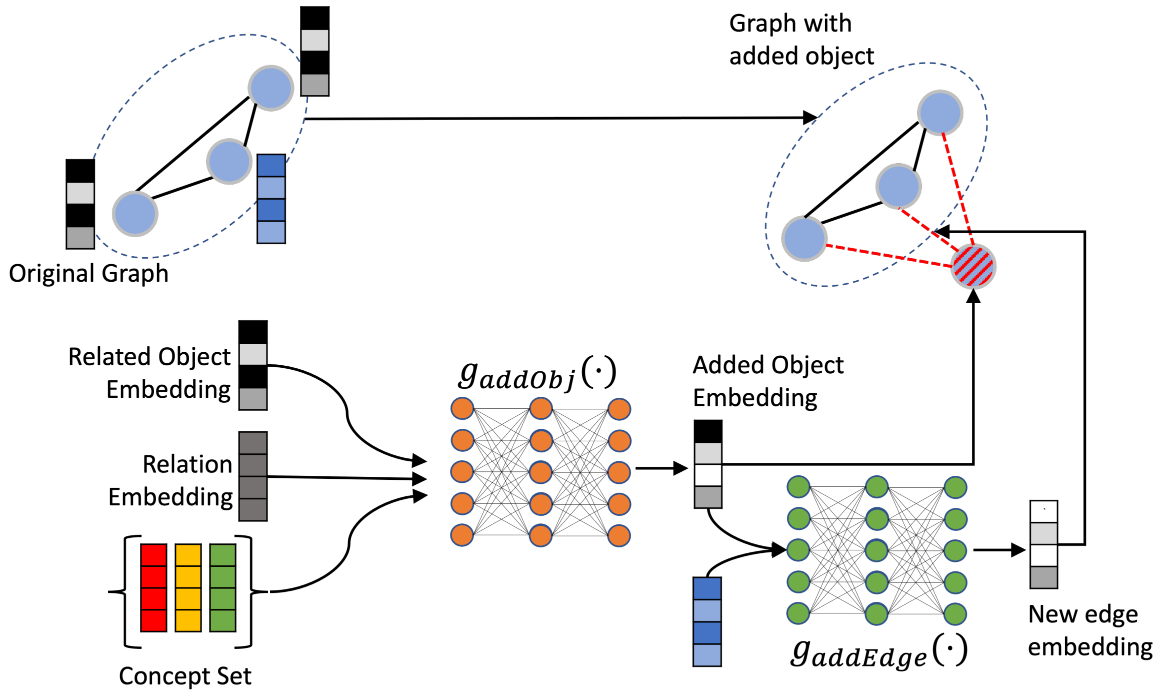
The loss used to train the *add network* weights is a weighted sum of losses equation 6 to equation 10 along with an object discriminator loss. The overall loss is given by the following expression.

$$L_{\text{overall\_add}} = \lambda_1 \ell_{\text{concepts}} + \lambda_2 \ell_{\text{relation}} + \lambda_3 \ell_{\text{objSup}} + \lambda_4 \ell_{\text{edgeSup}} + \lambda_5 \ell_{\text{edgeGAN}} + \lambda_6 \ell_{\text{objGAN}}$$

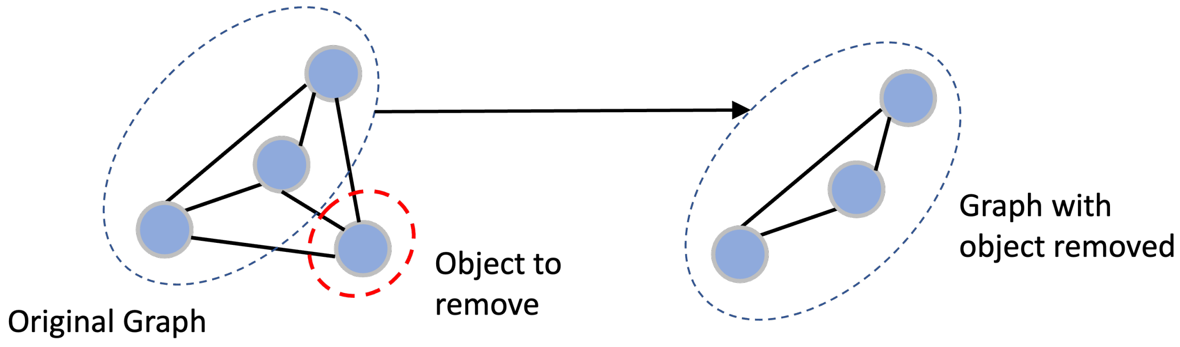
where, ℓ_{objGAN} and ℓ_{edgeGAN} above denotes the modified GAN loss (Goodfellow et al., 2014). Here



(a) Change operator overview.



(b) Add operator overview.



(c) Remove operator overview.

Figure 5: Overview of new operators (*change, add* and *remove*) added to the DSL.

$\lambda_1 = \lambda_2 = 1/(\text{num\_attrs})$, $\lambda_3 = \lambda_4 = 10^3$, $\lambda_6 = 1/(\text{num\_objects})$.

The object discriminator is a neural network with input dimension as 256 and a single 300 dimensional hidden layer with ReLU activation function. This discriminator is trained using the standard GAN objective ℓ_{objGAN}. Note, ℓ_{objGAN} has 2 parts – i) the loss for the generated (fake) object embedding using the *add network*, and ii) the loss for the real objects (all the unchanged object embeddings of the image). The former is unscaled but the latter one is scaled by a factor of $1/(\text{num\_objects})$.

The edge discriminator is a neural network with input dimension as $(256 * 3)$ and a single 300 dimensional hidden layer with ReLU activation function. As input to this discriminator network, we pass the concatenation of the two objects and the edge connecting them. This discriminator is trained using the standard GAN objective ℓ_{edgeGAN}. See Fig 5b for an overview of the add operator

D Additional Results

D.1 Detailed Performance for Zero-Shot Generalization on Larger Scenes

Table 11 below is a detailed version of the Table 3 in the main paper. This table compares the performance of NEUROSIM with baseline methods TIM-GAN, GeNeVA and IP2P for the zero-shot generalization to larger scenes (with ≥ 10 objects), while the models were trained on images with $3-8$ objects. Relative to the main paper's table 3, this table offers separate performance numbers for each of the *add, remove* and *change* instructions.

D.2 Image Retrieval Task

A task that is closely related to the image manipulation task is the task of *Text Guided Image Retrieval*, proposed by (Vo et al., 2019). Through this experiment, our is to demonstrate that NEUROSIM is highly effective in solving this task as well. In what follows, we provide details about this task, baselines, evaluation metric, how we adapted NEUROSIM for this task, and finally performance results in Table 12. This table is a detailed version of the Table 4 in the main paper.

Task Definition: Given an Image I, a text instruction T, and a database of images D, the task is to retrieve an image from the database that is semantically as close to the ground truth manipulated image as possible.

Note, for each such (I, T) pair, some image from the database, say $\widetilde{I} \in D$, is assumed to be the ideal image that should ideally be retrieved at rank-1. This, so called desired gold retrieval image might even be an image which is the ideal manipulated version of the original images I in terms of satisfying the instruction T perfectly. Or, image $\widetilde{I}$ may not be such an ideal manipulated image but it still may be the image in whole corpus D that comes closest to the ideal manipulated image.

In practice, while measuring the performance of any such system for this task, the gold manipulated image for (I, T) pair is typically inserted into the database D and such an image then serves as the desired gold retrieval image $\widetilde{I}$.

Baselines: Our baselines includes popular supervised learning systems designed for this task. The first baseline is TIRG proposed by Vo et al. (2019) where they combine image and text to get a joint embedding and train their model in a *supervised* manner using embedding of the desired retrieved image as supervision. For completeness, we also include comparison with other baselines – *Concat, Image-Only*, and *Text-Only* – that were introduced by Vo et al. (2019).

A recent model proposed by Chen et al. (2020) uses symbolic scene graphs (instead of embeddings) to retrieve images from the database. Motivated by this, we also retrieve images via the scene graph that is generated by the manipulation module of NEUROSIM. However, unlike Chen et al. (2020), the nodes and edges in our scene graph have associated vectors and make a novel use of them while retrieving. We do not compare our performance with (Chen et al., 2020) since its code is unavailable and we haven't been able to reproduce their numbers on datasets used in their paper. Moreover, (Chen et al., 2020) uses full supervision of the desired output image (which is converted to a symbolic scene graph), while we do not.

Evaluation Metric: We use Recall@k (and report results for $k = 1, 3$) for evaluating the performance of text guided image retrieval algorithms which is standard in the literature.

Retrieval using Scene Graphs: We use the scene graph generated by NEUROSIM as the latent representation to retrieve images from the database. We introduce a novel yet simple method to retrieve images using scene graph representation. For converting an image into the scene graph, we use the vi-

Method	Train Data Size	Add		Change		Remove	
		$R1$	$R3$	$R1$	$R3$	$R1$	$R3$
GeNeVA	$54K$	0.5	64.6	4.9	69.9	9.0	50.0
GeNeVA	$5.4K$	0.0	60.1	8.2	69.2	14.3	49.6
TIMGAN	$54K$	12.5	77.4	73.4	95.2	78.2	92.2
TIMGAN	$5.4K$	1.0	70.0	32.1	84.4	44.7	74.0
IP2P	$54K$	38.2	100.0	72.7	100.0	78.6	98.9
IP2P	$5.4K$	34.1	100.0	68.8	100.0	72.5	96.4
NeuroSIM	$5.4K$	3.8	46.6	68.2	95.8	90.7	94.3

Table 11: Detailed performance scores for NeuroSIM, TIM-GAN, GeNeVA and IP2P for zero-shot generalization to larger scenes (with ≥ 10 objects) from CIM-NLI-LARGE dataset, while models are trained on images with $3-8$ objects. Table has separate performance numbers for *add, remove,* and *change* instructions. Along with each method, we have also written the number of data points from CIM-NLI dataset that were used for training. $R1$ and $R3$ correspond to Recall@1 and Recall@3, respectively.

sual representation network of NeuroSIM. Given the scene graph G for the input image I and the manipulation instruction text T, NeuroSIM converts the scene graph into the changed scene graph $G_{\widetilde{I}}$, as described in Section C in Appendix. Now, we use this graph $G_{\widetilde{I}}$ as a query to retrieve images from the database D. For retrieval, we use the novel graph edit distance (GED) between $G_{\widetilde{I}}$ and the scene graph representation of the database images. The scene graph for each database image is also obtained using the visual representation network of NeuroSIM. The graph edit distance is given below.

$$GED(G_{\widetilde{I}}, G_D) = \begin{cases} \infty & |N_{\widetilde{I}}| \neq |N_{\widetilde{D}}| \\ \min_{\pi \in \Pi} \sum_{\forall i \in \{1, \cdots, |N_{\widetilde{I}}|\}} c(n_i, y_i) & \text{otherwise.} \end{cases}$$

where, $G_{\widetilde{I}} = (N_{\widetilde{I}}, V_{\widetilde{I}})$ and $G_D = (N_D, V_D)$. n_i and y_i are the node embeddings of the query graph $G_{\widetilde{I}}$ and scene graph G_D of an image from the database. $c(a, b)$ is the cosine similarities between embeddings a and b. This GED is much simpler than that defined in (Chen et al., 2020), since it does not need any hand designed cost for *change, removal,* or *addition* of nodes, or different attribute values. It can simply rely on the cosine similarities between node embeddings. We use the *Hungarian algorithm* (Kuhn, 1955) for calculating the optimal matching π of the nodes, among all possible matching Π. We use the negative of the cosine similarity scores between nodes to create the cost matrix for the Hungarian algorithm to process. This simple yet highly effective approach (See Table 4 in the main paper and Table 12 in the appendix), can be improved by more sophisticated techniques that include distance between edge embeddings and including notion of subgraphs in the GED. We leave this as future work. This result shows that our manipulation network edits the scene graph in a desirable manner, as per the input instruction.

D.3 Detailed Multi-hop Reasoning Performance

Table 14 below provides a detailed split of the performance numbers reported in Table 3 of the main paper across i) number of hops ($0-3$ hops) and ii) type of instructions *(add/remove/change)*. We observe that for *change* and *remove* instructions, NeuroSIM improves over TIM-GAN, GeNeVA and IP2P trained on $5.4K$ CIM-NLI data points by a significant margin ($\sim 20\%$ on 3-hop *change/remove* instructions). However, NeuroSIM lags behind TIM-GAN when the entire CIM-NLI labeled data is used to train TIM-GAN. We also observe that all the models perform poorly on the *add* instructions, as compared to *change* and *remove* instructions.

D.4 Detailed Performance for Different Cost Ratios β

Table 2 in Section 4 of the main paper showed the performance of NeuroSIM compared with TIM-GAN and GeNeVA for various values of β, where

Method	Train Data Size	Add			Change				Remove			
		1	2	3	0	1	2	3	0	1	2	3
Text-Only	$54K$	0.4	0.3	0.3	0.1	0.0	0.1	0.1	0.1	0.3	0.3	0.0
Image-Only	$54K$	35.3	33.4	32.3	20.1	23.2	16.9	19.8	46.3	41.3	53.1	57.8
Concat	$54K$	36.3	33.3	31.8	37.3	40.4	34.2	37.9	41.8	41.0	50.0	55.0
TIRG	$54K$	35.6	31.8	33.5	22.0	25.1	18.8	22.0	46.6	42.7	52.5	56.1
NEUROSIM	$5.4K$	96.2	95.3	95.3	83.3	82.9	81.3	78.7	79.6	77.4	86.4	82.2

Table 12: Performance scores (Recall@1) on the Image Retrieval task, comparing NEUROSIM with TIM-GAN and GeNeVA with increase in reasoning hops, for *add, remove,* and *change* instructions. Along with each method, number of data points from CIM-NLI used for training are written.

β is the ratio of the number of annotated (with output image supervision) image manipulation examples required by the supervised baselines, to the number of annotated VQA examples required to train NEUROSIM. In Table 13, we show a detailed split of the performance, for the *add, change,* and *remove* operators, across the same values of β as taken before.

We find that for the *change* operator, NEUROSIM performs better than TIM-GAN by a margin of $\sim 8\%$ (considering Recall@1) for $\beta \leq 0.1$. For the *remove* operator, NEUROSIM performs better than TIM-GAN by a margin of $\sim 4\%$ (considering Recall@1) for $\beta \leq 0.2$. Overall, NEUROSIM performs similar to TIM-GAN, for $\beta = 0.2$, for *remove* and *change* operators. All models perform poorly on the *add* operator as compared to the *change* and *remove* operators. We find that having full output image supervision allows TIM-GAN to reconstruct (copy) the unchanged objects from the input to the output for all the operators. This results in a higher recall in general but its effect is most pronounced in the Recall@3. NEUROSIM, on the other hand, suffers from rendering errors which makes the overall recall score (especially Recall@3) lower. We believe that improving image rendering quality would significantly improve the performance of NEUROSIM and we leave this as future work.

D.5 Results on Datasets from different domains

D.5.1 Minecraft Dataset

Dataset Creation: We create a new dataset having (Image, instruction) by building over the Minecraft dataset used in (Yi et al., 2018). Specifically, we create zero and one hop remove instructions and one hop add instructions similar to the creation of CIM-NLI. This dataset contains scenes and objects from the Minecraft video game and is used in prior works for testing Neuro-Symbolic VQA systems like NSCL (Mao et al., 2019) and NS-VQA (Yi et al., 2018). The setting of the Minecraft worlds dataset is significantly different from CLEVR in terms of concepts and attributes of objects and visual appearance.

Experiment: We use the above dataset for testing the addition and removal of objects using NeuroSIM (See Fig 6). We train NeuroSIM's decoder to generate images from scene graphs of the minecraft dataset. We assume access to a parser that gives us programs for an instruction. For removal, we use the same remove network as described above, while for addition, we assume access to the features of object to be added, which is added to the scene graph of the image and the decoder decodes the final image. See Figure 6 for a set of successful examples on the Minecraft dataset. We see that using our method, one can add and remove objects from the scene successfully, without using any output image as supervision during training. Though we have assumed the availability of a parser in the above set-up, training it jointly with other modules should be straightforward, and can be achieved using our general approach described in Section 3 of the main paper.

E End-to-end Training

The main objective of this work is to make use of weakly supervised VQA data for the image manipu-

Method	Instruction	$\beta = 0.054$		$\beta = 0.07$		$\beta = 0.1$		$\beta = 0.2$		$\beta = 0.54$	
		$R1$	$R3$	$R1$	$R3$	$R1$	$R3$	$R1$	$R3$	$R1$	$R3$
GeNeVA	add	0.0	57.3	–	–	–	–	–	–	0.7	63.6
	change	5.9	36.3	–	–	–	–	–	–	4.1	39.4
	remove	13.2	82.3	–	–	–	–	–	–	8.7	89.3
TIM-GAN	add	1.9	70.7	4.9	74.0	8.6	76.7	10.3	77.1	13.1	78.6
	change	41.0	72.1	42.9	73.5	49.8	77.3	62.5	84.2	78.3	92.3
	remove	49.6	79.5	47.0	91.9	53.9	93.1	65.3	96.8	78.0	98.5
IP2P	add	0.7	70.5	–	–	–	–	–	–	5.4	78.6
	change	54.9	72.8	–	–	–	–	–	–	61.5	78.9
	remove	62.0	87.2	–	–	–	–	–	–	76.0	96.3
NeuroSIM	add	4.9	30.9	6.4	34.8	5.7	34.7	5.9	38.9	5.6	35.0
	change	57.2	79.4	57.3	79.3	57.2	79.3	57.2	79.4	57.1	79.3
	remove	69.6	82.5	69.5	82.5	69.5	82.6	69.5	82.5	69.6	82.5

Table 13: Detailed performance comparison of NeuroSIM with TIM-GAN (Zhang et al., 2021), GeNeVA (El-Nouby et al., 2019) and IP2P (Brooks et al., 2023) with varying β levels, split across add, remove and change instructions. The '-' entries for GeNeVA and IP2P were not computed due to excessive training time (inference time as well in case of IP2P); Geneva's performance is abysmal even when using full data. TIM-GAN does the best among baselines in terms of its recall score at $\beta = 0.54$. We always use $100K$ VQA examples (5K Images, 20 questions per image) for our weakly supervised training. $R1$ and $R3$ correspond to Recall@1 and 3, respectively. For Recall, higher score is better.

lation task without using output image supervision. But a natural extension of our work is to use output image supervision as well, to improve the performance of NeuroSIM. We devised an experiment to compare how much performance boost can be obtained by utilizing ground truth output (manipulated) images as the supervision for different modules of NeuroSIM. This experiment demonstrates the value of end-to-end training for NeuroSIM and how it can exploit the supervised data. We refer to this variant as NeuroSIM(e2e). We begin with a pre-trained NeuroSIM model trained with VQA annotations and then fine-tune it using supervised manipulation data. The detailed results are given in Table 15. This experiment demonstrates that with a small amount of supervised data, the performance of NeuroSIM can be significantly improved (e.g., more than 9 points increase for the change instruction with only $5.4K$ supervision examples)

Given the significant increase in performance of NeuroSIM when using supervised data, we also test it's generalization capability (Analogous to Section 4.2, 4.3), and quality of scene graph retrieval (Analogous to Section 4.5).

From Table 16, we see that NeuroSIM(e2e) shows improved zero-shot generalization to larger scenes. Even when trained on just 5.4k CIM-NLI data, NeuroSIM(e2e) improves over TIM-GAN-54k by 3.9 R@1 points. A 5.3 point improvement over TIM-GAN is observed when full CIM-NLI data is used.

Next, we measure drop in performance with increasing reasoning hops. From Table 17, we see that NeuroSIM(e2e) achieves the lowest drop when compared to TIM-GAN. NeuroSIM(e2e) improves over weakly supervised NeuroSIM baseline by 6.6 R@1 points.

Finally, we measure the quality of scene graphs via retrieval. From Table 18, we see that super-

Method	Train Data Size	Add			Change				Remove			
		1	2	3	0	1	2	3	0	1	2	3
GeNeVA	$54K$	1.1	0.5	0.5	3.6	4.2	4.7	3.9	9.0	8.0	8.3	9.4
GeNeVA	$5.4K$	0.0	0.0	0.0	4.7	7.1	5.6	6.1	12.3	11.3	15.5	13.5
TIM-GAN	$54K$	7.6	16.1	15.7	85.8	74.1	78.0	75.4	82.2	68.3	81.9	79.7
TIM-GAN	$5.4K$	1.4	2.3	1.9	54.5	36.4	38.7	34.5	58.3	40.9	50.9	48.2
IP2P	$54K$	6.6	5.5	4.2	67.9	60.6	59.1	58.5	77.1	71.9	78.1	76.8
IP2P	$5.4K$	0.5	0.7	0.9	64.0	54.8	50.6	50.3	74.5	55.3	60.3	58.0
NEUROSIM	$5.4K$	4.6	5.0	5.1	59.5	57.9	55.8	55.7	69.6	66.6	71.8	70.4

Table 14: Performance scores (Recall@1) for NEUROSIM with TIM-GAN, GeNeVA and IP2P with increase in reasoning hops, for *add, remove,* and *change* instructions. Along with each method, number of data points from CIM-NLI used for training are written.

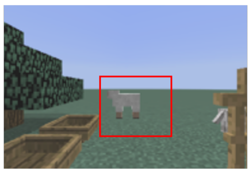
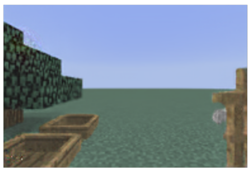
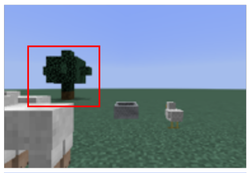
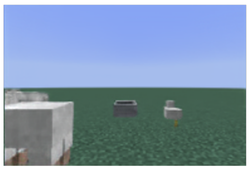
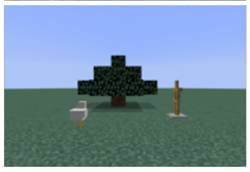
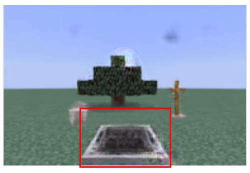
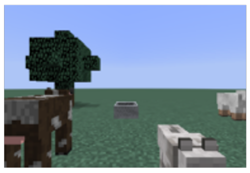
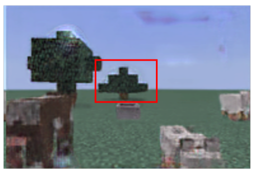
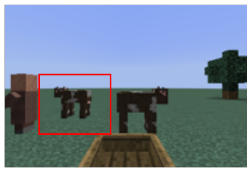
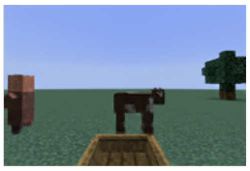
Input Image	**Instruction**	**Program**	**Output**
	Remove the sheep	scene() -> filter_class(['sheep']) –> remove()	
	Remove the tree to the left of chicken	scene() -> filter_class(['chicken']) –> relate(['left']) –> filter_class(['tree']) –> remove()	
	Add a minecart to the front of tree	scene() -> filter_class(['tree']) –> add(rel_concept=['frontOf'], class = ['minecart'])	
	Add a tree to the back of cow	scene() -> filter_class(['cow']) –> add(rel_concept=['backOf'], class = ['tree'])	
	Remove the cow to the left of tree	scene() -> filter_class(['tree']) –> relate(['left']) –> filter_class(['cow']) –> remove()	

Figure 6: Results for addition and removal of objects from images of the minecraft dataset

vised training significantly improves the scene graph quality, thus improving retrieval performance. Supervised training improves retrieval by 7.3 R@1 points over weakly supervised NEUROSIM baseline. These findings suggest that NEUROSIM(e2e) significantly outperforms other supervised approaches in almost all settings. One can fine-tune the image decoder and the visual representation network to further enhance the findings, which should greatly enhance the outcomes.

Instruction	Model	# of CIM-NLI examples used for training				
		5.4K	7K	10K	20K	54K
Add	GeNeVA	0.0	-	-	-	0.7
	TIM-GAN	1.9	4.9	8.6	10.3	13.1
	IP2P	0.7	-	-	-	5.4
	NEUROSIM	4.9	6.4	5.7	5.9	5.6
	NEUROSIM(e2e)	8.8	8.9	9.2	10.5	10.6
Change	GeNeVA	5.9	-	-	-	4.1
	TIM-GAN	41.0	42.9	49.8	62.5	78.3
	IP2P	54.9	-	-	-	61.5
	NEUROSIM	57.2	57.3	57.2	57.2	57.1
	NEUROSIM(e2e)	66.2	66.3	66.6	67.4	69.6
Remove	GeNeVA	13.2	-	-	-	8.7
	TIM-GAN	49.6	47.0	53.9	65.3	78.0
	IP2P	62.0	-	-	-	76.0
	NEUROSIM	69.6	69.5	69.5	69.5	69.6
	NEUROSIM(e2e)	69.6	69.5	69.5	69.5	69.6

Table 15: Performance comparison of NEUROSIM(e2e) with baselines using Recall@1. NEUROSIM(e2e) refers to NEUROSIM trained end-to-end by utilizing ground truth manipulated images as the supervision for NEUROSIM modules.

Model	Train Data Size	R1	R3
TIM-GAN	5.4K	30.2	80.7
TIM-GAN	54K	66.3	92.4
NEUROSIM	5.4K	63.7	89.1
NEUROSIM(e2e)	5.4K	70.2	92.6
NEUROSIM(e2e)	54K	71.6	91.7

Table 16: Zero-shot generalization to larger scenes (Extension of Table 3 of main paper).

F LLMs as few-shot parser

We also tested the semantic parsing ability of Large Language Models (LLMs), specifically GPT-4 for our task. The task of semantic parsing is given manipulation instruction text in natural language, generated the symbolic program by parsing the input text. To provide GPT-4 with context, we designed an extensive prompt that begins with our DSL followed by six different in-context examples representing various instruction types for few-shot learning. This prompt is then followed with the instruction text that we want to parse. We tested GPT-4 on a randomly sampled subset of our test dataset. For evaluation, we measured the accuracy of semantic parsing using an exact match between the generated symbolic program and the ground-truth symbolic program.

The detailed results are given in Table 19. Interestingly, we observed that GPT-4 performed poorly on Add instructions, achieving less than 10% of parsing accuracy. To address this, we prompted GPT-4 separately with additional few-shot examples for Add instructions, which led to the results displayed in the table. Even with the additional

Method	Train Data Size	Hops		
		ZH	MH	$\triangle$
TIM-GAN	5.4K	56.4	41.6	-14.8
TIM-GAN	54K	84.0	76.2	-7.8
NEUROSIM	5.4K	64.5	63.0	-1.5
NEUROSIM(e2e)	5.4K	69.4	67.3	-2.1
NEUROSIM(e2e)	54K	71.1	69.6	-1.5

Table 17: Performance with increasing reasoning hops (Extension of Table 3 of main paper).

Model	R1	R3
Text-Only	0.2	0.4
Image-Only	34.1	83.6
Concat	39.5	86.9
TIRG	34.8	84.6
NEUROSIM	85.8	92.9
NEUROSIM(e2e)	93.1	96.7

Table 18: Quality of scene graph measured via retrieval (Extension of Table 4 of main paper)

Instruction type	Hops				Total
	0	1	2	3	
Add	NA	18.9	30.5	37.5	29.5
Remove	91.5	85.1	80.0	84.9	85.6
Change	70.7	81.7	76.2	70.2	74.6
Total	81.5	55.6	55.9	58.1	60.5

Table 19: Few-shot parsing results of GPT-4

guidance, Add instructions remained significantly lower in accuracy compared to other instruction types. This analysis demonstrates that our reinforcement learning-based instruction parser outperforms GPT-4, at least on this dataset. It also highlights the need for more careful prompt engineering before LLMs like GPT-4 can be readily applied in our specific setting.

G Computational Resources

We trained all our models and baselines on 1 Nvidia Volta V100 GPU with 32GB memory and 512GB system RAM except IP2P which was trained on 8-A100 80 GB GPUs. Our image decoder training takes about 4 days of training time. Training of the VQA task takes $5-7$ days of training time and training the Manipulation networks take $4-5$ hours of training time.

H Hyperparameters and Validation Accuracies

H.1 Training for VQA Task

The hyperparameters for the VQA task are kept same as default values coming from the prior work (Mao et al., 2019). We refer the readers to (Mao et al., 2019) for more details. We obtained a question answering accuracy of 99.3% after training on the VQA task.

H.2 Training Semantic Parser

The semantic parser is trained to parse instructions. Learning of this module happens using the `REINFORCE` algorithm as described in Section C of this appendix. During `REINFORCE` algorithm, we search for positive rewards from the set $\{7, 8, 10\}$, and negative rewards from the set $\{0, 2, 3\}$. We finally choose a positive reward of 8 and negative reward of 2. For making this decision, we first train the semantic parser for 20 epochs and then calculate its accuracy by running it on the quantized scenes from the validation set. For a particular output program, we say it is correct if it leads to an object being selected (see Section C of the appendix for more information) and this is how the accuracy of the semantic parser is calculated. This accuracy is a proxy for the real accuracy. An alternative is to use annotated ground truth programs for calculating accuracy and then selecting hyperparameters. However, we do not use ground truth programs. All other hyperparameters are kept the same as used by (Mao et al., 2019) to train the parser on VQA task. We obtain a validation accuracy of 95.64% after training the semantic parser for manipulation instructions.

H.3 Training Manipulation Networks

The architecture details of the manipulation network are present in Section C of this appendix. We use batch size of 32, learning rate of 10^{-3}, and optimize using AdamW (Loshchilov and Hutter, 2019) with weight decay of 10^{-4}. Rest of the hyperparameters are kept the same as used in (Mao et al., 2019). During training, at every 5^{th} epochs, we calculate the manipulation accuracy by using the query networks that were trained while training the NEUROSIM on VQA data. This serves as a proxy to the validation accuracy.

- For the *change* network training, we use the query accuracy of whether the attribute that was supposed to change for a particular object, has changed correctly or not. Also, whether any other attribute has changed or not.
- For the *add* network training, we use the query accuracy of whether the attributes of the added object are correct or not. Also, whether the added object is in a correct relation with reference object or not.

We obtained a validation accuracy (based on querying) of 95.9% for the *add* network and an accuracy of 99.1% for the *change* network.

H.4 Image Decoder Training

The architecture of the image decoder is similar to (Johnson et al., 2018) but our input scene graph (having embeddings for nodes and edges) is directly processed by the graph neural network. We use a batch size of 16, learning rate of 10^{-5}, and optimize using Adam (Kingma and Ba, 2015) optimizer. The rest of the hyperparameters are same as (Johnson et al., 2018). We train the image decoder for a fixed set of $1000K$ iterations.

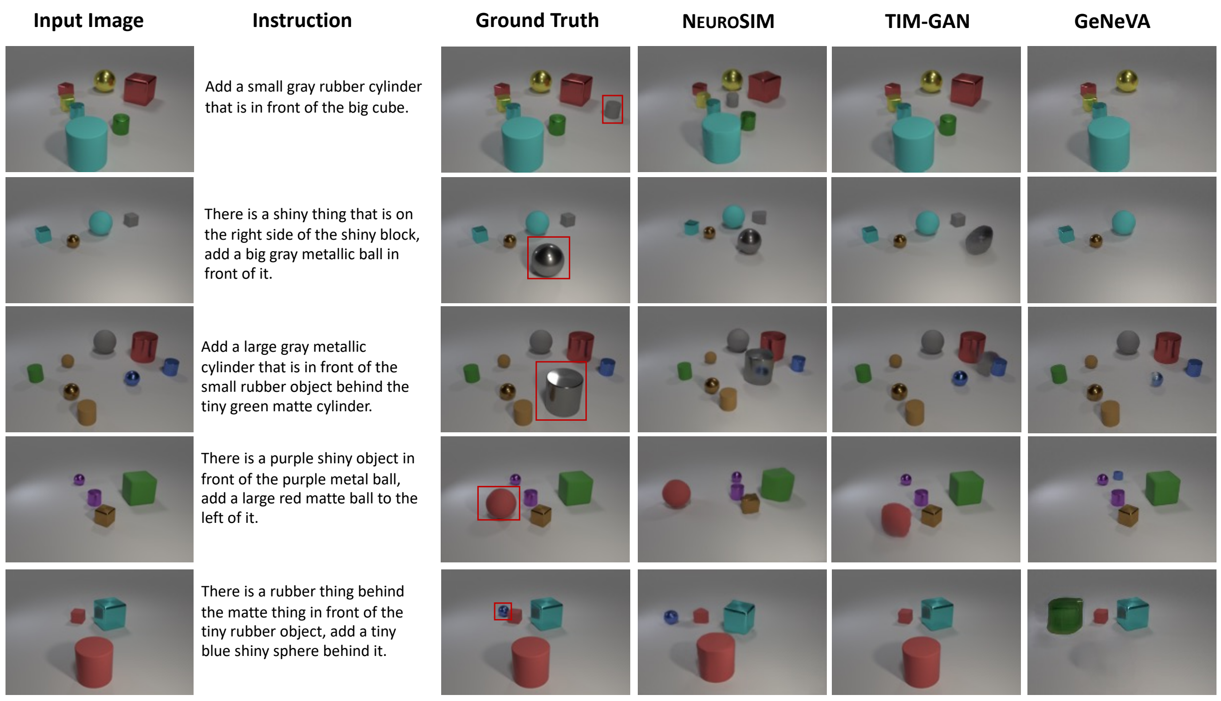
I Qualitative Analysis

Figures 7, 8, 9 compare the images generated by NEUROSIM, TIM-GAN, and GeNeVA on add, change and remove instructions respectively. NEUROSIM's advantage lies in semantic correctness of manipulated images. For example, see Figure 7 row #3,4; Figure 8 row #2; 9 all images. In these images, NEUROSIM was able to achieve semantically correct changes, while TIM-GAN, GeNeVA faced problems like blurry, smudged objects while adding them to the scene, removing incorrect objects from the scene, or not changing/partially changing the object to be changed. Images generated by TIM-GAN are better in quality as compared to NEUROSIM. We believe the reason for this is that TIM-GAN, being fully supervised, only changes a small portion of the image and has learned to copy a significant portion of the input image directly to the output. However, this doesn't ensure the semantic correctness of TIM-GAN's manipulation, as described above with examples where it makes errors. The images generated by NEUROSIM look slightly worse since the entire image is generated from object based embeddings in the scene graph. Improving neural image rendering from scene graphs can be a promising step to improve NEUROSIM.

J Errors

Figure 10 captures the images generated by our model where it has made errors. The kind of errors that NEUROSIM makes can be broadly classified into three categories.

- **[Rendering Errors]** This set includes images generated by our model which are semantically correct but suffer from rendering errors. The common rendering errors include malformed cubes, partial cubes, change in position of objects, and different lighting.
- **[Logical Errors]** This set includes images generated by our model which have logical errors. That is, manipulation instruction has been interpreted incorrectly and a different manipulation has been performed. This happens mainly due to an incorrect parse of the input instruction into the program, or manipulation network not trained to perfection. For example, *change* network changing attributes which were supposed to remain unchanged.
- **[VQA Errors]** The query networks are not ideal and have errors after they are trained on the VQA task. This in turn causes errors in supervision (obtained from query networks) while training the



Add a small gray rubber cylinder that is in front of the big cube.

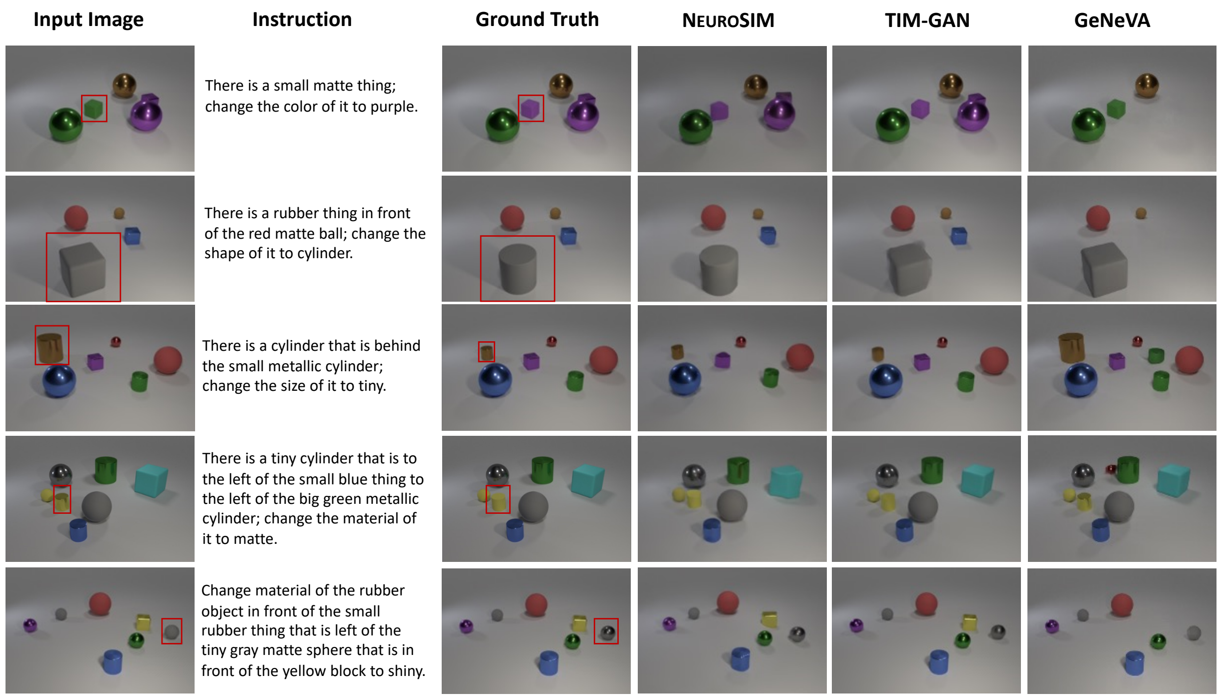
There is a shiny thing that is on the right side of the shiny block, add a big gray metallic ball in front of it.

Add a large gray metallic cylinder that is in front of the small rubber object behind the tiny green matte cylinder.

There is a purple shiny object in front of the purple metal ball, add a large red matte ball to the left of it.

There is a rubber thing behind the matte thing in front of the tiny rubber object, add a tiny blue shiny sphere behind it.

Figure 7: Visual comparison of NeuroSIM with TIM-GAN and GeNeVA for the *add* operator. The red bounding boxes in the ground truth output image indicate the objects required to add to the input image.



There is a small matte thing; change the color of it to purple.

There is a rubber thing in front of the red matte ball; change the shape of it to cylinder.

There is a cylinder that is behind the small metallic cylinder; change the size of it to tiny.

There is a tiny cylinder that is to the left of the small blue thing to the left of the big green metallic cylinder; change the material of it to matte.

Change material of the rubber object in front of the small rubber thing that is left of the tiny gray matte sphere that is in front of the yellow block to shiny.

Figure 8: Visual comparison of NeuroSIM with TIM-GAN and GeNeVA for the *change* operator. The red bounding boxes in the input and ground truth output image indicate the objects required to be changed.

manipulation networks and leads to a less than optimally trained manipulation network. Also, during inference, object embeddings may not be perfect due to the imperfections in the visual representation network and that leads to incorrect rendering.

K Ablations

Table 20 shows the performance of NeuroSIM when certain loss terms are removed while learning of the networks. This depicts the importance of loss terms that we have considered. In particular we test the performance of the network by removing

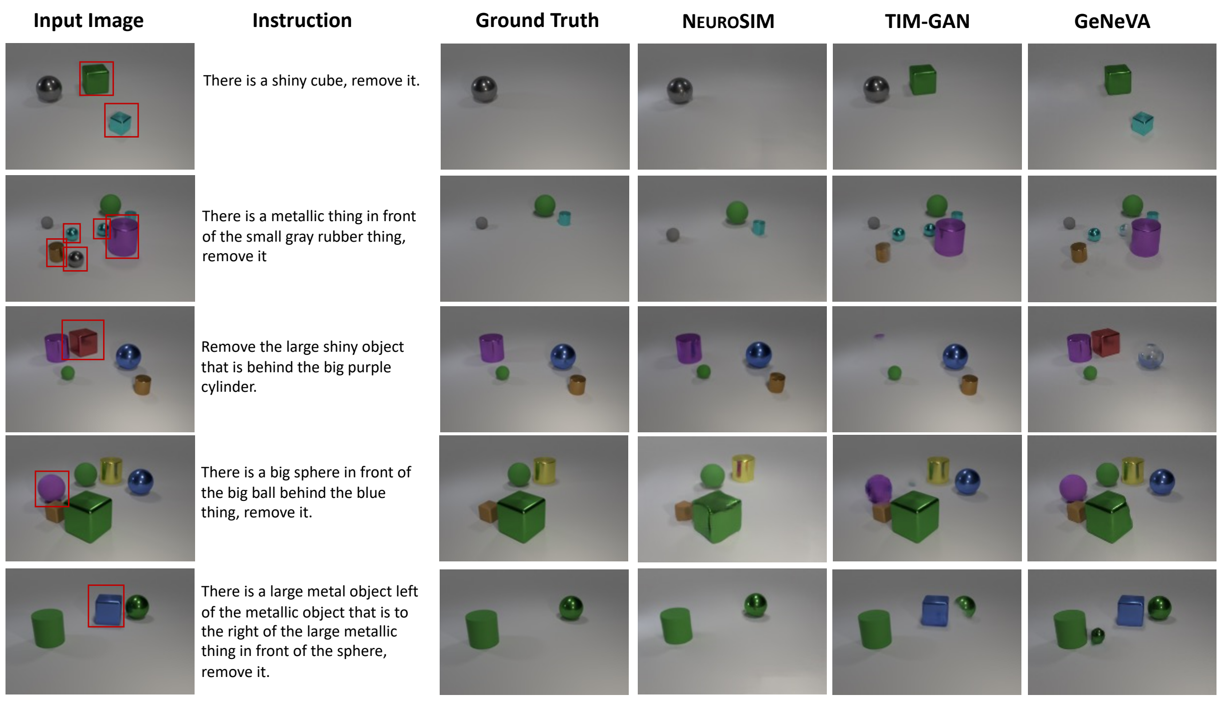

Input Image
Instruction
Ground Truth
NEUROSIM
TIM-GAN
GeNeVA
There is a shiny cube, remove it.
There is a metallic thing in front of the small gray rubber thing, remove it
Remove the large shiny object that is behind the big purple cylinder.
There is a big sphere in front of the big ball behind the blue thing, remove it.
There is a large metal object left of the metallic object that is to the right of the large metallic thing in front of the sphere, remove it.

Figure 9: Visual comparison of NEUROSIM with TIM-GAN and GeNeVA for the *remove* operator. The red bounding boxes in the input image indicate objects required to be removed.

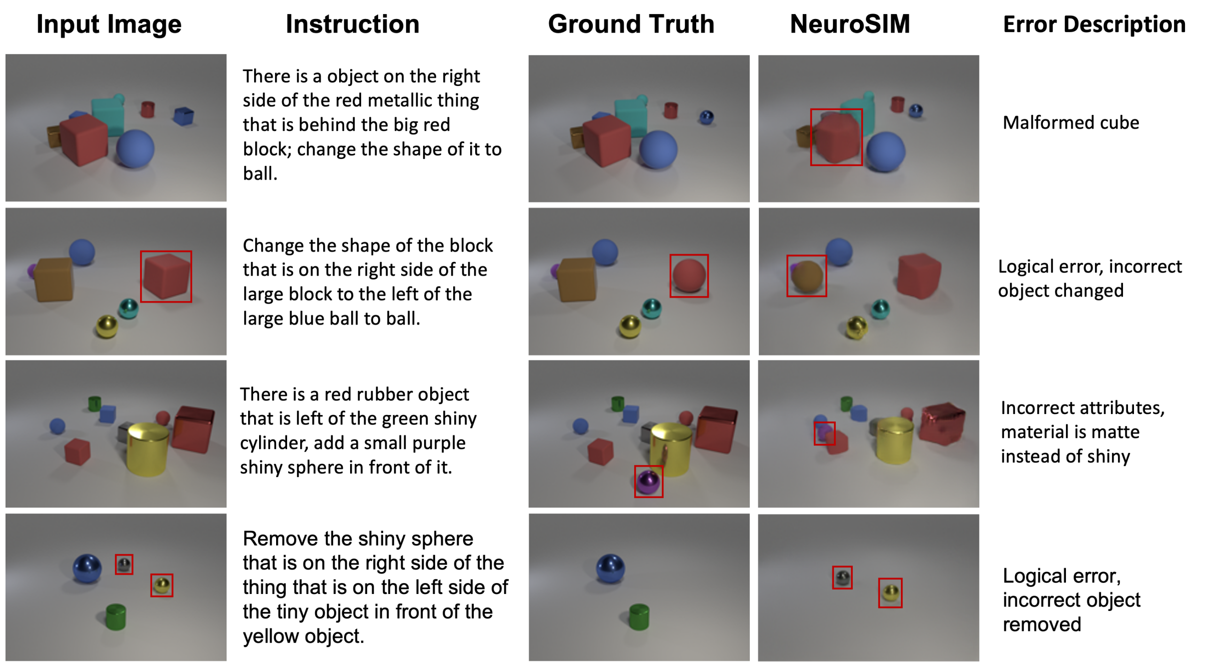

Input Image
Instruction
Ground Truth
NeuroSIM
Error Description
There is a object on the right side of the red metallic thing that is behind the big red block; change the shape of it to ball.
Malformed cube
Change the shape of the block that is on the right side of the large block to the left of the large blue ball to ball.
Logical error, incorrect object changed
There is a red rubber object that is left of the green shiny cylinder, add a small purple shiny sphere in front of it.
Incorrect attributes, material is matte instead of shiny
Remove the shiny sphere that is on the right side of the thing that is on the left side of the tiny object in front of the yellow object.
Logical error, incorrect object removed

Figure 10: Types of errors in NEUROSIM.

edge adversarial loss used by add network (row 2), object adversarial losses for both add and change networks (row 3, 5), self supervision losses used by add network (row 4), cyclic (row 6) and consistency (row 7) losses used by change network.

L Interpretability of NEUROSIM

NEUROSIM allows for interpretable image manipulation through programs which are generated as an intermediate representation of the input instruction. This is one of the major strengths of NEUROSIM, since it allows humans to detect where NEUROSIM failed. This is not possible with purely neural models, that behave as a black box. Knowing about the

Loss	R1	R3
ℓ	45.3	65.5
$\ell - \ell^{add}_{\text{edgeGAN}}$	43.7	66.0
$\ell - \ell^{add}_{\text{objGAN}}$	44.3	60.2
$\ell - \ell^{add}_{\text{objSup}} - \ell^{add}_{\text{edgeSup}}$	44.1	57.9
$\ell - \ell^{change}_{\text{objGAN}}$	44.9	61.5
$\ell - \ell^{change}_{\text{cycle}}$	36.5	51.1
$\ell - \ell^{change}_{\text{consistency}}$	31.0	44.8

Table 20: Ablations conducted by removing some loss terms. ℓ is the total loss before any ablation. For each loss term being removed, the superscript denotes which network it belongs to (add or change). Ablations are conducted for the setting where $\beta = 0.054$ (see main paper Section 4 for the definition of β)

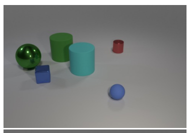
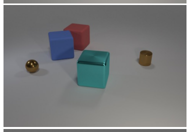
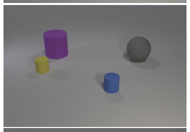
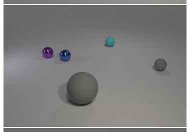
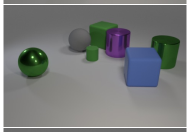
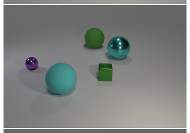
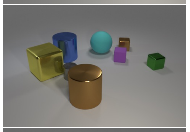
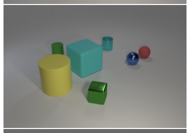
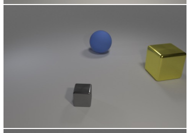
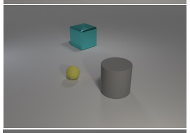
failure cases of NEUROSIM also means that it can be selectively trained to improve certain parts of the network (for eg individually training on change instructions to improve the change command, if the model is performing poorly on change instructions). We now assess the correctness of intermediate programs using randomly selected qualitative examples present in Figure 11. Since no wrong program was obtained in the randomly selected set, we find 2 more data points manually, to show some wrong examples.

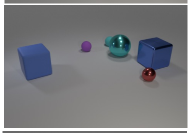
M Human Evaluation Details

See Table 21 for the questions (paraphrased) asked to the evaluators. Detailed instructions and an example of the questions provided to the evaluators can be found in Figure 12. A total of 10 evaluators, consisting of a mix of undergraduate and post-graduate students, were involved in the study. The same set of 30 random images were given to each evaluator. They were compensated at a rate three times the average hourly salary in the country of origin. Each evaluator was given upto 24 hours to complete the task.

N Simplifying Multi-Hop Instructions using NeuroSIM Modules

In this section, we provide details on our method of utilizing the trained semantic parser to convert the complex multi-hop instruction into a simplified 0 or 1 hop instruction. We generate three simplified templates one for each edit operation.
1. *Change the [attribute] of [size] [color] [material] [shape] to [attribute']*
2. *Remove the [size] [color] [material] [shape]*
3. *Add a [size] [color] [material] [shape] to the [relation] of [shape'].* Next, given a multi-hop instruction we parse it using our semantic parser which gives us the object's embedding on which either an operation is to be executed (in case of change and remove operations) or a new object has to be inserted in relation to it (in case of add operation). The trained query-networks predicts the symbolic values of the concepts in the placeholders. Example, if the MH instruction is "*Change the size of the big thing that is behind the metallic cylinder behind the purple object that is to the right of the big brown shiny object to tiny*" , we find the placeholder attributes to be operation=change, attribute=size, color=yellow, shape=cube, size=large, material=rubber, attribute'=tiny. Hence the simplified instruction becomes, "*Change the size of the large yellow rubber cube to tiny*". Add and Remove instructions follow similarly.

Input Image	Instruction	Generated Program	Correctness
	Add a tiny purple metal ball that is in front of the blue object that is behind the matte ball.	scene() -> filter(['rubber', 'sphere']) -> relate(['behind']) -> filter(['blue']) -> add(rel_concept=['front'], concept_set=['small', 'purple', 'metal', 'sphere'])	✓
	Remove the brown metal object that is left of the blue matte block that is left of the brown thing on the right side of the large cyan metal cube.	scene() -> filter(['large', 'cyan', 'metal', 'cube']) -> relate(['right']) -> filter(['brown']) -> relate(['left']) -> filter(['blue', 'rubber', 'cube']) -> relate(['left']) -> filter(['brown', 'metal']) -> remove()	✓
	Change the shape of the big gray thing to cube.	scene() -> filter(['large', 'gray']) -> change(attr=shape, concept=['cube'])	✓
	Remove the gray rubber thing in front of the gray matte sphere behind the large gray matte sphere.	scene() -> filter(['large', 'gray', 'rubber', 'sphere']) -> relate(['behind']) -> filter(['gray', 'rubber', 'sphere']) -> relate(['front']) -> filter(['gray', 'rubber']) -> remove()	✓
	There is a matte block that is in front of the big gray rubber object; change the material of it to shiny.	scene() -> filter(['large', 'gray', 'rubber']) -> relate(['front']) -> filter(['rubber', 'cube']) -> change(attr=material, concept=['metal'])	✓
	Remove the big ball that is to the right of the large thing in front of the purple metallic object	scene() -> filter(['purple', 'metal']) -> relate(['front']) -> filter(['large']) -> relate(['right']) -> filter(['large', 'sphere']) -> remove()	✓
	Change the material of the small metallic block in front of the brown metal block to rubber.	scene() -> filter(['brown', 'metal', 'cube']) -> relate(['front']) -> filter(['small', 'metal', 'cube']) -> change(attr=material, concept=['rubber'])	✓
	Remove the cylinder that is to the right of the matte cylinder that is in front of the small red matte thing.	scene() -> filter(['small', 'red', 'rubber']) -> relate(['front']) -> filter(['rubber', 'cylinder']) -> relate(['right']) -> filter(['cylinder']) -> remove()	✓
	Add a large purple shiny sphere behind the shiny cube behind the tiny gray object.	scene() -> filter(['small', 'gray']) -> relate(['behind']) -> filter(['metal', 'cube']) -> add(rel_concept=['behind'], concept_set=['large', 'purple', 'metal', 'sphere'])	✓
	Change the shape of the large object in front of the tiny yellow sphere to ball.	scene() -> filter(['small', 'yellow', 'sphere']) -> relate(['front']) -> filter(['large']) -> change(attr=shape, concept=['sphere'])	✓

	There is a blue thing behind the big blue block left of the thing that is in front of the blue matte block; change the color of it to brown.	scene() -> filter(['blue', 'rubber', 'cube']) -> relate(['front']) -> filter(['large', 'blue', 'cube']) -> relate(['left']) -> filter(['blue']) -> relate(['behind']) -> change(attr=color, concept=['brown'])
		Corrected program:
		scene() -> filter(['blue', 'rubber', 'cube']) -> relate(['front']) -> relate(['left']) -> filter(['large', 'blue', 'cube']) -> relate(['behind']) -> filter(['blue']) -> change(attr=color, concept=['brown'])
			✗
			✓

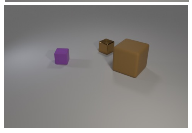
	Remove the brown object on the left side of the object right of the brown object that is behind the big brown block.	scene() -> filter(['large', 'brown', 'cube']) -> relate(['behind']) -> filter(['brown']) -> relate(['right']) -> filter(['brown']) -> relate(['left']) -> remove()
		Corrected program:
		scene() -> filter(['large', 'brown', 'cube']) -> relate(['behind']) -> filter(['brown']) -> relate(['right']) -> relate(['left']) -> filter(['brown']) -> remove()
			✗
			✓

Figure 11: Qualitative examples of generated programs by NeuroSIM.

Question 1:	**[Change]** Are all the attributes (color, shape, size, material, and relative position) of the changed object mentioned in the instructions identical between the ground truth image and the system-generated image? **[Add]** Are all the attributes (color, shape, size, material, and relative position) of the added object mentioned in the instructions identical between the ground truth image and the system-generated image? **[Remove]** Are same objects removed in ground truth image and the system-generated image?
Question 2:	**[Change]** Are all the attributes (color, shape, size, material, and relative position) of the remaining objects identical between the ground truth image and the system-generated image? **[Add]** Are all the attributes (color, shape, size, material, and relative position) of the remaining objects identical between the ground truth image and the system-generated image? **[Remove]** Are all the attributes (color, shape, size, material, and relative position) of the remaining objects identical between the ground truth image and the system-generated image?

Table 21: Questions asked to human evaluators for evaluating NEUROSIM and TIM-GAN. Note that there are some variations in the questions for Change, Add, and Remove instructions dues to different semantic nature of the instructions.

Human Evaluation

There are 30 questions. In each question, there are 5 images, an "Input" image, a "Reference Output" image (or ground truth image), and 3 randomly shuffled "Candidate Output" images generated by machine learning models.

Each object in an image has 4 properties:

- **Shape: cylinder, sphere, or cube**
- **Size: small or big**
- **Material: shiny "metallic" or matte "rubber"**
- **Color: green, red, purple, cyan, gray, blue, brown, or yellow**

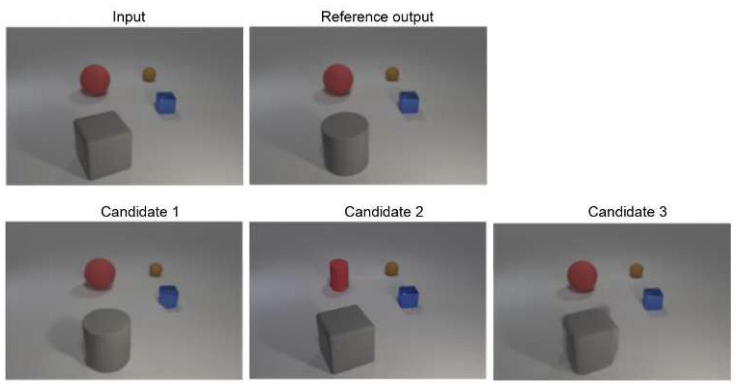
Between each input and ground truth image, one or more objects are different in terms of one of its properties (or whether it exists in the input or ground truth images or not). We call this the "changed object". To understand the "changed object" better please see the following examples:

There are 3 types of changes:

- **Change in property of one of the objects**
- **Removal of one or more objects**
- **Addition of one object**

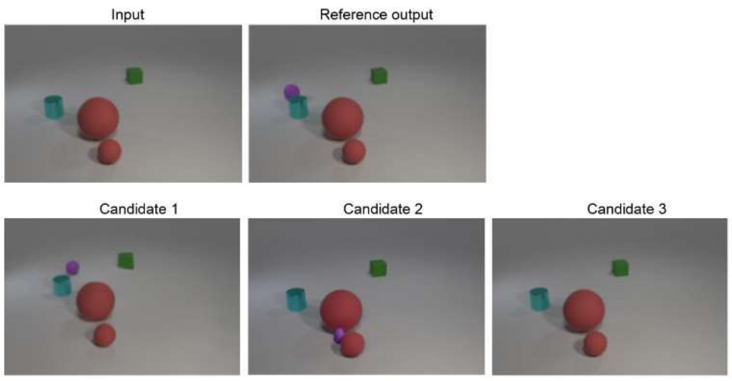
Example 1

Here the "large gray cube" in the Input Image is the "changed object", since it's shape has changed to "cylinder" in the Reference Output Image



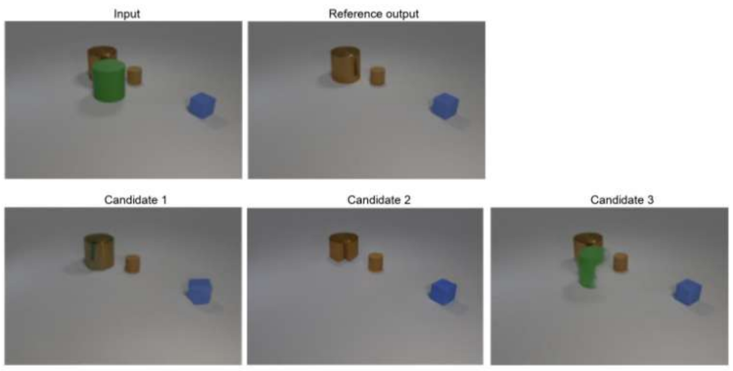
Example 2

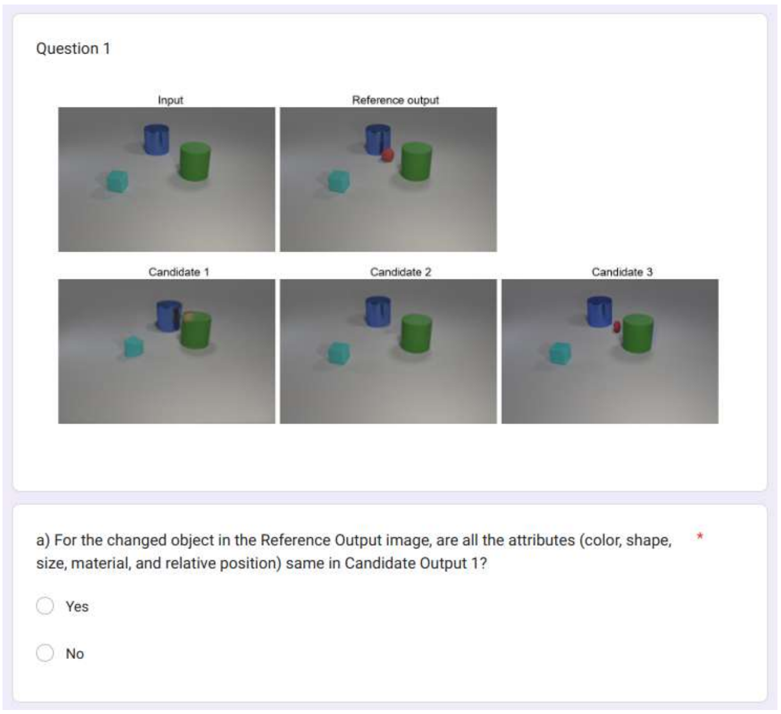
Here a “small pink sphere” in the Reference output image is the "changed object", since it doesn't exist in the Input image and has been added in the Reference Output Image.



Example 3

Here a “large green cylinder” in the Reference output image is the "changed object", since it exist in the Input image and but not in the Reference Output Image.



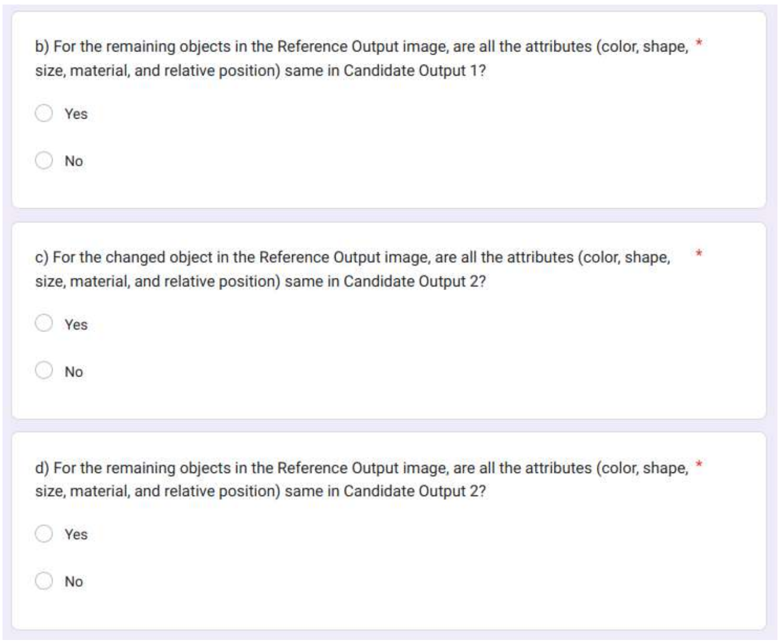


Question 1

a) For the changed object in the Reference Output image, are all the attributes (color, shape, size, material, and relative position) same in Candidate Output 1? *

Yes

No



b) For the remaining objects in the Reference Output image, are all the attributes (color, shape, size, material, and relative position) same in Candidate Output 1? *

Yes

No

c) For the changed object in the Reference Output image, are all the attributes (color, shape, size, material, and relative position) same in Candidate Output 2? *

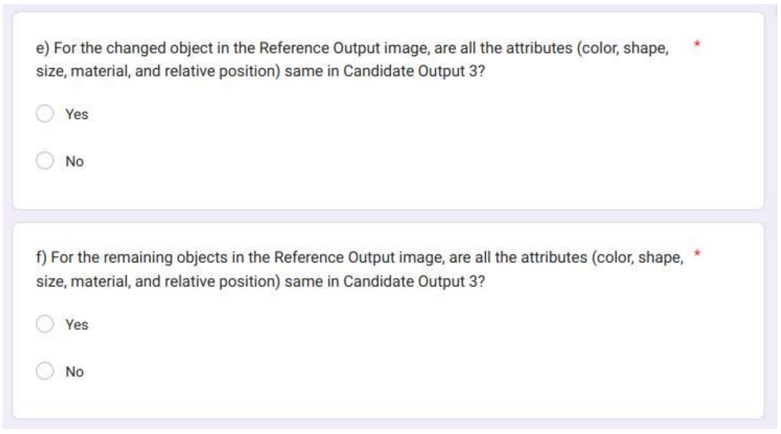
Yes

No

d) For the remaining objects in the Reference Output image, are all the attributes (color, shape, size, material, and relative position) same in Candidate Output 2? *

Yes

No



e) For the changed object in the Reference Output image, are all the attributes (color, shape, size, material, and relative position) same in Candidate Output 3? *

Yes

No

f) For the remaining objects in the Reference Output image, are all the attributes (color, shape, size, material, and relative position) same in Candidate Output 3? *

Yes

No

Figure 12: Screenshots for human evaluation study. See Section 4.7 for more details