# CURI: A Benchmark for Productive Concept Learning Under Uncertainty (Appendix)

## Contents

## 1 Example episodes from the dataset

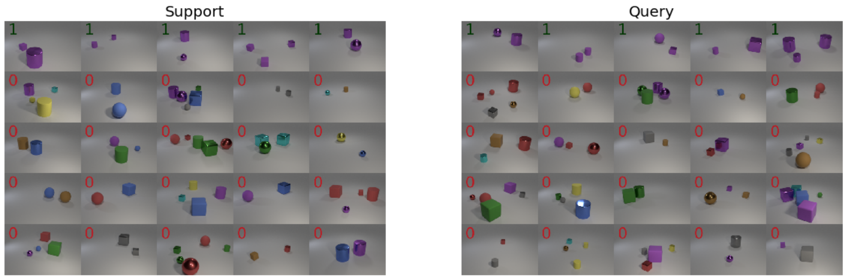
We show examples from the Concept IID split test set comprising the ground truth productive concept (top), along with the support and query sets for meta learning (rendered as images) and the alternate hypotheses which are consistent with the support set – that is, other hypotheses which could also have generated the positive and negative examples in the support set (Figures 1 to 4).

## 2 Additional Dataset Details

We first provide more details of the concept space $\mathcal{G}$, then explain how we obtain $\mathcal{H}$, the space of concepts for training and evaluation, provide more details of the structured splits, and finally explain the weight $w(h)$ based on which we sample concepts.

Productive Concept: for-all x in S and(>(8, locationX?( x ) ), =(purple, color?( x ) ) )

All objects are purple and the x-coordinate of all objects is less than 8



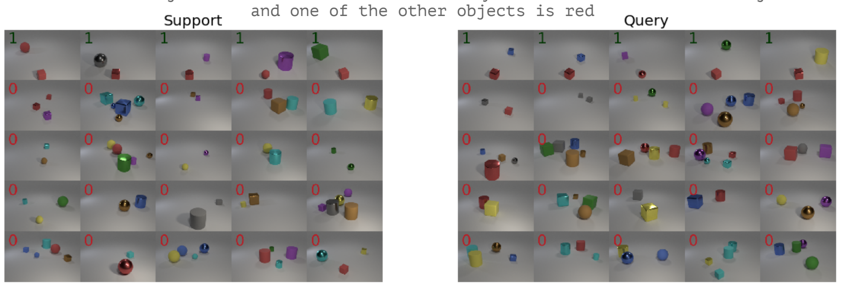
**Other Valid Hypotheses**

```
1) for-all x in S =(purple, color?(x))
   All objects are purple
   Log-Prob: -0.852

2) for-all x in S =(color?(x), purple)
   All objects are purple
   Log-Prob: -0.852

3) all(color?(S), purple)
   All objects are purple
   Log-Prob: -1.950

4) exists x in S and(=(metal, material?(x)), all(color?(S), purple))
   All objects are purple and there exists an object whose material is metal
   Log-Prob: -8.523

5) for-all x in S and(=(color?(x), purple), any(material?(S), metal))
   All objects are purple and there exists an object whose material is metal
   Log-Prob: -8.523
```

Figure 1: Qualitative Example of an Episode in CURI dataset. Best viewed zooming in, in color.

Productive Concept: exists x in S and(all(locationY?( S_{-x} ), 7 ), any(color?( S_{-x} ), red ) )

There exists an object in the scene such that the y-coordinate of all other objects is 7
and one of the other objects is red



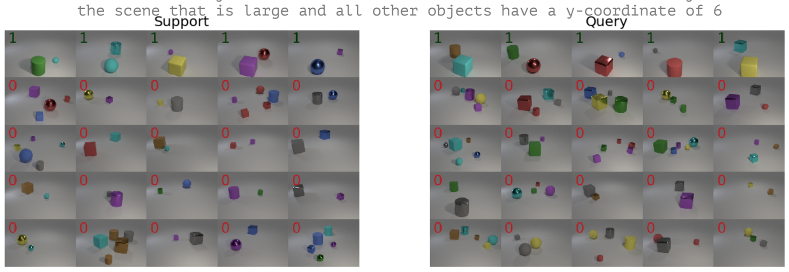
**Valid Hypotheses**

```
1) any(locationY?(S), 7)
   There exists an object in the scene whose y-coordinate is 7
   Log-Prob: -1.621

2) exists x in S >(locationY?(x), 6)
   There exists an object in the scene whose y-coordinate is greater than 6
   Log-Prob: -1.621

3) exists x in S any(locationY?(S_{-x}), 7)
   There exists an object in the scene such that an object other than it has a y-coordinate of 7
   Log-Prob: -1.621

4) exists x in S all(locationY?(S_{-x}), 7)
   There exists an object in the scene such that all other objects have a y-coordinate of 7
   Log-Prob: -1.647

5) exists x in S =(locationY?(x), 7)
   There exists an object in the scene whose y-coordinate is 7
   Log-Prob: -2.314
```

Figure 2: Qualitative Example of an Episode in CURI dataset. Best viewed zooming in, in color.

Productive Concept: exists x in S and(all(locationY?( S_{-x} ), 6 ), any(size?( S_{-x} ), 0.7 ) )

There exists an object in the scene such that there exists another object in the scene that is large and all other objects have a y-coordinate of 6



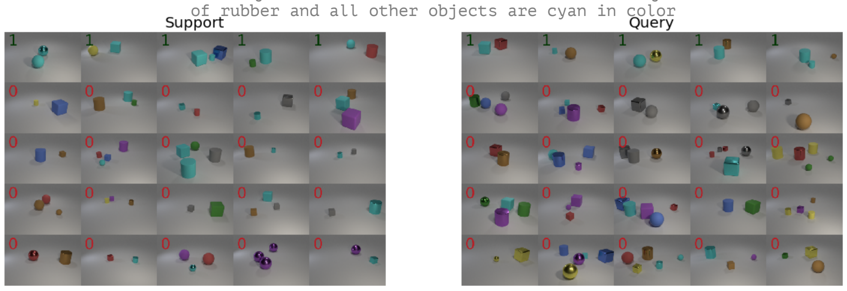
**Valid Hypotheses**

```
1) exists x in S all(locationY?(S_{-x}), 6)
   There exists an object in the scene such that all other objects have a y-coordinate of 6
   Log-Prob: -0.008

2) exists x in S >(count=(locationY?(S_{-x}), 6), count=(size?(S_{-x}), 0.35))
   There exists an object in the scene such that the count of all other objects whose y-coordinate is 6
   is greater than the count of all other objects whose size is small
   Log-Prob: -6.668

3) for-all x in S =(count=(locationY?(S), 6), count=(shape?(S), shape?(x))
   For all objects in the scene, the count of objects which have the same shape as it, and the count of
   all objects which have the a y-coordinate of 6 is equal
   Log-Prob: -6.690

4) for-all x in S =(count=(shape?(S), shape?(x)), count=(locationY?(S), 6))
   For all objects in the scene, the number of objects which have the same shape as it and the count of
   number of objects with y-coordinate of 6 is equal
   Log-Prob: -6.690

5) exists x in S and(>(locationY?(x), 5), >(size?(x), 0.35))
   There exists an object in the scene such that its y-coordinate is greater than 5 and its size is
   greater than small
   Log-Prob: -.451
```

Figure 3: Qualitative Example of an Episode in CURI dataset. Best viewed zooming in, in color.

Productive Concept: exists x in S and(all(material?( S_{-x} ), rubber ), all(color?( S_{-x} ), cyan ) )

There exists an object in the scene such that all other objects are made of rubber and all other objects are cyan in color



**Valid Hypotheses**

```
1) exists x in S and(any(material?(S_{-x}), rubber), all(color?(S_{-x}), cyan))
   There exists an object in the scene such that one of the other objects is made of
   rubber and all other objects are cyan in color
   Log-Prob: -0.983

2) exists x in S and(all(color?(S_{-x}), cyan), all(material?(S_{-x}), rubber))
   There exists an object in the scene such that all other objects are cyan and all
   other objects are made of rubber
   Log-Prob: -1.162

3) exists x in S and(all(material?(S_{-x}), rubber), all(color?(S_{-x}), cyan))
   There exists an object in the scene such that all other objects are cyan and all
   other objects are made of rubber
   Log-Prob: -1.162
```

Figure 4: Qualitative Example of an Episode in CURI dataset. Best viewed zooming in, in color.

## 2.1 More details of the grammar

We provide below the full grammar used to sample concepts, where $A \rightarrow B|C$ means that $A$ can expand to either $B$ or $C$ under the rules defined by the grammar. We always start expanding at the `START` token and then follow the rules of the grammar until we hit a terminal node (which does not have any expansions defined). As and where possible, we followed the insights from Piantadosi et al. [5] in choosing the sampling probabilities for various completions based on how well humans seem to be able to learn the corresponding primitive. For example, we sample utterances with disjunctions (`or`) less frequently since they are known to be difficult for humans to learn. Based on Kemp and Jern [4], we chose to represent location as a discrete entity, such that relative, and categorical notions of `left` or `right` simply become comparisons in the location space (location? $\mathbf{x}$ > location? $S_{-\mathbf{x}}$), unlike the CLEVR dataset [3] which defines categorical relational objects.

Here is the full grammar $\mathcal{G}$ used for sampling the concepts (as explained in the main paper, $S_{-\mathbf{x}} = S/\{\mathbf{x}\}$). Note that the grammar always generates strings in postfix notation and thus the operands in each expansion occur before the operation:

```
START -> λ S. BOOL exists= | λ S. BOOL for-all=

BOOL  -> BOOL BOOL and | BOOL BOOL or | BOOL not |
   C C = | SH SH = | M M = | SI SI = | L L = |
   NUM NUM = | SI SI > | L L > | NUM NUM > |
   SETFC C all | SETFSH SH all | SETFM M all |
   SETFSI SI all | SETFL L all | SETFC C any |
   SETFSH SH any | SETFM M any | SETFSI SI any |
   SETFL L any

NUM   -> SETFC C count= | SETFSH SH count= |
   SETFM M count= | SETFSI SI count= | SETFL L count=
NUM   -> 1 | 2 | 3

SETFC -> SET FC
SETFSH-> SET FSH
SETFM -> SET FM
SETFSI-> SET FSI
SETFL -> SET FL

C     -> gray | red | blue | green | brown | purple |
   cyan | yellow
C     -> OBJECT FC

SH    -> cube | sphere | cylinder
SH    -> OBJECT FSH

M     -> rubber | metal
M     -> OBJECT FM

SI    -> large | small
SI    -> OBJECT FSI

L     -> 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8
L     -> OBJECT FL

FC    -> color?
FSH   -> shape?
FM    -> material?
FSI   -> size?
FL    -> locationX? | locationY?

OBJECT-> x
SET: S | S_-x
```

### 2.2 Sampling

We sample 2000000 initial hypotheses from the CFG $\mathcal{G}$, and impose a maximum depth in the recursion tree of 6 when sampling. That is, no node has a depth larger than 6 in the recursion through which we generate concepts from the grammar $\mathcal{G}$. We then reject and filter the hypotheses to obtain a set of "interesting" hypotheses $\mathcal{H}$ used in the main paper explained in more detail below:
**Rejection Sampling:** We reject the following string combinations after sampling from the grammar $\mathcal{G}$:

- All programs which contain "$\lambda S$. for-all x" and "$S_{-\mathtt{x}}$" in the same program. This is asking that for all objects in a scene, a certain property is satisfied by everything other than the object, which is the same as saying, for all objects in the scene.
- All programs where we compare the properties of the same object to itself, e.g. `color? (x) == color?  (x)`, where color? can be any function applied to the object.
- All programs where we have the following string: `exists(color?(S) == color?(x))` or `for-all(color?(S) == color?(x))` where color? can be any function applied to the object.
- All programs which evaluate to true on schemas more than 10% of the time and less than 10 times. The former condition ensures that we work with concepts which are in some sense interesting and surprising (as opposed to concepts which are always trivially true), and the second condition ensures that we have unique schmeas or datapoints to place in the support and query sets, which both have 5 positive images each.

We provide examples of concepts which get rejected for being true too often below:

```
exists=x \in S or(
  =(locationX?( x ), locationY?( x ) ),
  any(color?( S ), brown )
)
exists=x \in S and(
  exists=(locationY?( S ), locationX?( x ) ),
  any(color?( S ), brown )
)
exists=x \in S or(
  all(color?( S ), gray ),
  all(color?( S ), brown )
)
```

See Section 3 for more details on the structured generalization splits which yeild train concepts $\mathcal{H}_{\text{train}}$ and test concepts $\mathcal{H}_{\text{test}}$.

### 2.3 Concept prior weight $w(h)$

We next explain the form of the prior weight $w(h)$ that we use for defining the prior over the concepts provided to the models (both oracles as well as deep learning models). Given $l(h)$, the number of tokens in the postfix serialization of the concept $h$, the unnormalized weight $\tilde{w}(h)$ is log-linear in the length, and is defined as follows:

$$\tilde{w}(h) \propto \exp -0.2 \cdot l(h) \tag{1}$$

Given a split $\Omega \in \{train, test\}$, the final, normalized weight is given as:

$$w(h) = \frac{\tilde{w}(h)}{\sum_{\mathcal{H}_\Omega} \tilde{w}(h)} \tag{2}$$

As explained in the main paper, the final prior for a hypothesis given a split $\Omega$ is $p(h) = \sum_{h' \in \mathcal{H}_\Omega} w(h)\mathbf{I}[h = h']$.
Our choice of the log-linear weight is inspired by the observation in cognitive science that longer boolean concepts are harder for people to learn [1].
Here are some examples of hypotheses with a high weight (computed on $\Omega = \text{train} \cup \text{test}$):

```
exists x in S =(2, count=(color?( $S_{-x}$ ), cyan ) ),
exists x in S >(locationY?( x ), 6 ),
=(count=(color?( S ), brown ), 3 ),
>(count=(locationX?( S ), 3 ), 2 ),
any(locationY?( S ), 6 ),
=(1, count=(locationY?( S ), 7 ) ),
=(3, count=(locationY?( S ), 3 ) ),
all(locationX?( S ), 2 ),
exists x in S all(locationY?( $S_{-x}$ ), 5 ),
=(2, count=(color?( S ), blue ) ),
for-all x in S not( >(6, locationX?( x ) ) ),
=(count=(color?( S ), gray ), 2 ),
=(2, count=(color?( S ), gray ) )
```

### 2.4 Execution on Images.

In order to create the perceptual inputs in the dataset $\mathcal{U}$, we sample images using the renderer for CLEVR from Johnson et al. [3], changing the range of objects to $[2, 5]$, to reduce clutter and enable easier learning of predicates like `any` and `all` for models. [1] The CLEVR renderer produces scenes $\mathbf{u}$ with pixels as well as an associated schema file $\mathbf{u}^s$ detailing the properties of all the objects sampled in the scene, including their location, shape, size, material, and rotation. Given the original CLEVR scenes, we map some of the properties to other floating point or integer values before execution. The exact mappings are:

1. "large" size $\rightarrow$ 0.7
2. "small" size $\rightarrow$ 0.35
3. "x-coordinate" of each object discretized to 8 bins
4. "y-coordinate" of each object discretized to 8 bins

Based on this, we convert our sampled concepts into postfix notation and execute them on the schemas using an operator stack. Concretely, execution of the concept $h \in \mathcal{H}$ on $\mathbf{u}^s$ yields a boolean true or false value $\{0, 1\}$. We execute each such hypothesis on a set of 990K images, yielding scores of how often a hypothesis is true for an image. We threshold this score to retain the subset of hypotheses which are true for no more than 10% of the images and are true at least for 10 images, to pick a subset of "interesting" hypotheses $\mathcal{H}'$ for training models.

**Bias.** The image dataset here sampled itself has a bias in terms of the location coordinates (in the pixel space). The CLEVR dataset generation process samples objects in the 3d (top-down x, y) coordinate space uniformly (from a grid of -3, to +3). However, since the camera is always looking into the scene from outside, the image formation geometry implies that in the camera / image coordinates most of the objects appear to be away from the scene and very few are close to the camera. Thus, in terms of the y-coordinates we observe in the image coordinates a bias in terms of the distribution not being unifrom. This also makes sense in general, as even in the real world, objects are not found very close to the camera or very far away from the camera in general. See Figure 5 for all the biases in the observation space $\mathbf{u}$ computed over 990K sampled images.

### 2.5 Audio.

To build the audio data, we use clips of orchestral instruments playing various pitches downloaded from `https://philharmonia.co.uk/resources/sound-samples/`. We make the following mappings of object properties:

- $x$ location $\rightarrow$ temporal location. larger $x$ bin means the note is played later.
- $y$ location $\rightarrow$ pitch. All pitches between the instruments are the same (up to octaves).
- color $\rightarrow$ instrument
    - gray $\rightarrow$ trumpet

[1]Since the chances of a constraint being true for all obejcts reduce exponentially as the number of objects increases.

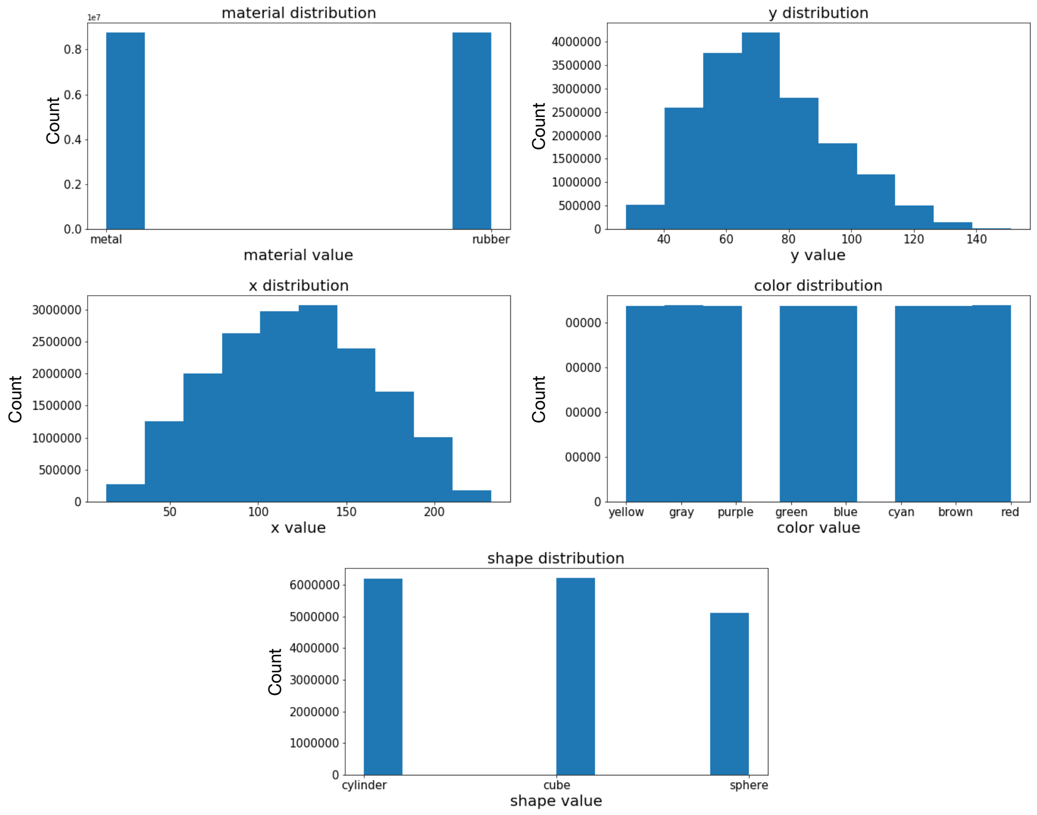


Figure 5: Histogram of properties found in inputs **u** in the dataset. We notice that the properties are all largely uniform with bias in x and y-coordinates towards the center of the image.

- red → clarinet
- blue → violin
- green → flute
- brown → oboe
- purple → saxaphone
- cyan → french-horn
- yellow → guitar

- shape → amplitude profile; either getting louder, getting softer, or constant volume
- size → total volume
- material → low-pass filtering or no filtering.

All binned quantities use the same number of bins as in the image domain.

### 2.6 Analysis of synonomy of concepts.

We next show an analysis of concepts which have the same evaluation signatures on a large set of 990K images, and are thus synonymous (in context of the dataset at hand). Note that while some of these concepts might be truly synonymous to each other (for example, $A > B$ is the same as $B < A$), others might be synonymous in context of the image distribution we work with. For example, size can never be greater than 0.7 in our dataset and location can never be greater than 8, and thus asking if location is greater than 8 or size is greater than 0.7 has the same semantics on our dataset. In Figure 6 we show each such "concept" or "meaning", which is a cluster of hypotheses which all evaluate to the same truth value and plot a histogram of how many hypotheses each cluster tends to have. We notice that most of the concepts have 1 synonym (i.e. there is only one concept with the particular) evaluation signature, with a long tail going upto 80 synonyms in a concept. In the Concept IID split we ensure that none of the concepts which have the same signature are found across the train/val/test splits.

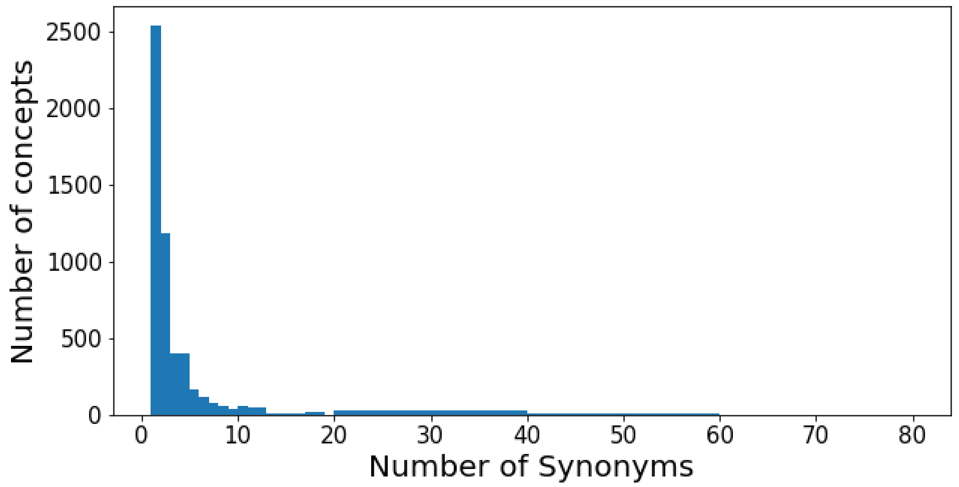


Figure 6: Histogram of number hypotheses with same evaluation signatures.

## 3 Detailed discussion of the structured splits

We provide more details on how each of the structured splits described in Sec. 3 of the main paper are created. Assuming access to $\mathcal{H}$, the space of concepts sampled and filtered from the grammar $\mathcal{G}$, we use various heuristics to produce the generalization splits in the paper:

- Instance IID: This split is trivial since $\mathcal{H}_{\text{train}} = \mathcal{H}_{\text{test}} = \mathcal{H}$
- Concept IID: This split divides concepts into train and test by picking concepts at random from $\mathcal{H}$ and assigning them to $\mathcal{H}_{\text{train}}$ or $\mathcal{H}_{\text{test}}$ while ensuring that no two concepts which are synonyms Section 2.6 are found in different splits.
- Boolean: This split forms cross product of all possible colors and `{and, or}` boolean operators, and partitions a subset of such combinations which we want to only occur in test. We use the following tokens for test:

  ```
  'green', 'or' | 'purple', 'and' | 'cyan', 'and' |
  'red', 'or' | 'green', 'and'
  ```

  We then create $\mathcal{H}_{\text{test}}$ to contain all concepts which have any of the combinations above. For example, if a concept has both `green` and `or` we would place it in $\mathcal{H}_{\text{test}}$. After every feasible candidate is placed in $\mathcal{H}_{\text{test}}$ based on this heurisitc, the remaining concepts in $\mathcal{H}$ are assigned to $\mathcal{H}_{\text{train}}$.
- Extrinsic: This split forms cross product of all possible colors and locations in the dataset, and partitions a subset of such combinations that we only want to occur in test. We use the following tokens for test (only a subset shown for illustration):

  ```
  '7', 'gray' | '1', 'red' | '3', 'purple' |
  '1', 'blue' | '8', 'cyan' | '5', 'yellow' |
  '5', 'green' | '3', 'yellow' | '7', 'purple' |
  '2', 'blue' | '3', 'cyan'
  ```

  We then create $\mathcal{H}_{\text{test}}$ to contain all concepts which have any of the combinations above. For example, if a concept has both `gray` and `7`, and is related to location, that is contains `locationX?` or `locationY?` keywords, we would place it in $\mathcal{H}_{\text{test}}$. After every feasible candidate is placed in $\mathcal{H}_{\text{test}}$ based on this heurisitc, the remaining concepts in $\mathcal{H}$ are assigned to $\mathcal{H}_{\text{train}}$.

- Intrinsic: This split forms cross product of all possible colors and materials in the dataset, and partitions a subset of such combinations that we only want to occur in test. We use the following tokens for test:

```
‘green’, ‘metal’ | ‘purple’, ‘rubber’ | ‘cyan’, ‘rubber’ |
‘red’, ‘metal’ | ‘green’, ‘rubber’
```

  We then create $\mathcal{H}_{\text{test}}$ to contain all concepts which have any of the combinations above. For example, if a concept has both `green` and `metal`, and is related to material, that is contains `material?` keyword, we would place it in $\mathcal{H}_{\text{test}}$. After every feasible candidate is placed in $\mathcal{H}_{\text{test}}$ based on this heurisitc, the remaining concepts in $\mathcal{H}$ are assigned to $\mathcal{H}_{\text{train}}$.

- Binding (color): This split takes all possible colors in the dataset, and partitions a subset of colors that we only want to occur in test. We use the following tokens for test:

```
‘purple’ | ‘cyan’ | ‘yellow’
```

  We then create $\mathcal{H}_{\text{test}}$ to contain all concepts which have any of the tokens above. For example, if a concept has `purple`, we would place it in $\mathcal{H}_{\text{test}}$. After every feasible candidate is placed in $\mathcal{H}_{\text{test}}$ based on this heurisitc, the remaining concepts in $\mathcal{H}$ are assigned to $\mathcal{H}_{\text{train}}$.

- Binding (shape): This split takes all possible shapes in the dataset, and partitions a subset of shapes that we only want to occur in test. We use the following tokens for test:

```
‘cylinder’
```

  We then create $\mathcal{H}_{\text{test}}$ to contain all concepts which have any of the tokens above. For example, if a concept has `cylinder`, we would place it in $\mathcal{H}_{\text{test}}$. After every feasible candidate is placed in $\mathcal{H}_{\text{test}}$ based on this heurisitc, the remaining concepts in $\mathcal{H}$ are assigned to $\mathcal{H}_{\text{train}}$.

- Complexity: This split partitions into train and test based on length of the postfix serialization of the concept. Specifically, concepts shorter than 10 tokens are placed in $\mathcal{H}_{\text{train}}$ and longer concepts are placed in $\mathcal{H}_{\text{test}}$.

## 4 Creating Support and Query Sets

We next explain how we go from the initial dataset $\mathcal{U}$ – which contains a large number of images, schema and sounds – and a concept space $\mathcal{H}_{\text{train}}$ and $\mathcal{H}_{\text{test}}$, to a dataset for meta learning. To create the training/validation/test sets for models, we sample a series of episodes, each containing a support set and a query set. We illustrate the sampling procedure for a training episode below:

**Support Set Sampling with Hard Negatives**

1. Pick a concept $h \sim p_{\text{train}}(h)$, with a preference for shorter hypotheses being more frequent based on the weights used to define the prior Section 2.3

2. Pick 5 images ($P$), uniformly at random from $\mathcal{U}$ such that $h(\mathbf{u}^s) = 1$, where the concept is evaluated on the schema to determine the label Section 2.4

3. Identify other concepts $h' \in \mathcal{H}$ s.t. $h(u(S)) = 1$ and $h' \neq h$

4. Pick images such that $h'(\mathbf{u}^s) = 1$ and $h(\mathbf{u}^s) = 0$ as negatives ($N$). If no such images exist, pick random images from $\mathcal{U}$ as negatives until we have 20 negatives.

5. Relabel any candidate negatives $\mathbf{u} \in N$ where $h(\mathbf{u}) = 1$, by removing them from $N$ and placing them in $P$.

6. Return $D_{\text{supp}} = P \cup N$.

The sampling procedure for the Query set iterates all the steps above (except step 1, where we choose the concept $h$). Step 3 and 4 outline a procedure for identifying hard negatives for training the model, by looking at other hypotheses which also explain a chosen set of positives $P$ and using them to clarify what the concept of interest is.

We give below an analogous procedure for easy negatives:

**Support Set Sampling with Easy Negatives**

1. Pick a concept $h \sim p_{\text{train}}(h)$, with a preference for shorter hypotheses being more frequent based on the weights used to define the prior Section 2.3

2. Pick 5 images ($P$), uniformly at random from $\mathcal{U}$ such that $h(\mathbf{u}^s) = 1$, where the concept is evaluated on the schema to determine the label Section 2.4

3. Pick 20 random images from $\mathcal{U}$ as negatives, N.

4. Relabel any candidate negatives $\mathbf{u} \in N$ where $h(\mathbf{u}) = 1$, by removing them from $N$ and placing them in $P$.

5. Return $D_{\text{supp}} = P \cup N$.

Similar to hard negatives, the sampling procedure for the Query set iterates all the steps above (except step 1, where we choose the concept $h$).

## 5 Reproducibility and Hyperparameters

For all the models in Figure. 5 in the main paper, we use the following hyperparameters. All the modalities are processed into a set of objects $\{o_i\}_{i=1}^{N}$ where each $o_i \in \mathbb{R}^{64}$ for image and sound models while for schema $o_i \in \mathbb{R}^{96}$. Further, the we use a learning rate of `1e-4` for image models, `1e-3` for schema models, and `5e-5` using the best learning rate for each modality across an initial sweep. The batch size for image and sound models is 8 episodes per batch, while for schema we use a batch size of 64. All models use the Adam optimizer. The overall scene representation across all the modalities is set to 256, that is, $\mathbf{u} \in \mathbb{R}^{256}$. All our models are initialized with the method proposed in [2], which we found to be crucial for training relation networks well. The initial representation from the first stage of the encoder (Figure. 5 in main paper) with the objects for images has a size of `10x8` *i.e.* there are 80 objects, while for sound representations have `38` objects. In the schema case the number of objects is the ground truth number of objects which is provided as input to the model.

We trained all the models on the training set, and picked the best performing checkpoints on training – measured in terms of mAP– to report all the results in the main paper. Our image and schema models are trained for 1 million steps (16 epochs for images, 128 epochs for schemas) while the sound models are trained for 500K steps, and checkpoints are stored after every 30K steps.

All our models fit on a GPU with 16GB capacity except the relation network trained with image inputs, which needs at 32GB GPU. We use the pytorch framework to implement all our models.

## 6 Model Architectures for Pooling

In this section we detail the exact architectures used for the different pooling operations we consider in this paper as shown in Figure. 5 center panel (main paper). We first establish some notation. Let $o_i \in \mathbb{R}^K$ be the output object feature from the modality specific encoder (Figure. 5, left panel, main paper), and let us denote by $O = \{o_i\}_{i=1}^{|O|}$ the set of features for each of the objects in the scene, which includes optional position information indicating where the object is present in the scene (Figure. 5). Let $N$ be the requested dimensionality of the feature space from the pooling operation. Given this, we can describe the pooling operations used as follows:

- **avg-pool**: We first average the representations across all the objects $\{o_i\}_{i=1}^{|O|}$ and then pass the averaged representation through an MLP with `256 x 512 x 384 x N` units with batch normalization and rectified linear unit nonlinearity in the hidden layers.

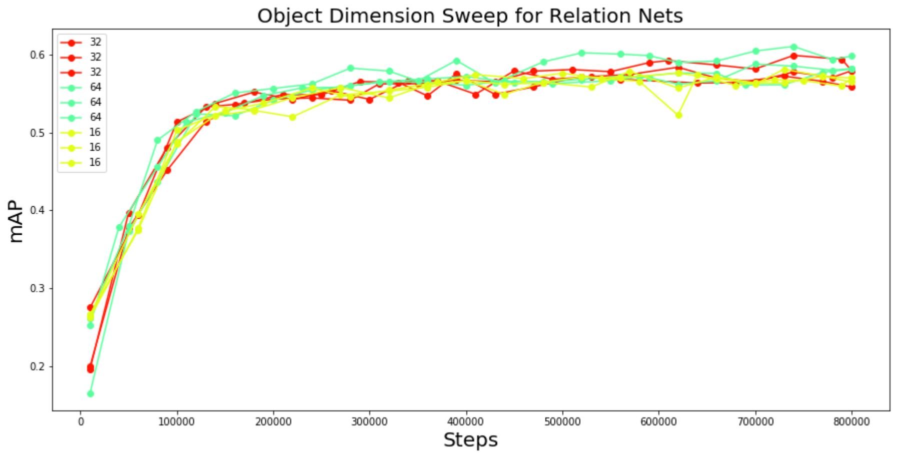


Figure 7: mAP on validation for hard negatives (y-axis) vs number of training steps (x-axis) for relation network models on images with different dimensionality for the object embedding $o_i$.

- **concat**: We first concatenate all the object representations in $O$, followed by an MLP with `256 x 512 x 256 x N` units with batch normalization and rectified linear units nonlinearity in the hidden layers.

- **relation-net**: For relation networks, following [6] we use relative position encoding that captures the relative positioning of the objects in a scene for image and sound modalities, and use the location information already present in the schema modality. Based on this, in the terminology of Santoro et al. [6] our $g()$ MLP has `256 x 256 x 256 x 256` hidden units with rectified linear unit non linearity and batch normalization whereas our $f()$ MLP has `256 x 256 x N` units with recitifed linear unit non linearlity and batch normalization in the middle (non-output) layers. Different from the original paper, we do not use dropout as we did not observe any overfitting in our experiments.

- **transformer**: We use a 2-head multi-head attention layer stacked 4 times, with the feedforward network dimenstions set to 512. After forwarding through this module, we take the output vectors $o_i'$ for each object processed through these initial layers and pool across objects by doing `max(), mean(), sum(), min()` operations and concatenating their outputs, similar to previous work by Wang et al. [7]. The final representation then does a linear projection of this concatenated vector to $N$, the dimensionality expected from the pooling module.

## 7 Additional Results

### 7.1 Hyperparameter sweeps – object feature dimensions

We next show the hyperparameter sweeps for image models in deterimining the choice of the dimensionality to represent each object $o_i$ for our image models (Figure 7). The same choice of dimensionality was replicated for sound models. In our initial sweeps, on the Concept IID split, across different choices of the dimensionality of objects, we found relation networks to outperform concat and global average pooling models substantially, and thus we picked the object dimensions based on what performs best for relation networks since overall we are interested in the best possible choice of models for a given split and modality. Based on the results in Figure 7 we picked $o_i \in \mathbb{R}^{64}$.

### 7.2 Image relation networks learning rate sweeps

We picked the learning rate for image models based on the best performing image relation network model, which across an initial sweep we found to yeild the best class of models. Figure 8 shows the performance of the models across learning rates of `{1e-4, 5e-4, 2.5e-4}`.

### 7.3 Sweep on use of Language

As explained in the main paper (Figure. 5), the parameter $\alpha$ controls the tradeoff between the query accuracy and the likelihood of the concept expressed as a prefix string. We generally found

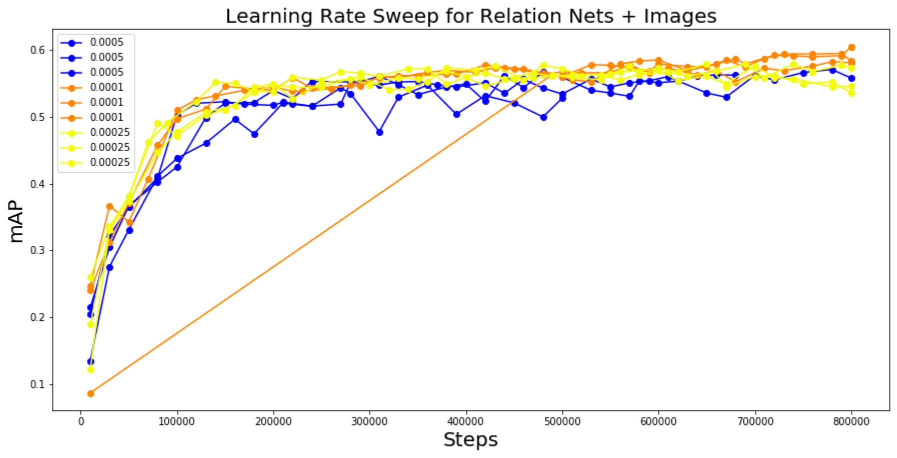


Figure 8: mAP on validation for hard negatives (y-axis) vs number of training steps (x-axis) for relation network models on images with different learning rates.

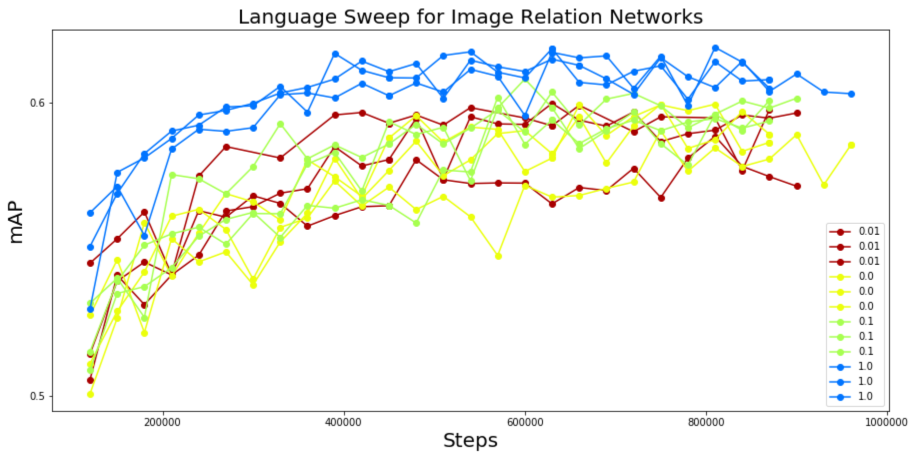


Figure 9: mAP on validation for hard negatives (y-axis) vs number of training steps (x-axis) for relation network models on images with different amounts of language usage by varying the parameter $\alpha$.

across a broad range of values in `{0.0, 0.01, 0.10, 1.0}` the models generally performed the best at $\alpha = 1.0$. Our initial experiments with $\alpha = 10.0$ suggested substantially worse performance so we discareded it from the sweep. See Figures 9 to 11 for the corresponding results.

### 7.4 Results on Easy Negatives

In Figure 12 we show results for the relation-net model on various splits, where easy negatives are used to populate the support and query sets during training and evaluation, unlike the case of hard negatives discussed in the main paper (Figure. 6). Notice that the compositionality gap (compositionality gap) is lower in general for easy negatives compared to the hard negatives as reported in the main paper. Further, we find that the best models are substantially closer to the strong oracle compared to Figure. 6 main paper, showing that on the easier, less compositional task it is easier for machine learning models to approach the strong oracle (especially in terms of accuracy). Finally, it is interesting to note that with easy negatives it appears that the best models outperform the weak oracle on the Counting split, while with the hard negatives one finds that the models are worse than the weak oracle, suggesting poor generalization for counting.

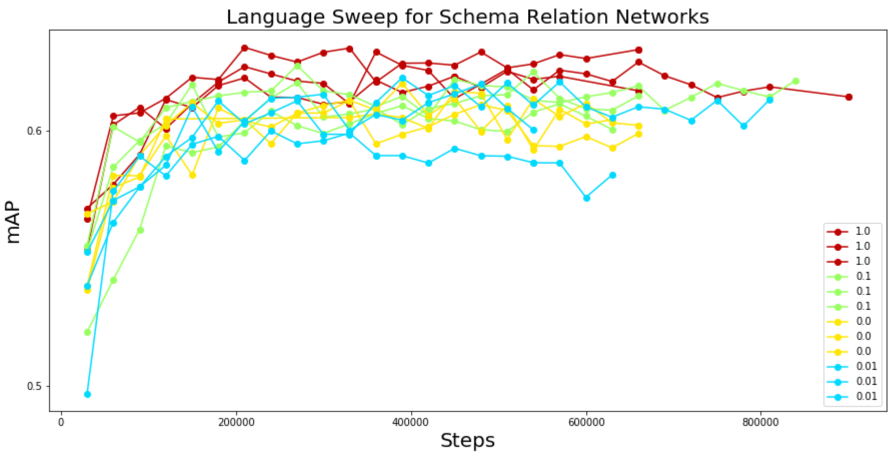


Figure 10: mAP on validation for hard negatives (y-axis) vs number of training steps (x-axis) for relation network models on schemas with different amounts of language usage by varying the parameter $\alpha$.

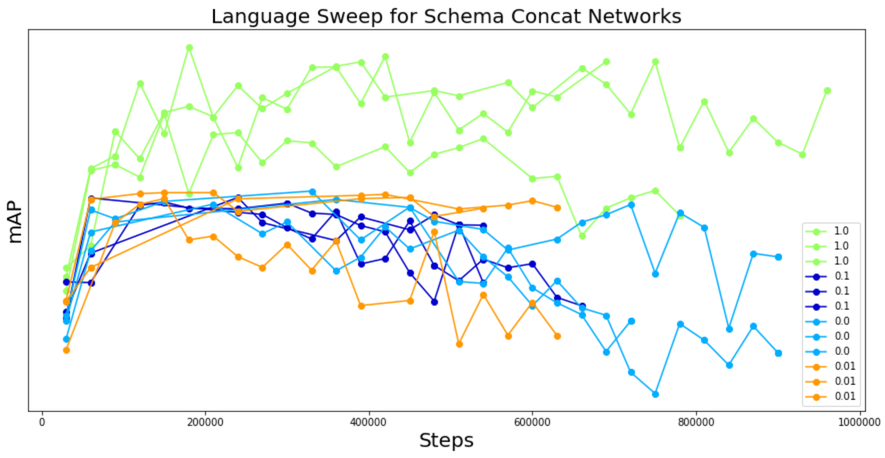


Figure 11: mAP on validation for hard negatives (y-axis) vs number of training steps (x-axis) for concat pooling models on schemas with different amounts of language usage by varying the parameter $\alpha$.

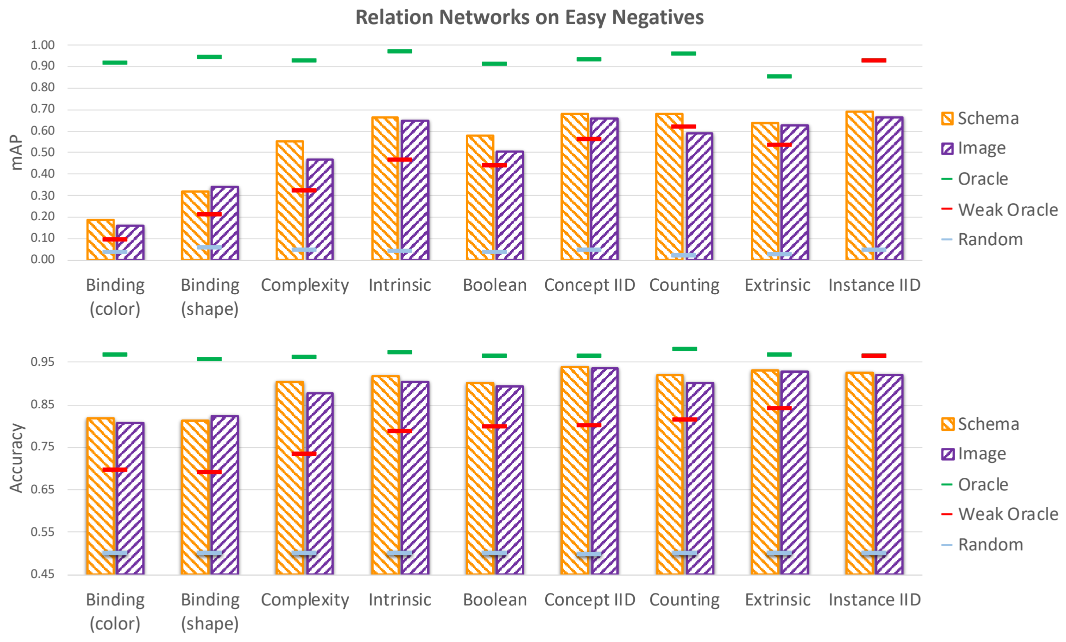


Figure 12: mAP (top) and accuracy (bottom) metrics for the different splits presented in the paper when easy negatives are used (ordered by their corresponding compositionality gap). **Yellow:** shows the schema relation-net model while **purple:** shows the image relation-net model. Notice that compared to Fig. 5 in the main paper, the compositionality gap is smaller and the models appear to be substantially closer to the strong oracle in this setting compared to when we use hard negatives.

## 7.5 Finer $\alpha$ sweep for Counting

Finally, we ran a finer alpha sweep for the Counting split since it appeared on our initial sweep that the counting split was not performing better with language. Concretely, we ran a new set of experiments sweeping over $\alpha$ values of `{0.01, 0.10, 1.0, 5.0, 10.0, 100.0}`. Across this broader range of values, we found models still did not show any statistically significant gains from using language *vs.* not for the Counting split.

## 7.6 Choice of metric: mAP *vs.* accuracy

In general, the mAP metric opens up a larger compositionality gap for the various splits than indicated by accuracy. For example, with hard negatives, while accuracy indicates a gap of 14.2% for Counting compared to 0% for Instance IID, mAP suggests a gap of 34.4% for Counting relative to 0% for Instance IID. For the Binding (color) split its 86.5% compositionality gap (mAP) *vs.* 34.0% for accuracy. mAP, while being more expensive to compute evaluates more thoroughly to test if a concept $h$ is truly learnt by the model, by probing its performance on a large, representative set of negatives $\mathcal{T}$, providing a more stringent test of compositional generalization.

## 7.7 Detailed results on all the splits in easy negatives setting.

In this section we provide the full results of all of the tested models on each of the splits considered in the paper, in the easy negatives setting. Tables 1-18 show the results of different models (sorted in a descending order based on accuracy or mAP (whichever is the metric reported) for each of the splits considered in the paper, in the case where models do not have access to language. Note that we did not evaluate transformer models or sound models in this setting as this is qualitatively less interesting than the hard negatives setting and is not the main focus of the paper.

## References

[1] Feldman, J. (2000, October). Minimization of boolean complexity in human concept learning. *Nature 407*(6804), 630–633.
[2] Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington (Eds.), *Proceedings of the Thirteenth*

Table 1: Performance on meta-test, sorted based on mAP (in %) on Binding (color) with easy negatives

| modality | pooling | mAP (mean) | mAP (std) |
|---|---|---|---|
| schema | relation-net | 18.6 | 1.6 |
| image | relation-net | 16.3 | 0.8 |
| image | avg-pool | 16.3 | 0.5 |
| image | concat | 15.6 | 0.5 |
| schema | avg-pool | 15.5 | 0.5 |
| schema | concat | 15.5 | 0.5 |

Table 2: Performance on meta-test, sorted based on mAP (in %) on Boolean with easy negatives

| modality | pooling | mAP (mean) | mAP (std) |
|---|---|---|---|
| schema | relation-net | 58.1 | 2.0 |
| schema | avg-pool | 55.0 | 2.1 |
| schema | concat | 54.7 | 2.2 |
| image | relation-net | 50.7 | 7.3 |
| image | avg-pool | 42.9 | 1.6 |
| image | concat | 40.9 | 1.7 |

Table 3: Performance on meta-test, sorted based on mAP (in %) on Counting with easy negatives

| modality | pooling | mAP (mean) | mAP (std) |
|---|---|---|---|
| schema | relation-net | 67.9 | 8.5 |
| schema | avg-pool | 64.4 | 7.3 |
| image | avg-pool | 62.2 | 6.6 |
| image | concat | 61.2 | 6.9 |
| schema | concat | 60.8 | 6.7 |
| image | relation-net | 58.8 | 5.7 |

Table 4: Performance on meta-test, sorted based on mAP (in %) on Extrinsic with easy negatives

| modality | pooling | mAP (mean) | mAP (std) |
|---|---|---|---|
| schema | relation-net | 63.7 | 1.4 |
| image | relation-net | 62.9 | 1.1 |
| image | concat | 61.8 | 2.4 |
| image | avg-pool | 61.5 | 1.4 |
| schema | avg-pool | 57.5 | 1.2 |
| schema | concat | 57.2 | 1.5 |

Table 5: Performance on meta-test, sorted based on mAP (in %) on Intrinsic with easy negatives

| modality | pooling | mAP (mean) | mAP (std) |
|---|---|---|---|
| schema | relation-net | 66.5 | 4.8 |
| image | relation-net | 64.9 | 5.3 |
| image | avg-pool | 64.4 | 4.5 |
| schema | avg-pool | 63.1 | 4.7 |
| image | concat | 62.7 | 4.5 |
| schema | concat | 60.8 | 4.6 |

Table 6: Performance on meta-test, sorted based on mAP (in %) on Concept IID with easy negatives

| modality | pooling | mAP (mean) | mAP (std) |
|---|---|---|---|
| schema | relation-net | 68.2 | 1.0 |
| image | relation-net | 65.8 | 0.4 |
| image | avg-pool | 65.1 | 0.4 |
| image | concat | 64.7 | 0.4 |
| schema | avg-pool | 63.3 | 0.4 |
| schema | concat | 61.9 | 0.2 |

Table 7: Performance on meta-test, sorted based on mAP (in %) on Instance IID with easy negatives

| modality | pooling | mAP (mean) | mAP (std) |
|---|---|---|---|
| schema | relation-net | 69.0 | 3.6 |
| image | avg-pool | 66.8 | 3.3 |
| image | relation-net | 66.5 | 3.2 |
| image | concat | 65.4 | 3.4 |
| schema | avg-pool | 63.5 | 3.1 |
| schema | concat | 63.3 | 3.1 |

Table 8: Performance on meta-test, sorted based on mAP (in %) on Complexity with easy negatives

| modality | pooling | mAP (mean) | mAP (std) |
|---|---|---|---|
| schema | relation-net | 55.1 | 1.8 |
| schema | avg-pool | 51.9 | 2.0 |
| schema | concat | 51.3 | 1.9 |
| image | avg-pool | 49.0 | 1.6 |
| image | concat | 48.2 | 1.7 |
| image | relation-net | 46.6 | 1.5 |

Table 9: Performance on meta-test, sorted based on mAP (in %) on Binding (shape) with easy negatives

| modality | pooling | mAP (mean) | mAP (std) |
|---|---|---|---|
| image | relation-net | 33.9 | 2.4 |
| schema | avg-pool | 32.1 | 1.7 |
| image | avg-pool | 31.9 | 1.8 |
| schema | relation-net | 31.7 | 1.6 |
| image | concat | 31.3 | 1.7 |
| schema | concat | 31.0 | 1.5 |

Table 10: Performance sorted based on accuracy (in %) on Binding (color) with easy negatives

| modality | pooling | accuracy | accuracy (Std) |
|---|---|---|---|
| schema | relation-network | 81.8 | 2.2 |
| image | relation-network | 80.9 | 2.0 |
| image | avg-pool | 80.4 | 1.9 |
| schema | concat | 79.4 | 2.0 |
| schema | avg-pool | 79.4 | 1.9 |
| image | concat | 79.2 | 1.9 |

Table 11: Performance sorted based on accuracy (in %) on Boolean with easy negatives

| modality | pooling | accuracy | accuracy (Std) |
|---|---|---|---|
| schema | relation-network | 90.3 | 1.8 |
| schema | avg-pool | 90.0 | 1.7 |
| schema | concat | 89.8 | 1.8 |
| image | relation-network | 89.4 | 2.0 |
| image | avg-pool | 89.1 | 1.8 |
| image | concat | 88.8 | 1.9 |

Table 12: Performance sorted based on accuracy (in %) on Counting with easy negatives

| modality | pooling | accuracy | accuracy (Std) |
|---|---|---|---|
| schema | relation-network | 92.0 | 5.4 |
| image | avg-pool | 90.6 | 5.0 |
| schema | avg-pool | 90.5 | 5.0 |
| schema | concat | 90.5 | 4.7 |
| image | concat | 90.1 | 5.1 |
| image | relation-network | 90.0 | 5.3 |

Table 13: Performance sorted based on accuracy (in %) on Extrinsic with easy negatives

| modality | pooling | accuracy | accuracy (Std) |
|---|---|---|---|
| image | avg-pool | 93.0 | 1.2 |
| schema | relation-network | 93.0 | 1.2 |
| image | concat | 92.9 | 1.2 |
| image | relation-network | 92.9 | 1.2 |
| schema | avg-pool | 92.4 | 1.3 |
| schema | concat | 92.3 | 1.2 |

Table 14: Performance sorted based on accuracy (in %) on Intrinsic with easy negatives

| modality | pooling | accuracy | accuracy (Std) |
|---|---|---|---|
| schema | relation-network | 91.9 | 3.4 |
| image | avg-pool | 91.6 | 3.3 |
| schema | avg-pool | 91.6 | 3.3 |
| image | concat | 91.4 | 3.2 |
| schema | concat | 91.3 | 3.5 |
| image | relation-network | 90.3 | 3.7 |

Table 15: Performance sorted based on accuracy (in %) on Concept IID with easy negatives

| modality | pooling | accuracy | accuracy (Std) |
|---|---|---|---|
| schema | relation-network | 94.0 | 0.2 |
| image | avg-pool | 93.6 | 0.1 |
| image | relation-network | 93.5 | 0.1 |
| image | concat | 93.3 | 0.1 |
| schema | avg-pool | 93.0 | 0.1 |
| schema | concat | 92.6 | 0.1 |

Table 16: Performance sorted based on accuracy (in %) on Instance IID with easy negatives

| modality | pooling | accuracy | accuracy (Std) |
|---|---|---|---|
| schema | relation-network | 92.6 | 2.8 |
| image | avg-pool | 92.3 | 2.7 |
| image | relation-network | 92.1 | 2.6 |
| image | concat | 91.9 | 2.8 |
| schema | avg-pool | 91.6 | 2.7 |
| schema | concat | 91.4 | 2.7 |

Table 17: Performance sorted based on accuracy (in %) on Complexity with easy negatives

| modality | pooling | accuracy | accuracy (Std) |
|---|---|---|---|
| schema | relation-network | 90.3 | 1.9 |
| schema | avg-pool | 89.7 | 1.9 |
| schema | concat | 89.3 | 2.1 |
| image | avg-pool | 88.2 | 2.1 |
| image | relation-network | 87.8 | 2.0 |
| image | concat | 87.7 | 2.3 |

Table 18: Performance sorted based on accuracy (in %) on Binding (shape) with easy negatives

| modality | pooling | accuracy | accuracy (Std) |
|---|---|---|---|
| image | relation-network | 82.4 | 2.1 |
| image | avg-pool | 81.9 | 2.1 |
| schema | avg-pool | 81.7 | 2.2 |
| image | concat | 81.7 | 2.2 |
| schema | relation-network | 81.2 | 2.6 |
| schema | concat | 80.9 | 2.2 |

*International Conference on Artificial Intelligence and Statistics*, Volume 9 of *Proceedings of Machine Learning Research*, Chia Laguna Resort, Sardinia, Italy, pp. 249–256. PMLR.

[3] Johnson, J., B. Hariharan, L. van der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick (2017). CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[4] Kemp, C. and A. Jern (2009). Abstraction and relational learning. *Neural Information Processing Systems (NeurIPS)*.

[5] Piantadosi, S. T., J. B. Tenenbaum, and N. D. Goodman (2016, July). The logical primitives of thought: Empirical foundations for compositional cognitive models. *Psychol. Rev. 123*(4), 392–424.

[6] Santoro, A., D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap (2017). A simple neural network module for relational reasoning. In *Neural Information Processing Systems (NeurIPS)*.

[7] Wang, D., M. Jamnik, and P. Lio (2020). Abstract diagrammatic reasoning with multiplex graph networks. In *International Conference on Learning Representations (ICLR)*.