COVR: A Test-Bed for Visually Grounded Compositional Generalization with Real Images

Anonymous EMNLP submission

Abstract

While interest in models that generalize at test time to new compositions has risen in recent years, benchmarks in the visually-grounded domain have thus far been restricted to synthetic images. In this work, we propose COVR, a new test-bed for visually-grounded compositional generalization with real images. To create COVR, we use real images annotated with scene graphs, and propose an almost fully automatic procedure for generating question-answer pairs along with a set of context images. COVR focuses on questions that require complex reasoning, including higher-order operations such as quantification and aggregation. Due to the automatic generation process, COVR facilitates the creation of compositional splits, where models at test time need to generalize to new concepts and compositions in a zero- or few-shot setting. We construct compositional splits using COVR and demonstrate a myriad of cases where state-of-the-art pre-trained language-and-vision models struggle to compositionally generalize.

1 Introduction

Models for natural language understanding (NLU) have exhibited remarkable generalization abilities on many tasks, when the training and test data are sampled from the same distribution. But such models still lag far behind humans when asked to generalize to an unseen combination of known concepts, and struggle to learn concepts for which only few examples are provided (Finegan-Dollak et al., 2018; Bahdanau et al., 2019a). Humans, conversely, do this effortlessly: for example, once humans learn the meaning of the quantifier *"all"*, they can easily understand the utterance *"all cheetahs have spots"* if they know what *"cheetahs"* and *"spots"* mean (Chomsky, 1957; Montague, 1970; Fodor and Pylyshyn, 1988). This ability, termed *compositional generalization*, is crucial for building models that generalize to new settings (Lake et al., 2018).

Figure 1: Compositional generalization in the VQA setup. COVR enables the creation of compositional splits such as the one depicted here, where quantification appears with conjunction only in the test set.

In recent years, multiple benchmarks have been created, illustrating that current NLU models *fail* to generalize to new compositions. However, these benchmarks focused on semantic parsing, the task of mapping natural language utterances to logical forms (Lake and Baroni, 2018; Kim and Linzen, 2020; Keysers et al., 2020). Visual question answering (VQA) is arguably a harder task from the perspective of compositional generalization, since the model needs to learn to compositionally "execute" the meaning of the question over images, without being exposed to an explicit meaning representation. For instance, in Fig. 1, a model should learn the meaning of the quantifier *"all"* from the first example, and the meaning of a conjunction of clauses from the second example, and then execute both operations compositionally at test time.

Existing VQA datasets for testing compositional generalization are mostly synthetic and contain a limited number of visual concepts and reasoning operators (Bahdanau et al., 2019a,b; Ruis et al., 2020). GQA (Hudson and Manning, 2018) uses

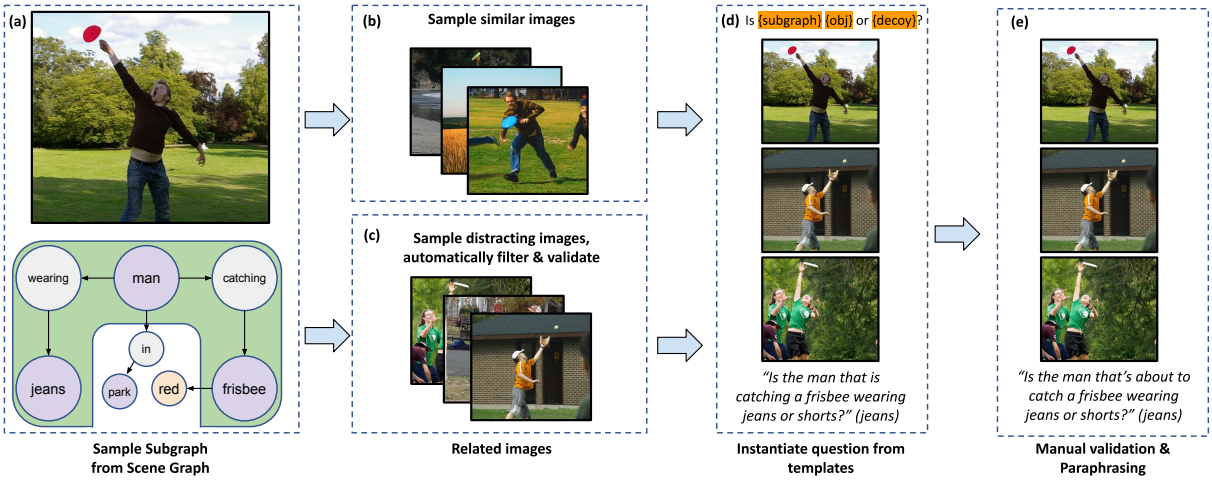

Figure 2: An overview of the dataset creation process.

real images with synthetic questions, but lacks logical operators available in natural language datasets, such as quantifiers and aggregations, and contains "reasoning shortcuts", due to a lack of challenging image distractors (Chen et al., 2020b). Other VQA datasets with real images and questions do not include a meaning representation for questions (Suhr et al., 2019; Antol et al., 2015) and thus cannot be easily used to create a compositional split.

In this work, we present COVR (**CO**mpositional **V**isual **R**easoning), a test-bed for visually-grounded compositional generalization with *real* images. We propose a process for automatically generating complex questions over sets of images (Fig. 2), where each example is annotated with the program corresponding to its meaning. We take images annotated with scene graphs (Fig. 2a) from GQA, Visual Genome (Krishna et al., 2016), and imSitu (Yatskar et al., 2016), automatically collect both similar and distracting images for each example and filter incorrect examples due to errors in the source scene graphs (Fig. 2b,c). We then use a template-based grammar to generate a rich set of complex questions that contain multi-step reasoning and higher-order operations on multiple images (2d). To further enhance the quality of the dataset, we manually validate the correctness of development and test examples through crowdsourcing and paraphrase the automatically generated questions into fluent English for a subset of the automatically generated dataset (2e). COVR contains 260k examples based on ∼70k images, with over 10k of the questions manually validated and paraphrased.

Our automatic generation process allows for the easy construction of *compositional data splits*, where models must generalize to new compositions, and is easily extendable with new templates and splits. We explore both the zero-shot setting, where models must generalize to new compositions, and the few-shot setting, where models need to learn new constructs from a small number of examples.

We evaluate state-of-the-art pre-trained models on a wide range of compositional splits, and expose generalization weaknesses in 13 out of the 22 setups, where the *generalization score* we define is low (0%-75%). Moreover, results show it is not trivial to characterize the conditions under which generalization occurs, and we conjecture generalization is harder when it requires that the model learn to combine complex/large structures. We encourage the community to use COVR to further explore compositional splits and investigate visually-grounded compositional generalization.

2 Dataset Creation

The goal of COVR is to facilitate the creation of VQA compositional splits, with questions that require a high degree of compositionality on both the textual and visual input.

Task definition Examples in COVR are $(q, \mathcal{I}, a)$ triples, where q is a complex question, $\mathcal{I}$ is a set of images, and a the expected answer. Unlike most visually-grounded datasets , which contain 1-2 images, each example in COVR contains up to 5 images. This allows us to (a) generate questions with higher-order operators, and (b) detect good distracting images. Also, questions are annotated with programs corresponding to their meaning, which enables creating compositional splits.

High-level overview Fig. 2 provides an overview of the data generation process. Given an image and

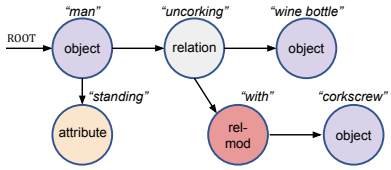

Figure 3: An example subgraph that shows all supported nodes, referring to *"standing man uncorking a wine bottle with a corkscrew"*. The types of the nodes are inside the circle, and their name above it.

its annotated scene graph describing the image objects and their relations, we iterate through a set of subgraphs. For example, in Fig. 2a the subgraph corresponds to *"a man is catching a frisbee and wearing jeans"*. Next, we sample images with related subgraphs: (a) images that contain the same subgraph (Fig. 2b), and (b) images that contain a similar subgraph, to act as distracting images (e.g., images with a man catching a ball or a woman catching a frisbee, Fig. 2c). To ensure the quality of the distracting images, we propose models for automatic filtering and validation (§2.2). Next (Fig. 2d), we instantiate questions from a template-based grammar by filling slots with values computed from the selected subgraphs, and automatically obtain a program for each question. Last, we balance the answers and question types, and use crowdsourcing to manually validate and provide fluent paraphrases for the evaluation set (Fig. 2e).

2.1 Extracting Subgraphs

A scene graph describes objects in an image, their attributes and relations. We use existing datasets with annotated scene graphs, specifically imSitu (Yatskar et al., 2016) and GQA (Hudson and Manning, 2018), which contains a clean version of Visual Genome's human-annotated scene graphs (Krishna et al., 2016). While Visual Genome's scenes have more detailed annotations, imSitu has a more diverse set of relations, such as *"person measuring the length of the wall with a tape"*, which also introduces ternary relations (*"uncorking"*, in Fig. 3).

Given a scene graph, we extract all of its *subgraphs* using the rules described next. A subgraph is a directed graph with the following node types: `object`, `attribute`, `relation` and `rel-mod` (relation modifier), where every node has a name which describes it (e.g. *"wine"*). Fig. 3 shows an example subgraph. A valid subgraph has a root `object` node, and has the following structure: Every `object` has an outgoing edge to ≤ 2 `relation` nodes and ≤ 1 `attribute` node. Every `relation` node has an outgoing edge to exactly one `object` node and optionally multiple `rel-mod` nodes. Every `rel-mod` node has an outgoing edge to exactly one `object` node. The depth of the subgraph is constrained such that every path from the root to a leaf has at most two `relation` nodes. For brevity, we refer to subgraphs with a textual description (*"standing man uncorking a wine bottle with a corkscrew"*).

Type	Example
VERIFYATTR	Is the sink that is below a towel white?
CHOOSEATTR	Is the man that is wearing a jersey running or looking up?
QUERYATTR	What is the color of the cat that is on a black floor?
COMPARECOUNT	There are more coffee tables that are in living room than couches that are in living room
COUNT	How many people are erasing a mark from a paper?
VERIFYCOUNT	There is at most 1 cup that is behinda man that is wearing a jacket
COUNTGROUPBY	How many images contain exactly 2 men that are in water?
VERIFYCOUNT-GROUPBY	There is at least 1 image that contain exactly 2 women that are carrying surfboard
VERIFYLOGIC	Are there both people that are packing a tea into a mason jar and people that are packing a salt into a bag?
VERIFYQUANT	No boys with large trees behind them are wearing jeans
VERIFYQUANT-ATTR	Do all dogs that are on a bed have the same color?
CHOOSEOBJECT	The woman that is wearing dress is carrying a bottle or a purse?
QUERYOBJECT	What is the woman that is wearing glasses holding?
VERIFYSAME-ATTR	Does the pillow that is on a bed and the pillow that is on a couch have the same color?
CHOOSEREL	Is the sitting man holding a hat or wearing it?

Table 1: A list of all question templates with examples. Template slots are underlined.

2.2 Finding Related Images

Given a subgraph g, we pick candidate *context images* that will be part of the example images $\mathcal{I}$. We want to only pick *related* images, i.e., images that share sub-structure with g. Images that contain g in their scene graph will be used to generate counting or quantification questions. Images with different but similar subgraphs to g will be useful as image distractors, which are important to avoid "reasoning shortcuts" (Cirik et al., 2018; Agrawal et al., 2018; Chen et al., 2020a,b; Bitton et al., 2021). For example, for the question in Fig. 2e, it is not necessary to perform all reasoning steps if only one man is showing in all images in $\mathcal{I}$.

To find images with similar subgraphs, We define the *edit distance* between two graphs to be the minimum number of operations required to transform one graph to another, where the only valid operation is to *substitute* a node with another node that has the same type. A good distracting

Type	Preconditions	Example subgraph(s)	Template	Example output
VERIFY-ATTR	(1) g's root has attribute (2) Distracting images have 2 different nodes, one of which is the root's attribute	white sink below towel	Is the G-NOATTRIBUTE, ATTRIBUTE?	Is the sink that is below a towel, white?
CHOOSE-OBJECT	(1) Distracting images have 2 different nodes, one of which is the object	woman lighting a cigar on fire using a candle	G-SUBJECT is REL OBJ or DECOYOBJ?	The woman that is lighting a cigar on fire is lighting it using a candle or a lighter?
COMPARE-COUNT	(no conditions)	woman wearing dark blue jacket, woman wearing white jacket	There are COMPARATIVE G than G2	There are less women that are wearing a dark blue jacket than women that are wearing a white jacket

Table 2: A subset of the question templates with their preconditions and examples for how questions are instantiated. Template slots are shown with a background color. See text for further explanation.

image will (a) contain a subgraph at edit distance 1 or 2, and (b) not contain the subgraph g itself. For the question in Fig. 2e, a good distracting image will contain, for example, *"a man holding a frisbee wearing shorts"* or *"a woman catching a frisbee wearing jeans"*. We extract related images by querying all scene graphs that exhibit the mentioned requirements using a graph-based database.[1]

Filtering overlapping distractors A drawback of the above method is that the edit distance between two subgraphs could be 1, but the two subgraphs might still semantically *overlap*. For example, a subgraph *"woman using phone"* is not a good distractor for *"woman holding phone"* since *"holding"* and *"using"* are not mutually exclusive. A similar issue arises with attributes and objects (e.g. *"man"* and *"person"*, *"resting"* and *"sitting"*).

We thus add a step when sampling distracting images to filter such cases. We define $m(x_1, x_2)$ to be the probability that node x_1 and x_2 are mutually exclusive. For example, for an object x, $m(x, \cdot)$ should return high probabilities for all objects except x, its synonyms, hypernyms, and hyponyms. When computing the edit distance between two subgraphs, we only consider nodes x_1, x_2 such that $m(x_1, x_2) > 0.5$. To learn $m(x, \cdot)$, we fine-tune RoBERTa (Liu et al., 2019) separately on nouns, attributes and relations, with a total of 6,366 manually-annotated word pairs, reaching accuracies of 94.4%, 95.7% and 82.5% respectively. See App. A for details.

Incomplete scene graphs A good distracting image should not contain the source subgraph g. However, because scene graphs are often incomplete, naïve selection from scene graphs can yield a high error rate. To mitigate this issue, we fine-tune LXMERT (Tan and Bansal, 2019), a pre-trained multi-modal classifier, to recognize if a given subgraph is in an image or not. We do not assume that this classifier will be able to comprehend complex subgraphs, and only use it to recognize *simple* subgraphs, that contain ≤ 2 `object` nodes, and up to a single `relation` and `attribute`. We train the model on binary-labeled image-subgraph pairs, where we sample a simple subgraph g_s and its image I from the set of all subgraphs, and use I as a positive example, and an image I' that contains a subgraph g'_s as a negative example, where I' is a distracting image for I, according to the procedure described above. For example, for the subgraph *"man wearing jeans"* the model will be trained to predict 'True' for an image that contains it, and 'False' for an image with *"man wearing shorts"*, but not *"man with jeans"*.

After training, we filter out candidate distracting images for the subgraph g if the model outputs a score above a certain threshold τ for *all* of the simple graphs in g. We adjust τ such that the probability of correctly identifying missing subgraphs is above 95% according to a manually-annotated set of 441 examples. See App. B for details.

2.3 Template-based Question Generation

Once we have a subgraph, an image, and a set of related images, we can generate questions. For a subgraph g and its related images, we generate questions from a set of 15 manually-written templates (Table 1), that include operators such as quantifiers, counting, and "group by". Further extending this list is trivial in our framework. Each template contains slots, filled with values conditioned on the subgraph g as described below. Since we have 15 templates with multiple slots, and each slot can be filled with different types of subgraphs, we get a large set of possible question types and programs.

We define a set of *preconditions* for each template (Table 2) which specifies (a) the types of subgraphs that can instantiate each slot, and (b) the images that can be used as distractors. The

[1] https://neo4j.com/

Measurement	Train	Test
# total questions	250.0k	10.6k
# unique questions	170.8k	8.7k
# unique answers	3972	1109
# unique images	69.6k	7.0k
# unique anonymized programs	338	177
# True/False (T/F) questions	133.3k	5.5k
# "X or Y" questions	50.0k	1.9k
# "how many ..." questions	33.3k	1.5k
# Open questions	33.3k	1.6k
avg. # question words (a/p)	14.3	13.5/11.8
avg. # images per question	3.8	3.4

Table 3: Statistics for COVR for both the training and test sets (development and test combined). a/p stands for automatically-generated vs paraphrased.

first precondition type ensures that the template is supported by the subgraph—e.g., to instantiate a question that verifies an attribute (VERIFYATTR), the root object must have an outgoing edge to an `attribute`. The second precondition type ensures that we have relevant distractors. For example, for VERIFYATTR, we only consider distracting subgraphs if the attribute connected to their root is *different* from the attribute of g's root. In addition, the distracting subgraphs must have at least one more different node. Otherwise, we cannot refer to the object unambiguously: if we have a *"white dog"* as a distracting image for a *"black dog"*, the question *"Is the dog black?"* will be ambiguous.

When a template, subgraph, and set of images satisfy the preconditions, we instantiate a question by filling the template slots. There are three types of slots: First, slots with the description of g or a subset of its nodes. E.g., in VERIFYATTR (Table 2) we fill the slot G-NOATTRIBUTE with the description of g without the attribute node (*"sink that is below a towel"*), and the slot ATTRIBUTE with the name of that attribute (*"white"*). In CHOOSEOBJECT (second row), we fill the slots G-SUBJECT, REL and OBJ with different subsets of g: *"woman lighting a cigar on fire"*, *"using"* and *"candle"*.

The second slot type fills the description of a *different* subgraph than g, or a subset of its nodes. In COMPARECOUNT (third row), we fill G2 with the description of another subgraph, sampled from the distracting images. Similarly, in CHOOSEOBJECT, we fill DECOYOBJ with the node *"lighter"*. Last, some slots are filled from a closed set of words that describe reasoning operations: In COMPARECOUNT, we fill COMPARATIVE with one of *"less"*, *"more"* or *"same number"*.

Once slots are filled, we compute the corresponding program for the question. The list of program operators and a sample program are in App. F.

2.4 Quality Assurance and Analysis

We perform the generation process separately on the training and validation set of GQA and imSitu graph scenes, which yields 13 million and 1 million questions respectively. We split the latter set into *development* and *test* sets of equal size, making sure that no two examples with the same question text appear in both of them.

Balancing the dataset Since we generate a question for every valid combination of subgraph and template, the resulting set of questions possibly contains correlations between the language of the question and the answer, and has skewed distributions over answers, templates, and subgraph structures. To overcome the first issue, for all questions where it is possible we create two examples with the same question, but a different set of images and a different answer (77,072 questions appear with at least two different answers in the training set) Then, to balance our dataset, we use a heuristic procedure that leads to a uniform distribution over templates, and balances the distribution over answers and over the size and depth of subgraphs (App. C). We provide statistics about our dataset in Table 3 and the answer distribution in App. D. Overall, COVR contains ∼250,000 training, 5,345 development and 5,253 test examples.

Manual Validation Examples thus far are generated automatically. We manually evaluated examples and found that 80% are correct. The most common issue is incorrect count answers due to missing scene graphs annotations (e.g., only 3 trees annotated when there are more). To increase the evaluation set quality, we validate all test examples through crowdsourcing. Workers get a question, images and an answer, and reject it if they find it invalid. We ask workers to reject questions if they are uncertain, leading to a 29% rejection rate.

Paraphrasing Since questions are generated automatically, we ask workers to paraphrase all test questions into fluent English, while maintaining their meaning. Examples are given in App. E.

3 Compositional Splits

Our generation process lets us create questions and corresponding programs with a variety of reasoning operators. We now how COVR can be used to generate challenging compositional splits. We propose

Test name	Training	Generalization
HAS-QUANT-COMP & HAS-QUANT-ALL	No man that is next to a horse is standing All birds are black	All computer mice that are on a mouse pad are black
HAS-COUNT & HAS-ATTR	There are 3 dogs What is the black dog holding?	There are 3 black dogs
HAS-COUNT & PP/V	The horse is pulling people with a rope or a leash? There are two people next to a tree.	There are two children cleaning the path with a broom
LEXICAL-X/LEXICAL-Y (Lexical Split)	All men wear jeans. All women are standing.	No men wearing jeans are standing.
HAS-SAMEATTR-COLOR	What is the color of X? Do all X have the same material?	Do all X have the same color?
TPL-CHOOSEOBJECT	What is the man carrying? Is the man X or Y?	The man is carrying a X or Y?
TPL-VERIFYQUANTATTR	Does the sitting dog and the standing cat have the same color? All dogs are standing	Do all dogs have the same color?
TPL-VERIFYATTR	Is the table that is under donuts dark or tan?	Is the towel that is on a floor pink?
TPL-VERIFYCOUNT ∪ TPL-VERIFYCOUNTGROUPBY	How many images contain at least 2 men that are in water?	There are at least 2 images that contain exactly 2 blankets that are on bed

Table 4: List of the zero-shot compositional splits. Top half shows splits where we hold out examples where two properties co-occur, bottom half shows splits where we hold out questions with a single property or a union of two (see text). Background colors highlight different reasoning steps that the model is trained or tested on.

Property	Description
HAS-X	True if p contains an operator of type X, X $\in$ { QUANTIFIER (QUANT), COMPARATIVE (COMPAR), GROUPBY (GROUP), NUMBER (NUM), ATTRIBUTE (ATTR), SAMEATTR}.
HAS-X-Y	Same as HAS-X, where Y is a specific instance of X (e.g., ALL if X is QUANTIFIER).
RM/V	True if g's structure contains either a `rel-mod` node or an `object` node with two outgoing edges to `relation` nodes (V-shape).
TPL-X	True if the question originated from the template X.
ANS-X	True if answer is of type X $\in$ { NUM, ATTR, NOUN}
LEXICAL-X	True if g contains a node with the name X.

Table 5: List of the types of properties given a program p and subgraph g on which the question was based on.

two setups: (1) *Zero-shot*, where training questions provide examples for all required reasoning steps, but the test questions require a new *composition* of these reasoning steps, and (2) *Few-shot*, where the model only sees a small number of examples for a specific reasoning type.

Because each question is annotated with its program, we can define binary properties over the program and answer, where a property is a binary predicate that typically defines a reasoning type in the program. For example, the property HAS-QUANT is true iff the program contains a quantifier, and HAS-QUANT-NONE iff it contains the quantifier NONE. We can create any compositional split that is a function of such properties. We list the types of properties used in this work in Table 5.

All compositional splits are based on the original training/validation splits from Visual Genome and imSitu to guarantee that no image appears in both the training set and any of the test sets. Splits are created simply by removing certain questions from the train and test splits. If we do not remove any question, we get an i.i.d split.

Zero-shot We test if a model can answer questions where two properties co-occur, when during training it has only seen questions that have at most one of these properties. For a pair of properties, we filter from the training set all examples that have *both* properties, and keep only such examples for the evaluation set.For example, the split in the first row of Table 4 (top) shows a split of the two properties HAS-QUANT-COMP and HAS-QUANT-ALL.

The zero-shot setup can also be used to test examples with a single property that is unseen during training (Table 4, bottom), or a *union* of two properties, assuming that the model has seen examples for all the reasoning steps that the unseen property requires. For example, in TPL-CHOOSEOBJECT we test on a template that is entirely unseen during training, since we have the templates CHOOSEATTR and VERIFYOBJECT.

Another popular zero-shot test is the *program split*. In a similar fashion to the compositional generalization tests in Finegan-Dollak et al. (2018), we randomly split programs after anonymizing the names of nodes, and hold out 20% of the programs to test how models perform on program structures that were not seen during training. We also perform a *lexical split*, where we hold out randomly selected pairs of node names (i.e., names of objects, relations or attributes) such that the model never sees this pair together in the same program during training. We hold out 20% of all pairs.

Few-shot In this setup, we test if a model can handle examples with a given property, when it has only seen a small number M of examples with this property during training. For a given property, we create this split by filtering from the original training set all examples that have this property,

Model	COVR	COVR-PARAPH.
MAJ	27.3	27.3
MAJTEMPL	42.4	42.4
LXM_{TEXT}	46.6	41.0
LXM_{iid}	72.0	62.7
$\text{LXM}_{\text{RANDNEG}}$	58.8	56.2
Humans	-	93.1

Table 6: Results on both the generated and paraphrased versions of the test-unseen set in the i.i.d. split.

except for M examples. From the evaluation set, we keep only examples that have this property.

4 Experiments

Experimental Setup We consider the following baselines: (a) MAJ, the majority answer in the training set, and (b) MAJTEMPL: an oracle-based baseline that assumes perfect knowledge of the template from which the question was generated, and predicts the majority answer for that template. For templates that include two possible answers (*"candle or lighter"*), it randomly picks one.

Our main model is LXMERT (Tan and Bansal, 2019), a pre-trained vision-and-language model which provides a representation for question-image pairs, (q, I). We modify LXMERT to accept N images by running LXMERT with (q, I) as input for each image $I \in \mathcal{I}$, and then passing the N LXMERT-computed representations through two transformer layers with a concatenated `[CLS]` token. We pass the `[CLS]` token representation through a classifier layer to predict the answer. The classifier layer and the added transformer layers are randomly initialized, and fine-tuned, along with LXMERT, on COVR. In addition, we evaluate a text-only baseline LXM_{TEXT} that only sees the input text (image representations are zeroed out) to estimate a lower bound on performance without any reasoning on images.

For the compositional splits, we evaluate LXMERT trained on the entire data (LXM_{iid}), the text baseline (LXM_{TEXT}), and the compositionally-trained models LXMERT_{500}, LXMERT_{0} for the few-shot (M=500) and zero-shot setups, respectively. To control the training size, we also evaluate $\text{LXM}_{\text{iid-size}}$, a model trained with a similar data size as the compositionally-trained model, by uniformly downsampling the training set. All models are evaluated on the same subset of the test-unseen compositional split. To focus on the generalization gap, we define a generalization score ("Gen. Score") that measures the proportion of the gap between LXM_{TEXT} and our upper-bound, $\text{LXMERT}_{\text{iid–size}}$, that is closed by a model. In all compositional splits, we early-stop using the subset of the development set that does not contain any of the compositional properties we test on (Teney et al., 2020).

Results First, we show how models perform on paraphrased and automatically-generated questions in the i.i.d setup in Table 6. The difference between LXM_{TEXT} and MAJTEMPL is small (4.2%), suggesting that the answer to most questions cannot be inferred without looking at the images. We also show that when the model is trained with random images instead of distracting ones ($\text{LXM}_{\text{RANDNEG}}$), accuracy drops by 13.2%, showing the importance of training on good distracting images. In addition, there is still a large gap from human performance, at 93.1%, which we measure by evaluating human answers on 160 questions. Finally, we observe a 9.3% performance drop when training on the automatically-generated examples and testing on the paraphrased examples. Accuracy per template is shown in App. G.

Next, we report results on the compositional splits. We show results on automatically-generated questions (not paraphrased), to disentangle the effect of compositional generalization from transfer to natural language. App. H reports results for the paraphrased test set, where generalization scores are lower, showing that transfer to natural language makes compositional generalization even harder.

Table 7 shows results in the **few-shot** setup, where in 7 out of 12 setups the generalization score is lower than 75. LXMERT generalizes better in cases where the withheld operator is similar to an operator that appears in the training set. For instance, HAS-QUANT-ALL generalizes better than HAS-QUANT since it sees many examples with the quantifiers *"some"* and *"none"*, HAS-COMPAR-MORE generalizes better than HAS-COMPAR, and HAS-LOGIC-AND has a relatively high generalization score of 76%. This suggests that when the model has some representation for a reasoning type it can generalize better to new instances of it.

The large difference between the nearly-perfect score of HAS-NUM-3 (97%), and the low score of HAS-NUM-3-ANS-3 (34%), where in both the number 3 is rarely seen in the question, and in the latter it is also rare as an answer, suggests that the model learns good number representations just from seeing numbers in the answers. Other cases where the generalization scores are low are HAS-

Split	Filtered	LXM_{TEXT}	LXM_{500}	Gen. Score	$\text{LXM}_{\text{iid-size}}$	LXM_{iid}
HAS-QUANT	33.3k	56.0	65.1		78.5	80.6
HAS-QUANT-ALL	21.3k	55.8	68.4		76.3	78.3
HAS-QUANT-COMP	30.5k	55.5	70.4		78.4	80.8
HAS-COMPAR	16.7k	51.5	52.9		72.3	72.6
HAS-COMPAR-MORE	7.7k	51.4	68.9		75.0	75.0
HAS-GROUPBY	33.3k	48.8	68.7		74.3	75.9
HAS-LOGIC	16.7k	51.9	78.7		83.8	82.2
HAS-LOGIC-AND	9.6k	51.5	76.3		84.0	81.3
HAS-NUM-3	7.8k	53.8	77.9		78.6	80.7
HAS-NUM-3-ANS-3	11.8k	39.0	51.1		74.5	75.3
HAS-RM/V	42.7k	39.0	60.1		70.2	72.6
ANS-NUM	33.3k	34.5	41.3		68.5	69.5

Table 7: Non-paraphrased test-unseen results in the few-shot setup. 'Filtered' shows the number of examples that were filtered out of the training set in each split.

Split	Filtered	LXM_{TEXT}	LXM_{0}	Gen. Score	$\text{LXM}_{\text{iid-size}}$	LXM_{iid}
HAS-QUANT-COMP & HAS-QUANT-ALL	20.5k	55.2	47.0		76.4	78.6
HAS-COUNT & HAS-ATTR	24.3k	43.2	69.1		69.1	71.0
HAS-COUNT & RM/V	19.4k	34.9	74.7		78.9	80.9
HAS-SAMEATTR-COLOR	29.6k	53.6	75.5		75.5	77.1
TPL-CHOOSEOBJECT	16.7k	47.3	2.0		63.8	71.7
TPL-VERIFYQUANTATTR	16.7k	55.3	49.1		78.1	78.7
TPL-VERIFYATTR	16.7k	61.7	42.8		74.9	79.2
TPL-VERIFYCOUNT ∪ TPL-VERIFYCOUNTGROUPBY	33.3k	49.9	52.3		84.5	85.7
Program Split	41k ± 5k	47.0 ± 4.2	63.2 ± 7.8		72.2 ± 1.8	74.4 ± 1.8
Lexical Split	57k ± 3k	49.6 ± 1.4	73.6 ± 1.1		73.9 ± 0.0	75.4 ± 0.4

Table 8: Non-paraphrased test-unseen results in the zero-shot setup. Filtered shows number of examples that were filtered out of the training set. A red rectangle under "Gen. Score" illustrates that LXM_0 is lower than LXM_{TEXT}. '&' indicates holding out the intersection of two sets of questions, '∪' indicates holding out the union of the two.

QUANT, where quantifiers appear in only 500 examples, and HAS-COMPAR, where comparatives appear in only 500 examples. Fig. 5 in App. I shows performance on the development set as M, the number of held-out examples the model sees, increases. We observe model performance is much lower when $M = 125$, and improves rapidly as M increases. This shows that models can acquire new skills rapidly from hundreds of examples, but not from a handful of examples, like humans.

Table 8 shows results for the **zero-shot** setup. A model that sees examples for a compound in the scope of the quantifier, but never in the context of the quantifier ALL, fails to generalize (HAS-QUANT-COMP & HAS-QUANT-ALL, score is lower than the text baseline). The model also fails to generalize to the template CHOOSEOBJECT, although it saw at training time the necessary parts in the templates CHOOSEATTR and VERIFYOBJECT. Similarly, the model fails to generalize to the template VERIFYQUANTATTRIBUTE although it saw examples for comparing object attributes and with quantifiers, and to TPL-VERIFYCOUNT ∪ TPL-VERIFYCOUNTGROUPBY, where we hold out all verification questions with counting, even though the model sees verification questions and counting in other templates. Last, the model struggles to generalize in the program split.

Conversely, the model generalizes well to counting questions where the subgraph contains a `rel-mod` node, a v-shape structure (HAS-COUNT & RM/V) or an attribute (HAS-COUNT & HAS-ATTR), and in the lexical split, where the model is tested on unseen combinations of names of nodes.

A possible explanation for the above is that compositional generalization is harder when the model needs to learn to combine large/complex structures, and does well when composing more atomic constructs. However, further characterizing the conditions under which compositional generalization occurs is an important question for future work.

5 Conclusion

We present COVR, a test-bed for visually-grounded compositional generalization with real images. COVR is created automatically except for manual validation and paraphrasing, and allows us to create a suite of compositional splits. COVR can be easily extended with new templates and splits to encourage the community to further understand compositional generalization. Through COVR, we expose a wide range of cases where models struggle to compositionally generalize.

References

Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. 2018. Don't just assume; look and answer: Overcoming priors for visual question answering. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4971–4980. IEEE Computer Society.

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. 2018. Bottom-up and top-down attention for image captioning and visual question answering. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6077–6086.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. 2015. VQA: Visual Question Answering. In *International Conference on Computer Vision (ICCV)*.

Dzmitry Bahdanau, Harm de Vries, Timothy J. O'Donnell, Shikhar Murty, Philippe Beaudoin, Yoshua Bengio, and Aaron C. Courville. 2019a. CLOSURE: assessing systematic generalization of CLEVR models. *CoRR*, abs/1912.05783.

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. 2019b. Systematic generalization: What is required and can it be learned? In *International Conference on Learning Representations*.

Yonatan Bitton, Gabriel Stanovsky, Roy Schwartz, and Michael Elhadad. 2021. Automatic generation of contrast sets from scene graphs: Probing the compositional consistency of GQA. *CoRR*, abs/2103.09591.

Long Chen, Xin Yan, Jun Xiao, Hanwang Zhang, Shiliang Pu, and Yueting Zhuang. 2020a. Counterfactual samples synthesizing for robust visual question answering. In *CVPR*.

Zhenfang Chen, Peng Wang, Lin Ma, Kwan-Yee K. Wong, and Qi Wu. 2020b. Cops-ref: A new dataset and task on compositional referring expression comprehension.

Noam Chomsky. 1957. *Syntactic Structures*. Mouton.

Volkan Cirik, Louis-Philippe Morency, and Taylor Berg-Kirkpatrick. 2018. Visual referring expression recognition: What do systems actually learn? In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 781–787, New Orleans, Louisiana. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Jerry A. Fodor and Zenon W. Pylyshyn. 1988. *Connectionism and Cognitive Architecture: A Critical Analysis*, page 3–71. MIT Press, Cambridge, MA, USA.

Drew Arad Hudson and Christopher D. Manning. 2018. Compositional attention networks for machine reasoning. In *International Conference on Learning Representations*.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*.

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Online. Association for Computational Linguistics.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. 2016. Visual genome: Connecting language and vision using crowdsourced dense image annotations.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *ICML*.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel Gershman. 2018. Building machines that learn and think like people. *The Behavioral and brain sciences*, 40:e253.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Richard Montague. 1970. Universal grammar. *Theoria*, 36(3):373–398.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M. Lake. 2020. A benchmark for systematic generalization in grounded language understanding. *CoRR*, abs/2003.05161.

Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. 2019. A corpus for

reasoning about natural language grounded in photographs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6418–6428, Florence, Italy. Association for Computational Linguistics.

Hao Tan and Mohit Bansal. 2019. LXMERT: Learning cross-modality encoder representations from transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5100–5111, Hong Kong, China. Association for Computational Linguistics.

Damien Teney, Kushal Kafle, Robik Shrestha, Ehsan Abbasnejad, Christopher Kanan, and Anton van den Hengel. 2020. On the value of out-of-distribution testing: An example of goodhart's law. *CoRR*, abs/2005.09241.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Mark Yatskar, Luke Zettlemoyer, and Ali Farhadi. 2016. Situation recognition: Visual semantic role labeling for image understanding. In *Conference on Computer Vision and Pattern Recognition*.

A Filtering Overlapping Distracors

We take the published RoBERTa (Large, Liu et al. 2019) model that is already fine-tuned on MNLI (Williams et al., 2018), and further fine-tune it separately on pairs of nouns, attributes and relations to predict whether a pair of words or phrases are mutually exclusive. To leverage the knowledge learned during pre-training, we use the same setup as the training on MNLI, where the model is given two phrases and predicts one of three classes: "contradiction", "entailment" and "neutral". We train all models for 50 epochs with a learning rate of $3e^{-5}$.

For the **nouns** models, we use 2,366 manually annotated pairs of nouns for training and validation. The model is trained to predict "contradiction" whenever nouns are mutually-exclusive, i.e. when none of the words is a synonym, hypernym, or hyponym of the other, and "neutral" otherwise (we do not use the entailment class). We randomly shuffle the order of pairs for regularization. We get an accuracy score of 94.4% on 20% of the pairs which were held-out for validation. Simialrly we train a model that predicts mutual-exclusiveness of **attributes** over 3,053 pairs, and get an accuracy of 95.7%.

Unlike the other two models, for the **relations** model we do not require complete mutual-exclusiveness, and we do not assume symmetrical annotations, i.e., that if $m(x_1, x_2)$ then $m(x_2, x_1)$, to increase the probability of finding pairs where m returns a score higher than 0.5 for a relation x. For example, we annotate pairs such that $m($*"riding on"*, *"near"*$) = 1$ but $m($*"near"*, *"'riding on'*$) = 0$, since *most often*, if some object is hanging on another object, the annotation of the relations between the two objects in Visual Genmoe will be specific, i.e. *"riding on"* or *"on"* and not *"near"*. This way, for a question such as *"Is the man riding a motorcycle"* we might get distracting images with a man *"standing near"* a motorcycle, but for a question such as *"Is the man near a motorcycle"* we will not get distracting image with a man *"riding"* a motorcycle, as then the question will be ambiguous. Note that while this can potentially introduce some noise (i.e., in some rare cases *"a man riding a motorcycle"* might be annotated as if the man is *"near"* a motorcycle), such mistakes can be overridden with the second validation that we use (incomplete scene graphs, App B). We annotate 917 pairs of relations, where every pair is annotated in both directions. We get an accuracy of 82.5% on the held-out set.

B Incomplete Scene Graphs

We use LXMERT (Tan and Bansal, 2019) to train a classifier that predicts whether a simple subgraph exists in an image. See §2.2 for details on the data we train on. We extract image and objects features with the bottom-up top-down attention method of Anderson et al. (2018) similarly to LXMERT, and fine-tune the pre-trained model. To extract the training data, we use all subgraphs from all images for which we have at least one valid negative image (from both the training and test sets). This results in 6,520,367 positive and negative examples. Since we need the model to predict results not only on the test set, but also on the training set, we split all examples (training and test) into 5 splits based on their image, and train 5 different models, where each model does not see a different fifth of the images during training. Then, to predict whether a simple subgraph exists in an image, we use the model that was not trained on that image.

We manually annotate 441 examples where we determine if a simple subgraph exists in an image and use these annotations for early stopping and to adjust a threshold τ. We use this threshold to filter out candidate distracting images for a subgraph g if the model outputs a score above a certain threshold τ for *all* of the simple graphs in g. Note that each negative example is a candidate distracting image to some subgraph g. We use g to further adjust τ in the following way. By definition, a candidate simple graph of a distracting image has a non-empty set d of nodes that are different than g. Based on our annotated examples, we found that the model should have a different threshold τ for different *types* of nodes in d. Specifically, we found that the model performed best when d contained nodes of type `object`, then `relation`, and finally `attribute`. Thus, we use a different τ for each type: $\tau = 0.05$ for `object`, $\tau = 0.1$ for `relation` and $\tau = 0.5$ for `attribute`. If there are more than one type of nodes in d, we take the one that gives the maximal τ.

C Downsampling & Balancing

We use the following downsampling method to balance the dataset and reduce bias as much as possible, separately for the training, development and test sets. In high-level, we start with a total of N

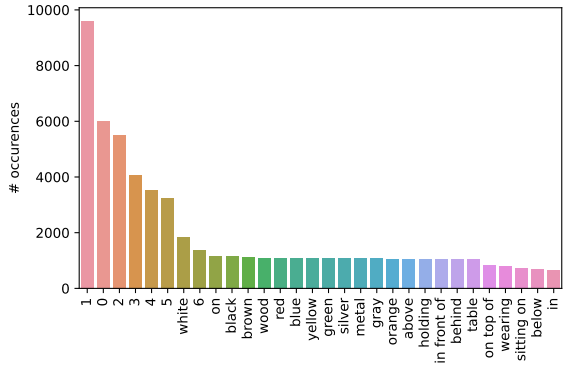

Figure 4: Distribution of the top-30 answers in the training set (excluding 'True'/'False').

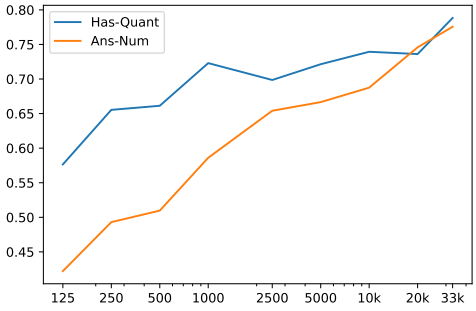

Figure 5: Effect of M on development set accuracy on two compositional splits from 125 examples and up to the maximal amount of training examples for the split. Log scale is used for the X-axis.

questions and group them by their templates, such that we have T groups. We then use a heuristic ordering method that prioritizes or balances different desired features, described next, and finally we take the top $S = \frac{N}{T}$ questions from each group, such that we get an equal number of questions per template. The ordering method is defined as follows, starting with an empty list L_t for a template t. Each question is automatically annotated with the following three features: (1) whether this question appears at least twice with different answers, (2) the answer to the question and (3) the structure of the source subgraph for that question, specifically a tuple with its size and its depth. We first add to t all questions where the first feature is positive (in all cases this was less than S). Then, to balance between the different question answers, at each step until $|L_t| = S$, we count the appearances of all answers and sample an answer a from the answers that appeared least in L_t. Then, we count the appearances of all subgraph structures, and sample a question with answer a, such that its subgraph structure appeared least. We stop once $|L_t| = S$.

D Answers Distribution

We show the distribution of the top answers of the training set in Fig. 4.

E Paraphrasing Samples

We show examples for crowd-sourced paraphrasing in Table 9.

F Programs

We list all program operators in Table 10, and a sample program in Table 11.

G Accuracy per Template

Tables 12 and 13 show the accuracy for each template for both the non-paraphrased and the paraphrased versions.

H Compositional Results on COVR-PARAPH.

We report results on the compositional splits when they are evaluated on the *paraphrased* questions in Tables 14 and 15.

I Effect of M

We show how M, the number of examples the model sees from the compositional subset, affects the accuracy in Figure 5.

Auto-generated question	Rephrase
Does the trees that are behind a zebra and the trees that are behind a fire hydrant have the same color?	Are the trees behind a zebra the same color as those behind a fire hydrant?
There is 1 bottle that is on bench that is in front of tree	Is there a bottle on a bench in front of a tree?
No forks that are on a white plate are silver	None of the forks on a white plate are silver.
Do all boats that are in a harbor have the same color?	Are all the boats in the harbor the same color?
Is the person that is wearing a yellow jacket skiing?	Is the person in the yellow jacket skiing?
How many images with mushrooms that are on a pizza that is on a table?	The pizza on the table - how many mushrooms are on it?
Is there either a girl that is holding a bouquet and is wearing dress or a girl that is holding a book and is wearing a hat?	Is there either a girl holding a bouquet and wearing a dress, or a girl holding a book and wearing a hat?
There are less boats that are on water than surfboards that are on water	There are fewer boats on water than surfboards.
There are at least 4 people that are buttering pan	There are four or more people buttering a pan.
What is the material of the table that is under a coffee mug?	What material is the table under the coffee mug?

Table 9: Examples for crowd-sourced paraphrasing.

Operator	**Input**	**Output**
`All`	(1) objects, (2) subprogram	Returns 'True' iff 'subprogram' returns 'True' for *all* 'objects'.
`Some`	(1) objects, (2) subprogram	Returns 'True' iff 'subprogram' returns 'True' for *any* of the 'objects'.
`None`	(1) objects, (2) subprogram	Returns 'True' iff 'subprogram' returns 'True' for *none* of the objects.
`QueryName`	(1) object	Returns the name of 'object'
`Find`	(1) name	Returns all objects from all scenes that are named 'name'.
`Filter`	(1) objects (2) attribute_value	Returns only objects in 'objects' that have 'attribute_value'.
`Count`	(1) objects	Returns the size of 'objects'
`Or`	(1) bool1 (2) bool2	Returns 'bool1' OR 'bool2'
`And`	(1) bool1 (2) bool2	Returns 'bool1' AND 'bool2'
`eq`	(1) number1 (2) number2	Returns 'True' iff number1 == number2
`gt`	(1) number1 (2) number2	Returns 'True' iff number1 > number2
`lt`	(1) number1 (2) number2	Returns 'True' iff number1 < number2
`geq`	(1) number1 (2) number2	Returns 'True' iff number1 $\geq$ number2
`leq`	(1) number1 (2) number2	Returns 'True' iff number1 $\leq$ number2
`Unique`	(1) objects	Assumes 'objects' contain a single object. Returns the object in the list.
`UniqueImages`	(1) objects	Returns a set (without duplicates) of all images of the given 'objects'.
`GroupByImages`	(1) objects	Returns (image, objects_in_image) tuples where all object in 'objects' that are in the same image are grouped together and coupled with that image.
`KeepIfValuesCountEq/ KeepIfValuesCountGt/ KeepIfValuesCountLt`	(1) (key, list) tuples (2) size	Returns only tuples where the size of 'list' is equal/greater than/less than 'size'.
`QueryAttribute`	(1) object (2) attribute_name	Returns the attribute value (e.g., *"red"*) of the 'attribute_name' (e.g., *"color"*) of 'object'.
`VerifyAttribute`	(1) object (2) attribute_value	Returns "True" iff 'object' has the attribute 'attribute_value'.
`WithRelation`	(1) objects1, (2) objects2 (3) relation	Returns all objects from 'objects1' that have the relation 'relation' with any of the objects in 'objects2.
`WithRelationObject`	(1) objects1, (2) objects2 (3) relation	Same as `WithRelation`, except it returns objects from 'objects2'.

Table 10: All program operators.

Index	Operator	Arguments
1	`Find`	*"table"*
2	`Filter`	↱ 1, *"wood"*
3	`Find`	*"book"*
4	`WithRelation`	↱ 3, ↱ 2, *"on"*
5	`GroupByImages`	↱ 4
6	`KeepIfValuesCountEq`	↱ 5, 2
7	`Count`	↱ 6

Table 11: The program for the question *"How many images contain exactly 2 books that are on wood table?"*. The symbol ↱ with a row index next to it indicates that it is replaced with the output of the operator of the row in that index.

Template	**LXM$_{\text{TEXT}}$**	**LXM$_{\text{iid}}$**
VERIFYATTR	61.7	79.2
CHOOSEATTR	65.4	73.4
QUERYATTR	39.1	65.7
COMPARECOUNT	51.0	72.1
COUNT	22.7	70.4
VERIFYCOUNT	50.2	85.0
COUNTGROUPBY	48.7	86.7
VERIFYCOUNT-GROUPBY	48.6	67.2
VERIFYLOGIC	51.9	82.2
VERIFYQUANTIFIER	56.9	82.3
VERIFYQUANTIFIER-ATTR	55.3	78.7
CHOOSEOBJECT	47.3	71.7
QUERYOBJECT	8.7	28.5
VERIFYSAMEATTR	51.2	75.4
CHOOSEREL	58.7	63.6

Table 12: Accuracy score per template (i.i.d setup) on COVR automatically-generated questions, test set.

Template	**LXM$_{\text{TEXT}}$**	**LXM$_{\text{iid}}$**
VERIFYATTR	52.3	71.2
CHOOSEATTR	63.5	67.0
QUERYATTR	30.8	61.3
COMPARECOUNT	50.0	56.1
COUNT	21.3	59.5
VERIFYCOUNT	48.1	77.0
COUNTGROUPBY	40.2	62.7
VERIFYCOUNT-GROUPBY	43.9	79.5
VERIFYLOGIC	47.1	75.2
VERIFYQUANTIFIER	54.8	72.1
VERIFYQUANTIFIER-ATTR	53.0	69.5
CHOOSEOBJECT	25.7	52.3
QUERYOBJECT	6.2	24.0
VERIFYSAMEATTR	45.9	62.0
CHOOSEREL	50.4	53.7

Table 13: Accuracy score per template (i.i.d setup) on COVR (paraphrased), test set.

Split	Filtered	LXM_{TEXT}	LXM_{500}	Gen. Score	$LXM_{iid\text{-}size}$	LXM_{iid}
HAS-QUANT	33.3k	53.9	53.6		70.7	70.9
HAS-QUANT-ALL	21.3k	48.2	48.2		71.9	68.2
HAS-QUANT-COMP	30.5k	54.8	52.5		69.6	69.8
HAS-COMPAR	16.7k	50.5	48.0		52.4	56.4
HAS-COMPAR-MORE	7.7k	50.3	58.5		59.9	65.3
HAS-AGGREGATE	33.3k	41.9	59.7		68.3	70.2
HAS-LOGIC	16.7k	47.1	70.2		79.1	75.2
HAS-LOGIC-AND	9.6k	48.2	74.9		77.7	76.9
HAS-NUM-3	7.8k	44.8	64.1		71.0	73.8
HAS-NUM-3-ANS-3	11.8k	31.3	47.0		60.0	61.7
HAS-RM/V	42.7k	31.6	49.2		58.5	62.3
ANS-NUM	33.3k	30.0	32.5		57.2	60.0

Table 14: Same splits and experiments as in Table 7, evaluated on the paraphrased questions.

Split	Filtered	LXM_{TEXT}	LXM_0	Gen. Score	$LXM_{iid\text{-}size}$	LXM_{iid}
HAS-QUANT-COMP+ HAS-QUANT-ALL	20.5k	53.3	44.0		72.5	69.1
HAS-COUNT+HAS-ATTR	24.3k	42.1	57.9		58.2	58.8
HAS-COUNT & RM/V	19.4k	29.8	59.4		64.6	69.3
HAS-SAMEATTR-COLOR	29.6k	50.0	60.5		68.7	67.3
TPL-CHOOSEOBJECT	16.7k	25.7	0.7		44.4	52.3
TPL-VERIFYQUANTATTR	16.7k	53.0	45.3		75.7	69.5
TPL-VERIFYATTR	16.7k	52.3	55.6		67.5	71.2
TPL-VERIFYCOUNT ∪ TPL-VERIFYCOUNTGROUPBY	33.3k	46.6	48.9		75.0	78.0
Program Split	41k ± 5k	47.3 ± 2. 2	58.6 ± 10.8		69.6 ± 0.7	70.0 ± 1.0
Lexical Split	57k ± 3k	42.8 ± 0.4	63.3 ± 1.7		63.6 ± 1.4	65.2 ± 1.5

Table 15: Same splits and experiments as in Table 8, evaluated on the paraphrased questions.