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

Ben Bogin[1]   **Shivanshu Gupta**[2]   **Matt Gardner**[3]   **Jonathan Berant**[1,3]
[1]Tel-Aviv University   [2]University of California Irvine   [3]Allen Institute for AI
{ben.bogin,joberant}@cs.tau.ac.il, shivag5@uci.edu,
mattg@allenai.org

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).
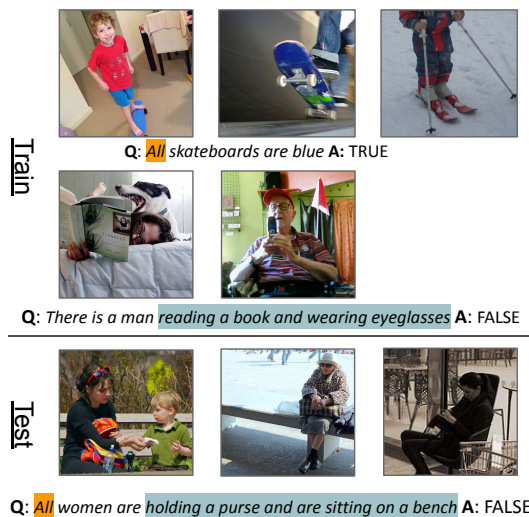

Figure 1: Compositional generalization in the VQA setup. COVR enables the creation of compositional splits such as the one depicted here, where quantification appears with conjunction only in the test set.

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 typically use synthetic images and contain a limited number of visual concepts and reasoning operators (Bahdanau et al., 2019a,b; Ruis
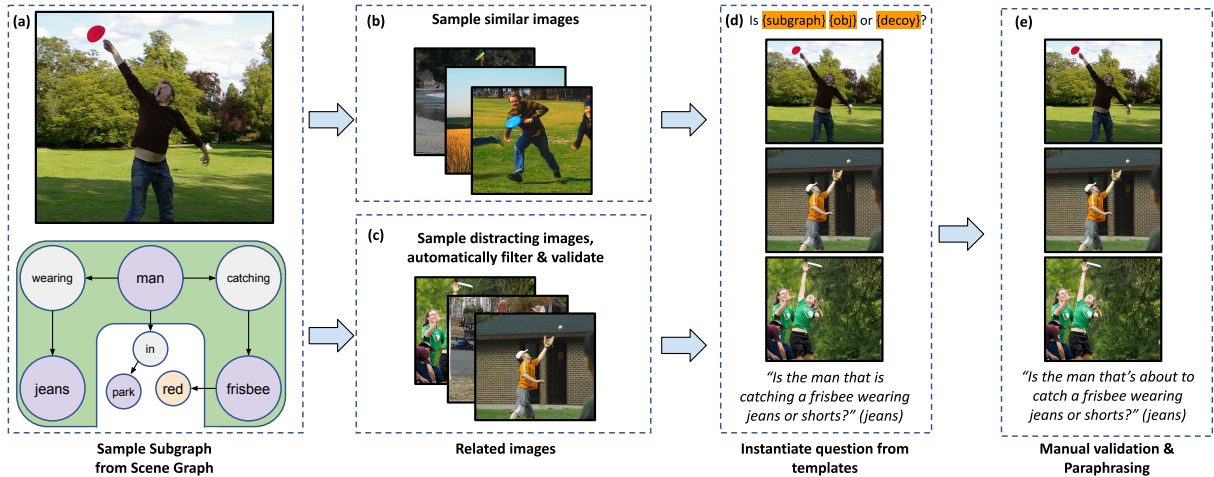
Figure 2: An overview of the dataset creation process.

et al., 2020), or focus on generalization to unseen *lexical* constructions rather than unseen reasoning skills (Hudson and Manning, 2019; Agrawal et al., 2017). Other datasets such as GQA (Hudson and Manning, 2018) use real images with synthetic questions, but lack logical operators available in natural language datasets, such as quantifiers and aggregations, and contain "reasoning shortcuts", due to a lack of challenging image distractors (Chen et al., 2020b).

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 262k examples based on ~89k images, with 13.9k 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 10 out of the 21 setups, where the *generalization score* we define is low (0%-70%). 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.[1]

2 Related Work

Prior work on grounded compositional generalization has typically tested generalization in terms of *lexical* and *syntactic* constructions, while we focus on compositions in the space of *meaning* by testing unseen program compositions. For example, the "structure split" in Hudson and Manning (2019) splits examples based on their surface form. As a result, identical programs are found in both the training and the test splits. The "content split" from the same work tests generalization to unseen lexical concepts. This is a different kind of skill than the one we address, where we assume the

---

[1] The dataset and our codebase can be found at https://github.com/benbogin/covr-dataset.

model sees all required concepts during training. C-VQA (Agrawal et al., 2017) splits are based only on question-answer pairs and not on the question meaning. CLEVR-CoGenT (Johnson et al., 2017) tests for unseen combinations of attributes, thus focuses more on visual generalization. Other datasets that do split samples based on their programs (Bahdanau et al., 2019a,b; Ruis et al., 2020) are using synthetic images with a small set of entities and relations ($\leq$ 20).

GQA (Hudson and Manning, 2018) uses real images with synthetic questions, which in theory could be used to create compositional splits. However, our work uses multiple images which allows testing reasoning operations over sets and not only over entities, such as counting, quantification, comparisons, etc. Thus, the space of possible compositional splits in COVR is much larger than GQA. If we anonymize lexical items in GQA programs (i.e., replace them with a single placeholder), the number of distinct programs (79) is too low to create a rich set of splits where all operators appear in the training set. In contrast, COVR contains 640 anonymized programs, allowing us to create a large number of splits. Moreover, our process for finding high-quality distracting images mitigates issues with reasoning shortcuts, where models solve questions due to a lack of challenging image distractors (Chen et al., 2020b). Finally, other VQA datasets with real images and questions that were generated by humans (Suhr et al., 2019; Antol et al., 2015) do not include a meaning representation for questions and thus cannot be easily used to create compositional splits.

3 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.
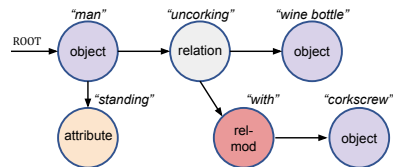

Figure 3: An example subgraph that shows all supported nodes, referring to *"standing man uncorking a wine bottle with a corkscrew"*. The types of the nodes are inside the circle, and their name above it.

High-level overview Fig. 2 provides an overview of the data generation process. Given an image and 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 (§3.2). 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 and a subset of the training set (Fig. 2e).

3.1 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

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 behind a 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.

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 goes through at most two `relation` nodes. For brevity, we refer to subgraphs with a textual description (*"standing man uncorking a wine bottle with a corkscrew"*).

3.2 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; Kervadec et al., 2021). For example, for the question in Fig. 2e, it is not necessary to perform all reasoning steps if there is only one man in all images in $\mathcal{I}$.

To find images with similar subgraphs, we define the *edit distance* between two graphs to be the min-

imum 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 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.[2]

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 objects and attributes (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 performing node substitutions (to compute the edit distance) 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

---

[2]https://neo4j.com/

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.

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. We also use our trained model to filter out cases where an object is annotated in the scene graph, but the image contains other instances of that object (e.g., if in an image with a crowd full of people, only a few are annotated). See App. B for details.

3.3 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. Specifically, if we anonymize lexical terms (i.e. replace nouns, relations and attributes with a single placeholder) there are 640 different

Measurement	Train	Dev.+Test
# total questions	248.1k	13.9k
# unique questions	122.0k	7.6k
# unique answers	3666	1268
# unique images	79.0k	9.5k
# unique anonymized programs	635	291
# True/False (T/F) questions	133.3k	7.5k
# "X or Y" questions	50.0k	2.9k
# "how many ..." questions	33.3k	1.8k
# Open questions	31.5k	1.7k
avg. # question words (a/p)	14.0	13.5/11.9
avg. # images per question	4.4	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.

programs in total.

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 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. Each row in the program is an operator (such as `Find`, `Filter`, `All`) with a given set of arguments (e.g. *"black"*) or dependencies (input from other operator). The list of program operators and a sample program are in App. G.

3.4 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, whenever possible, we create two examples with the same question, but a different set of images and answers (186,841/75.3% 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 nearly uniform distribution over templates, and balances the distribution over answers and over the size and depth of subgraphs as much

as possible (App. C). We provide statistics about our dataset in Table 3 and the answer distribution in App. D.

Manual Validation Examples thus far are generated automatically. To increase the evaluation set quality, we validate all development and test examples through crowdsourcing. Workers receive the question, images and answer, and reject it if they find it invalid, leading to a 17% rejection rate (see App. E.1 for more details, including error analysis).

Paraphrasing Since questions are generated automatically, we ask workers to paraphrase 3990 examples from the training set, and all development and test questions into fluent English, while maintaining their meaning. Paraphrasing examples are given in App. E.2.

Overall, after validation, COVR contains 262,069 training, 6,891 development and 7,024 test examples. See App. F for examples from the validation set.

4 Compositional Splits

Our generation process lets us create questions and corresponding programs with a variety of reasoning operators. We now show how COVR can be used to generate challenging compositional splits. We propose 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 (Bahdanau et al., 2019a; Yin et al., 2021).

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 ques-

Test name	Training	Generalization
HAS-QUANT-COMPSCOPE & 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 & RM/V/C	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 . / No 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 ∈ { 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).
HAS-QUANT-COMPSCOPE	True if the quantifier's scope is "complex", i.e., includes an attribute or a relation.
RM/V/C	True if g's structure contains either a Rel-Mod node, an object node with two outgoing edges to relation nodes (V-shape) or chain of more than a single relation (C).
TPL-X	True if the question originated from the template X.
ANS-X	True if answer is of type X ∈ { 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.

tions 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-COMPSCOPE and HAS-QUANT.

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 any pair together in the same program during training. We create 3 random splits where 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, except for M examples. From the evaluation set, we keep only examples that have this property.

5 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.

Model	COVR		COVR-PARAPH.	
	Development	Test	Development	Test
MAJ	26.4	26.8	26.4	26.8
MAJTEMPL	42.1	41.5	42.1	41.5
VB$_{\text{TEXT}}$	45.6	44.8	40.9	39.9
VB$_{\text{iid}}$	69.2	67.6	61.1	57.9
VB$_{\text{EASYDISTRACTORS}}$	56.4	56.9	50.2	49.0
Humans	-	-	-	91.9

Table 6: Results on both the generated and paraphrased versions of the development and test set for the i.i.d. split.

For templates that include two possible answers (*"candle or lighter"*), it randomly picks one.

We use the Volta framework (Bugliarello et al., 2021) to train and evaluate different pre-trained vision-and-language models. We use Volta's controlled setup models (that have a similar number of parameters and pre-training data) of VisualBERT (Li et al., 2019) and ViLBERT (Lu et al., 2019). In this section we show results only for VisualBERT, and results for ViLBERT can be found in App. H, showing mostly similar scores.

A vision-and-language model provides a representation for question-image pairs, (q, I). We modify the implementation to accept N images by running the model with (q, I) as input for each image $I \in \mathcal{I}$, and then passing the N 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 all parameters are fine-tuned during training.

To estimate a lower bound on performance without any reasoning on images, we evaluate a text-only baseline VB$_{\text{TEXT}}$ that only sees the input text (image representations are zeroed out).

For the compositional splits, we evaluate VisualBERT trained on the entire data (VB$_{\text{iid}}$), the text baseline (VB$_{\text{TEXT}}$), and the compositionally-trained models VB$_{250}$, VB$_0$ for the few-shot (M=250) and zero-shot setups, respectively. To control the training size, we also evaluate VB$_{\text{iid-size}}$, 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 development compositional split. To focus on the generalization gap, we define a generalization score ("Gen. Score") that measures the proportion of the gap between VB$_{\text{TEXT}}$ and our upper-bound, VB$_{\text{iid–size}}$,

that is closed by a model. In all compositional splits, we train the models 8 epochs and 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 VB$_{\text{TEXT}}$ and MAJTEMPL is small (3.3%), 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 (VB$_{\text{EASYDISTRACTORS}}$), accuracy drops by 10.7%, showing the importance of training on good distracting images. In addition, there is still a large gap from human performance, at 91.9%, which we estimate by evaluating human answers on 160 questions. Finally, we observe a 9.7% performance drop when training on the automatically-generated examples and testing on the paraphrased examples. Accuracy per template is shown in App. H.

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 5 out of 11 setups the generalization score is ≤ 70. VB$_{250}$ generalizes better in cases where the withheld operator is similar to an operator that appears in the training set. For instance, HAS-QUANT-ALL has higher generalization score compared to HAS-QUANT since it sees many examples with the quantifiers *"some"* and *"none"*, HAS-COMPAR-MORE has a higher score compared to HAS-COMPAR, and HAS-LOGIC-AND has a perfect generalization score. This suggests that when the model has some representation for a reasoning type it can generalize better to new instances of it.

The large gap between the nearly-perfect score of HAS-NUM-3 (92%), and the low score of HAS-NUM-3-ANS-3 (0%), 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-QUANT,

Split	Filtered	VB_{TEXT}	VB_{250}	Gen. Score	$VB_{iid\text{-}size}$	VB_{iid}
HAS-QUANT	33.3k	50.8	55.8		78.1	80.5
HAS-QUANT-ALL	21.1k	50.7	69.2		75.2	77.6
HAS-QUANT-COMPSCOPE	22.8k	50.8	60.4		78.1	80.3
HAS-COMPAR	16.7k	54.3	57.5		76.4	80.0
HAS-COMPAR-MORE	7.6k	53.8	79.8		81.3	84.0
HAS-GROUPBY	33.3k	38.1	56.5		62.8	65.7
HAS-LOGIC	16.7k	50.4	69.3		75.4	77.5
HAS-LOGIC-AND	8.8k	50.4	74.8		72.3	75.2
HAS-NUM-3	6.9k	44.8	68.8		70.8	71.9
HAS-NUM-3-ANS-3	12.1k	26.2	25.8		65.6	66.8
ANS-NUM	33.3k	26.4	32.2		64.9	67.0

Table 7: Non-paraphrased test 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	VB_{TEXT}	VB_0	Gen. Score	$VB_{iid\text{-}size}$	VB_{iid}
HAS-QUANT-COMPSCOPE & HAS-QUANT-ALL	12.9k	50.8	57.7		77.3	78.2
HAS-COUNT & HAS-ATTR	37.6k	41.2	58.7		62.6	63.6
HAS-COUNT & RM/V/C	36.9k	40.5	74.1		82.2	82.0
HAS-SAMEATTR-COLOR	27.3k	49.8	66.0		71.2	72.6
TPL-CHOOSEOBJECT	16.7k	52.0	1.6		62.6	66.4
TPL-VERIFYQUANTATTR	16.7k	50.4	71.2		76.9	80.3
TPL-VERIFYATTR	16.7k	49.6	0.0		75.4	73.5
TPL-VERIFYCOUNT ∪ TPL-VERIFYCOUNTGROUPBY	33.3k	49.8	41.7		77.6	79.8
Program Split	48k ± 11k	43.8 ± 4.5	49.5 ± 3.6		61.5 ± 4.4	64.8 ± 4.7
Lexical Split	40k ± 3k	46.4 ± 0.1	71.5 ± 1.1		72.2 ± 1.5	73.0 ± 1.7

Table 8: Non-paraphrased test 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 VB_0 is lower than VB_{TEXT}. '&' indicates holding out the intersection of two sets of questions, '∪' indicates holding out the union of the two.

where quantifiers appear in only 250 examples, HAS-QUANT-COMPSCOPE, where the scope of the quantifier is complex, and HAS-COMPAR, where comparatives appear in only 250 examples. Fig. 13 (App. H) shows performance on the development set as M, the number of examples with the tested property that the model is shown during training, increases. We observe model performance is much lower when $M = 50$ and improves rapidly as M increases. This shows that models 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 where the quantifier scope is complex, but never in the context of the quantifier ALL, fails to generalize (HAS-QUANT-COMP & HAS-QUANT-ALL, 26%). 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 VERIFYATTR, 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 questions with the Count operator where the subgraph contains a complex sub-graph (HAS-COUNT & RM/V/C) 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 performs better when composing more atomic constructs. However, further characterizing the conditions under which compositional generalization occurs is an important question for future work.

6 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.

Acknowledgements

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.

Aishwarya Agrawal, Aniruddha Kembhavi, Dhruv Batra, and Devi Parikh. 2017. C-VQA: A compositional split of the visual question answering (VQA) v1.0 dataset. *ArXiv*, abs/1704.08243.

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.

Emanuele Bugliarello, Ryan Cotterell, Naoaki Okazaki, and Desmond Elliott. 2021. Multimodal pretraining unmasked: A meta-analysis and a unified framework of vision-and-language BERTs. *Transactions of the Association for Computational Linguistics*.

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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

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.

Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.

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 Hudson and Christopher D Manning. 2019. Learning by abstraction: The neural state machine. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

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

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. 2017. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1988–1997.

Corentin Kervadec, Theo Jaunet, Grigory Antipov, Moez Baccouche, Romain Vuillemot, and Christian Wolf. 2021. How transferable are reasoning patterns in vqa? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4207–4216.

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.

Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. 2019. Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*.

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*.

Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

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

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 91–99, Cambridge, MA, USA. MIT Press.

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*.

Pengcheng Yin, Hao Fang, Graham Neubig, Adam Pauls, Emmanouil Antonios Platanios, Yu Su, Sam Thomson, and Jacob Andreas. 2021. Compositional generalization for neural semantic parsing via span-level supervised attention. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2810–2823, Online. Association for Computational Linguistics.

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".

To collect the list of pairs to annotate that will be used for fine-tuning, we fetched all pairs that have been used in Visual Genome within the same context. For **attributes**, we took all pairs of attributes that have appeared within the context of the same object (this way, we will be likely to collect *"red"* and *"green"* since they appear within the context objects such as *"apple"*, but not *"red"* and *"grilled"*). For **nouns**, we consider all pairs of nouns that have been used with the same relation. For **relations**, we consider all pairs of relations that have been used with the same pair of nouns. While there are other resources that could have been useful for fine-tuning (e.g. WordNet, Fellbaum 1998), we did not use any such external knowledge base since it allowed us to have exact control on the subtleties of the data in our training context.

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 internal order of each pair for regularization. We get an accuracy score of 94.4% on 20% of the pairs which were held-out for validation. Similarly 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 images 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 will hopefully 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 §3.2 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) as performed in LXMERT's paper, 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.

The described procedure can be used to detect unannotated objects, however, it will not be useful in the non-rare case where an object is annotated in the scene graph, but the image contains more similar objects in the image (e.g. there is a crowd full of people in the image, but only a few of the people are annotated). We thus add another verification step for each simple sub-graph g. First, we take the annotated positions of all instances of g in the scene. For example, if there are three annotated *"apples"*, we will take the positions (bounding boxes annotations from Visual Genome) of all three. Then, we use our trained LXMERT model with the textual description of g (e.g. *"apple"*) and the image, but this time we zero-out the image parts that contain the apples according to their annotated positions.[3] Essentially, we are querying the model if there are any other *"apples"* other than those that are annotated. We use a similar procedure as before to find the best threshold, 0.5. Since the LXMERT model is never trained with zeroed-out parts, during the described fine-tuning procedure we also zero-out 15% of the bounding boxes.

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. At a high-level, we start with a total of N 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

---
[3]LXMERT uses pre-calculated features of bounding boxes that are proposed by Faster-RCNN (Ren et al., 2015), thus we zero-out proposed bounding boxes that overlap with the annotated bounding boxes.
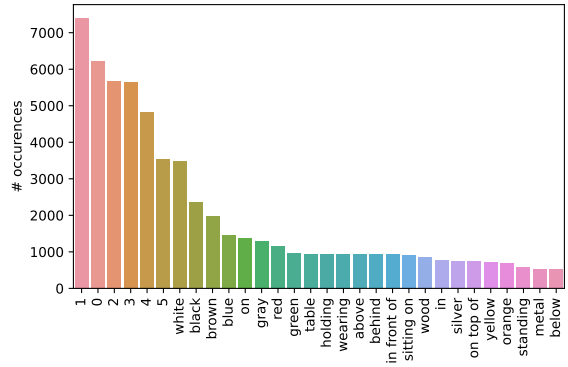

Figure 4: Distribution over the top 30 answers in the training set (excluding true/false).
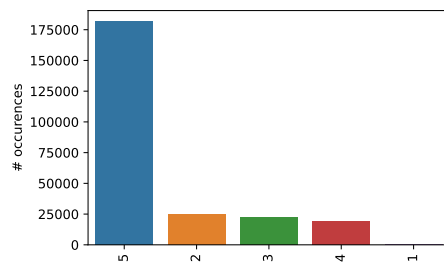

Figure 5: Distribution over the number of images each question in the training set contains.

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 L_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   Additional Statistics

Answers distribution   We show the distribution over the top 30 answers of the training set in Fig. 4, excluding true/false answers. As can be seen, the most common answers are the numbers 0-5, followed by common colors, attributes and relations.

Number of images distribution   We show in Fig. 5 the distribution over the number of images each question in the training set contains. As can be seen, most questions contain exactly 5 images.
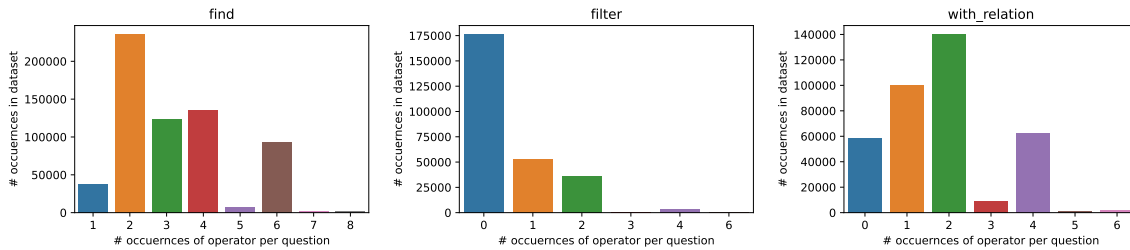
Figure 6: Distributions over the number of occurrences that specific operators appear in a single question in the training set.

Occurrences of operators in questions We show in Fig. 6, for a selected set of operators (`find`, `filter`, and `with_relation`), the distribution over the number of occurrences of that operator in a single question (e.g. the program for the question *"There is 1 green banana on a tree that is next to a man"* contains 3 `find` operators, one `filter` and two `with_relation`). The graphs show that `find` appears between one to eight times in a single question, and `filter` and `with_relation` between zero to six. Note that a question that contains six `with_relation` does not imply that a single reference to an object contains six relations, since a question can contain more than one object reference (e.g. in COMPARECOUNT).

E   Crowdsourcing Details

We use Amazon Mechanical Turk (AMT) for two different tasks: validation of questions and paraphrasing them.

E.1   Validation

We wanted to make sure that our validation and test examples are of high-quality by manually validating that the question is logically valid, there are no ambiguous object references, and the answer is correct. To maintain high-quality work in AMT, we first created a qualification task by annotating ourselves 100 examples, finding that the percentage of valid questions from the automatically generated samples was 83%. Workers were asked to choose one of the following options: *"Answer is CORRECT"*, *"I cannot understand the question"*, *"I cannot determine if the answer is correct"* or *"Answer is WRONG"*. We filtered workers by their performance: workers that have gained over 85% accuracy were given feedback and were approved

for the main task that contained the rest of the questions. During their work, we have repeatedly sampled the annotations of the workers and gave feedback where necessary, and also measured the accuracy of their submissions: all workers got an accuracy of between 95% to 98%. Workers were paid 0.5$ for a batch of 5 questions. Screenshots of the instructions and the HIT can be seen in Figures 7 and 8.

Analysis We sample 40 examples that were filtered out by the annotators to analyze the different causes for invalid generated questions. We find that most errors (70%) were due to problematic scene graph annotations: either because of missing annotations (53%) or wrong annotations (17%). The former type would make the answers to questions that require counting to be incorrect, and also questions that ask about a specific object (e.g. a question about *"a man wearing a hat"* will be invalid if there's more than one such man), and in general, means that our automatic validation mechanism failed to recognize that object. The latter type (wrong annotations, in contrast to missing annotations) can cause any question to be incorrect, and could not have been detected by our automatic validation methods. Other errors were questions about color (6%) that were not accurate (e.g. a question about whether two benches are brown will be given the answer 'true' since they are both annotated brown, but in practice, they could have significantly different shades of brown which might lead to the correct answer 'false'). The rest of the errors (16%) were due to various issues that make the answer unclear, such as questions that require to count feathers or meat.

- Our goal, with your help, is to teach the AI to answer complex questions given a set of images as context.
- For each machine generated question, you will get up to 5 images, and a predicted answer. **You will have to indicate whether the answer is CORRECT or WRONG, or if the answer cannot be determined.**
- Most examples are correct, but it is important to catch the wrong ones.
- If you cannot understand the question, select *"I cannot understand the question"*.
  Note that questions are sometimes ungrammatical or phrased unnaturally, and this is fine. Only select this option if the question can not be understood.
- If you cannot tell whether the answer is correct, select *"I cannot determine if the answer is correct"*.
  This could happen, for example, if the question refers to a specific object but there's more than one in the images, if objects are too small to count, etc. (see examples below)

- You may need to enlarge the image to see more details. You can enlarge an image by clicking on it (a new tab will open)
- When you see parentheses (...) in the machine generated question, it suggests that the method of answering is to **first** find the object inside the parentheses, **and only then** use that object to answer the rest of the question.
- Thanks!

Examples (click to collapse / expand)

| Correct | Correct | Correct | Wrong | Cannot determine | Cannot determine |

Machine generated question:

All knives that are on a plate are white

Machine generated answer:

False

Images:
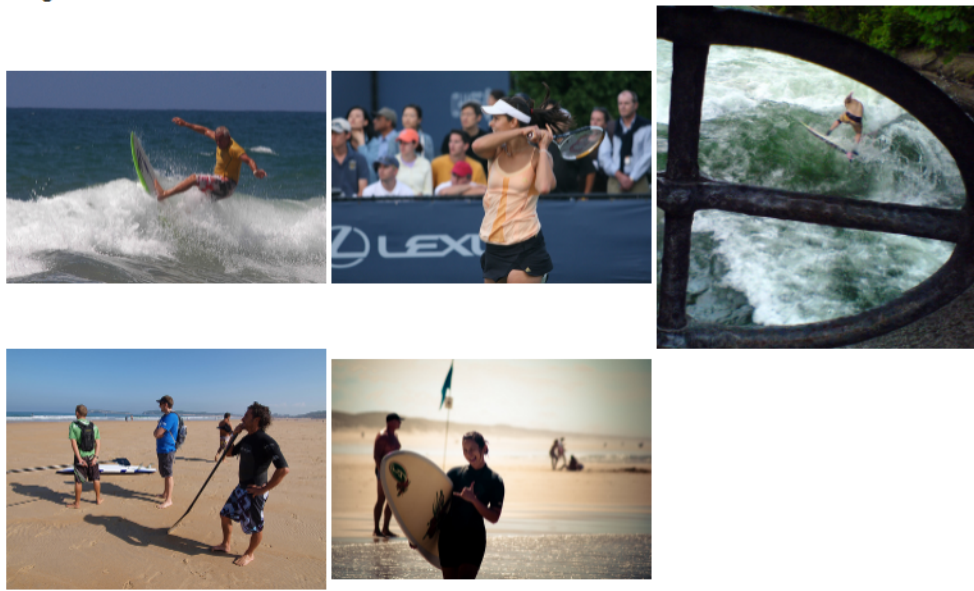

Expected annotation:

Answer is CORRECT

Some of the knives that are on a table are gray/silver.

Figure 7: The instructions for the AMT validation task.

Question 1:

Machine generated question	There is at least 1 image that contain exactly 3 barefoot men that are wearing shorts
Machine generated answer	True

Images:

Is answer correct?

Answer is WRONG

Comment (can be left blank):

Optional. Tell us if you see a problem with this example.

Figure 8: An example input for the AMT validation task. The expected annotation is "Answer is WRONG".

Auto-generated question	Paraphrase
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.

- Our goal, with your help, is to teach the AI to understand complex questions.
- We will present to you questions that were automatically generated by a machine. Your task is to rephrase them to natural spoken english while **preserving the exact meaning** correcting any grammatical errors.
- **Enter the rephrased question in the box below.**
- This is a long term task with thousands of future HITs.
- **20% additional bonus** will be rewarded for every 500 HITs completed.
- Try to be creative and to change the language of the question.
- For eg. You could consider changing a declarative (eg. "There is ...") question to interrogative (eg. "Is there ...?") and vice versa.
- But make sure that the rephrase is **grammatical** and the **meaning of the question does not change** at all. This means that the reasoning involved in the question and the possible answers do not change. More specifically, this includes
    - **Not changing words to opposites** even if the answer is expected to remain the same. Eg. "A is more than B" to "B is less than A", "A is behind B" to "B is in front of A" etc. (See example 8).
    - **Not changing words to similar words** unless you are certain it won't change the meaning of the question. (See examples 9, 10 and 11).
- When you see such a parentheses (...) in the machine generated question, it suggests that the method of answering is to **first** find the object inside the parentheses, and **only then** use that object to answer the rest of the question. Your paraphrase should be **consistent** with this requirement. Note the paraphrase itself doesn't need to contain parantheses nor does it need to contain their contents verbatim.
- All rephrases will be verified (by multiple workers) in a different project for quality assurance.
- If the machine generated question cannot be understood, please enter "**cannot understand**" (all lowercase) in comment box.
- Thanks!

Examples (click to collapse / expand)

| (1) Good | (2) Good | (3) Good | (4) Good | (5) Good | (6) Good | (7) Bad | (8) Bad | (9) Bad | (10) Bad | (11) Good |

Machine generated question:

Is the people standing next to a wood table or sitting on it?

Rephrased Question:

Are the people standing next to a wooden table or sitting on it?

> Notice the correction of "Is the people" to "Are the people" and "wood table" to "wooden table". A correct rephrase will correct all ungrammaticalities and make the question fluent.

Figure 9: The instructions and examples for the AMT question rephrasing task.

Question 1:

Machine generated question:

How many images contain at least 2 gray elephants?

Rephrased Question:

Comment:

Optional. Tell us if you see a probelm with this example, or with the instructions.

Figure 10: An example HIT for the AMT question rephrasing task.

E.2 Paraphrasing

For the question paraphrasing task, we again conducted a qualification task in addition to the final task. All potential workers were first added to the qualification task and asked to paraphrase 10 questions each. The paraphrases were then manually analyzed for meaning preservation and fluency and only the workers with very good performance were added to the final task which was used to paraphrase the bulk of the questions. In either case, we shared feedback with the workers via Google spreadsheets (one for each worker). Additionally, we regularly sampled and analysed the workers' paraphrases in the final task and used the same spreadsheets to share any necessary feedback. The workers were asked to periodically check their feedback spreadsheets and the workers that ignored the feedback were disqualified from the final task. We qualified 14 workers to the final task most of whom wrote good paraphrases. We only had to disqualify one worker for not taking note of their feedback.

Both the qualification and final tasks had the same instructions, examples and HIT interface. Screenshots can be seen in Figures 9 and 10. Workers were paid $0.7 for every task completed in both AMT tasks – with 5 questions per task. Additionally, as shown in Figure 10, workers were provided a comment box to leave comments in case they could not understand the question. Comments were left for a very small fraction of questions (less than 2%), mostly to indicate questions that were invalid or unclear. We removed all questions with comments in the final datasets. Some examples of the crowd-sourced paraphrases are shown in Table 9.

F Dataset examples

Fig. 11 and 12 show 10 selected validated and paraphrased examples from the validation set, demonstrating the variety of the questions and relevant distracting images.

G Programs

We list all program operators in Table 11, together with their input arguments/dependencies and output. A sample program can be found in Table 10.

H Additional Results

Results per pattern Tables 12 and 13 show the accuracy for each template for both the non-paraphrased and the paraphrased versions, for models that were trained on all data. The results of

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 10: 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.

the text-only baseline, VB_{TEXT}, show that indeed the model is struggling to get more than a random baseline performance of 50% in most patterns. The scores of the main model, VB_{iid}, show that open questions are hardest (QUERYATTR, QUERYOBJECT), and that there's a rather large variance between the performance on the different patterns.

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. The generalization scores are lower than the results for the non-paraphrased data, showing that transfer to natural language makes compositional generalization harder.

Effect of M We show how M, the number of examples the model sees from the compositional subset, affects the accuracy in Figure 13. The graph shows that using 50 examples barely has an effect, and that most of the improvement is achieved when increasing the number of examples from 125 to 2500. Increasing it further shows diminishing improvements.

ViLBERT Results To assess whether the results we get are specific to the model that we used (VisualBERT), we run additional compositional tests on a different model, ViLBERT, using Volta's framework (Bugliarello et al., 2021). The model has the same number of parameters and was trained on the same pre-training data. Results in Tables 16 and 17 show that for most of the compositional splits, both of our tested models get similar generalization scores.

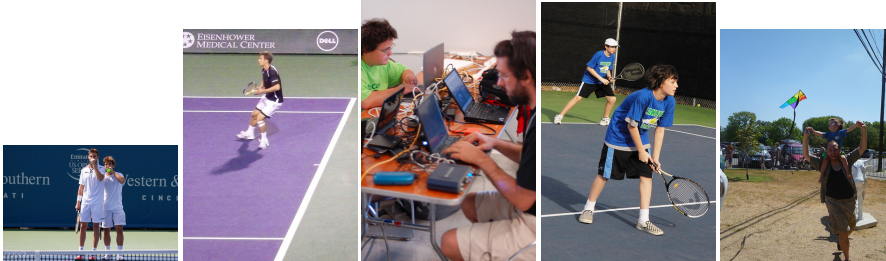**What is being held by the man in the jacket?** umbrella

No men are surfing on a white surfboard. True
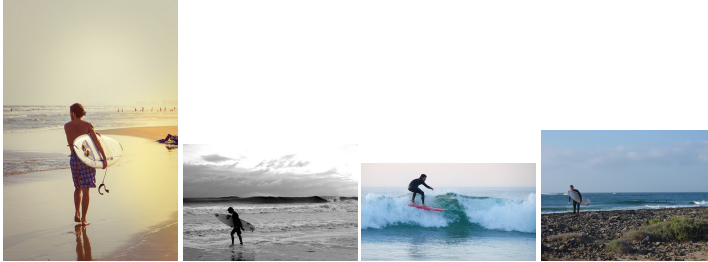
At least one image depicts a girl hitting a ball while wearing gloves. False

There are precisely two men wearing white shorts in how many images? 1

There is at least 1 image with exactly 2 dark bottles on a counter. True
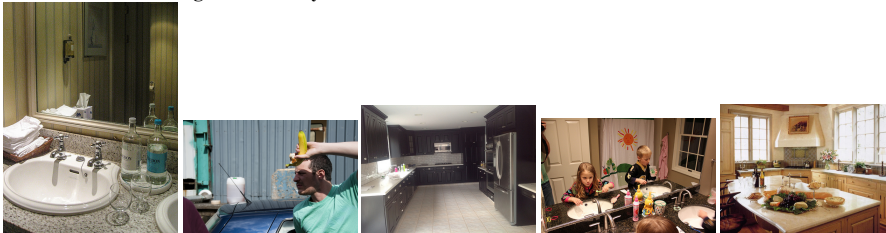
Figure 11: Selected examples from COVR validation set.

There are fewer plates on the table than there are on the wall. True
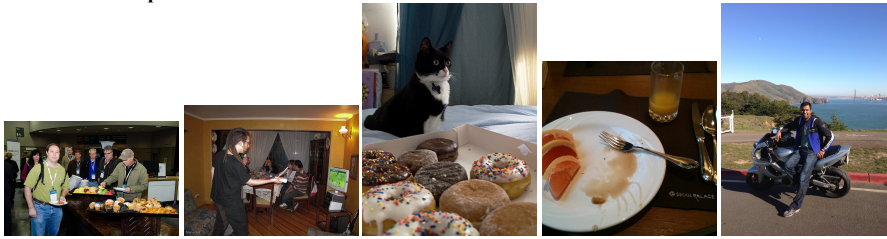
Are there 2 boys wearing a hat? False

In how many of the images can you see exactly two green cones? 2

Is the plate on a brown table the same color as the plate that is on a black table? True

All of the cats on a laptop are dark. False

Figure 12: Selected examples from COVR validation set.

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 given 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 11: All program operators.

Template	VB$_{\text{TEXT}}$	VB$_{\text{iid}}$
VERIFYATTR	49.6	73.5
CHOOSEATTR	52.0	69.5
QUERYATTR	36.2	47.1
COMPARECOUNT	54.3	80.0
COUNT	26.2	73.1
VERIFYCOUNT	50.0	87.7
COUNTGROUPBY	26.6	60.3
VERIFYCOUNT-GROUPBY	49.5	71.2
VERIFYLOGIC	50.4	77.5
VERIFYQUANTIFIER	51.2	80.7
VERIFYQUANTIFIER-ATTR	50.4	80.3
CHOOSEOBJECT	52.0	66.4
QUERYOBJECT	11.0	14.4
VERIFYSAMEATTR	52.1	62.3
CHOOSEREL	58.1	65.9

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

Template	VB$_{\text{TEXT}}$	VB$_{\text{iid}}$
VERIFYATTR	46.5	69.0
CHOOSEATTR	48.5	62.1
QUERYATTR	23.3	35.2
COMPARECOUNT	50.9	67.8
COUNT	25.2	67.6
VERIFYCOUNT	50.2	80.9
COUNTGROUPBY	29.6	46.0
VERIFYCOUNT-GROUPBY	50.0	67.1
VERIFYLOGIC	47.7	71.1
VERIFYQUANTIFIER	50.0	68.7
VERIFYQUANTIFIER-ATTR	50.0	75.6
CHOOSEOBJECT	19.6	41.9
QUERYOBJECT	6.3	8.3
VERIFYSAMEATTR	47.9	50.7
CHOOSEREL	51.8	52.2

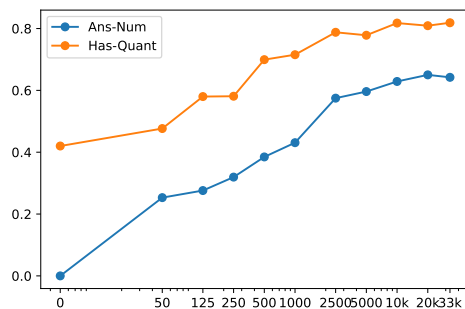Table 13: Accuracy score per template (i.i.d setup) on COVR (paraphrased), test set.

Figure 13: Effect of M on development set accuracy on two compositional splits, starting with no examples at all of the tested compositional property (zero-shot) and up to the maximal amount of available training examples with that property. Log scale is used for the X-axis.

Split	Filtered	VB_{TEXT}	VB_{250}	Gen. Score	$VB_{iid\text{-}size}$	VB_{iid}
HAS-QUANT	33.3k	50.0	48.8		70.9	72.1
HAS-QUANT-ALL	21.1k	50.2	58.7		70.1	74.1
HAS-QUANT-COMPSCOPE	22.8k	49.8	53.8		70.3	72.5
HAS-COMPAR	16.7k	50.9	44.8		64.8	67.8
HAS-COMPAR-MORE	7.6k	50.8	60.7		70.6	72.9
HAS-GROUPBY	33.3k	39.8	54.0		57.8	56.6
HAS-LOGIC	16.7k	47.7	55.7		71.9	71.1
HAS-LOGIC-AND	8.8k	49.6	72.7		73.4	73.0
HAS-NUM-3	6.9k	42.7	65.6		63.5	63.5
HAS-NUM-3-ANS-3	12.1k	24.2	25.4		54.5	51.6
ANS-NUM	33.3k	27.3	28.8		57.9	57.3

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

Split	Filtered	VB_{TEXT}	VB_0	Gen. Score	$VB_{iid\text{-}size}$	VB_{iid}
HAS-QUANT-COMP & HAS-QUANT-ALL	12.9k	50.1	54.5		68.9	75.0
HAS-COUNT & HAS-ATTR	37.6k	40.9	52.7		53.0	55.7
HAS-COUNT & RM/V/C	36.9k	39.9	68.5		76.3	76.5
HAS-SAMEATTR-COLOR	27.3k	47.7	59.3		61.6	63.5
TPL-CHOOSEOBJECT	16.7k	19.6	1.6		34.8	41.9
TPL-VERIFYQUANTATTR	16.7k	50.0	57.8		72.0	75.6
TPL-VERIFYATTR	16.7k	46.5	8.1		64.0	69.0
TPL-VERIFYCOUNT ∪ TPL-VERIFYCOUNTGROUPBY	33.3k	50.1	47.7		73.7	74.3
Program Split	48k ± 11k	38.6 ± 2.5	42.8 ± 3.8		52.3 ± 2.0	55.5 ± 2.7
Lexical Split	40k ± 3k	42.7 ± 1.3	61.2 ± 0.5		63.0 ± 0.7	64.3 ± 0.3

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

Split	Filtered	VLB_{TEXT}	VLB_{250}	Gen. Score	$VLB_{iid\text{-}size}$	VLB_{iid}
HAS-QUANT	33.3k	51.1	65.6		78.5	80.7
HAS-QUANT-ALL	21.1k	50.9	60.9		74.1	77.6
HAS-QUANT-COMPSCOPE	22.8k	50.9	67.5		78.1	80.2
HAS-COMPAR	16.7k	54.3	64.2		76.8	77.7
HAS-COMPAR-MORE	7.6k	53.8	79.8		79.0	81.7
HAS-GROUPBY	33.3k	37.2	55.6		63.4	61.3
HAS-LOGIC	16.7k	50.2	72.1		76.2	75.0
HAS-LOGIC-AND	8.8k	50.0	70.9		75.5	74.5
HAS-NUM-3	6.9k	43.8	67.7		68.8	71.9
HAS-NUM-3-ANS-3	12.1k	25.4	27.5		66.4	68.0
ANS-NUM	33.3k	24.1	38.5		64.0	65.8

Table 16: Same splits and experiments as in Table 7 for VILBERT.

Split	Filtered	VLB_{TEXT}	VLB_{250}	Gen. Score	$VLB_{iid\text{-}size}$	VLB_{iid}
HAS-QUANT-COMPSCOPE & HAS-QUANT-ALL	12.9k	50.8	64.7		76.4	77.4
HAS-COUNT & HAS-ATTR	37.6k	39.6	57.4		62.6	62.6
HAS-COUNT & RM/V/C	36.9k	39.9	71.4		81.1	81.9
HAS-SAMEATTR-COLOR	27.3k	48.5	64.7		67.7	71.4
TPL-CHOOSEOBJECT	16.7k	51.0	2.0		58.9	63.8
TPL-VERIFYQUANTATTR	16.7k	50.4	61.2		76.1	78.2
TPL-VERIFYATTR	16.7k	50.2	0.0		70.0	75.4
TPL-VERIFYCOUNT ∪ TPL-VERIFYCOUNTGROUPBY	33.3k	49.6	29.5		78.0	77.1
Program Split	48k ± 11k	43.6 ± 4.5	49.0 ± 3.1		61.9 ± 4.5	64.6 ± 4.9
Lexical Split	40k ± 3k	45.6 ± 0.7	71.5 ± 1.4		70.6 ± 0.7	73.6 ± 0.9

Table 17: Same splits and experiments as in Table 8, for VILBERT.