Latent Compositional Representations Improve Systematic Generalization in Grounded Question Answering

Ben Bogin[1] **Sanjay Subramanian**[2] **Matt Gardner**[2] **Jonathan Berant**[1,2]
[1]Tel-Aviv University [2]Allen Institute for AI
{ben.bogin,joberant}@cs.tau.ac.il, {sanjays,mattg}@allenai.org

Abstract

Answering questions that involve multi-step reasoning requires decomposing them and using the answers of intermediate steps to reach the final answer. However, state-of-the-art models in grounded question answering often do not explicitly perform decomposition, leading to difficulties in generalization to out-of-distribution examples. In this work, we propose a model that computes a representation and denotation for all question spans in a bottom-up, compositional manner using a CKY-style parser. Our model effectively induces latent trees, driven by end-to-end (the answer) supervision only. We show that this inductive bias towards tree structures dramatically improves systematic generalization to out-of-distribution examples compared to strong baselines on an arithmetic expressions benchmark as well as on CLOSURE, a dataset that focuses on systematic generalization of models for grounded question answering. On this challenging dataset, our model reaches an accuracy of 92.8%, significantly higher than prior models that almost perfectly solve the task on a random, in-distribution split.

1 Introduction

Humans can effortlessly interpret new natural language utterances, as long as they are composed of previously-observed primitives and structure (Fodor and Pylyshyn, 1988). Neural networks, on the other hand, do not exhibit this *systematicity*: while they generalize well to examples sampled from the same distribution as the training set, they have been shown to struggle in generalizing to out-of-distribution (OOD) examples that contain new compositions in both grounded question answering (Bahdanau et al., 2019a,b) and semantic parsing (Finegan-Dollak et al., 2018; Keysers et al., 2020). For example, consider the question in Fig. 1. This question requires querying the size of objects, comparing colors, identifying spatial relations and computing intersections between sets of objects. Neural networks tend to succeed whenever these concepts are combined in ways that were seen during training time. However, they commonly fail whenever these concepts are combined in novel ways at test time.

A possible reason for this phenomenon is the expressivity of modern architectures such as LSTMs (Hochreiter and Schmidhuber, 1997) and Transformers (Vaswani et al., 2017), where rich representations that depend on the entire input are computed. The fact that token representations are contextualized by the entire utterance potentially lets the model avoid step-by-step reasoning, "collapse" multiple reasoning steps, and rely on shortcuts (Jiang and Bansal, 2019; Subramanian et al., 2020). Such failures are revealed when evaluating models for systematic generalization on OOD examples. This stands in contrast to pre-neural log-linear models, where hierarchical representations were explicitly constructed over the input (Zettlemoyer and Collins, 2005; Liang et al., 2013).

In this work, we propose a model for visual question answering (QA) that, analogous to these classical pre-neural models, computes for every span in the input question a *representation* and a *denotation*, that is, the set of objects in the image that the span refers to (see Fig. 1). Denotations for long spans are recursively computed from shorter spans using a bottom-up CKY-style parser *without* access to the entire input, leading to an inductive bias that encourages compositional computation. Because training is done from the final answer only, the model must effectively learn to induce latent trees that describe the compositional structure of the problem. We hypothesize that this explicit grounding of the meaning of sub-spans through hierarchical computation should result in better gen-

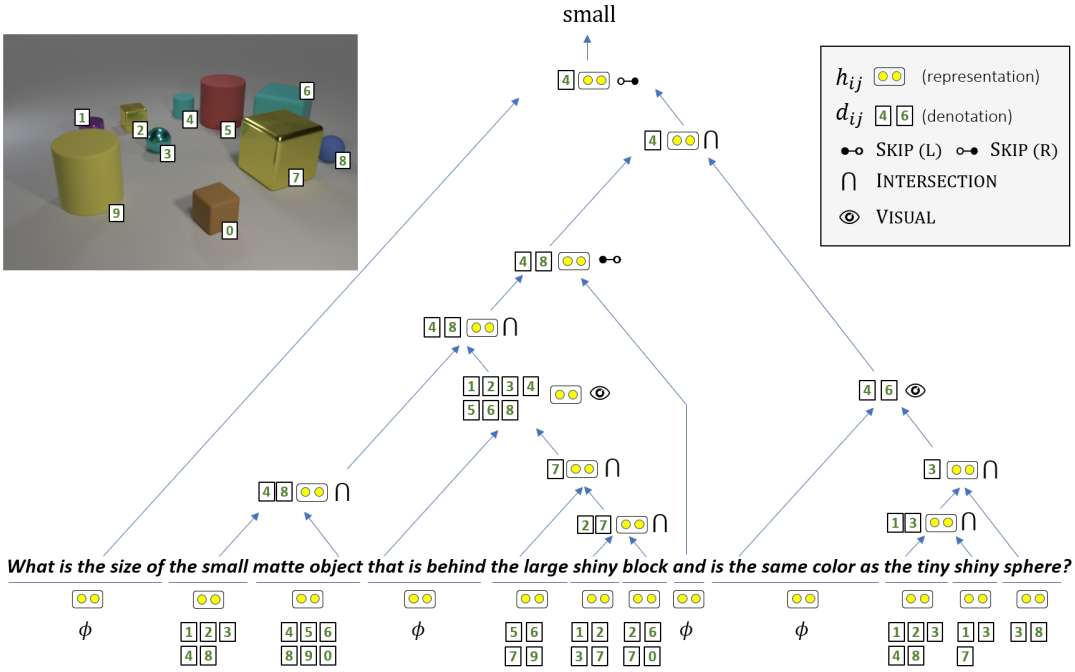


Figure 1: An example from CLOSURE illustrating how our model learns a latent structure over the input, where a representation and denotation is computed for every span (for denotation we show the set of objects with probability > 0.5). For brevity, some phrases were merged to a single node of the tree. For each phrase, we show the split point and module with the highest probability, although all possible split points and module outputs are softly computed. SKIP(L) and SKIP(R) refer to taking the denotation of the left or right sub-span, respectively.

eralization to new compositions.

We evaluate our approach in two setups: (a) a synthetic arithmetic expressions dataset, and (b) CLOSURE (Bahdanau et al., 2019b), a visual QA dataset that focuses on systematic generalization. On a random train/test split of the data (*i.i.d split*), both our model and prior baselines obtain near perfect performance. However, on splits that require systematic generalization to new compositions (*compositional split*) our model dramatically improves performance: for the arithmetic expressions problem, a vanilla Transformer fails to generalize and obtains 2.9% accuracy, while our model, Grounded Latent Trees (GLT), gets 98.4%. On CLOSURE, our model's accuracy is at 92.8%, 20% absolute points higher than strong baselines and even 15% points higher than models that use gold structures at training time or depend on domain-knowledge.

To conclude, we propose a model with an inherent inductive bias for copositional computation, which leads to large gains in systematic generalization, and induces latent structures that are useful for understanding its inner workings. Our work suggests that despite the undeniable success of general-purpose architectures built on top of contextualized representations, restricting information flow inside the network can greatly benefit compositional generalization.[1]

2 Compositional Generalization

Natural language is mostly compositional; humans can understand and produce a potentially infinite number of novel combinations from a closed set of known components (Chomsky, 1957; Montague, 1970). For example, a person would know what a *"winged giraffe"* is even if she's never seen one, assuming she knows the meaning of *"winged"* and *"giraffe"*. This ability, which we term *compositional generalization*, is fundamental for building robust models that effectively learn from limited data (Lake et al., 2018).

Neural networks have been shown to generalize well in many language understanding tasks (Devlin et al., 2019; Raffel et al., 2019), when using i.i.d splits. However, when models are evaluated on splits that require compositional generalization, a significant drop in performance is observed. For example, in SCAN (Lake and Baroni, 2018) and

[1] Our code and data can be found at https://github.com/benbogin/glt-grounded-latent-trees-qa.

gSCAN (Ruis et al., 2020), synthetically generated commands are mapped into a sequence of actions. When tested on unseen command combinations, models perform poorly. A similar case was shown in text-to-SQL parsing Finegan-Dollak et al. (2018), where splitting the training examples by the template of the target SQL query resulted in a dramatic drop in performance. SQOOP (Bahdanau et al., 2019a) shows the same phenomena on a synthetic visual QA task, which tests for generalization over unseen combinations of object properties and relations. This also led to developing methods that construct compositional splits automatically (Keysers et al., 2020).

In this work, we focus on answering complex grounded questions over images. The CLEVR benchmark (Johnson et al., 2017a) contains pairs of synthetic images and questions that require multi-step reasoning, e.g., *"Are there any large cyan spheres made of the same material as the large green sphere?"*. While this task is mostly solved, with an accuracy of 97%-99% (Perez et al., 2018; Hudson and Manning, 2018), recent work (Bahdanau et al., 2019b) introduced CLOSURE: a new set of questions with identical vocabulary but different structure than CLEVR, asked on the same set of images. They evaluated generalization of different model families and showed that all fail on a large fraction of the examples.

The most common approach for grounded QA is based on end-to-end differentiable models such as FiLM (Perez et al., 2018), MAC (Hudson and Manning, 2018) LXMERT (Tan and Bansal, 2019), and UNITER (Chen et al., 2019). These high-capacity models do not explicitly decompose the problem into smaller sub-tasks, and are thus prone to fail on compositional generalization. A different approach (Yi et al., 2018; Mao et al., 2019) is to parse the image into a symbolic or distributed knowledge graph with objects, attributes (color, size, etc.), and relations, and then parse the question into an executable logical form, which is deterministically executed. Last, Neural Module Networks (NMNs; Andreas et al. 2016) parse the question into an executable program as well, but execution is learned: each program module is a neural network designed to perform an atomic task, and modules are composed to perform complex reasoning. The latter two model families construct compositional programs and have been shown to generalize better on compositional splits (Bahdanau et al., 2019a,b) compared to fully differentiable models. However, programs are not explicitly tied to spans in the input question, and search over the space of possible programs is not differentiable, leading to difficulties in training.

In this work, we learn a latent structure for the question and tie each question span to an executable module in a differentiable manner. Our model balances the distributed and the symbolic approaches: we learn from downstream supervision only and output an inferred tree of the question, describing how the answer was computed. We base our model on work on latent tree parsers (Le and Zuidema, 2015; Liu et al., 2018; Maillard et al., 2019; Drozdov et al., 2019) that produce representations for all spans, and compute a soft weighting over all possible trees. We extend these parsers to answer grounded questions, grounding sub-trees in image objects. Closest to our work is Gupta and Lewis (2018), where denotations are computed for each span. However, they do not compute compositional representations for the spans, limiting the expressivity of their model. Additionally, they work with a knowledge graph rather than images.

3 Model

In this section, we give a high-level overview of our proposed Grounded Latent Trees (GLT) model (§3.1), explain our grounded CKY-based parser (§3.2), and describe the architecture details (§3.3, §3.4) and training procedure (§3.5).

3.1 High-level overview

Problem setup Our task is visual QA, where given a question $q = (q_0, \ldots, q_{n-1})$, and an image I, we aim to output an answer $a \in \mathcal{A}$ from a fixed set of natural language phrases. We train a model from a training set $\{(q^i, I^i, a^i)\}_{i=1}^N$. We assume we can extract from the image up to n_{obj} features vectors of objects, and represent them as a matrix $V \in \mathbb{R}^{n_{\text{obj}} \times h_{\text{dim}}}$ (details on object detection and representation are in §3.4).

Our goal is to compute for every question span $q_{ij} = (q_i, \ldots, q_{j-1})$ a *representation* $\mathbf{h}_{ij} \in \mathbb{R}^{h_{\text{dim}}}$ and a *denotation* $\mathbf{d}_{ij} \in [0,1]^{n_{\text{obj}}}$, which we interpret as the probability that the question span refers to each object. We compute $\mathbf{h}_{ij}$ and $\mathbf{d}_{ij}$ in a bottom-up fashion, using CKY (Cocke, 1969; Kasami, 1965; Younger, 1967). Algorithm 1 provides a high-level description of the procedure.

Algorithm 1

Require: question q, image I, word embedding matrix E, visual representations matrix V

1: H: tensor holding representations $\mathbf{h}_{ij}, \forall i, j$ s.t. $i < j$
2: D: tensor holding denotations $\mathbf{d}_{ij}, \forall i, j$ s.t. $i < j$
3: **for** $i = 1 \ldots n$ **do**
4: $\quad \mathbf{h}_i = E_{q_i}, \mathbf{d}_i = f_{\text{ground}}(E_{q_i}, V)$ (see §3.4)
5: **for** $l = 1 \ldots n$ **do**
6: $\quad$ compute $\mathbf{h}_{ij}, \mathbf{d}_{ij}$ for all entries s.t $j - i = l$
7: $p(a \mid q, I) = \text{softmax}(W[\mathbf{h}_{0n}; \mathbf{d}_{0n}V])$
8: **return** $\arg\max_a p(a \mid q, I)$

We compute representations and denotations for length-1 spans (we use $\mathbf{h}_i = \mathbf{h}_{i(i+1)}$, $\mathbf{d}_i = \mathbf{d}_{i(i+1)}$ for brevity) by setting the representation $\mathbf{h}_i = E_{q_i}$ to be the corresponding word representation in an embedding matrix E, and grounding each word in the image objects: $\mathbf{d}_i = f_{\text{ground}}(E_{q_i}, V)$ (lines 3-4; f_{ground} function described in §3.4). Then, we recursively compute representations and denotations of larger spans (lines 5-6). Last, we pass the representation of the entire question ($\mathbf{h}_{0n}$) together with the weighted sum of the visual representations ($\mathbf{d}_{0n}V$) through a softmax layer to produce a final answer distribution (line 7), using a learned classification matrix $W \in \mathbb{R}^{\mathcal{A} \times 2h_{\text{dim}}}$.

Computing $\mathbf{h}_{ij}, \mathbf{d}_{ij}$ for all spans requires overcoming some challenges. Each span representation $\mathbf{h}_{ij}$ should be a function of two *sub-spans* $\mathbf{h}_{ik}, \mathbf{h}_{kj}$. We use the term sub-spans to refer to all adjacent pairs of spans that cover q_{ij}, formally, $\{(q_{ik}, q_{kj})\}_{k=i+1}^{j-1}$. However, we have no supervision for the "correct" split point k. Our model (§3.2) considers all possible split points and learns to induce a latent tree structure from the final answer only. We show that this leads to a compositional structure and denotations that can be inspected at test time, providing an interpretable layer.

In §3.3 we describe the form of the *composition functions*, which compute both span representations and denotations from two sub-spans. These functions must be expressive enough to accommodate a wide range of interactions between sub-spans, but not create reasoning shortcuts that might hinder compositional generalization.

3.2 Grounded chart parsing

We now describe how to recursively compute $\mathbf{h}_{ij}, \mathbf{d}_{ij}$ from previously computed representations and denotations. In standard CKY-parsing, each constituent over a span q_{ij} is constructed by combining two sub-spans q_{ik}, q_{kj} that meet at a split point k. Similarly, we define a representation $\mathbf{h}_{ij}^k$ that is conditioned on the split point and constructed from previously-computed representations of two sub-spans:

$$\mathbf{h}_{ij}^k = f_h(\mathbf{h}_{ik}, \mathbf{h}_{kj}), \tag{1}$$

where $f_h(\cdot)$ is a *composition function* (§3.3).

Since we want the loss to be differentiable with respect to its input, we do not pick a particular value k, but instead use a continuous relaxation. Specifically, we compute the probability $p_H(k \mid i, j)$ that k is the split point for the span q_{ij}, given the tensor H of all computed representations of shorter spans. We then define the representation of the span $\mathbf{h}_{ij}$ to be the expected representation over all possible split points:

$$\mathbf{h}_{ij} = \sum_k p_H(k \mid i, j) \cdot \mathbf{h}_{ij}^k = \mathbb{E}_{p_H(k|\cdot)}[\mathbf{h}_{ij}^k]. \tag{2}$$

The split point distribution is defined as $p_H(k \mid i, j) \propto \exp(\mathbf{s}^T \mathbf{h}_{ij}^k)$, where $\mathbf{s} \in \mathbb{R}^{h_{\text{dim}}}$ is a parameter vector that determines what split points are likely. Figure 2 illustrates computing $\mathbf{h}_{ij}$.

Next, we turn to computing the denotation $\mathbf{d}_{ij}$ of each span. Conceptually, computing $\mathbf{d}_{ij}$ can be analogous to $\mathbf{h}_{ij}$; that is, a function f_d will compute $\mathbf{d}_{ij}^k$ for every possible split point k, and we will define $\mathbf{d}_{ij} = E_{p_H(k|\cdot)}[\mathbf{d}_{ij}^k]$. However, the function f_d (see §3.3) interacts with the visual representations of all objects and is thus computation-

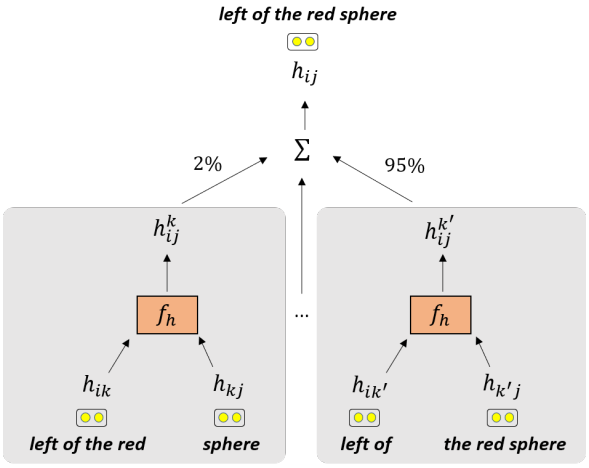


Figure 2: Illustration of how h_{ij} is computed. First, we consider all possible split points and compose pairs of sub-spans using f_h. Then, a weight is computed for all representations, and the output is their weighted sum.

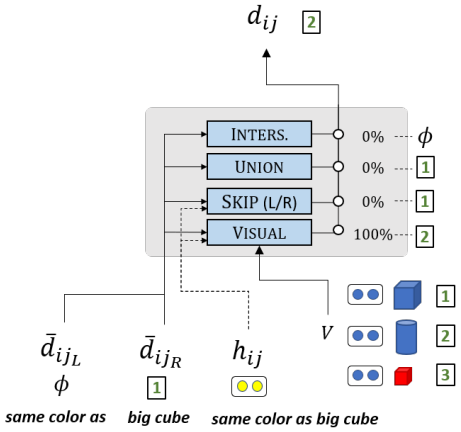


Figure 3: Illustration of how d_{ij} is computed. We compute the denotations of all modules, and a weight for each one of the modules. The span denotation is then the weighted sum of the module outputs.

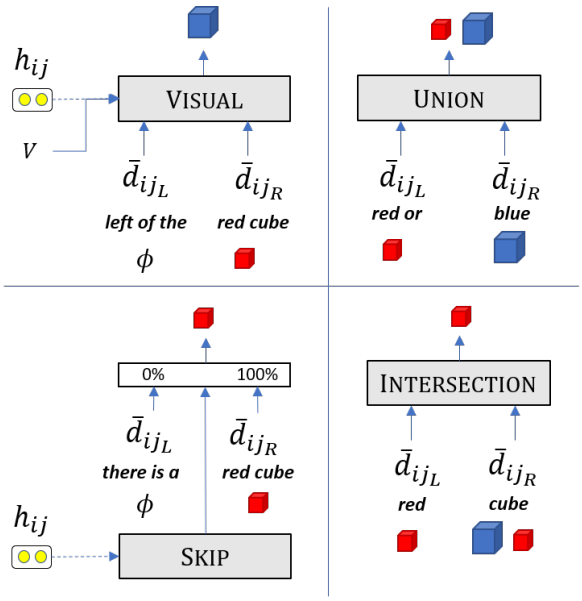


Figure 4: The different modules used with their inputs and expected output.

ally costly. Therefore, we propose a less expressive but more efficient approach, where $f_d(\cdot)$ is applied only once for each span q_{ij}.

Specifically, we compute the expected denotation of the left and right sub-spans of q_{ij}:

$$\bar{\mathbf{d}}_{ij_L} = \mathbb{E}_{p_H(k|\cdot)}[\mathbf{d}_{ik}] \quad \in \mathbb{R}^{n_{\text{obj}}} \quad (3)$$

$$\bar{\mathbf{d}}_{ij_R} = \mathbb{E}_{p_H(k|\cdot)}[\mathbf{d}_{kj}] \quad \in \mathbb{R}^{n_{\text{obj}}}. \quad (4)$$

If $p_H(k \mid \cdot)$ puts most probability mass on a single split point k', then the expected denotations will be similar to picking that particular split point.

Now we can compute $\mathbf{d}_{ij}$ given the expected sub-span denotations and representations with a single application of $f_d(\cdot)$:

$$\mathbf{d}_{ij} = f_d(\bar{\mathbf{d}}_{ij_L}, \bar{\mathbf{d}}_{ij_R}, \mathbf{h}_{ij}), \quad (5)$$

which is substantially more efficient than the alternative $\mathbb{E}_{p(k|\cdot)}[f_d(\mathbf{d}_{ik}, \mathbf{d}_{kj}, \mathbf{h}_{ij})]$: in our implementation f_d is applied $O(n^2)$ times versus $O(n^3)$ with the alternative solution. This is important for making training tractable in practice.

3.3 Composition functions

We now describe the exact form of the composition functions f_h and f_d.

Composing representations We first describe the function $f_h(\mathbf{h}_{ik}, \mathbf{h}_{kj})$, used to compose the representations of two sub-spans (Eq. 1). The goal of this function is to compose the "meanings" of two adjacent spans, *without* having access to the rest of the question or to the denotations of the sub-spans. For example, composing the representations of *"same"* and *"size"* to a representation for *"same size"*. At a high-level, composition is based on a generic attention mechanism. Specifically, we use attention to form a convex sum of the representations of the two sub-spans (Eq. 6-7), and apply a non-linear transformation with a residual connection (Eq. 8).

$$\alpha^k_{ij} = \text{softmax}\left([\mathbf{a}_L \mathbf{h}_{ik}, \mathbf{a}_R \mathbf{h}_{kj}]\right) \quad \in \mathbb{R}^2 \quad (6)$$

$$\hat{\mathbf{h}}^k_{ij} = \alpha_{(1)} W_L \mathbf{h}_{ik} + \alpha_{(2)} W_R \mathbf{h}_{kj} \quad \in \mathbb{R}^{h_{\text{dim}}} \quad (7)$$

$$f_h(\mathbf{h}_{ik}, \mathbf{h}_{kj}) = \text{FF}_{\text{rep}}\left(\hat{\mathbf{h}}^k_{ij}\right) + \hat{\mathbf{h}}^k_{ij} \quad \in \mathbb{R}^{h_{\text{dim}}} \quad (8)$$

where $\mathbf{a}_L, \mathbf{a}_R \in \mathbb{R}^{h_{\text{dim}}}$, $W_L, W_R \in \mathbb{R}^{h_{\text{dim}} \times h_{\text{dim}}}$, and $\text{FF}_{\text{rep}}(\cdot)$ is a linear layer of size $h_{\text{dim}} \times h_{\text{dim}}$ followed by a non-linear activation.[2]

Composing denotations Next, we describe the function $f_d(\bar{\mathbf{d}}_{ij_L}, \bar{\mathbf{d}}_{ij_R}, \mathbf{h}_{ij})$, used to compute the span denotation $\mathbf{d}_{ij}$ (Eq. 5). Importantly, this function has access only to words in the span q_{ij} and not to the entire input utterance. We would like $f_d(\cdot)$ to support both simple compositions that depend only on the denotations of sub-spans, as well as more complex functions that take into account the visual representations of different objects (spatial relations, colors, etc.).

[2]We also use Dropout and Layer-Norm (Ba et al., 2016) throughout the paper, omitted for simplicity.

We define four modules in $f_d(\cdot)$ for computing denotations and let the model learn when to use each module (we show in §4 that two modules suffice, but four improve interpretability). The modules are: SKIP, INTERSECTION, UNION, and a general-purpose VISUAL function, where only VISUAL uses the visual representations V. As illustrated in Fig. 3, each module m outputs a denotation vector $\mathbf{d}_{ij}^m \in [0,1]^{n_{\text{obj}}}$, and the denotation $\mathbf{d}_{ij}$ is a weighted average of the four modules:

$$p(m|i,j) \propto \exp(W_{\text{mod}}\mathbf{h}_{ij}) \quad \in \mathbb{R}^4 \quad (9)$$

$$\mathbf{d}_{ij} = \sum_m p(m|i,j) \cdot \mathbf{d}_{ij}^m \quad \in \mathbb{R}^{n_{\text{obj}}}, \quad (10)$$

where $W_{\text{mod}} \in \mathbb{R}^{h_{\text{dim}} \times 4}$. Next, we define the four modules (see Fig. 4).

SKIP In many cases, only one of the left or right sub-spans have a meaningful denotation: for example, for the sub-spans *"there is a"* and *"red cube"*, we should only keep the denotation of the right sub-span. To that end, the SKIP module weighs the two denotations and sums them:

$$(c_{ij}^{(1)}, c_{ij}^{(2)}) = \text{softmax}\left(W_{\text{sk}}\mathbf{h}_{ij}\right) \quad \in \mathbb{R}^2 \quad (11)$$

$$\mathbf{d}_{ij}^{\text{sk}} = c_{ij}^{(1)} \cdot \mathbf{d}_{ij_L} + c_{ij}^{(2)} \cdot \mathbf{d}_{ij_R} \quad \in \mathbb{R}^{n_{\text{obj}}}, \quad (12)$$

where $W_{\text{sk}} \in \mathbb{R}^{h_{\text{dim}} \times 2}$.

INTERSECTION and UNION We define two simple modules that only use the denotations $\bar{\mathbf{d}}_{ij_L}$ and $\bar{\mathbf{d}}_{ij_R}$. The first module corresponds to intersection of two sets, and the second to union:

$$\mathbf{d}_{ij}^{\text{int}} = \min\left(\bar{\mathbf{d}}_{ij_L}, \bar{\mathbf{d}}_{ij_R}\right) \quad \in \mathbb{R}^{n_{\text{obj}}}, \quad (13)$$

$$\mathbf{d}_{ij}^{\text{uni}} = \max\left(\bar{\mathbf{d}}_{ij_L}, \bar{\mathbf{d}}_{ij_R}\right) \quad \in \mathbb{R}^{n_{\text{obj}}}, \quad (14)$$

where $\min(\cdot)$ and $\max(\cdot)$ are computed element-wise, per object. We show in §4.2 that while these two modules are helpful for interpretability, their effect on performance is relatively small, and they can be omitted for simplicity.

VISUAL This module is responsible for compositions that involve visual computation, such as computing spatial relations (*"left of the red sphere"*) and comparing attributes of objects (*"has the same size as the red sphere"*). Unlike other modules, in addition to sub-span denotations it also uses the visual representations of the objects, $V \in \mathbb{R}^{n_{\text{obj}} \times h_{\text{dim}}}$. For example, for the sub-spans *"left of"* and *"the red object"*, we expect the function to ignore $\bar{\mathbf{d}}_{ij_L}$ (since the denotation of *"left to"* is irrelevant), and return a denotation with high probability for objects that are left to objects with high probability in $\bar{\mathbf{d}}_{ij_R}$.

To determine whether an object with index o should have high probability in the output, we need to consider its relation to all other objects. A simple scoring function might be $(\mathbf{h}_{ij} + \mathbf{v}_{o_1})^T(\mathbf{h}_{ij} + \mathbf{v}_{o_2})$, which will capture the relation between all pairs of objects conditioned on the span representation. However, this computation is quadratic in n_{obj}. Instead, we propose a linear alternative that again leverages expected denotations of sub-spans. Specifically, we compute the expected visual representation of the right sub-span and process this representation with a feed-forward layer:

$$\overline{v}_R = \bar{\mathbf{d}}_R V \in \mathbb{R}^{h_{\text{dim}}}, \quad (15)$$

$$\mathbf{q}_R = \text{FF}_R\left(W_h\mathbf{h}_{ij} + \overline{v}_R\right) \in \mathbb{R}^{h_{\text{dim}}}. \quad (16)$$

We use the right sub-span because the syntax in CLEVR is mostly right-branching, but a symmetric term can be computed if needed. Then, we generate a representation $\mathbf{q}(o)$ for every object that is conditioned on the span representation $\mathbf{h}_{ij}$, the object probability under the sub-span denotations, and its visual representation. The final object probability is based on the interaction of $\mathbf{q}(o)$ and $\mathbf{q}_R$:

$$\mathbf{q}(o) = \text{FF}_{\text{vis}}\left(W_h\mathbf{h}_{ij} + \mathbf{v}_o + \bar{\mathbf{d}}_L(o)\mathbf{s}_1 + \bar{\mathbf{d}}_R(o)\mathbf{s}_2\right)$$

$$\mathbf{d}_{ij}^{\text{vis}}(o) = \sigma\left(\mathbf{q}(o)^T\mathbf{q}_R\right)$$

where $W_{\text{h}} \in \mathbb{R}^{h_{\text{dim}} \times h_{\text{dim}}}$, $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{R}^{h_{\text{dim}}}$ are learned embeddings and FF_{vis} is a feed-forward layer of size $h_{\text{dim}} \times h_{\text{dim}}$ with a non-linear activation. This is the most expressive module we propose.

Relation to CCG Our approach is related to classical linguistic formalisms, such as CCG (Steedman, 1996), that tightly couple syntax and semantics. Under this view, one can consider the representations $\mathbf{h}$ and denotations $\mathbf{d}$ as analogous to syntax and semantics, and our composition functions and modules perform syntactic and semantic neural composition.

3.4 Grounding

In lines 3-4 of Algorithm 1, we initialize the representations and denotations of length-1 spans. The representation $\mathbf{h}_i$ is initialized as the corresponding word embedding E_{q_i}, and the denotation

is computed with a *grounding function*. A simple implementation for f_{ground} would be $\sigma(\mathbf{h}_i^\top V)$, based on the dot product between the word representation and the visual representations of all objects. However, in the case of a co-referring pronoun (*"it"*), we want to ground the pronoun to the denotation of a previous span. We now describe how we address this case.

Coreference Sentences such as *"there is a red sphere; what is its material?"* are harder to answer with a CKY parser, since the denotation of *"its"* depends on the denotation of a distant span. We propose a simple heuristic for this issue, that addresses the case where the referenced object is the denotation of a previous sentence. This solution could be potentially expanded in future work, to a wider array of coreference phenomena.

In every example that comprises two sentences:[3] (a) We compute the denotation $\mathbf{d}^{\text{first}}$ for the entire first sentence as described (standard CKY); (b) We ground each word in the second sentence as proposed above: $\hat{\mathbf{d}}_i^{\text{second}} = \sigma(\mathbf{h}_i^{\text{second}\top} V)$; (c) For each word in the second sentence, we predict whether it co-refers to $\mathbf{d}^{\text{first}}$ using a learned gate $(r_1, r_2) = \text{softmax}\left(\text{FF}_{\text{coref}}(\mathbf{h}_i^{\text{second}})\right)$, where $\text{FF}_{\text{coref}} \in \mathbb{R}^{\mathbf{h}_{\text{dim}} \times 2}$. (d) We define $\mathbf{d}_i^{\text{second}} = r_1 \cdot \mathbf{d}^{\text{first}} + r_2 \cdot \hat{\mathbf{d}}_i^{\text{second}}$.

Visual representation Next, we describe how we compute the visual embedding matrix V. Two common approaches to obtain visual features are (1) computing a feature map for the entire image and letting the model learn to attend to the correct feature position (Hudson and Manning, 2018; Perez et al., 2018); and (2) predicting the locations of objects in the image, and extracting features just for these objects (Anderson et al., 2018; Tan and Bansal, 2019; Chen et al., 2019). We use the latter approach, since it simplifies learning over discrete sets, and has better memory efficiency – the model only attends to a small set of objects rather then the entire image feature map.

Specifically, we run CLEVR images through a ResNet101 model (He et al., 2016), pre-trained on ImageNet (Russakovsky et al., 2015). This model outputs a feature map V_{all} of size $W \times H \times D$, where $D = 512$ and $W = H = 28$. We then use an object detector, Faster R-CNN (Ren et al., 2015), which predicts the location $\text{bb}_{\text{pred}} \in \mathbb{R}^4$ of all objects in the image, in the format of bounding boxes (horizontal and vertical positions, width and height). We use these predicted locations to compute V_{pred}, containing only the features in V_{all} that are predicted to contain an object according to bb_{pred}. Since Faster R-CNN was trained on real images, we adapt it to CLEVR images by training it to predict bounding boxes of 5,000 objects from CLEVR images (and 1,000 images used for validation), using gold scene data. The bounding boxes and features are extracted and fixed as a preprocessing step.

Finally, to compute V, in a similar fashion to LXMERT and UNITER we augment the object representations in V_{pred} with their position embeddings, and pass them through a single Transformer self-attention layer to add context about other objects: $V = \text{TransformerLayer}(V_{\text{pred}} W_{\text{feat}} + \text{bb}_{\text{pred}} W_{\text{pos}})$, where $W_{\text{feat}} \in \mathbb{R}^{D \times h_{\text{dim}}}$ and $W_{\text{pos}} \in \mathbb{R}^{4 \times h_{\text{dim}}}$.

Complexity Similar to CKY, we go over all $O(n^2)$ spans in a sentence, and for each span compute $\mathbf{h}_{ij}^k$ for each of the possible $O(n)$ splits (there is no grammar constant since the grammar has effectively one rule). To compute denotations $\mathbf{d}_{ij}$, for all $O(n^2)$ spans, we perform a linear computation over all n_{obj} objects. Thus, the algorithm runs in time $O(n^3 + n^2 n_{\text{obj}})$, with similar memory consumption. This is higher than end-to-end models that do not compute explicit span representations.

3.5 Training

The model is fully differentiable, and we train with maximum likelihood, maximizing the log probability $\log p(a^* \mid q, I)$ of the correct answer a^* (see Algorithm 1).

4 Experiments

In this section, we evaluate our model on both in-distribution and out-of-distribution splits.

4.1 Arithmetic expressions

It has been shown that neural networks can be trained to perform numerical reasoning (Zaremba and Sutskever, 2014; Kaiser and Sutskever, 2016; Trask et al., 2018; Geva et al., 2020). However, models are often evaluated on expressions that are similar to the ones they were trained on, where only the numbers change. To test for generalization, we create a simple dataset and evaluate on

[3] in CLEVR, we split sentences based on semi-colons.

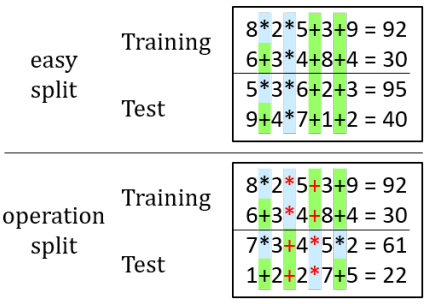


Figure 5: Arithmetic expressions: unlike the easy setup, we evaluate models on expressions with operations ordered in ways unobserved at training time. Flipped operator positions are in red.

two splits that require learning the correct operator precedence and outputs. In the first split, sequences of operators that appear at test time do not appear at training time. In the second split, the test set contains longer sequences compared to the training set.

We define an arithmetic expression as a sequence containing n numbers with $n-1$ arithmetic operators between each pair. The answer a is the result evaluating the expression.

Evaluation setups The sampled operators are addition and multiplication, and we take only expressions such that $a \in \{0, 1, \ldots, 100\}$ to train as a multi-class problem. During training, we randomly pick the length n to be up to n_{train}, and during test time we choose a fixed length n_{test}. We evaluate on three setups. In the easy split, we choose $n_{\text{train}} = n_{\text{test}} = 8$, and the sequence of operators is randomly drawn from a uniform distribution for both training and test examples. In this setup, we only check that the exact same expression is not shared between the training and test set. In the compositional split, we randomly pick 3 positions, and for each one randomly assign exactly one operator that will appear at training time. On the test set, the operators in all three positions are flipped, so that they now contain the unseen operator (see Fig. 5). The same lengths are used as in the easy split. Finally, in the length split, we train with $n_{\text{train}} = 8$ and test with $n_{\text{test}} = 10$. Examples for all setups are generated on-the-fly for 3 million steps with a batch size of 100.

Models We compare GLT to a standard Transformer, where the input is the expression, and the output is predicted using a classification layer over the `[CLS]` token. All models are trained with cross-entropy loss given the correct answer.

	Easy split	Op. split	Len. split
Transformer	**100.0** $\pm$ 0.0	2.9 $\pm$ 1.1	10.4 $\pm$ 2.4
GLT	99.9 $\pm$ 0.2	**98.4** $\pm$ 0.7	**94.9** $\pm$ 1.1

Table 1: Arithmetic expressions results for easy split, operation-position split, and length split.

For both models, we use an in-distribution validation set for hyper-parameter tuning. For the Transformer, we use 15 layers with a hidden size of 200 and feed-forward layer size of 300. For our model, we use $h_{\text{dim}} = 400$. Since in this setup we do not have an image or any grounded input, we only compute $\mathbf{h}_{ij}$ for all spans, and define $p(a \mid q) = \text{softmax}(W\mathbf{h}_{0n})$.

GLT layers are almost entirely recurrent, that is, the same parameters are used to compute representations for spans of all lengths. The only exception are layer-normalization parameters, which are not shared across layers. Thus, at test time when processing an expression longer than observed at training time, we use the layer-normalization parameters (total of $2 \cdot h_{\text{dim}}$ parameters per layer) from the longest span seen at training time.[4]

Results Results are reported in Table 1. We see that both models almost completely solve the in-distribution setup, but on out-of-distribution splits the Transformer performs poorly, while GLT shows only a small drop in accuracy.

4.2 CLEVR and CLOSURE

We evaluate performance on grounded complex questions using CLEVR (Johnson et al., 2017a), consisting of 100,000 synthetic images with multiple objects of different shapes, colors, materials and sizes. 864,968 questions were synthetically created using 80 different templates, including simple questions (*"what is the size of red cube?"*) and questions requiring multi-step reasoning (see Figure 1). The split in this dataset is i.i.d: templates used for training are the same as those in the validation and test sets.

To test compositional generalization after training on CLEVR, we use the recent CLOSURE dataset (Bahdanau et al., 2019b), which includes seven new question templates, with a total of

[4]Removing layer normalization leads to improved accuracy of 99% on the arithmetic expressions length split, but training on CLEVR becomes too slow.

	Train Programs	Test Programs	Deterministic Execution	CLEVR	CLOSURE
MAC	no	no	no	98.5	72.4
FiLM	no	no	no	97.0	60.1
GLT (our model)	no	no	no	98.4	**92.8** $\pm$ 3.0
NS-VQA	yes	no	yes	**100**	77.2
PG+EE (18K prog.)	yes	no	no	95.4	-
PG-Vector-NMN	yes	no	no	98.0	71.3
GT-Vector-NMN	yes	yes	no	98.0	94.4

Table 2: Test results for all models on CLEVR and CLOSURE. "Train Programs" stands for models trained with gold program, "Test Programs" for oracle models evaluated using gold programs, and "Deterministic Execution" for models that depend on domain-knowledge for execution (execution is not learned).

25,200 questions, asked on the CLEVR validation set images. The new templates are created by taking referring expressions of various types from CLEVR and combining them in novel ways.

A problem found in CLOSURE is that sentences from the template `embed_mat_spa` are ambiguous. For example, in the question *"Is there a sphere on the left side of the cyan object that is the same size as purple cube?"*, the phrase *"that is the same size as purple cube"* can modify either *"the sphere"* or *"the cyan object"*, but the answer in CLOSURE is always the latter. Therefore, we deterministically compute both of the two possible answers and keep two sets of question-answer pairs of this template for the entire dataset. We evaluate models[5] on this template by taking the maximum score over these two sets (such that models must be consistent and choose a single interpretation for the template to get a perfect score).

Baselines We evaluate against the baselines presented in Bahdanau et al. (2019b). The most comparable baselines are MAC (Hudson and Manning, 2018) and FiLM (Perez et al., 2018), which are differentiable and do not use any program annotations. We also compare to NMNs that require at least a few hundred program examples for training. We show results for PG+EE (Johnson et al., 2017b) and an improved version, PG-Vector-NMN (Bahdanau et al., 2019b). Last, we compare to NS-VQA, which in addition to parsing the question, also parses the scene into a knowledge graph. NS-VQA also requires domain-knowledge and data, as it parses the image into a knowledge graph based on gold data from CLEVR (objects color, shape, location, etc.).

Setup Baseline results are taken from previous papers (Bahdanau et al., 2019b; Hudson and Manning, 2018; Yi et al., 2018; Johnson et al., 2017b), except for MAC and FiLM on CLOSURE, which we re-executed due to the aforementioned evaluation change. For GLT, we use CLEVR's validation set for hyper-parameter tuning, and run 4 experiments to compute mean and variance on CLOSURE test set. We train for 40 epochs and perform early-stopping on CLEVR's validation set. We use $h_{\text{dim}} = 400$.

Because of our model's high run-time and memory demands (see §3.4), we found that running on CLEVR and CLOSURE, where question length goes up to 42 tokens, is difficult. Thus, we delete function words that typically have empty denotations and can be safely skipped,[6] reducing the maximum length to 25.

CLEVR and CLOSURE In this experiment we compare results on i.i.d and compositional splits. Results are in Table 2. We see that GLT performs well on CLEVR and gets the highest score on CLOSURE, improving by almost 20 points over comparable models. GLT is competitive even with the oracle GT-Vector-NMN which uses gold programs at **test time**.

Removing intersection and union As described in §3.3, we defined two modules specifically for CLEVR, (INTERSECTION and UNION). We remove these modules to evaluate performance without them, and see that the model suffers only a small loss in accuracy and generalization: accuracy on CLEVR (validation set) is 98.0 $\pm$ 0.3, and

[5] We update the scores on CLOSURE for MAC, FiLM and GLT due to this change in evaluation. The scores for the rest of the models were not affected.

[6] The removed tokens are punctuations, 'the', 'there', 'is', 'a', 'as', 'it', 'its', 'of', 'are', 'other', 'on', 'that'.

	CLOSURE FS	C.Humans
MAC	90.2	**81.5**
FiLM	-	75.9
GLT (our model)	**96.1** $\pm$ 0.9	72.8
NS-VQA	92.9	67.0
PG-Vector-NMN	88.0	-
PG+EE (18K prog.)	-	66.6

Table 3: Test results in the few-shot setup and for CLEVR-Humans.

accuracy on CLOSURE (test set) is 90.1 $\pm$ 7.1. Removing these modules leads to more cases where the VISUAL function is used, effectively performing intersection and union as well. While the drop in performance and generalization is mild, this model is harder to interpret since the VISUAL function performs multiple functions.

Few-shot We test GLT in a few-shot (FS) setup, where we add a few out-of-distribution examples. Specifically, we use 36 questions for each CLOSURE template, with a total of 252 examples. Similar to Bahdanau et al. (2019b), we take a model that was trained on CLEVR and fine-tune it by oversampling CLOSURE examples (300 times) and adding them to the original training set. To make results comparable to Bahdanau et al. (2019b), we perform model selection based on the CLOSURE validation set, and evaluate on the test set. As we see in Table 3, GLT gets the best accuracy. If we perform model selection based on CLEVR alone (the preferred way to evaluate in the OOD setup, Teney et al. 2020), accuracy on CLOSURE is 94.2 $\pm$ 2.1, which is still highest.

CLEVR-Humans To test the performance of GLT on real natural language, we test on CLEVR-Humans (Johnson et al., 2017b), which consists of 32,164 questions based on images from CLEVR. These questions, asked and answered by humans, contain new words and reasoning steps that were not seen in CLEVR. We take a model that was trained on CLEVR and fine-tune it on CLEVR-Humans training set, similar to prior work. We use GloVe (Pennington et al., 2014) for the embeddings of words unseen in CLEVR. We show results in Table 3. We see that GLT gets better results than models that use programs, showing its flexibility to learn new concepts and phrasings, but lower results compared to more flexible models like MAC and FILM (see error analysis below).

4.3 Error analysis

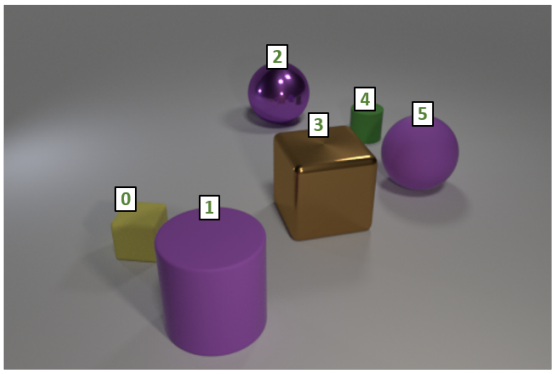
We sampled 25 questions with wrong predictions on CLEVR, CLOSURE, and CLEVR-Humans to analyze model errors. On **CLEVR**, most errors (84%) are due to problems in visual processing of the images such as grounding the word *"rubber"* to a metal object, problems in bounding box prediction or questions that require subtle spatial relation reasoning, such as identifying if an object is left to another object of different size, when they are at an almost identical x-position. The remaining errors (16%) are due to failed comparisons of numbers or attributes (*"does the red object have the same material as the cube"*).

On **CLOSURE**, 60% of the errors were similar to those seen in CLEVR, e.g. problematic visual processing or failed comparisons. We've found that in 4% of cases, the execution of the VISUAL module was wrong, e.g. it collapsed two reasoning steps (both intersection and finding objects of same shape), but did not output the correct denotation. Other errors (36%) are in the predicted latent tree, where the model was uncertain about the split point and softly predicted more than one tree, resulting in wrong answer predictions. In some cases (16%) this was due to question ambiguity (see §4.2), and in others cases the cause was unclear (e.g., for the phrase *"same color as the cube"* the model gave similar probability for the split after *"same"* and after *"color"*, leading to a wrong denotation for that span).

On **CLEVR-Humans**, we see that the model successfully learned certain new "concepts" such as colors (*"gold"*), superlatives (*"smallest", "largest"*), relations (*"the reflecting object"*), positions (*"back left"*) and negation (see Fig. 6). It also answered correctly questions with different style than CLEVR (*"Are there more blocks or balls?"*, *"... cube being covered by ..."*). However, the model fails on other new concepts, such as the *"all"* quantifier, arithmetic computations (*"how many more... are there than...?"*), and others (*"What is the most common shape?"*).

4.4 Interpretability

A key advantage of latent trees is interpretability – one can analyze the computation structure of a given question. Next, we analyze when are model outputs interpretable, and discuss how interpretability is affected by the limitations of GLT and relates to its generalization abilities. Addi-



How many objects are not purple and not metallic? (2)

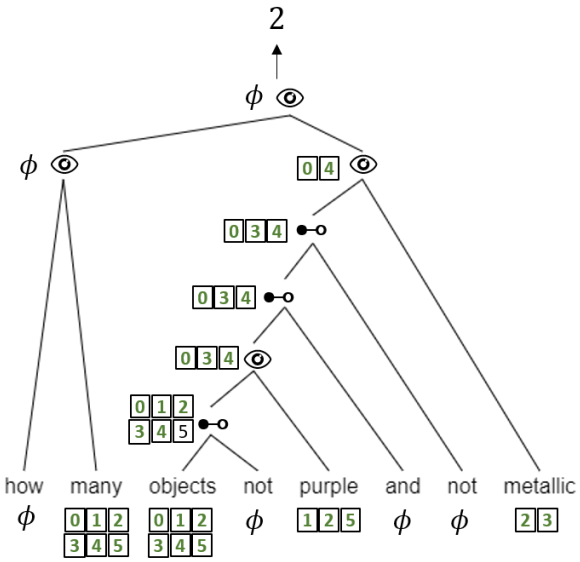
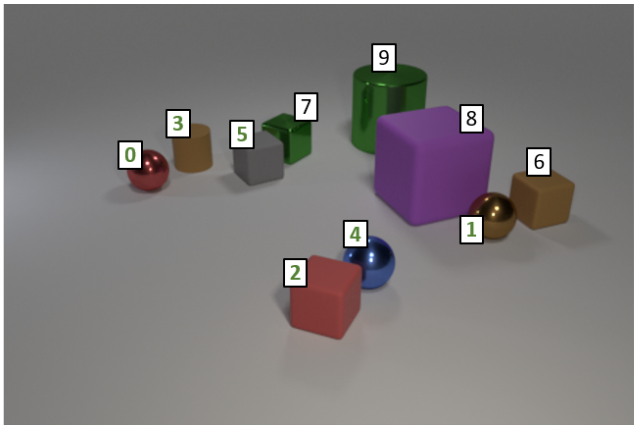


Figure 6: An example from CLEVR-Humans. The model learned to negate (*"not"*) using the VISUAL module (negation is not part of CLEVR).



what shape appears most often? (cube)

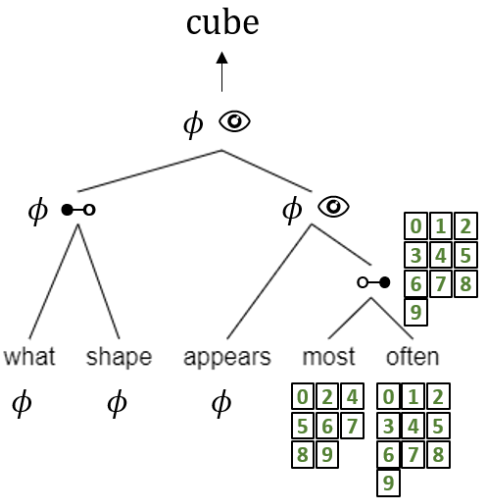


Figure 7: An example from CLEVR-Humans. This question requires reasoning steps that are not explicitly mentioned in the input. This results in a correct answer but non-interpretable output.

tional output examples can be seen in Appendix A.

The model predicts a denotation for each span, which is a probability for all objects in the image. Thus, for every question span that should correspond to a set of objects, the output is interpretable, as can be seen in Fig. 1. Having interpretable tree structures helps analyze ambiguous questions, such as the ones found in CLEVR and CLOSURE.

However, span denotations are not always distributions over objects, but rather a number or an attribute. For example, in comparison questions (*"is the number of cubes higher than the number of spheres?"*) a fully interpretable model would have a numerical denotation for each group of objects. GLT solves such questions, by grounding the objects correctly and leaving the counting and arithmetic comparison to the answer function (line 7 in Algorithm 1). However, this comes at a cost to interpretability (see Fig. 10). In the numerical comparison example, it is easy to inspect the grounding of objects, but hard to tell what is the count for each group, which is likely to affect generalization as well. A future research direction is to learn richer denotation structure.

Another case where interpretability is suboptimal is counting. Due to the expressivity of the answer function, the denotation in counting questions does not necessarily contain only the objects to be counted. For example, for a question such as *"how many cubes are there"*, the most interpretable model would only have all the cubes in the denotation of the entire question. However, GLT often outputs non-interpretable probabilities for the objects. In such cases, the outputs are interpretable for sub-spans of the question (*"cubes are there"*), as seen in Fig. 6. This issue could be addressed by pre-training or injecting different count modules, as shown by Subramanian et al. (2020).

Finally, the hardest case is when the required reasoning steps are not explicitly mentioned in the question. For example, the question *"what is the most common shape?"* requires to count the different shapes in the image, then take the shape with the maximum count. While our model answers this question correctly (see Fig. 7), it does so by "falling back" to the flexible answer function, rather than by explicitly performing the required computation. In future work, we will explore combining the compositional generalization abilities of our model, which grounds intermediate answers to spans, with the advantages of NMNs, that support more flexible reasoning.

5 Conclusion

We propose a model for grounded question answering that strongly relies on compositional computation. We show our model leads to large gains in a systematic generalization setup and provides an interpretable structure that can be inspected by humans and sheds light on the model's inner workings. Our work suggests that generalizing to unseen language structures can benefit from a strong inductive bias in the network architecture. By limiting our model to compose non-contextualized representations in a recursive bottom-up manner, we outperform state-of-the-art models a challenging compositional generalization task. Our model also obtains high performance on real natural language questions in the CLEVR-humans dataset. In future work, we plan to investigate the structures revealed by our model in other grounded question answering setups, and to allow the model more freedom to incorporate non-compositional signals, which go hand in hand with compositional computation in natural language.

Acknowledgements

This research was partially supported by The Yandex Initiative for Machine Learning, and the European Research Council (ERC) under the European Union Horizons 2020 research and innovation programme (grant ERC DELPHI 802800). We thank Jonathan Herzig for his useful comments. This work was completed in partial fulfillment for the Ph.D degree of Ben Bogin.

References

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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6077–6086.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 39–48.

Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *ArXiv*, abs/1607.06450.

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

Dzmitry Bahdanau, Harm de Vries, Timothy J. O'Donnell, Shikhar Murty, Philippe Beaudoin, Yoshua Bengio, and Aaron C. Courville. 2019b. Closure: Assessing systematic generalization of clevr models. *ArXiv*, abs/1912.05783.

Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. 2019. Uniter: Learning universal image-text representations. *ArXiv*, abs/1909.11740.

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

John Cocke. 1969. *Programming Languages and Their Compilers: Preliminary Notes*. New York University, USA.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Andrew Drozdov, Patrick Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive auto-encoders. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1129–1141, Minneapolis, Minnesota. 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*. MIT Press, Cambridge, MA, USA.

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting numerical reasoning skills into language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 946–958, Online. Association for Computational Linguistics.

Nitish Gupta and Mike Lewis. 2018. Neural compositional denotational semantics for question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2152–2161, Brussels, Belgium. Association for Computational Linguistics.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9:1735–1780.

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

Yichen Jiang and Mohit Bansal. 2019. Avoiding reasoning shortcuts: Adversarial evaluation, training, and model development for multi-hop QA. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2726–2736, Florence, Italy. Association for Computational Linguistics.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. 2017a. Clevr: A di-

agnostic dataset for compositional language and elementary visual reasoning. In *CVPR*.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. 2017b. Inferring and executing programs for visual reasoning. In *ICCV*.

Lukasz Kaiser and Ilya Sutskever. 2016. Neural gpus learn algorithms. *International Conference on Learning Representations*.

T. Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA†.

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*.

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.

Phong Le and Willem Zuidema. 2015. The forest convolutional network: Compositional distributional semantics with a neural chart and without binarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1155–1164, Lisbon, Portugal. Association for Computational Linguistics.

Percy Liang, Michael I Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

Yang Liu, Matt Gardner, and Mirella Lapata. 2018. Structured alignment networks for matching sentences. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1554–1564, Brussels, Belgium. Association for Computational Linguistics.

Jean Maillard, Stephen Clark, and Dani Yogatama. 2019. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *ArXiv*, abs/1705.09189.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. 2019. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations*.

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

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *AAAI*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683.

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. *ArXiv*, abs/2003.05161.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, and et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

Mark Steedman. 1996. Surface structure and interpretation.

Sanjay Subramanian, Ben Bogin, Nitish Gupta, Tomer Wolfson, Sameer Singh, Jonathan Berant, and Matt Gardner. 2020. Obtaining faithful interpretations from compositional neural networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5594–5608, Online. 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. *ArXiv*, abs/2005.09241.

Andrew Trask, Felix Hill, Scott E Reed, Jack Rae, Chris Dyer, and Phil Blunsom. 2018. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*, pages 8035–8044.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B. Tenenbaum. 2018. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *Advances in Neural Information Processing Systems*, pages 1039–1050.

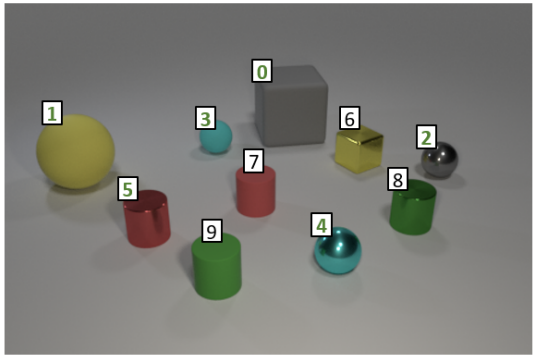
D.H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

Wojciech Zaremba and Ilya Sutskever. 2014. Learning to execute. *ArXiv*, abs/1410.4615.

Luke Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*.

A Output Examples

We show 3 additional examples of our model outputs, along with the induced trees and denotations in the following pages.



How many things are either green objects in front of the tiny green metal object or tiny objects in front of the large sphere? (5)

{all} = 0 1 2 / 3 4 5 / 6 7 8 / 9

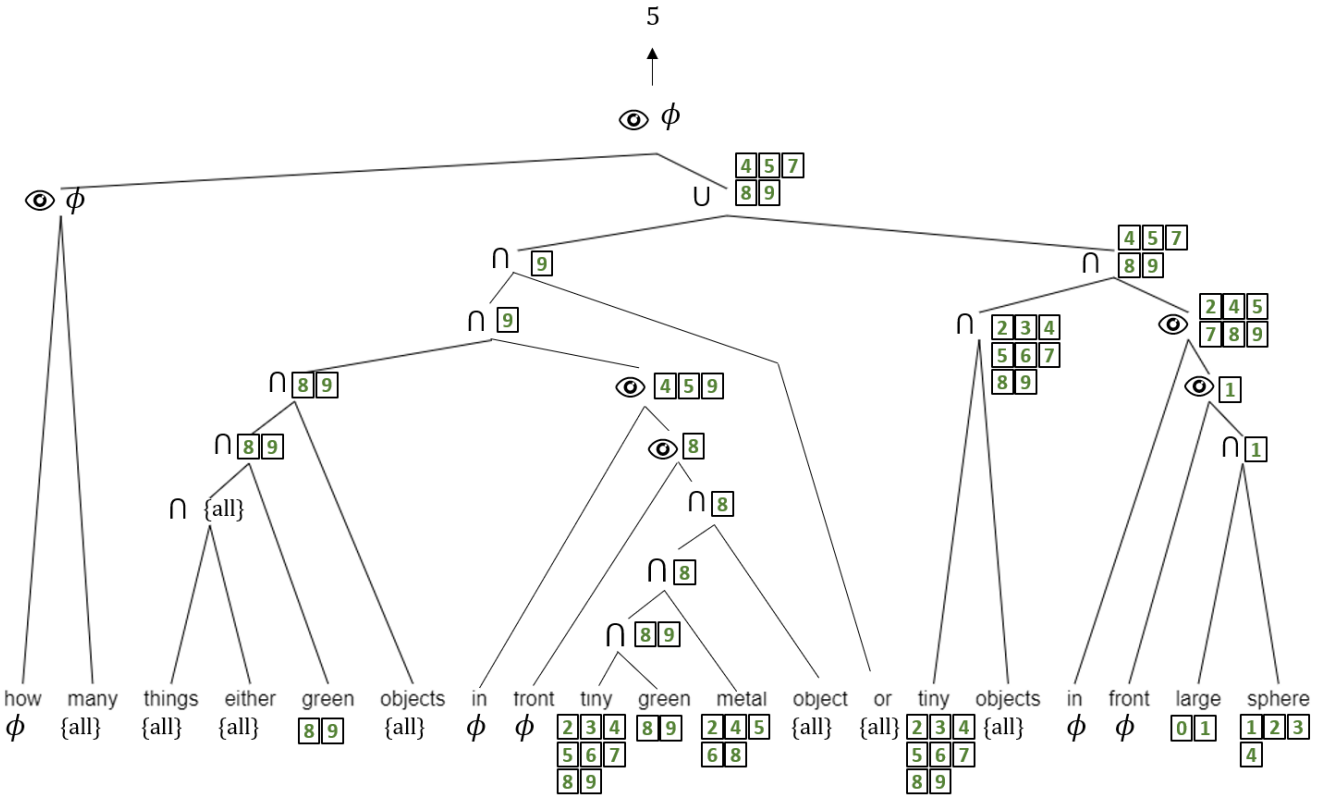
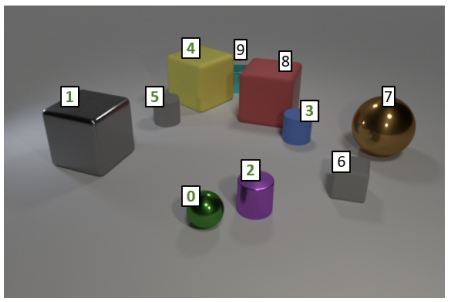


Figure 8: An example from CLOSURE. ∪ stands for the UNION module.



What is the color of the cube being covered by the yellow cube and the red cube? (cyan)

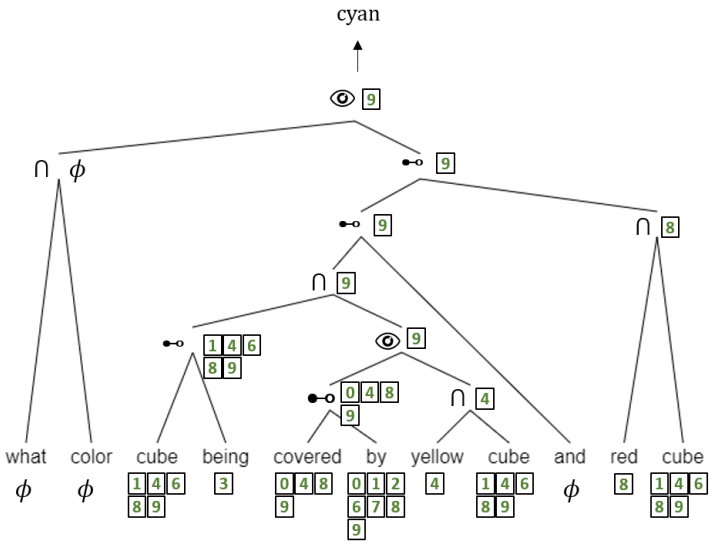
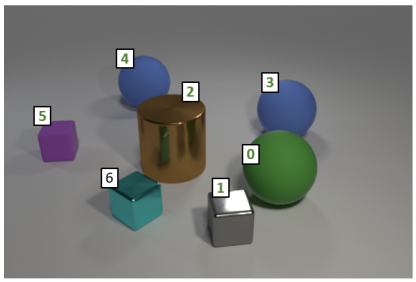


Figure 9: An example from CLEVR-Humans.



Are there more objects behind the large green ball than tiny cyan matte cylinders? (yes)

{all} =

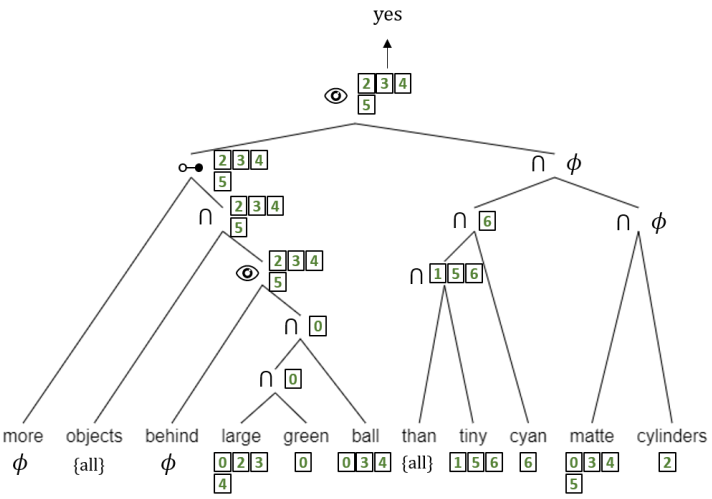


Figure 10: An example from CLEVR.