Explainable and Explicit Visual Reasoning over Scene Graphs

Jiaxin Shi¹* Hanwang Zhang² Juanzi Li¹
¹Tsinghua University  ²Nanyang Technological University
shijx12@gmail.com; hanwangzhang@ntu.edu.sg; lijuanzi@tsinghua.edu.cn

Abstract

We aim to dismantle the prevalent black-box neural architectures used in complex visual reasoning tasks, into the proposed eXplainable and eXplicit Neural Modules (XNMs), which advance beyond existing neural module networks towards using scene graphs — objects as nodes and the pairwise relationships as edges — for explainable and explicit reasoning with structured knowledge. XNMs allow us to pay more attention to teach machines how to “think”, regardless of what they “look”. As we will show in the paper, by using scene graphs as an inductive bias, 1) we can design XNMs in a concise and flexible fashion, i.e., XNMs merely consist of 4 meta-types, which significantly reduce the number of parameters by 10 to 100 times, and 2) we can explicitly trace the reasoning-flow in terms of graph attentions. XNMs are so generic that they support a wide range of scene graph implementations with various qualities. For example, when the graphs are detected perfectly, XNMs achieve 100% accuracy on both CLEVR and CLEVR CoGenT, establishing an empirical performance upper-bound for visual reasoning; when the graphs are noisily detected from real-world images, XNMs are still robust to achieve a competitive 67.5% accuracy on VQAv2.0, surpassing the popular bag-of-objects attention models without graph structures.

1. Introduction

The prosperity of A.I. — mastering super-human skills in game playing [22], speech recognition [1], and image recognition [8, 20] — is mainly attributed to the “winning streak” of connectionism, more specifically, the deep neural networks [15], over the “old-school” symbolism, where their controversy can be dated back to the birth of A.I. in 1950s [18]. With massive training data and powerful computing resources, the key advantage of deep neural networks is the end-to-end design that generalizes to a large spec-

*The work was done when Jiaxin Shi was an intern at Nanyang Technological University.
Neural module networks (NMNs) [3, 12, 10, 17, 9, 26] show a promising direction in conferring reasoning ability for the end-to-end design by learning to compose the networks on-demand from the language counterpart, which implies the logical compositions. Take the question “How many objects are left of the red cube?” as an example, we can program the reasoning path into a composition of functional modules [17]: Attend[cube], Attend[red], Relate[left], and Count, and then execute them with the input image. We attribute the success of NMNs to the eXplainable and eXplicit (dubbed X) language understanding. By explicitly parsing the question into an explainable module assembly, NMNs effectively prevent the language-to-reasoning shortcut, which are frequent when using the implicit fused question representations [4, 6] (e.g., the answer can be directly inferred according to certain language patterns).

However, the vision-to-reasoning shortcut still exists as an obstacle on the way of NMNs towards the real X visual reasoning. This is mainly because that the visual perception counterpart is still attached to reasoning [17], which is inevitably biased to certain vision patterns. For example, on the CLEVR CoGenT task, which provides novel object attributes to test the model’s generalization ability (e.g., cubes are blue in the training set but red in the test set), we observe significant performance drop of existing NMNs [13, 17] (e.g., red cubes in the test set cannot be recognized as “cube”). Besides, the reusability of the current module design is limited. For example, the network structure of the Relate module in [17] must be carefully designed using a series of dilated convolutions to achieve good performance. Therefore, how to design a complete inventory of X modules is still an tricky engineering.

In this paper, we advance NMN towards X visual reasoning by using the proposed eXplainable and eXplicit Neural Modules (XNMs) reasoning over scene graphs. By doing this, we can insulate the “low-level” visual perception from the modules, and thus can prevent reasoning shortcut of both language and vision counterpart. As illustrated in Figure 1, a scene graph is the knowledge representation of a visual input, where the nodes are the entities (e.g., cylinder, horse) and the edges are the relationships between entities (e.g., left, ride). In particular, we note that scene graph detection per se is still a challenging task in computer vision [28], therefore, we allow XNMs to accept scene graphs with different detection qualities. For example, the left-hand side of Figure 1 is one extreme when the visual scene is clean and closed-vocabulary, e.g., in CLEVR [12], we can have almost perfect scene graphs where the nodes and edges can be represented by one-hot class labels; the right-hand side shows another extreme when the scene is cluttered and open-vocabulary in practice, the best we have might be merely a set of object proposals. Then, the nodes are RoI features and the edges are their concatenations.

Thanks to scene graphs, our XNMs only have 4 meta-types: 1) AttendNode, finding the queried entities, 2) AttendEdge, finding the queried relationships, 3) Transfer, transforming the node attentions along the attentive edges, and 4) Logic, performing basic logical operations on attention maps. All types are fully X as their outputs are pure graph attentions that are easily traceable and visible. Moreover, these meta modules are only specific to the generic graph structures, and are highly reusable to constitute different composite modules for more complex functions. For example, we do not need to carefully design the internal implementation details for the module Relate as in [17]; instead, we only need to combine AttendEdge and Transfer in XNMs.

We conduct extensive experiments on two visual Q&A benchmarks and demonstrate the following advantages of using XNMs reasoning over scene graphs:

1. We achieve 100% accuracy by using the ground-truth scene graphs and programs on both CLEVR [12] and CLEVR-CoGent, revealing the performance upper-bound of XNMs, and the benefits of disentangling “high-level” reasoning from “low-level” perception.
2. Our network requires significantly less parameters while achieves better performance than previous state-of-the-art neural module networks, due to the conciseness and high-reusability of XNMs.
3. XNMs are flexible to different graph qualities, e.g., it achieves competitive accuracy on VQA v2.0 [6] when scene graphs are noisily detected.
4. We show qualitative results to demonstrate that our XNMs reasoning is highly explainable and explicit.

2. Related Work

Visual Reasoning. It is the process of analyzing visual information and solving problems based on it. The most representative benchmark of visual reasoning is CLEVR [12], a diagnostic visual Q&A dataset for compositional language and elementary visual reasoning. The majority of existing methods on CLEVR can be categorized into two families: 1) holistic approaches [12, 21, 19, 11], which embed both the image and question into a feature space and infer the answer by feature fusion; 2) neural module approaches [3, 10, 13, 17, 9, 26], which first parse the question into a program assembly of neural modules, and then execute the modules over the image features for visual reasoning. Our XNM belongs to the second one but replaces the visual feature input with scene graphs.

Neural Module Networks. They dismantle a complex question into several sub-tasks, which are easier to answer and more transparent to follow the intermediate outputs. Modules are pre-defined neural networks that implement the corresponding functions of sub-tasks, and then are as-
Scene graphs have been shown effective in boosting several vision-language tasks [10, 13, 17]. In particular, the program generator is trained based on the human annotations of desired layout or with the help of reinforcement learning due to the non-differentiability of layout selection. Recently, Hu et al. [9] proposed StackNMM, which replaces the hard-layout with soft and continuous module layout and performs well even without layout annotations at all. Our XNM experiments on VQA2.0 follows their soft-program generator.

Recently, NS-VQA [26] firstly built the reasoning over the object-level structural scene representation, improving the accuracy on CLEVR from the previous state-of-the-art 99.1% [17] to an almost perfect 99.8%. Their scene structure consists of objects with detected labels, but lacked the relationships between objects, which limited its application on real-world datasets such as VQA2.0 [6]. In this paper, we propose a much more generic framework for visual reasoning over scene graphs, including object nodes and relationship edges represented by either labels or visual features. Our scene graph is more flexible and more powerful than the table structure of NS-VQA.

Scene Graphs. This task is to produce graph representations of images in terms of objects and their relationships. Scene graphs have been shown effective in boosting several vision-language tasks [14, 24, 27, 5]. To the best of our knowledge, we are the first to design neural module networks that can reason over scene graphs. However, scene graph detection is far from satisfactory compared to object detection [25, 28, 16]. To this end, our scene graph implementation also supports cluttered and open-vocabulary in real-world scene graph detection, where the nodes are merely RoI features and the edges are their concatenations.

3. Approach

We build our neural module network over scene graphs to tackle the visual reasoning challenge. As shown in Figure 2, given an input image and a question, we first parse the image into a scene graph and parse the question into a module program, and then execute the program over the scene graph. In this paper, we propose a set of generic base modules that can conduct reasoning over scene graphs — eXplainable and eXplicit Neural Modules (XNMs) — as the reasoning building blocks. We can easily assemble these XNMs to form more complex modules under specific scenarios. Besides, our XNMs are totally attention-based, making all the intermediate reasoning steps transparent.

3.1. Scene Graph Representations

We formulate the scene graph of an image as \((V, E)\), where \(V = \{v_1, \cdots, v_N\}\) are graph nodes corresponding to \(N\) detected objects, and \(v_i\) denotes the feature representation of the \(i\)-th object. \(E = \{e_{ij} | i, j = 1, \cdots, N\}\) are graph edges corresponding to relations between each object pairs, and \(e_{ij}\) denotes the feature representation of the relation from object \(i\) to object \(j\) (Note that edges are directed).

Our XNMs are generic for scene graphs of different quality levels of detection. We consider two extreme settings in this paper. The first is the ground-truth scene graph with labels, denoted by GT, that is, using ground-truth objects as nodes, ground-truth object label embeddings as node features, and ground-truth relation label embeddings as edge features. In this setting, scene graphs are annotated with fixed-vocabulary object labels and relationship labels, e.g., defined in CLEVR dataset [12]. We collect all the \(C\) labels into a dictionary, and use an embedding matrix \(D \in \mathbb{R}^{C \times d}\) to map a label into a \(d\)-dimensional vector. We represent the nodes and edges using the concatenation of their corresponding label embeddings.

The second setting is totally detected and label-agnostic, denoted by Det, that is, using detected objects as nodes, RoI visual features as node features, and the fusion of two node features as the edge features. For example, the edge features can be represented by concatenating the two related node features, i.e., \(e_{ij} = [v_i; v_j]\). As another example, in CLEVR where the edges are only about spatial relationships, we use the difference between detected coordinates of object pairs as the edge embedding. More details are in Section 4.

We use the GT setting to demonstrate the performance upper-bound of our approach when a perfect scene graph detector is available along with the rapid development of visual recognition, and use the Det setting to demonstrate the practicality in open domains.

3.2. X Neural Modules

As shown in Figure 1, our XNMs have four meta-types and are totally attention-based. We denote the node attention weight vector by \(a \in [0, 1]^N\) and the weight of the \(i\)-th node by \(a_i\). The edge attention weight matrix is denoted by \(W \in [0, 1]^{N \times N}\), where \(W_{ij}\) represents the weight of edge from node \(i\) to node \(j\).

AttendNode[query]. This most basic and intuitive operation is to find the relevant objects given an input query (e.g., find all ["cubes"] ). For the purpose of semantic computation, we first encode the query into a vector \(q\). This X module takes the query vector as input, and produces the node attention vector by the following function:

\[
a = f(V, q).
\] (1)

The implementation of \(f\) is designed according to a specific scene graph representation, as long as \(f\) is differentiable and \(\text{range}(f) = [0, 1]\).

AttendEdge[query]. Though object attention is a widely-used mechanism for better visual understanding, it is unable to capture the interaction between objects and thus
is weak in the complex visual reasoning [29]. This X module aims to find the relevant edges given an input query (e.g., find all edges that are “left”). After encoding the query into q, we compute the edge attention matrix by the following function:

$$W = g(E, q),$$  

(2)

where g is defined according to a specific scene graph representation, as long as g is differentiable and range(g) = [0, 1].

**Transfer.** With the node attention vector a and the edge attention matrix W, we can transfer the node weights along the attentive relations to find new objects (e.g., find objects that are “left” to the “cube”). Thanks to the graph structure, to obtain the updated node attention a’, we merely need to perform a simple matrix multiplication:

$$a' = \text{norm}(W^T a),$$  

(3)

where norm asserts the values in [0, 1] by dividing the maximum value if any entry exceeds 1. Here, W_{ij} indicates how many weights will flow from object i to object j, and \(a'_i = \sum_{j=1}^{N} W_{ij}a_j\) is the total received weights of object i. This module reallocates node attention in an efficient and fully-differentiable manner.

**Logic.** Logical operations are crucial in complex reasoning cases. In XNM, logical operations are performed on one or more attention weights to produce a new attention. We define three logical X modules: **And**, **Or**, and **Not**. Without loss of generality, we discuss all these logical modules on node attention vectors, and the extension to edge attention is similar. The **And** and **Or** modules are binary, that is, take two attentions as inputs, while the **Not** module is unary. The implementation of these logical X modules are as follows:

$$\text{And}(a^1, a^2) = \min(a^1, a^2), \text{Not}(a) = 1 - a,$$

$$\text{Or}(a^1, a^2) = \max(a^1, a^2).$$

These four meta-types of XNs constitute the base of our graph reasoning. They are explicitly executed on attention maps, and all intermediate results are explainable. Besides, these X modules are totally differentiable. We can flexibly assemble them into composite modules for more complex functions, which can be still trained end-to-end.

### 3.3. Implementations

To apply XNs in practice, we need to consider these questions: (1) How to implement the attention functions \(f, g\) in Eq. (1) and Eq. (2)? (2) How to compose our X modules into composite reasoning modules? (3) How to predict the answer according to the attentive results? (4) How to parse the input question to an executable module program?

#### 3.3.1 Attention Functions

We use different attention functions for different scene graph settings. In the GT setting, as annotated labels are mostly mutually exclusive (e.g., “red” and “green”), we compute the node attention using the softmax function over the label space. Specifically, given a query vector \(q \in \mathbb{R}^d\), we first compute its attention distribution over all labels by \(b = \text{softmax}(D - q)\), where length(D) = \(C\) and \(b_c\) represents the weight of the \(c\)-th label. Then we capture the

---

*Figure 2: To answer a question about an image, we need to 1) parse the image into a scene graph, 2) parse the question into a module program, and 3) reasoning over the scene graph. Here, we show the reasoning details of an example from CLEVR. The nodes and edges in red are attended. Scene is a dummy placeholder module that attends all nodes. All intermediate steps of our XNs are explainable and explicit.*

8379
node and edge attention by summing up corresponding label weights:

\[ a_i = f(V, q)_i = \sum_{c \in C_i} b_c, \quad W_{ij} = g(E, q)_{ij} = \sum_{c \in C_{ij}} b_c, \]  

(5)

where \( C_i \) and \( C_{ij} \) denote the (multi-) labels of node \( i \) and edge \( ij \) respectively.

In the Det setting, we use the sigmoid function to compute the attention weights. Given the query \( q \in \mathbb{R}^d \), the node and edge attentions are:

\[ a_i = f(V, q)_i = \text{sigmoid}(\text{MLP}(v_i)^	op q), \]

\[ W_{ij} = g(E, q)_{ij} = \text{sigmoid}(\text{MLP}(e_{ij})^	op q), \]  

(6)

where the MLP maps \( v_i \) and \( e_{ij} \) to the dimension \( d \).

### 3.3.2 Composite Reasoning Modules

We list our composite reasoning modules and their implementations (i.e., how they are composed by basic X modules) in the top section of Table 1. For example, Same module is to find other objects that have the same attribute value as the input objects (e.g., find other objects with the same [“color”]). In particular, Describe used in Same is to obtain the corresponding attribute value (e.g., describe one object’s [“color”]), and will be introduced in the following section.

### 3.3.3 Feature Output Modules

Besides the above reasoning modules, we also need another kind of modules to map the intermediate attention to a hidden embedding \( h \) for feature representation, which is fed into a softmax layer to predict the final answer, or into some modules for further reasoning. We list our output modules in the bottom section of Table 1. Exist and Count sum up the node attention weights to answer yes/no and counting questions. Compare is for attribute or number comparisons, which takes two hidden features as inputs. Describe[query] is to transform the attentive node features to an embedding that describes the specified attribute value (e.g., what is the [“color”] of attended objects).

To implement the Describe module, we first obtain the “raw” attentive node feature by

\[ \bar{v} = \frac{1}{\sum_{i=1}^{N} a_i} \sum_{i=1}^{N} a_i v_i, \]  

(7)

and then project it into several “fine-grained” sub-spaces — describing different attribute aspects such as color and shape — using different transformation matrices. Specifically, we define \( K \) projection matrices \( M_1, \ldots, M_K \) to map \( \bar{v} \) into different aspects (e.g., \( M_1 \bar{v} \) represents the color, \( M_2 \bar{v} \) represents the shape, etc.), where \( K \) is a hyper-parameter related to the specific scene graph vocabulary. The output feature is computed by

\[ \text{Describe}(a, q) = \sum_{k=1}^{K} c_k (M_k \bar{v}), \]  

(8)

where \( c = \text{Softmax}(\text{MLP}(q)) \) represents a probability distribution over these \( K \) aspects, and \( c_k \) denotes the \( k \)-th probability. The mapping matrixes can be learned end-to-end automatically.

### Table 1: Our composite modules (the top section) and output modules (the bottom section). MLP() consists of several linear and ReLU layers.

<table>
<thead>
<tr>
<th>Modules</th>
<th>In → Out</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersect</td>
<td>( a^1, a^2 \to a' )</td>
<td>And[( a^1, a^2 )]</td>
</tr>
<tr>
<td>Union</td>
<td>( a^1, a^2 \to a' )</td>
<td>Or[( a^1, a^2 )]</td>
</tr>
<tr>
<td>Filter</td>
<td>( a, q \to a' )</td>
<td>And(a, AttendNode(q))</td>
</tr>
<tr>
<td>Same</td>
<td>( a, q \to a' )</td>
<td>Filter(Not(a), Describe(a, q))</td>
</tr>
<tr>
<td>Relate</td>
<td>( a, q \to a' )</td>
<td>Transfer(a, AttendEdge(q))</td>
</tr>
<tr>
<td>Exist</td>
<td>( a \to h )</td>
<td>MLP(( \sum_i a_i ))</td>
</tr>
<tr>
<td>Count</td>
<td>( h^1, h^2 \to h' )</td>
<td>MLP(( h^1 - h^2 ))</td>
</tr>
<tr>
<td>Compare</td>
<td>( a, q \to h )</td>
<td>Eq. (8)</td>
</tr>
</tbody>
</table>

### 3.3.4 Program Generation & Training

For datasets that have ground-truth program annotations (e.g., CLEVR), we directly learn an LSTM sequence-to-sequence model [23] to convert the word sequence into the module program. However, there is no layout annotations in most real-world datasets (e.g., VQA v2.0). In this case, following StackMN [9], we make soft module selection with a differentiable stack structure. Please refer to their papers for more details.

We feed our output features from modules (cf. Table 1) into a softmax layer for the answer prediction. We use the cross entropy loss between our predicted answers and ground-truth answers to train our XNMs.

### 4. Experiments

#### 4.1. CLEVR

**Settings.** The CLEVR dataset [12] is a synthetic diagnostic dataset that tests a range of visual reasoning abilities. In CLEVR, images are annotated with ground-truth object positions and labels, and questions are represented as functional programs that consists of 13 kinds of modules. Except the “Unique” module, which does not have actual operation, all the remaining 12 modules can correspond to our modules in Table 1. CLEVR modules “Equal_attribute”, “Equal_integer”, “Greater_than” and “Less_than” have the same implementation as our Compare, but with different
Table 2: Comparisons between neural module networks on the CLEVR dataset. Top section: results of the official test set; Bottom section: results of the validation set (we can only evaluate our GT setting on the validation set since the annotations of the test set are not public [12]). The program option “scratch” means totally without program annotations, “supervised” means using trained end-to-end parser, and “GT” means using ground-truth programs. Our reasoning modules are composed of highly-reusable X modules, leading to a very small number of parameters. Using the ground-truth scene graphs and programs, we can achieve a perfect reasoning on all kinds of questions.

<table>
<thead>
<tr>
<th>Method</th>
<th>Program</th>
<th>#Modules</th>
<th>#Param.</th>
<th>Overall</th>
<th>Count</th>
<th>Compare</th>
<th>Exist</th>
<th>Query</th>
<th>Compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human [12]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>92.6</td>
<td>86.7</td>
<td>86.4</td>
<td>96.6</td>
<td>95.0</td>
<td>96.0</td>
</tr>
<tr>
<td>N2NMN [10]</td>
<td>scratch</td>
<td>12</td>
<td>-</td>
<td>69.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>N2NMN [10]</td>
<td>supervised</td>
<td>12</td>
<td>-</td>
<td>83.7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PG+EE [13]</td>
<td>supervised</td>
<td>39</td>
<td>40.4M</td>
<td>96.9</td>
<td>92.7</td>
<td>98.7</td>
<td>97.1</td>
<td>98.1</td>
<td>98.9</td>
</tr>
<tr>
<td>StackNMN [9]</td>
<td>scratch</td>
<td>9</td>
<td>7.32M</td>
<td>93.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>StackNMN [9]</td>
<td>supervised</td>
<td>9</td>
<td>7.32M</td>
<td>96.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>XNM-Det</td>
<td>supervised</td>
<td>12</td>
<td>0.55M</td>
<td>97.7</td>
<td>96.0</td>
<td>98.0</td>
<td>98.7</td>
<td>98.4</td>
<td>97.6</td>
</tr>
<tr>
<td>NS-VQA [26]</td>
<td>supervised</td>
<td>12</td>
<td>-</td>
<td>99.8</td>
<td>99.7</td>
<td>99.9</td>
<td>99.9</td>
<td>99.8</td>
<td>99.8</td>
</tr>
<tr>
<td>XNM-Det</td>
<td>supervised</td>
<td>12</td>
<td>0.55M</td>
<td>97.8</td>
<td>96.0</td>
<td>98.1</td>
<td>98.6</td>
<td>98.7</td>
<td>97.8</td>
</tr>
<tr>
<td>XNM-Det</td>
<td>GT</td>
<td>12</td>
<td>0.55M</td>
<td>97.9</td>
<td>96.2</td>
<td>98.1</td>
<td>98.8</td>
<td>98.7</td>
<td>97.8</td>
</tr>
<tr>
<td>XNM-GT</td>
<td>supervised</td>
<td>12</td>
<td>0.22M</td>
<td>99.9</td>
<td>99.9</td>
<td>99.9</td>
<td>99.8</td>
<td>99.8</td>
<td>99.9</td>
</tr>
<tr>
<td>XNM-GT</td>
<td>GT</td>
<td>12</td>
<td>0.22M</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

parameters. There are 4 attribute categories in CLEVR, so we set the number of mapping matrices \( K = 4 \).

We reused the trained sequence-to-sequence program generator of [13, 17], which uses prefix-order traversal to convert the program trees to sequences. Note that their modules are bundled with input, e.g., they regard Filter[red] and Filter[green] as two different modules. This will cause serious sparseness in the real-world case. We used their program generator, but unpack the module and the input (e.g., Filter[red] and Filter[green] are the same module with different input query).

In the GT setting, we performed reasoning over the ground-truth scene graphs. In the Det setting, we built the scene graphs by detecting objects and using RoI features as node embeddings and the differences between detected coordinates as edge embeddings. Since CLEVR does not provide the bounding box or segmentation annotations of objects, it is hard to directly train an object detector. NS-VQA [26] trained a Mask R-CNN [7] for object segmentation by ‘hacking’ the rendering process [12], which could perform very well due to the simplicity of visual scenes of CLEVR. However, as we expected to explore X modules in a noisier case, we chose the trained attention modules of TbD-net [17] as our object detector. Specifically, we enumerated all possible combinations of object attributes (e.g., red, cube, metal, large), and tried to find corresponding objects using their attention modules (e.g., intersection of the output mask of Attend[red], Attend[cube], Attend[metal] and Attend[large], and then regarded each clique as a single object). The detected results have some frequent mistakes, such as inaccurate position, wrongly merged nodes (two adjacent objects with the same attribute values are recognized as one). These detection noises allow us to test whether our XNMs are robust enough.

**Goals.** We expect to answer the following questions according to the CLEVR experiments: **Q1:** What is the upper bound of our X reasoning when both the vision and language perceptions are perfect? **Q2:** Are our XNMs robust for noisy detected scene graphs and parsed programs? **Q3:** What are the parameter and data efficiency, and the convergence speed of XNMs? **Q4:** How is the explainability of XNMs?

**Results.** Experimental results are listed in Table 2. **A1:** When using the ground-truth scene graphs and programs, we can achieve 100% accuracy, indicating an inspiring upper-bound of visual reasoning. By disentangling “high-level” reasoning from “low-level” perception and using XNMs, we may eventually conquer the visual reasoning challenge with the rapid development of visual recognition. **A2:** With noisy detected scene graphs, we can still achieve a competitive 97.9% accuracy using the ground-truth programs, indicating that our X reasoning are robust to different quality levels of scene graphs. When replacing the ground-truth programs with parsed programs, the accu-
accuracy drops by 0.1% in both GT and Det settings, which is caused by minor errors of the program parser.

**A3:** Due to the conciseness and high-reusability of X modules, our model requires significantly less parameters than existing models. Our GT setting only needs about 0.22M parameters, taking about 500MB memory with batch size of 128, while PG+EE [13] and TbD-net [17] bundle modules and inputs together, leading to a large number of modules and parameters.

To explore the data efficiency, we trained our model with a partial training set and evaluated on the complete validation set. Results are displayed in the left part of Figure 3. We can see that our model performs much better than other baselines when the training set is small. Especially, our GT setting can still achieve a 100% accuracy even with only 10% training data. The right part shows the accuracy at each training epoch. We can see that our X reasoning converges very fast.

**A4:** As our XNMs are attention-based, the reasoning process is totally transparent and we can easily show intermediate results. Figure 4 displays two examples of CLEVR. We can see all reasoning steps are clear and intuitive.

### 4.2. CLEVR-CoGenT

**Settings.** The CLEVR-CoGenT dataset is a benchmark to study the ability of models to recognize novel combinations of attributes at test-time, which is derived from CLEVR but has two different conditions: in Condition A all cubes are colored one of gray, blue, brown, or yellow, and all cylinders are one of red, green, purple, or cyan; in Condition B the color palettes are swapped. The model is trained using the training set of Condition A, and then is tested using Condition B to check whether it can generalize well to the novel attribute combinations. We train our model on the training set of Condition A, and report the accuracy of both conditions.

**Goals.** Q1: Can our model perform well when meeting the novel attribute combinations? Q2: If not, what actually causes the reasoning shortcut?

**Results.** Results of CLEVR-CoGenT are displayed in Table 3. A1: When using the ground-truth scene graphs, our XNMs perform perfectly on both Condition A and Condition B. Novel combinations of attributes in Condition B do not cause the performance drop at all. However, when using the detected scene graphs, where node embeddings are RoI features that fuse all attribute values, our generalization results on Condition B drops to 72.1%, suffering from the dataset shortcut just like other existing models [13, 17].

<table>
<thead>
<tr>
<th>Method</th>
<th>Program</th>
<th>Condition A</th>
<th>Condition B</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG+EE [13]</td>
<td>supervised</td>
<td>96.6</td>
<td>73.7</td>
</tr>
<tr>
<td>TbD-net [17]</td>
<td>supervised</td>
<td><strong>98.8</strong></td>
<td><strong>75.4</strong></td>
</tr>
<tr>
<td>XNM-Det</td>
<td>supervised</td>
<td>98.1</td>
<td>72.6</td>
</tr>
<tr>
<td>NS-VQA [26]</td>
<td>supervised</td>
<td>99.8</td>
<td>63.9</td>
</tr>
<tr>
<td>XNM-GT</td>
<td>GT</td>
<td>98.3</td>
<td>72.2</td>
</tr>
<tr>
<td>XNM-GT</td>
<td>supervised</td>
<td><strong>100</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Filter (purple)</th>
<th>Filter (cube)</th>
<th>Filter (cylinder)</th>
<th>Query (shapes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>![cylinder]</td>
<td>![cylinder]</td>
<td>![cylinder]</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: Failure cases of our Det setting on Condition B of CoGenT.
Figure 4: Reasoning visualizations of two CLEVR samples. Question 1: What number of objects are either big objects that are behind the big gray block or tiny brown rubber balls? Question 2: The other small shiny thing that is the same shape as the tiny yellow shiny object is what color? We plot a dot for each object and darker (red) dots indicate higher attention weights.

4.3. VQA v2.0

Settings. VQA v2.0 [6] is a real-world visual Q&A dataset which does not have annotations about scene graphs and module programs. We used the grounded visual features of [2] as node features, and concatenated node embeddings as edge features. We set $K = 1$ and fused the question embedding with our output feature for answer prediction. Following [2], we used softmax over objects for node attention computation.

Goals. We used VQA v2.0 to demonstrate the generality and robustness of our model in the practical case.

Results. We list the results in Table 4. We follow Stack-NMN [9] to build the module program in a stacked soft manner, but our model can achieve better performance as our reasoning over scene graphs is more powerful than their pixel-level operations.

Recall that [13, 17] are not applicable in open-vocabulary input, and [26] relies on the fixed label representation, so it is hard to apply them on practical datasets. In contrast, our XNMs are flexible enough for different cases.

5. Conclusions

In this paper, we proposed X neural modules (XNMs) that allows visual reasoning over scene graphs, represented by different detection qualities. Using the ground-truth scene graphs and programs on CLEVR, we can achieve 100% accuracy with only 0.22M parameters. Compared to existing neural module networks, XNMs disentangle the “high-level” reasoning from the “low-level” visual perception, and allow us to pay more attention to teaching A.I. how to “think”, regardless of what they “look”. We believe that this is an inspiring direction towards explainable machine reasoning. Besides, our experimental results suggest that visual reasoning benefits a lot from high-quality scene graphs, revealing the practical significance of the scene graph research.

Acknowledgments. The work is supported by NSFC key projects (U1736204, 61661146007, 61533018), Ministry of Education and China Mobile Research Fund (No. 20181770250), THUNUS NExT Co-Lab, and Alibaba-NTU JRI.
References


