META MODULE NETWORK FOR COMPOSITIONAL VISUAL REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

There are two main lines of research on visual reasoning: neural module network (NMN) with *explicit* multi-hop reasoning through handcrafted neural modules, and monolithic network with *implicit* reasoning in the latent feature space. The former excels in interpretability and compositionality, while the latter usually achieves better performance due to model flexibility and parameter efficiency. In order to bridge the gap of the two, we present Meta Module Network (MMN), a novel hybrid approach that can efficiently utilize a Meta Module to perform versatile functionalities, while preserving compositionality and interpretability through modularized design. The proposed model first parses an input question into a functional program through a Program Generator. Instead of handcrafting a task-specific network to represent each function like traditional NMN, we use Recipe Encoder to translate the functions into their corresponding recipes (specifications), which are used to dynamically instantiate the Meta Module into Instance Modules. To endow different instance modules with designated functionality, a Teacher-Student framework is proposed, where a symbolic teacher pre-executes against the scene graphs to provide guidelines for the instantiated modules (student) to follow. In a nutshell, MMN adopts the meta module to increase its parameterization efficiency, and uses recipe encoding to improve its generalization ability over NMN. Experiments conducted on the GQA benchmark demonstrates that: (i) MMN achieves significant improvement over both NMN and monolithic network baselines; (ii) MMN is able to generalize to unseen but related functions.

1 INTRODUCTION

Visual reasoning requires a model to learn strong compositionality and generalization abilities, i.e., understanding and answering compositional questions without having seen similar semantic compositions before. Such compositional visual reasoning is a hallmark for human intelligence that endows people with strong problem-solving skills given limited prior knowledge. Recently, neural module networks (NMNs) (Andreas et al., 2016a;b; Hu et al., 2017; Johnson et al., 2017b; Hu et al., 2018; Mao et al., 2019) have been proposed to perform such complex reasoning tasks. First, NMN needs to pre-define a set of functions and explicitly encode each function into unique shallow neural networks called modules, which are composed dynamically to build an instance-specific network for each input question. This approach has high compositionality and interpretability, as each module is specifically designed to accomplish a specific sub-task and multiple modules can be combined to perform unseen combinations during inference. However, with increased complexity of the task, the set of functional semantics and modules also scales up. As observed in Hudson & Manning (2018), this leads to higher model complexity and poorer scalability on more challenging scenarios.

Another line of research on visual reasoning is focused on designing monolithic network architecture, such as MFB (Yu et al., 2017), BAN (Kim et al., 2018), DCN (Nguyen & Okatani, 2018), and MCAN (Yu et al., 2019). These black-box methods have achieved state-of-the-art performance on more challenging realistic image datasets like VQA (Hudson & Manning, 2019a), surpassing the aforementioned NMN approach. They use a unified neural network to learn general-purpose reasoning skills (Hudson & Manning, 2018), which is known to be more flexible and scalable without making strict assumption about the inputs or designing operation-specific networks for the pre-defined functional semantics. As the reasoning procedure is conducted in the latent feature space, the reasoning process is difficult to interpret. Such a model also lacks the ability to capture the compositionality of questions, thus suffering from poorer generalizability than module networks.

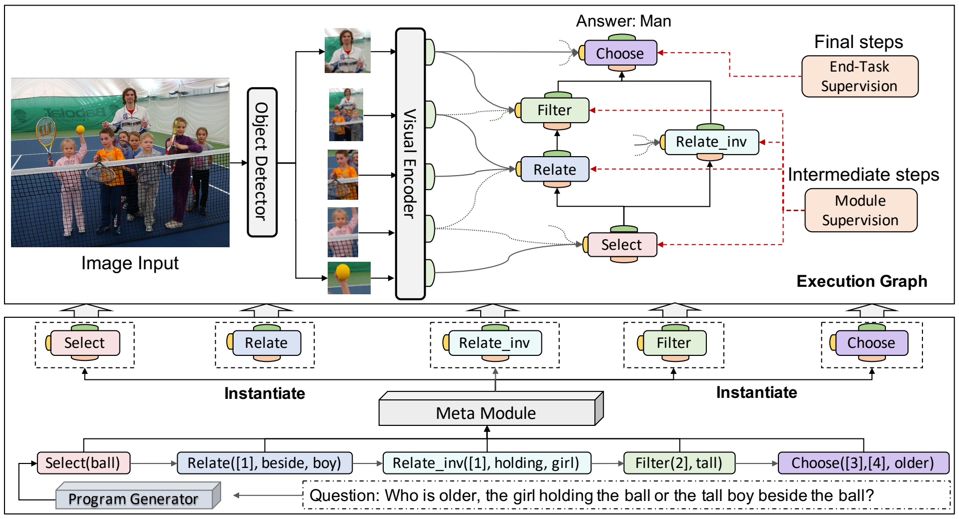

Figure 1: The model architecture of Meta Module Network: the lower part describes how the question is translated into programs and instantiated into operation-specific modules; the upper part describes how execution graph is built based on the instantiated modules.

Motivated by this, we propose a Meta Module Network (MMN) to bridge the gap, which preserves the merit of interpretability and compositionality of traditional module networks, but without requiring strictly defined modules for different semantic functionality. As illustrated in Figure 1, instead of handcrafting a shallow neural network for each specific function like NMNs, we propose a flexible meta (parent) module $g(*, *)$ that can take a function recipe f as input and instantiates a (child) module $g_f(*) = g(*, f)$ to accomplish the functionality specified in the recipe. These instantiated modules with tied parameters are used to build an execution graph for answer prediction. The introduced meta module empowers the MMN to scale up to accommodate a larger set of functional semantics without adding complexity to the model itself. To endow each instance module with the designated functionality, we introduce module supervision to enforce each module $g_f(*)$ to imitate the behavior of its symbolic teacher learned from ground-truth scene graphs provided in the training data. The module supervision can dynamically disentangle different instances to accomplish small sub-tasks to maintain high compositionality.

Our main contributions are summarized as follows. (i) We propose Meta Module Network for visual reasoning, in which different instance modules can be instantiated from a meta module. (ii) Module supervision is introduced to endow different functionalities to different instance modules. (iii) Experiments on GQA benchmark validate the outperformance of our model over NMN and monolithic network baselines. We also qualitatively provide visualization on the inferential chain of MMN to demonstrate its interpretability, and conduct experiments to quantitatively showcase the generalization ability to unseen functional semantics.

2 Meta Module Network

The visual reasoning task (Hudson & Manning, 2019a) is formulated as follows: given a question Q grounded in an image I, where $Q = \{q_1, \cdots, q_M\}$ with q_i representing the i-th word, the goal is to select an answer $a \in \mathbb{A}$ from a set $\mathbb{A}$ of possible answers. During training, we are provided with an additional scene graph G for each image I, and a functional program P for each question Q. During inference, scene graphs and programs are not provided.

Figure 1 provides an overview of Meta Module Network (MMN), which consists of three components: (i) Visual Encoder (Sec. 2.1), which consists of self-attention and cross-attention layers on top of an object detection model, transforming an input image into object-level feature vectors; (ii) Program Generator (Sec. 2.2), which generates a functional program from the input question; (iii) Meta Module (Sec. 2.3), which can be instantiated to different instance modules to execute the program for answer prediction. The following sub-sections describe each component in detail.

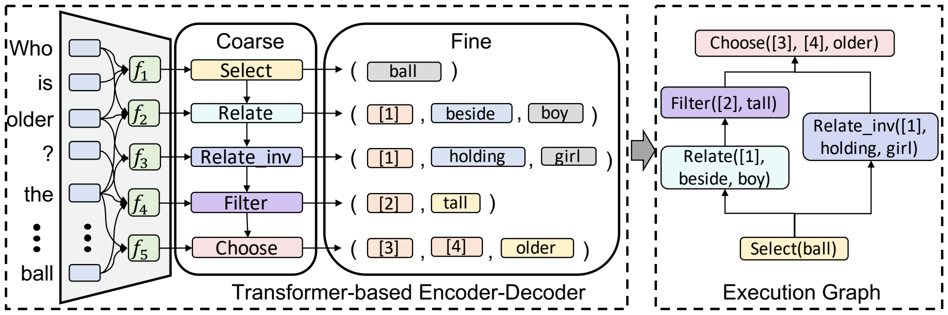

Figure 2: Architecture of the Coarse-to-fine Program Generator: the left part depicts the coarse-to-fine two-stage generation; the right part depicts the resulting execution graph.

2.1 Visual Encoder

The Visual Encoder is based on a pre-trained object detection model (Ren et al., 2015; Anderson et al., 2018) that extracts from image I a set of regional features $\mathbf{R} = \{\mathbf{r}_i\}_{i=1}^{N}$, where $\mathbf{r}_i \in \mathbb{R}^{D_v}$, N denotes the number of region of interest, and D_v denotes the feature dimension. Similar to a Transformer block (Vaswani et al., 2017), we first use two self-attention networks, SA_q and SA_r, to encode the question and the regional features as $\hat{\mathbf{Q}} = SA_q(Q, Q; \phi)$ and $\hat{\mathbf{R}} = SA_r(\mathbf{R}, \mathbf{R}; \phi)$, respectively, where $\hat{\mathbf{Q}} \in \mathbb{R}^{M \times D}$, $\hat{\mathbf{R}} \in \mathbb{R}^{N \times D}$, and D is the network's hidden dimension. Based on this, a cross-attention network CA is applied to use the question as guidance to refine the visual features into $\mathbf{V} = CA(\hat{\mathbf{R}}, \hat{\mathbf{Q}}; \phi) \in \mathbb{R}^{N \times D}$, where $\hat{\mathbf{Q}}$ is used as the query vector, and ϕ denotes all the parameters in the Visual Encoder. The attended visual features $\mathbf{V}$ will then be fed into the meta module, detailed in Sec. 2.3. We visualize the encoder in the Appendix for better illustration.

2.2 Program Generator

Similar to other programming languages, we define a set of syntax rules for building valid programs and a set of semantics to determine the functionality of each program. Specifically, we define a set of functions $\mathcal{F}$ with their fixed arity $n_f \in \{1, 2, 3, 4\}$ based on the semantic string provided in Hudson & Manning (2019a). The definitions for all the functions are provided in the Appendix. The defined functions can be divided into 10 different categories based on their abstract semantics (e.g., "relate, verify, filter"), and each abstract function type is further implemented with different realizations depending on their arguments (e.g., "verify_attribute, verify_geometric, verify_relation").

In total, there are 48 different functions defined, whose returned values could be *List of Objects*, *Boolean* or *String*. A program P is viewed as a sequence of function calls $f_1, \cdots, f_L$. For example, in Figure 2, f_2 is `Relate([1], beside, boy)`, the functionality of which is to find a boy who is beside the objects returned by f_1 : `Select(ball)`. Formally, we call `Relate` the "function name", `[1]` the "dependency", and `beside, boy` the "arguments". By exploiting the dependency relationship between functions, we build an execution graph for answer prediction.

In order to generate syntactically plausible programs, we follow Dong & Lapata (2018) and adopt a coarse-to-fine two-stage generation paradigm, as illustrated in Figure 2. Specifically, the Transformer-based program generator (Vaswani et al., 2017) first decodes a sketch containing only function names, and then fills the dependencies and arguments into the sketch to generate the program P. Such a two-stage generation process helps guarantee the plausibility and grammaticality of synthesized programs. We apply the known constraints to enforce the syntax in the fine-grained generation stage. For example, if function `Filter` is sketched, we know there are two tokens required to complete the function. The first token should be selected from the dependency set (`[1]`, `[2]`, ...), while the second token should be selected from the attribute set (e.g., `color`, `size`). With these syntactic constraints, our program synthesizer can achieve a 98.8% execution accuracy.

2.3 Meta Module

Instead of learning a full inventory of task-specific modules for different functions as in NMN (Andreas et al., 2016b), we design an abstract Meta Module that can instantiate a generic meta mod-

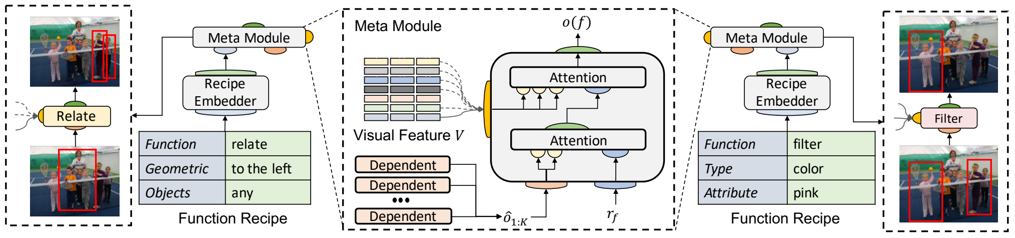

Figure 3: Illustration of the instantiation process for "Relate" and "Filter" functions.

ule into instance modules based on an input function recipe, which is a set of pre-defined key-value pairs specifying the properties of the function. As exemplified in Figure 3, when taking `Function:relate; Geometric:to the left` as the input, the Recipe Embedder produces a recipe vector to transform the meta module into a "geometric relation" module, which can search for target objects that the current object is to the left of.

Two-layered Attention: The left part of Figure 3 demonstrates the computation flow in Meta Module based on multi-head attention network (Vaswani et al., 2017). Specifically, a Recipe Embedder encodes a function recipe into a real-valued vector $\mathbf{r}_f \in \mathbb{R}^D$. In the first attention layer, $\mathbf{r}_f$ is fed into an attention network g_d as the query vector to incorporate the output ($\hat{\mathbf{o}}_{1:K}$) of neighbor modules on which the current module is dependent. The intermediate output ($\mathbf{o}_d$) from this attention layer is further fed into a second attention network g_v to incorporate the visual representation $\mathbf{V}$ of the image. The final output from the is denoted as $g(\mathbf{r}_f, \hat{\mathbf{o}}_{1:K}, \mathbf{V}) = g_v(g_d(\mathbf{r}_f, \hat{\mathbf{o}}_{1:K}), \mathbf{V})$.

Instantiation & Execution: Here is how the instantiation process of Meta Module works. First, we feed a function f to instantiate the meta module g into an instance module $g_f(\hat{\mathbf{o}}_{1:K}, \mathbf{V}; \psi)$, where ψ denotes the parameters of the meta module. The instantiated module is then used to build the execution graph on the fly as depicted in Figure 1. Each module g_f outputs $\mathbf{o}(f) \in \mathbb{R}^D$, which acts as the message passed to its neighbor modules. For brevity, we use $\mathbf{o}(f_i)$ to denote the MMN's output at the i-th function f_i. The final output $\mathbf{o}(f_L)$ of function f_L will be fed into a softmax-based classifier for answer prediction. During training, we optimize the parameters ψ (in Meta Module) and the parameters ϕ (in Visual Encoder) to maximize the likelihood $p_{\phi,\psi}(a|P, Q, \mathbf{R})$ on the training data, where a is the answer, and $P, Q, \mathbf{R}$ are programs, questions and visual features, respectively.

As demonstrated, Meta Module Network excels over standard module network in the following aspects. (i) The parameter space of different functions is shared, which means similar functions can be jointly optimized, benefiting from more efficient parameterization. For example, `query_color` and `verify_color` share the same partial parameters related to the input `color`. (ii) Our Meta Module can accommodate larger function semantics by using function recipes and scale up to more complex reasoning scenes. (iii) Since all the functions are embedded into the recipe space, functionality of an unseen recipe can be inferred from its neighboring recipes (see Sec. 3.4 for details), which equips our Meta Module with better generalization ability to unseen functions.

2.4 Module Supervision

In this sub-section, we explain how to extract supervision signals from scene graphs and programs provided in the training data, and how to adapt these learning signals during inference when no scene graphs or programs are available. We call this "Module Supervision", which is realized by a Teacher-Student framework as depicted in Figure 4. First, we define a Symbolic Executor as the 'Teacher', which can traverse the ground-truth scene graph provided in training data and obtain intermediate results by executing the programs. The 'Teacher' exhibits these results as guideline γ for the 'Student' instance module g_f to adhere to during training.

Symbolic Teacher: We first pre-execute the program $P = f_1, \cdots, f_L$ on the ground-truth scene graph G provided in the training data to obtain all the intermediate execution results. According to the function definition (see Appendix for details), the intermediate results are either *List of Objects* or *Boolean*. If the result is: (i) Non-empty *List of Objects*: use the first element's vertexes $[x_1, y_1, x_2, y_2]$ to represent it; (ii) Empty *List of Objects*: use dummy vertexes $[0, 0, 0, 0]$ as the default representation; (iii) "True" from *Boolean*: use the vertexes from last step to represent it; (iv) "False" from *Boolean*: use dummy vertexes as in (ii). Therefore, the intermediate results from

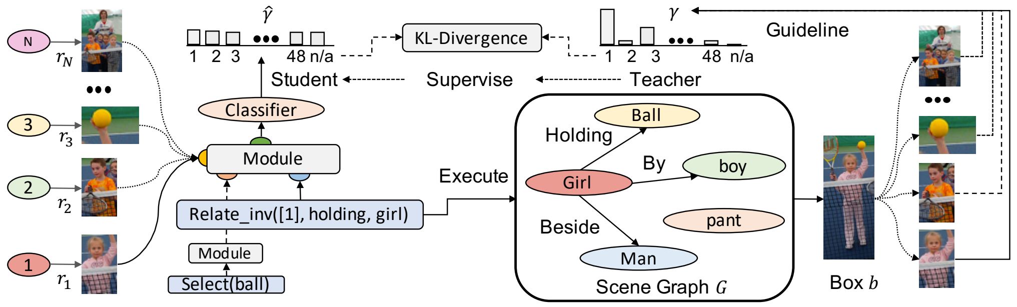

Figure 4: Illustration of the Module Supervision process: the symbolic teacher executes the function on the scene graph to obtain the bounding box b, which is then aligned with bounding boxes from the object detection model to compute the distribution guideline γ for supervision.

$(f_1, \cdots, f_{L-1})$ are unified into a series of quadruples denoted as $\{b_i\}_{i=1}^{L-1}$.

Knowledge Transfer: As no scene graphs are provided during inference, we need to train a Student to mimic the Symbolic Teacher in associating objects between input images and generated programs for end-to-end model training. To this end, we compare the execution results from the Symbolic Teacher with object detection results from the Visual Encoder to provide learning guideline for the Student. Specifically, for the i-th step function f_i, we compute the overlap between its execution result b_i and all the model-detected regions R as $a_{i,j} = \frac{Intersect(b_i, r_j)}{Union(b_i, r_j)}$. If $\sum_j a_{i,j} > 0$, which means that there exists detected bounding boxes overlapping with the ground-truth object, we normalize $a_{i,j}$ over R to obtain a guideline distribution $\gamma_{i,j} = \frac{a_{i,j}}{\sum_j a_{i,j}}$ and append an extra 0 in the end to obtain $\gamma_i \in \mathbb{R}^{N+1}$. If $\sum_j a_{i,j} = 0$, which means no detected bounding box has overlap with the ground-truth object (or $b_i = [0, 0, 0, 0]$), we use the one-hot distribution $\gamma_i = [0, \cdots, 0, 1] \in \mathbb{R}^{N+1}$ as the learning guideline. The last bit represents “No Match”.

Student Training: To explicitly teach the student module g_f to follow the learning guideline from the Symbolic Teacher, we add an additional head to each module output $\mathbf{o}(f_i)$ to predict the execution result distribution, denoted as $\hat{\gamma}_i = softmax(MLP(\mathbf{o}(f_i)))$. During training, we propel the instance module to align its prediction $\hat{\gamma}_i$ with the guideline distribution γ_i by minimizing their KL divergence $KL(\gamma_i || \hat{\gamma}_i)$. Formally, given the quadruple of $(P, Q, \mathbf{R}, a)$ and the pre-computed guideline distribution γ, we propose to add KL divergence to the standard loss function with a balancing factor η: $\mathcal{L}(\phi, \psi) = -\log p_{\phi,\psi}(a|P, Q, \mathbf{R}) + \eta \sum_{i=1}^{L-1} KL(\gamma_i || \hat{\gamma}_i)$.

3 Experiments

In this section, we conduct the following experiments. (i) We evaluate the proposed Meta Module Network on the GQA v1.1 dataset (Hudson & Manning, 2019a), and compare with the state-of-the-art methods. (ii) We provide visualization of the inferential chains and perform fine-grained error analysis based on that. (iii) We design synthesized experiments to quantitatively measure our model’s generalization ability towards unseen functional semantics.

3.1 Experimental Setup

Dataset The GQA dataset contains 22M questions over 140K images. This full “all-split” dataset has unbalanced answer distributions, thus, is further re-sampled into a “balanced-split” with a more balanced answer distribution. The new split consists of 1M questions. Compared with the VQA v2.0 dataset (Goyal et al., 2017), the questions in GQA are designed to require multi-hop reasoning to test the reasoning skills of developed models. Compared with the CLEVR dataset (Johnson et al., 2017a), GQA greatly increases the complexity of the semantic structure of questions, leading to a more diverse function set. The real-world images in GQA also bring in a bigger challenge in visual understanding. In GQA, around 94% of questions need multi-hop reasoning, and 51% questions are about the relationships between objects. Following Hudson & Manning (2019a), the main evaluation metrics used in our experiments are accuracy, consistency, plausibility, and validity.

Model	Binary	Open	Consistency	Plausibility	Validity	Accuracy
Bottom-up (Anderson et al., 2018)	66.64	34.83	78.71	84.57	96.18	49.74
MAC (Hudson & Manning, 2018)	71.23	38.91	81.59	84.48	96.16	54.06
GRN (Guo et al., 2019)	74.93	41.24	87.41	84.68	96.14	57.04
LCGN (Hu et al., 2019)	73.77	42.33	84.68	84.81	**96.48**	57.07
BAN (Kim et al., 2018)	76.00	40.41	91.70	**85.58**	96.16	57.10
LXMERT (Tan & Bansal, 2019)	77.16	45.47	89.59	84.53	96.35	60.33
NSM (Hudson & Manning, 2019b)	**78.94**	**49.25**	**93.25**	84.28	96.41	**63.17**
MCAN (Yu et al., 2019)	75.87	42.15	87.72	84.57	96.20	57.96
NMN (Andreas et al., 2016b)	72.88	40.53	83.52	**84.81**	**96.39**	55.70
MMN (Ours)	**78.49**	**44.31**	**92.16**	84.29	96.13	**60.33**

Table 1: Comparison of MMN single model with published state-of-the-art methods on the blind test2019 set, as reported on the leaderboard as of Sep. 2019.

Implementation Details The dimensionality of input image features D_v is 2048, extracted from the bottom-up-attention model (Anderson et al., 2018)[1]. For each image, we keep the top 48 bounding boxes ranked by confidence score with the positional information of each bounding box in the form of [top-left-x, top-left-y, bottom-right-x, bottom-right-y], normalized by the image width and height. Both the Meta Module and the Visual Encoder have a hidden dimension D of 512 with 8 heads. GloVe embeddings (Pennington et al., 2014) are used to encode both questions and function keywords with 300 dimensions. The total vocabulary size is 3761, including all the functions, objects, and attributes. For training, we first use the 22M unbalanced “all-split” to bootstrap our model with a mini-batch size 2048 for 3-5 epochs, then fine-tune on the “balanced-split” with a mini-batch size 256. The testdev-balanced split is used for selecting the best model.

3.2 Experimental Results

We report our experimental results on the test2019 split (from the public GQA leaderboard) in Table 1. First, we observe significant performance gain from MMN over NMN (Andreas et al., 2016b), which demonstrates the effectiveness of the proposed meta module mechanism. Further, we observe that our model outperforms the VQA state-of-the-art monolithic model MCAN (Yu et al., 2019) by a large margin, which demonstrates the strong compositionality of our module-based approach. Overall, our single model achieves competitive performance (tied top 2) among published approaches. Notably, we achieve the same performance as LXMERT (Tan & Bansal, 2019), which is pre-trained on large-scale out-of-domain datasets. The performance gap with NSM (Hudson & Manning, 2019b) is debatable since our model is self-contained without relying on well-tuned external scene graph generation model (Xu et al., 2017; Yang et al., 2016; Chen et al., 2019).

To verify the contribution of each component in MMN, we perform several ablation studies: (1) *w/o Module Supervision vs. w/ Module Supervision*. We investigate the influence of module supervision by changing the hyper-parameter η from 0 to 2.0. (2) *Attention Supervision vs. Guideline*: We investigate different module supervision strategies, by directly supervising multi-head attention in multi-modal fusion stage (Figure 1). Specifically, we supervise different number of heads or the mean/max over different heads. (3) *w/o Bootstrap vs w/ Bootstrap*: We investigate the effectiveness of bootstrapping in training to validate the influence of pre-training on the final model performance.

Results are summarized in Table 2. From Ablation (1), we observe that without module supervision, our MMN achieves decent performance improvement over MCAN (Yu et al., 2019), but with much fewer parameters. By increasing η from 0.1 to 0.5, accuracy steadily improves, which reflects the importance of module supervision. Further increasing the value of η did not improve the performance empirically. From Ablation (2), we observe that directly supervising the attention weights in different Transformer heads only yields marginal improvement, which justifies the effectiveness of the implicit regularization in MMN. From Ablation (3), we observe that bootstrapping is an important step for MMN, as it explores more data to better regularize functionalities of reasoning modules. It is also observed that the epoch number of bootstrap also influences the final model performance. Choosing the optimal epoch size can lead to a better initialization for the following fine-tuning stage.

[1]`https://github.com/peteanderson80/bottom-up-attention`

Ablation (1)	Accuracy	Ablation (2)	Accuracy	Ablation (3)	Accuracy
6-Layered MCAN	57.4	Ours + AS (1 head)	57.5	w/o Bootstrap	58.4
Ours w/o MS	58.1	Ours + AS (2 head)	58.0	w/o Fine-tuning	56.5
Ours + MS ($\eta = 0.1$)	59.1	Ours + AS (4 head)	58.0	Bootstrap (2 epochs)	59.2
Ours + MS ($\eta = 0.5$)	60.0	Ours + AS (Mean)	58.1	Bootstrap (3 epochs)	59.6
Ours + MS ($\eta = 1.0$)	59.8	Ours + AS (Max)	58.2	Bootstrap (4 epochs)	60.0
Ours + MS ($\eta = 2.0$)	59.5	Ours + Guideline	60.0	Bootstrap (5 epochs)	59.8

Table 2: Three sets of ablation study on GQA. MS: Module Supervision; AS: Attention Supervision; w/o Bootstrap: Directly training on the balanced-split.

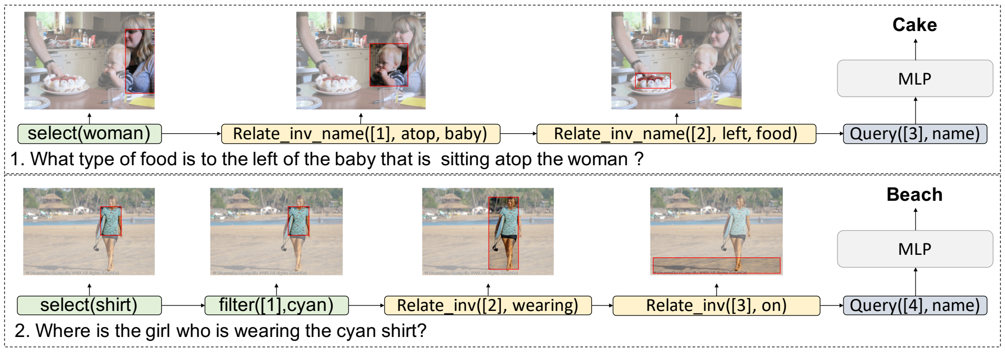
Cake
MLP
select(woman)
Relate_inv_name([1], atop, baby)
Relate_inv_name([2], left, food)
Query([3], name)
1. What type of food is to the left of the baby that is sitting atop the woman ?
Beach
MLP
select(shirt)
filter([1],cyan)
Relate_inv([2], wearing)
Relate_inv([3], on)
Query([4], name)
2. Where is the girl who is wearing the cyan shirt?

Figure 5: Visualization of the inferential chains learned by our model.

3.3 Interpretability and Error Analysis

To demonstrate the interpretability of MMN, Figure 5 provides some visualization results to show the inferential chain during reasoning. As shown, the model correctly executes the intermediate results and yields the correct final answer. To better interpret the model's behavior, we also perform quantitative analysis to diagnose the errors in the inferential chain. Here, we held out a small validation set to analyze the execution accuracy of different functions. Our model obtains Recall@1 of 59% and Recall@2 of 73%, which indicates that the object selected by the symbolic teacher has 59% chance of being top-1, and 73% chance as the top-2 by the student model, significantly higher than random-guess Recall@1 of 2%, demonstrating the effectiveness of module supervision.

Furthermore, we conduct detailed analysis on function-wise execution accuracy to understand the limitation of MMN. Results are shown in Table 3. Below are the observed main bottlenecks: (i) relation-type functions such as `relate`, `relate_inv`; and (ii) object/attribute recognition functions such as `query_name`, `query_color`. We hypothesize that this might be attributed to the quality of visual features from standard object detection models (Anderson et al., 2018), which does not capture the relations between objects well. Besides, the object and attribute classification network is not fine-tuned on GQA. This suggests that scene graph modeling for visual scene understanding is critical to surpassing NSM (Hudson & Manning, 2019b) on performance.

3.4 Analysis on Generalization

To demonstrate the generalization ability of the meta module, we perform additional experiments to validate whether the recipe representation can generalize to unseen functions. Specifically, we held out all the training instances containing `verify_shape`, `relate_name`, `choose_name` to quantitatively measure model's on these unseen functions. Standard NMN (Andreas et al., 2016b) fails to handle these unseen functions, as it requires training instances for the randomly initialized shallow network for these unseen functions. In contrast, MMN can transform the unseen functions

Return Type	Binary					Objects			String
Abstract Function	verify	choose	compare	exist	and/or	filter	select	relate	query
Accuracy	0.74	0.79	0.88	0.88	0.99	0.67	0.61	0.44	0.61

Table 3: Error analysis on different functions. "Objects" functions only appear in the intermediate step, "String" function only appears in the final step, "Binary" functions can occur in both cases.

Function	Verify_color			Relate_name			Choose_name		
Methods	NMN	MMN	Full-Shot	NMN	MMN	Full-Shot	NMN	MMN	Full-Shot
Accuracy	50%	61%	74%	5%	23%	49%	50%	62%	79%

Table 4: Comparison between MMN and NMN on generalization ability to unseen functions.

into recipe format and exploits the structural similarity with its related functions to infer its semantic functionality. For example, if the training set contains `verify_size` (function: verify, type: size, attr: ?) and `filter_shape` (function: filter, type: shape, attr: ?) functions in the recipes, an instantiated module is capable of inferring the functionality of an unseen but similar function `verify_shape` (function: verify, type:shape, attr: ?) from the recipe embedding space. Table 4 shows that the zero-shot accuracy of the proposed meta module is significantly higher than NMN (equivalent to random guess), which demonstrates the generalization ability of MMN and validate the extensibility of the proposed recipe encoding. Instead of handcrafting new modules every time when new functional semantics comes in like NMN (Andreas et al., 2016b), our MMN is more flexible and extensible for handling growing function sets under incremental learning.

4 Related Work

Monolithic Networks: Most monolithic networks for visual reasoning resort to attention mechanism for multimodal fusion (Zhu et al., 2017; 2016; Zhou et al., 2017; Yu et al., 2019; 2017; Kim et al., 2016; 2018; Kafle et al., 2018; Li et al., 2019; Hu et al., 2019). To realize multi-hop reasoning on complex questions, SAN (Yang et al., 2016), MAC (Hudson & Manning, 2018) and MuRel (Cadene et al., 2019) models have been proposed. However, their reasoning procedure is built on a general-purpose reasoning block, which can not be disentangled to perform specific tasks, resulting in limited model interpretability and compositionality.

Neural Module Networks: By parsing a question into a program and executing the program through dynamically composed neural modules, NMN excels in interpretability and compositionality by design (Andreas et al., 2016a;b; Hu et al., 2017; Johnson et al., 2017b; Hu et al., 2018; Yi et al., 2018; Mao et al., 2019; Vedantam et al., 2019). However, its success is mostly restricted to the synthetic CLEVR dataset, whose performance can be surpassed by simpler methods such as relational network (Santoro et al., 2017) and FiLM (Perez et al., 2018).

Our MMN is a module network in concept, thus possessing high interpretability and compositionality. However, different from traditional NMN, MMN uses only one Meta Module for program execution recurrently, similar to an LSTM cell (Hochreiter & Schmidhuber, 1997) in Recurrent Neural Network. This makes MMN a monolithic network in practice, which ensures strong empirical performance without sacrificing model interpretability.

State of the Art on GQA: GQA was introduced in Hudson & Manning (2019a) for real-world visual reasoning. Simple monolithic networks (Wu et al., 2019), MAC netowrk (Hudson & Manning, 2018), and language-conditioned graph neural networks (Hu et al., 2019; Guo et al., 2019) have been developed for this task. LXMERT (Tan & Bansal, 2019), a large-scale pre-trained encoder, has also been tested on this dataset. Recently, Neural State Machine (NSM) (Hudson & Manning, 2019b) proposed to first predict a probabilistic scene graph, then perform multi-hop reasoning over the graph for answer prediction. The scene graph serves as a strong prior to the model. Our model is designed to leverage dense visual features extracted from object detection models, thus orthogonal to NSM and can be enhanced with their scene graph generator once it is publicly available. Different from the aforementioned approaches, MMN also performs explicit multi-hop reasoning based on predicted programs, so the inferred reasoning chain can be directly used for model interpretability.

5 Conclusion

In this paper, we propose Meta Module Network that bridges the gap between monolithic networks and traditional module networks. Our model is built upon a Meta Module, which can be instantiated into an instance module performing specific functionalities. Our approach significantly outperforms baseline methods and achieves comparable performance to state of the art. Detailed error analysis shows that relation modeling over scene graph could further boost MMN for higher performance. For future work, we plan to incorporate scene graph prediction into the proposed framework.

REFERENCES

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *CVPR*, 2018.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. In *NAACL*, 2016a.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *CVPR*, 2016b.

Remi Cadene, Hedi Ben-Younes, Matthieu Cord, and Nicolas Thome. Murel: Multimodal relational reasoning for visual question answering. In *CVPR*, 2019.

Tianshui Chen, Weihao Yu, Riquan Chen, and Liang Lin. Knowledge-embedded routing network for scene graph generation. In *CVPR*, 2019.

Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. In *ACL*, 2018.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *CVPR*, 2017.

Dalu Guo, Chang Xu, and Dacheng Tao. Graph reasoning networks for visual question answering. *arXiv preprint arXiv:1907.09815*, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *ICCV*, 2017.

Ronghang Hu, Jacob Andreas, Trevor Darrell, and Kate Saenko. Explainable neural computation via stack neural module networks. In *ECCV*, 2018.

Ronghang Hu, Anna Rohrbach, Trevor Darrell, and Kate Saenko. Language-conditioned graph networks for relational reasoning. In *ICCV*, 2019.

Drew A Hudson and Christopher D Manning. Compositional attention networks for machine reasoning. In *ICLR*, 2018.

Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *CVPR*, 2019a.

Drew A Hudson and Christopher D Manning. Learning by abstraction: The neural state machine. In *NeurIPS*, 2019b.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017a.

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

Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. Dvqa: Understanding data visualizations via question answering. In *CVPR*, 2018.

Jin-Hwa Kim, Kyoung-Woon On, Woosang Lim, Jeonghee Kim, Jung-Woo Ha, and Byoung-Tak Zhang. Hadamard product for low-rank bilinear pooling. In *ICLR*, 2016.

Jin-Hwa Kim, Jaehyun Jun, and Byoung-Tak Zhang. Bilinear attention networks. In *NeurIPS*, 2018.

Linjie Li, Zhe Gan, Yu Cheng, and Jingjing Liu. Relation-aware graph attention network for visual question answering. *arXiv preprint arXiv:1903.12314*, 2019.

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

Duy-Kien Nguyen and Takayuki Okatani. Improved fusion of visual and language representations by dense symmetric co-attention for visual question answering. In *CVPR*, 2018.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

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

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *NeurIPS*, 2017.

Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. In *EMNLP*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Ramakrishna Vedantam, Karan Desai, Stefan Lee, Marcus Rohrbach, Dhruv Batra, and Devi Parikh. Probabilistic neural-symbolic models for interpretable visual question answering. In *ICML*, 2019.

Chenfei Wu, Yanzhao Zhou, Gen Li, Nan Duan, Duyu Tang, and Xiaojie Wang. Deep reason: A strong baseline for real-world visual reasoning. *arXiv preprint arXiv:1905.10226*, 2019.

Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *CVPR*, 2017.

Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. In *CVPR*, 2016.

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *NeurIPS*, 2018.

Zhou Yu, Jun Yu, Jianping Fan, and Dacheng Tao. Multi-modal factorized bilinear pooling with co-attention learning for visual question answering. In *ICCV*, 2017.

Zhou Yu, Jun Yu, Yuhao Cui, Dacheng Tao, and Qi Tian. Deep modular co-attention networks for visual question answering. In *CVPR*, 2019.

Yiyi Zhou, Rongrong Ji, Jinsong Su, Yongjian Wu, and Yunsheng Wu. More than an answer: Neural pivot network for visual qestion answering. In *ACMMM*, 2017.

Chen Zhu, Yanpeng Zhao, Shuaiyi Huang, Kewei Tu, and Yi Ma. Structured attentions for visual question answering. In *ICCV*, 2017.

Yuke Zhu, Oliver Groth, Michael Bernstein, and Li Fei-Fei. Visual7w: Grounded question answering in images. In *CVPR*, 2016.

A Appendix

A.1 Visual Encoder and Multi-head Attention

The visual encoder and multi-head attention network is illustrated in Figure 6 and Figure 7, respectively.

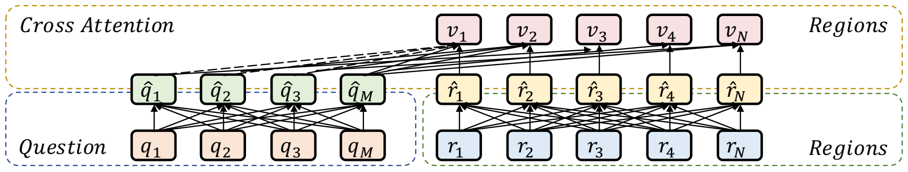
Cross Attention
Regions
Question
Regions
v_1 v_2 v_3 v_4 v_N
$\hat{q}_1$ $\hat{q}_2$ $\hat{q}_3$ $\hat{q}_M$
$\hat{r}_1$ $\hat{r}_2$ $\hat{r}_3$ $\hat{r}_4$ $\hat{r}_N$
q_1 q_2 q_3 q_M
r_1 r_2 r_3 r_4 r_N

Figure 6: Illustration of the Visual Encoder described in Section. 2.1.

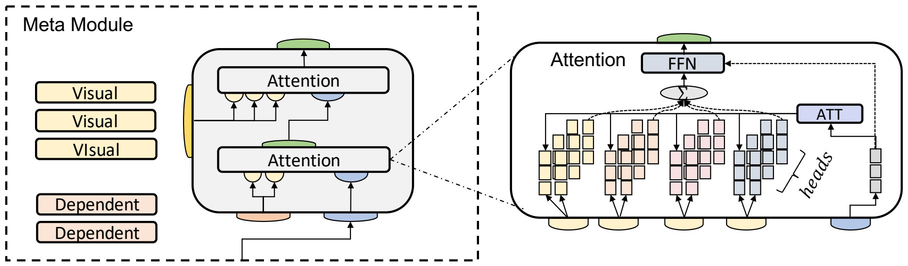

Figure 7: Illustration of the multi-head attention network used in the Meta Module.

A.2 Recipe Embedding

The recipe embedder is illustrated in Figure 8.

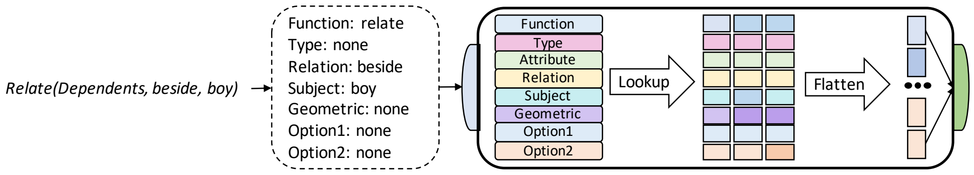

Figure 8: Illustration of the recipe embedder.

A.3 Function Statistics

The function statistics is listed in Table 5.

A.4 Function Description

The detailed function descriptions are provided in Figure 9.

A.5 Inferential Chains

More inferential chains are visualized in Figure 10 and Figure 11.

Type	Relate	Select	Filter	Choose	Verify	Query	Common	Differ	Bool	Exist	All
Funcs	5	1	8	12	5	6	2	6	2	1	48

Table 5: The statistics of different functions.

Type	Overrides	arg0 (? Means Dependency)	arg1	arg2	arg3	Output
Relationship	Relate	?	Relation	-	-	Region
	Relate_with_name	?	Relation	Object	-	
	Relate_invese	?	Relation	-	-	
	Relate_inverse_with_name	?	Relation	Object	-	
	Relate_with_same_attribute	?	Relation	Attribute	-	
Selection	Select	-	Object	-	-	Region
Filter	Filter_horizontal_position	?	H-Position	-	-	Region
	Filter_Vertical_position	?	V-Position	-	-	
	Filter_with_color	?	Color	-	-	
	Filter_with_shape	?	Shape	-	-	
	Filter_with_activity	?	Activity	-	-	
	Filter_with_material	?	Material	-	-	
	Filter_with_color_noteq	?	Color	-	-	
	Filter_with_shape_noteq	?	Shape	-	-	
Choose	Choose_name	?	Name1	Name2	-	Answer
	Choose_scene	-	Scene1	Scene2	-	
	Choose_color	?	Color1	Color2	-	
	Choose_shape	?	Shape1	Shape2	-	
	Choose_horizontal_position	?	H-Position1	H-Position2	-	
	Choose_vertical_position	?	V-Position1	V-Position2	-	
	Choose_relation_name	?	Relation1	Relation2	Name	
	Choose_relation_inverse_name	?	Relation1	Relation2	Name	
	Choose_younger	?	?	-	-	
	Choose_older	?	?	-	-	
	Choose_healthier	?	?	-	-	
	Choose_less_healthier	?	?	-	-	
Verify	Verify_color	?	Color	-	-	Answer
	Verify_shape	?	Shape	-	-	
	Verify_scene	-	Scene	-	-	
	Verify_relation_name	?	Relation	Name	-	
	Verify_relation_inv_name	?	Relation	Name	-	
Query	Query_name	?	-	-	-	Answer
	Query_color	?	-	-	-	
	Query_shape	?	-	-	-	
	Query_scene	-	-	-	-	
	Query_horizontal_position	?	-	-	-	
	Query_vertical_position	?	-	-	-	
Common	Common_color	[? ? .. ?]	-	-	-	Answer
	Common_material	[? ? .. ?]	-	-	-	
Different	Different_name	[? ? .. ?]	-	-	-	Answer
	Different_name	?	?	-	-	
	Different_color	?	?	-	-	
Same	Same_name	[? ? .. ?]	-	-	-	Answer
	Same_name	?	?	-	-	
	Same_color	?	?	-	-	
And	And	?	?	-	-	Answer
Or	Or	?	?	-	-	Answer
Exist	Exist	?	?	-	-	Answer

Figure 9: The function definitions and their corresponding outputs.

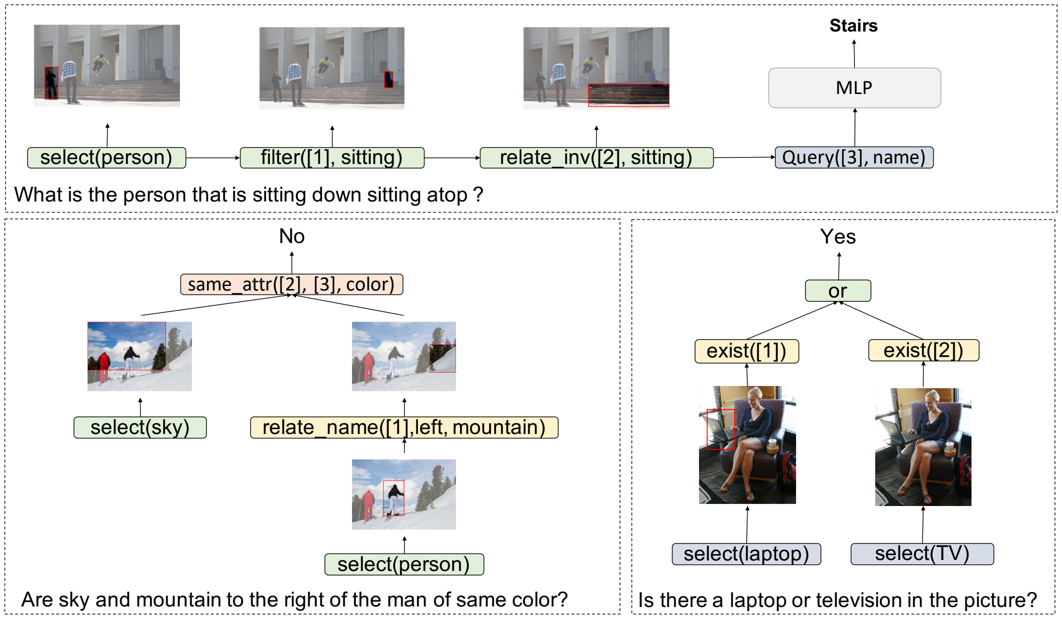
Stairs
MLP
select(person)
filter([1], sitting)
relate_inv([2], sitting)
Query([3], name)
What is the person that is sitting down sitting atop ?
No
same_attr([2], [3], color)
select(sky)
relate_name([1],left, mountain)
select(person)
Are sky and mountain to the right of the man of same color?
Yes
or
exist([1])
exist([2])
select(laptop)
select(TV)
Is there a laptop or television in the picture?

Figure 10: More examples on visualization of the inferential chains learned by our model.

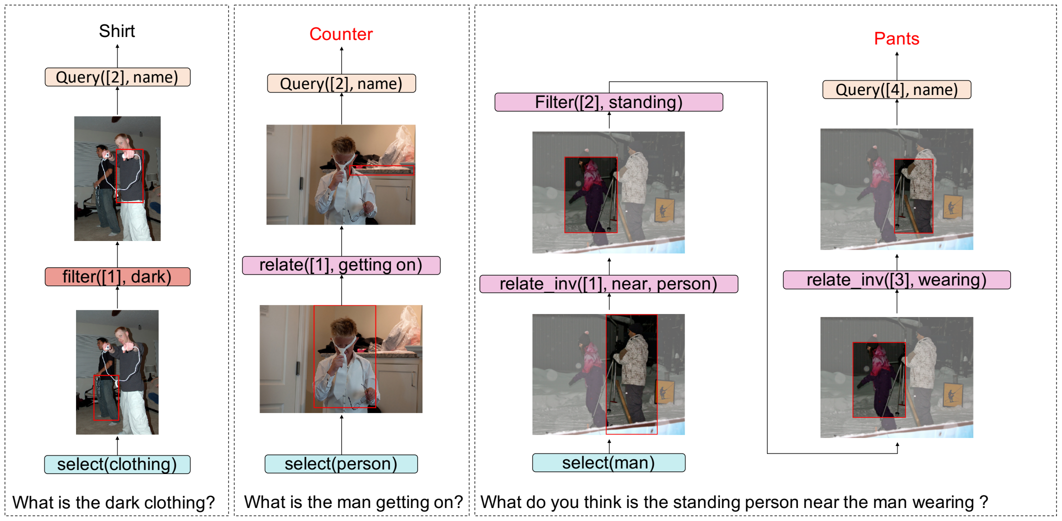

Figure 11: More examples on visualization of the inferential chains learned by our model.