Meta Module Network for Compositional Visual Reasoning

Wenhu Chen[†], Zhe Gan[*], Linjie Li[*], Yu Cheng[*], William Wang[†], Jingjing Liu[*]
[†]University of California, Santa Barbara, CA, USA
[*]Microsoft, Bellevue, WA, USA
{wenhuchen, william}@cs.ucsb.edu,
{zhe.gan, lindsey.li, yu.cheng, jingjl}@microsoft.com

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 between the two and leverage the merits of both, we present Meta Module Network (MMN), a novel hybrid approach that can 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 similar to traditional NMN, we propose a Meta Module, which can read a recipe (function specifications) to dynamically instantiate the task-specific Instance Modules for compositional reasoning. To endow different instance modules with designated functionalities, we design a symbolic teacher which can execute against provided scene graphs to generate guidelines for the instantiated modules (student) to follow during training. Experiments conducted on the GQA benchmark demonstrates that MMN outperforms both NMN and monolithic network baselines, with good generalization ability to handle unseen functions.[1]

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) [2, 3, 14, 20, 13, 27] 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 subtask and multiple modules can be combined to perform unseen combinations during inference. However, with increased complexity of the task, the set of functional seman-

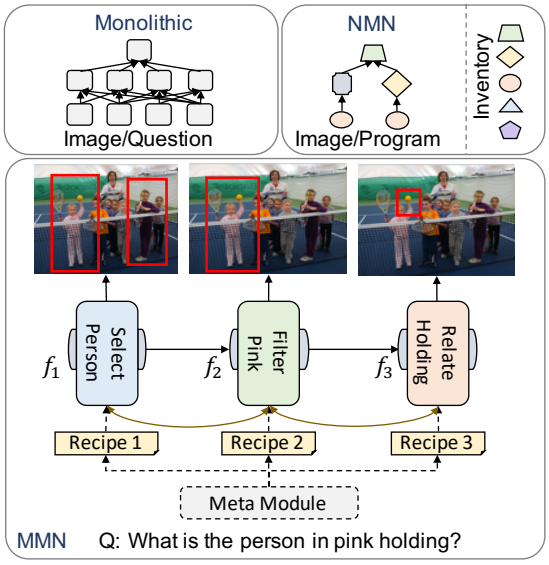

Figure 1. Overview of Meta Module Network (MMN) for visual reasoning. (Top) Monolithic architecture uses an instance-agnostic network to perform general-purpose reasoning, while Neural Module Network (NMN) builds an instance-specific network from a pre-defined inventory of neural modules with independent parameterization. (Bottom) In MMN, we define an abstract Meta Module Network, which can take different function recipes (specifications) as input to instantiate designated instance modules to accomplish different sub-tasks.

[1]Code will be released upon acceptance.

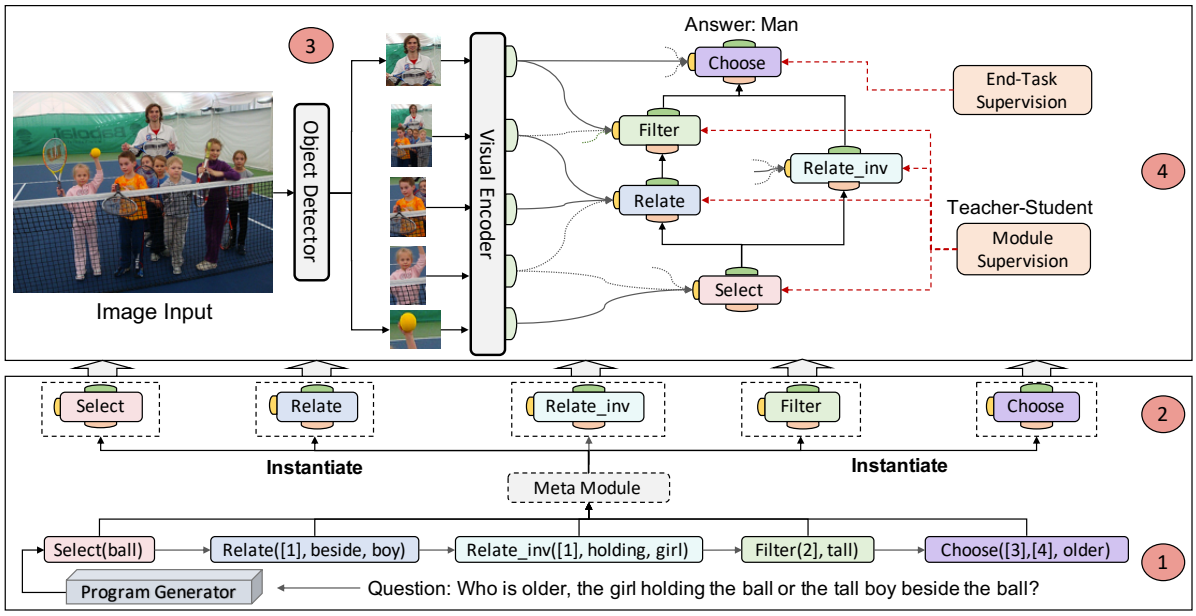

Figure 2. The model architecture of Meta Module Network: the lower part describes how the question is translated into programs and instantiated into operation-specific modules; the upper part describes the visual encoder and supervision. Circles denote i-th steps.

tics scales up, so are the modules. For example, in the recent GQA dataset [16], a much larger set of functions with varied arity will be involved, compared with previous CLEVR dataset [19]. In the standard NMN framework [2, 14, 27], a large amount of independent module networks are needed to implement these diverse functions, which leads to higher model/sample complexity and poorer generalization.

Another line of research on visual reasoning is focused on designing monolithic network architectures, such as MFB [45], BAN [22], DCN [28], and MCAN [44]. These black-box models have achieved strong performance on challenging datasets, such as VQA [4, 10] and GQA [17], surpassing the NMN approach. More recently, multi-modal pre-training algorithms [37, 26, 35, 7] have been proposed to further achieve state-of-the-art performance on different tasks, including VQA [10], NLVR^2 [36], and VCR [46]. They use a unified neural network to learn general-purpose reasoning skills [16], which is known to be more flexible and scalable without strict assumptions about input or function-specific design for the pre-defined functional semantics. However, as the reasoning procedure is conducted in the latent feature space, it is difficult to interpret. Such model also lacks the ability to capture the compositionality of questions, thus suffering from poorer generalizability than module networks.

In order to bridge the gap, we propose a Meta Module Network (MMN) as depicted in Figure 1. MMN is based on a novel Meta Module (a monolithic neural network), which does not require handcrafted neural architectures for individual functions. The Meta Module can take a recipe (function description) and instantiate an instance modules to accomplish designated sub-task specified in the recipe. The instance modules are instantiated on the fly to build an execution graph to perform compositional visual reasoning.

MMN draws inspiration from Meta Learning [34, 31, 9] and frame the module network as a learning-to-learn problem. Instead of directly teaching the different hand-crafted neural modules to solve different sub-tasks, MMN proposes to teach a meta learner (module) to understand a recipe to output a function which can solve a specific sub-task. Specifically, a (parent) meta module is a function $g(*,*)$, which take a function recipe f (an embedding vector) as input to output a (child) instance function $g_f(*) = g(*, f)$ for problem solving. To endow each instance module g_f with the designated functionality f, we introduce a Teacher-Student module supervision to enforce each module $g_f(*)$ to imitate the behavior (e.g. where to attend in the image, whether to attend to the input object, etc) of a "symbolic teacher". The teacher can traverse the scene graph to provide the expected outputs of the given functions, which are provided as guidelines for the instance module to follow. As different instance module are supervised differently, their functionalities are thus disentangled, which achieves high compositionality. The introduced Meta Module paradigm can help accommodate a larger set of functional semantics without increasing model/sample complexity. Moreover, the meta learner can generalize in the recipe embedding space to infer unseen functions.

As illustrated in Figure 2, in order to answer a question about an image, (i) we first adopt a coarse-to-fine semantic parser to parse questions in to programs based on predefined functions; (ii) the Meta Module is instantiated into

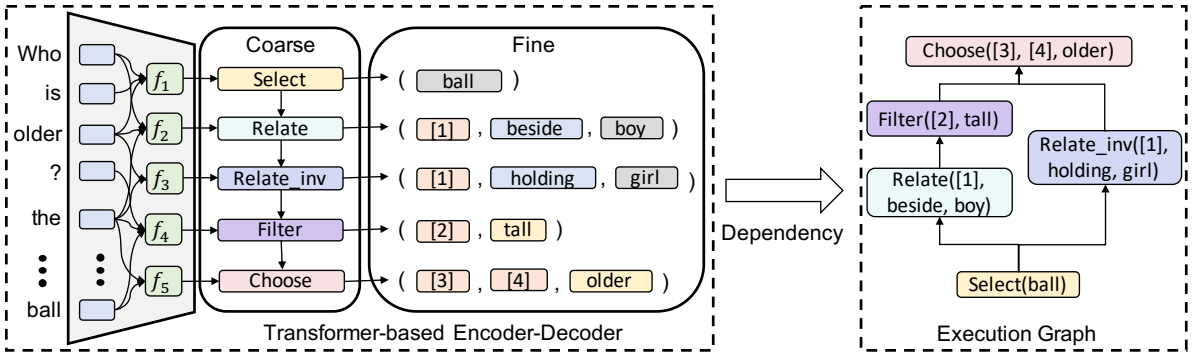

Figure 3. Architecture of the coarse-to-fine Program Generator: the left part depicts the coarse-to-fine two-stage generation; the right part depicts the resulting execution graph based on the dependency relationship.

different instance modules based on the parsed functions, which are then used to compose the execution graph; (*iii*) the visual encoder encodes the given detection features and feed it to the instance modules; (*iv*) during training, we use a Teacher-Student framework to provide module-wise supervision to disentangle different instance (student) modules jointly with the standard QA supervision. At test time, we use the trained student modules to predict the answer.

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

2. Meta Module Network

The visual reasoning task [17] 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 2 provides an overview of Meta Module Network (MMN), which consists of three components: (*i*) Program Generator (Sec. 2.1), which generates a functional program from the input question; (*ii*) Visual Encoder (Sec. 2.2), 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; (*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.

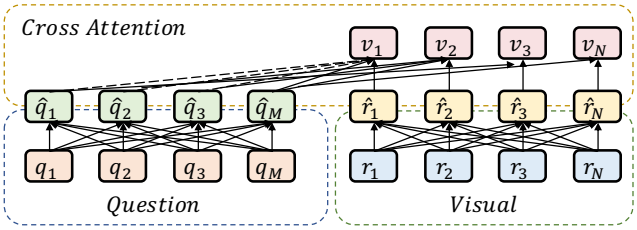

Figure 4. Illustration of the Visual Encoder described in Sec. 2.2.

2.1. 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 [17]. The definitions for all the functions are provided in the Appendix. The defined functions can be divided into 10 different categories based on their coarse-grained functionality (*e.g.*, "relate, verify, filter, choose"), and each abstract function type is further implemented with different realizations based on their fine-grained functionality. For example, the "verify" category can be further implemented with "verify_attribute, verify_geometric, verify_relation, verify_color", and these different realizations take different arguments as inputs.

In total, there are 48 different functions defined, whose returned values could be *List of Objects*, *Boolean* or *String*, where *Object* specifically refers to the detected bounding box and *String* could refer to object name, attributes, relations, etc. A program P is viewed as a sequence of function calls $f_1, \cdots, f_L$. For example, in Figure 3, 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" (previous execu-

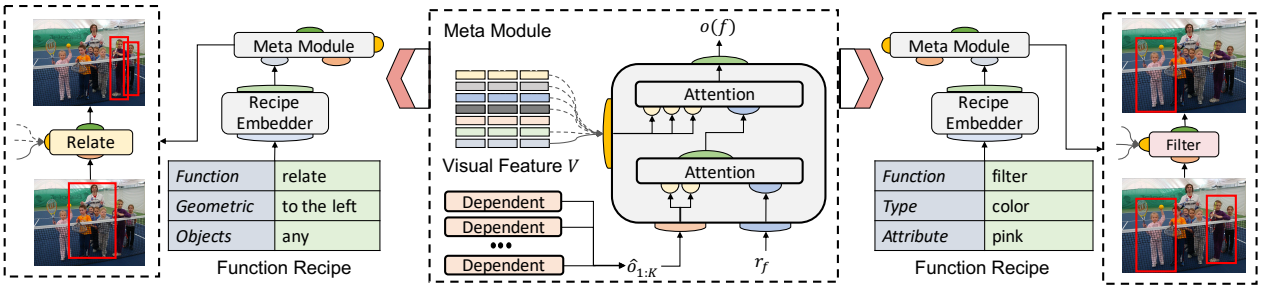

Figure 5. Illustration of the instantiation process for "Relate" and "Filter" functions.

tion results), and `beside, boy` the "arguments". By exploiting the dependency relationship between functions, we build an execution graph, where each node represents a function and each edge denotes a dependency relationship between the connected two nodes.

In order to generate syntactically plausible programs, we follow [8] and adopt a coarse-to-fine two-stage generation paradigm, as illustrated in Figure 3. Specifically, the Transformer-based program generator [38] first encodes the question as the context vectors, and then decodes a sketch step by step, the sketch only contains the function name without arguments. Once the sketch is decoded, the arity and types of the decoded functions are determined, for example, after generating "Relate", we are certain that there will be three arguments following this function while the first argument is the dependency. The sketch is thus expanded as "Relate(#1, #2, #3)", where "#i" denotes i-th unfilled slots. We then apply a fine-grained generator to fill in the slots of dependencies and arguments for the sketch as concrete program P. During the slot-filling phase, we constrain the type at each time step to greatly reduce the complexity. Such a two-stage generation process helps guarantee the plausibility and grammaticality of synthesized programs. 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 to shrink the search space, our program synthesizer can achieve a 98.8% execution accuracy (returning the same result as the ground truth after execution). In contrast, a canonical transformer-based encoder-decoder [38] could only reach an execution accuracy of 93%, which greatly lags behind our parser.

2.2. Visual Encoder

The Visual Encoder is based on a pre-trained object detection model [32, 1] 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. As illustrated in Figure 4, similar to a Transformer block [38], we first use two self-attention networks, SA_q and SA_r, to encode the question and the visual features as $\hat{\mathbf{Q}} = SA_q(Q, Q; \phi_q)$ and $\hat{\mathbf{R}} = SA_r(\mathbf{R}, \mathbf{R}; \phi_r)$, 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 $\mathbf{V} = CA(\hat{\mathbf{R}}, \hat{\mathbf{Q}}; \phi_c) \in \mathbb{R}^{N \times D}$, where $\hat{\mathbf{Q}}$ is used as the query vector, and $\phi = \{\phi_q, \phi_r, \phi_c\}$ 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.

2.3. Meta Module

As opposed to having a full inventory of task-specific parameterized modules for different functions as in NMN [3], we design an abstract Meta Module that can be instantiated 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 5, when taking recipe `Function:relate; Geometric:to the left` as the input, the Recipe Embedder produces a recipe vector to instantiate the abstract Meta Module into a "geometric relation" module, which specifically searches for target objects that the current object is to the left of. When taking recipe `Function:filter; Type:color; Attribute:pink` as input, the Embedder will instantiate the Meta Module into a "filter pink" module, which specifically looks for the objects with pink color in the input objects. Our Meta Module draws inspiration from meta learning [34, 31, 9], where we teach the meta learner to generate different functions to accomplish different sub-tasks described in the recipe.

Two-layered Attention: Figure 5 demonstrates the computation flow in Meta Module, which is built upon two-leveled multi-head attention network [38]. 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 dependent modules. The intermediate output ($\mathbf{o}_d$) from this attention layer is further fed into a second attention network g_v to incorporate the visual representa-

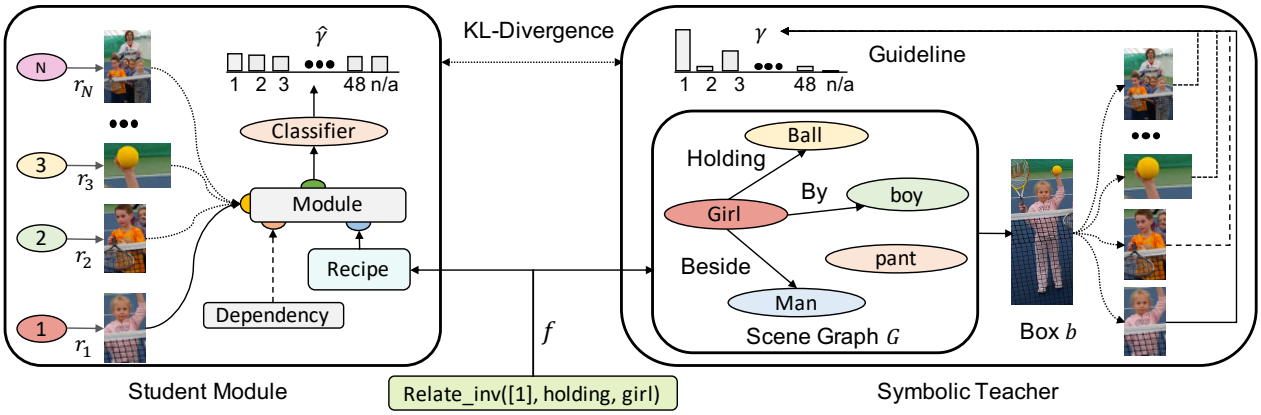

Figure 6. Illustration of the Module Supervision process: the symbolic teacher executes the function on the scene graph to obtain the bounding box b, which is then aligned with bounding boxes from the object detection model to compute the supervision guideline γ.

tion $\mathbf{V}$ 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: The instantiation is accomplished by feeding a function f to the meta module g, which results in a wrapper function $g_f(\hat{\mathbf{o}}_{1:K}, \mathbf{V}; \psi)$ known as instance module, where ψ denotes the parameters of the module. 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.

Scalability & Generalization: As demonstrated, Meta Module Network excels over standard module network in the following aspects. (i) The parameter space ψ 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 enables MMN to generalize to unseen but related functions.

2.4. Module Supervision

In this sub-section, we explain how to apply the intermediate supervision signals from scene graphs provided in the training data to supervise the instance modules. Such supervision is realized by a Teacher-Student framework as depicted in Figure 6. First, we define a Symbolic Executor as the 'Teacher', which can take the input function f and traverse the ground-truth scene graph to obtain intermediate results (distribution over the objects on the ground truth scene graph). The 'Teacher' exhibits it as guideline γ for the 'Student' instance module g_f to adhere to.

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 $(f_1, \cdots, f_{L-1})$ 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 ob-

ject (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 provided 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 $\hat{\gamma}_i = softmax(MLP(\mathbf{o}(f_i)))$. Instead of explicitly modeling the attention like PVR [24], our method adopts the classification to implicitly reconstruct its attention. 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 [17], 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 [10], the questions in GQA are designed to require multi-hop reasoning to test the reasoning skills of developed models. Compared with the CLEVR dataset [19], 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 [17], the main evaluation metrics used in our experiments are accuracy, consistency, plausibility, and validity.

Implementation Details The dimensionality of input image features D_v is 2048, extracted from the bottom-up-attention model [1][2]. 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 [29] 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.

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

Model	Traning Data	Binary	Open	Accuracy
Bottom-up [1]	V[1]+NL[2]	66.64	34.83	49.74
MAC [16]	V+NL	71.23	38.91	54.06
GRN [11]	V+NL	74.93	41.24	57.04
LCGN [15]	V+NL	73.77	42.33	57.07
BAN [22]	V+NL	76.00	40.41	57.10
PVR [24]	V+NL+Prog[3]	74.58	42.10	57.33
LXMERT [37]	V+NL+PT[4]	77.16	45.47	60.33
NSM [18]	V+NL+SGM[5]	**78.94**	**49.25**	**63.17**
MCAN [44]	V+NL	75.87	42.15	57.96
NMN [3]	V+NL+Prog	72.88	40.53	55.70
MMN (Ours)	V+NL+Prog	**78.90**	**44.89**	**60.83**

Table 1. Comparison of MMN single model with published state-of-the-art methods on the blind test2019 set, as reported on the leaderboard in Sep. 2019. Here, V[1] denotes visual feature, NL[2] denotes natural language, Prog[3] denotes program, PT[4] denotes pre-training and SGM[5] denotes scene graph 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 [3], 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 [44] by a large margin, which demonstrates the strong compositionality of our module-based approach. Overall, our single model achieves competitive performance (top 2) among published approaches. Notably, we achieve higher performance than LXMERT [37], which is pre-trained on large-scale out-of-domain datasets. The performance gap with NSM [18] is debatable since our model is standalone without relying on well-tuned external scene graph generation model [41, 42, 6].

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 hyperparameter η from 0 to 2.0 to see how much influence the module supervision has on the model performance. (2) *Explicit vs. Implicit*: We investigate different module supervi-

Ablation (1)	Accuracy	Ablation (2)	Accuracy	Ablation (3)	Accuracy
6-Layered MCAN	57.4	Ours + Explicit (1 head)	57.5	Ours w/o Bootstrap	58.4
Ours w/o MS	58.1	Ours + Explicit (2 head)	58.0	Ours w/o Fine-tuning	56.5
Ours + MS ($\eta = 0.1$)	59.1	Ours + Explicit (4 head)	58.0	Ours + Bootstrap (2 epochs)	59.2
Ours + MS ($\eta = 0.5$)	**60.4**	Ours + Explicit (Mean)	58.1	Ours + Bootstrap (3 epochs)	59.9
Ours + MS ($\eta = 1.0$)	60.1	Ours + Explicit (Max)	58.2	Ours + Bootstrap (4 epochs)	**60.4**
Ours + MS ($\eta = 2.0$)	59.5	Ours + Implicit	**60.4**	Ours + Bootstrap (5 epochs)	60.0

Table 2. Ablation study on GQA validation. Explicit/Implicit: Module Supervision; w/o Bootstrap: Directly training on the balanced-split.

sion strategies, by either explicitly supervising multi-head attention in second cross-modal attention layer (Figure 5) or implicitly supervising the behavior with an auxiliary classifier depicted in Figure 6. For the explicit supervision, we also use the guideline distribution to minimize its KL-divergence with attention weights extracted from the attention block aggregated with mean/max operation over different heads. (3) *w/o Bootstrap vs w/ Bootstrap*: We investigate the effectiveness of bootstrapping in training to validate whether we could use the large-scale unbalanced split to benefit on the model's performance.

We further report the ablation results for the validation split in Table 2. From Ablation (1), we observe that without module supervision, our MMN already achieves decent improvement over 6-layered MCAN [44]. Since all the modules have shared parameters, our model has similar parameters as 1-layered MCAN. The result reflects the effectiveness of our compact parameterization. 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 explicitly supervising the attention weights in different Transformer heads only yields marginal improvement, which justifies the effectiveness and flexibility of the implicit supervision in MMN. From Ablation (3), we observe that bootstrapping is an critical for MMN, as it explores more data to better regularize functionalities of reasoning modules. It is also observed that the bootstrap time could also influences the final model performance.

3.3. Interpretability and Error Analysis

To demonstrate the interpretability of MMN, Figure 7 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 [1], 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 [18] on performance.

3.4. Analysis on Generalization

Similar to Meta Learning [9], we also evaluate whether our meta module has learned the ability to adapt to unseen sub-tasks. To evaluate such generalization ability, we perform additional experiments, where 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 [3] 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 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 [3], our MMN is more flexible and extensible for handling growing function sets. Such observation further validates the value of

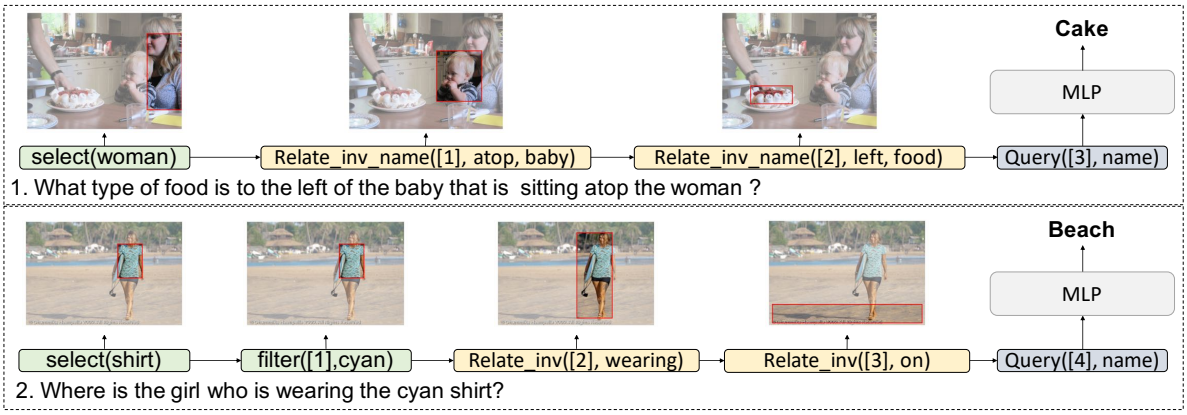

Figure 7. Visualization of the inferential chains learned by our model.

Return Type	Binary						Objects			String	
Function Category	verify	choose	compare	exist	and	or	filter	select	relate	query[object]	query[scene]
Accuracy	0.74	0.79	0.88	0.88	0.97	0.95	0.67	0.61	0.44	0.61	0.65

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(MMN)	NMN	MMN	Full-Shot(MMN)	NMN	MMN	Full-Shot(MMN)
Accuracy	50%	61%	74%	5%	23%	49%	50%	62%	79%

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

proposed method to adapt more challenging environment where we need to handle unknown functions.

4. Related Work

Monolithic Network: Most monolithic networks for visual reasoning resort to attention mechanism for multimodal fusion [48, 49, 47, 44, 45, 23, 22, 21, 25, 15]. To realize multi-hop reasoning on complex questions, SAN [42], MAC [16] and MuRel [5] models have been proposed. However, their reasoning procedure is built on a general-purpose reasoning block, which can not be disentangled, 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 [2, 3, 14, 20, 13, 43, 27, 39]. However, its success is mostly restricted to the synthetic CLEVR dataset, whose performance can be surpassed by simpler methods such as relational network [33] and FiLM [30]. 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 [12] in Recurrent Neural Network. This makes MMN a monolithic network in practice, which ensures strong empirical performance without sacrificing model interpretability.

GQA Models: GQA was introduced in [17] for real-world visual reasoning. Simple monolithic networks [40], MAC netowrk [16], and language-conditioned graph neural networks [15, 11] have been developed for this task. LXMERT [37], a large-scale pre-trained encoder, has also been tested on this dataset. Recently, Neural State Machine (NSM) [18] 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 to demonstrate inferred reasoning chain.

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 MMN to further improve its performance on visual reasoning.

References

[1] 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. 4, 6, 7

[2] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. In *NAACL*, 2016. 1, 2, 8

[3] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *CVPR*, 2016. 1, 4, 6, 7, 8

[4] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *ICCV*, 2015. 2

[5] Remi Cadene, Hedi Ben-Younes, Matthieu Cord, and Nicolas Thome. Murel: Multimodal relational reasoning for visual question answering. In *CVPR*, 2019. 8

[6] Tianshui Chen, Weihao Yu, Riquan Chen, and Liang Lin. Knowledge-embedded routing network for scene graph generation. In *CVPR*, 2019. 6

[7] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Learning universal image-text representations. *arXiv preprint arXiv:1909.11740*, 2019. 2

[8] Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. In *ACL*, 2018. 4

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017. 2, 4, 7

[10] 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. 2, 6

[11] Dalu Guo, Chang Xu, and Dacheng Tao. Graph reasoning networks for visual question answering. *arXiv preprint arXiv:1907.09815*, 2019. 6, 8

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997. 8

[13] Ronghang Hu, Jacob Andreas, Trevor Darrell, and Kate Saenko. Explainable neural computation via stack neural module networks. In *ECCV*, 2018. 1, 8

[14] 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. 1, 2, 8

[15] Ronghang Hu, Anna Rohrbach, Trevor Darrell, and Kate Saenko. Language-conditioned graph networks for relational reasoning. In *ICCV*, 2019. 6, 8

[16] Drew A Hudson and Christopher D Manning. Compositional attention networks for machine reasoning. In *ICLR*, 2018. 2, 6, 8

[17] Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *CVPR*, 2019. 2, 3, 6, 8

[18] Drew A Hudson and Christopher D Manning. Learning by abstraction: The neural state machine. In *NeurIPS*, 2019. 6, 7, 8

[19] 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*, 2017. 2, 6

[20] 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*, 2017. 1, 8

[21] Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. Dvqa: Understanding data visualizations via question answering. In *CVPR*, 2018. 8

[22] Jin-Hwa Kim, Jaehyun Jun, and Byoung-Tak Zhang. Bilinear attention networks. In *NeurIPS*, 2018. 2, 6, 8

[23] 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. 8

[24] Guohao Li, Xin Wang, and Wenwu Zhu. Perceptual visual reasoning with knowledge propagation. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 530–538. ACM, 2019. 6

[25] Linjie Li, Zhe Gan, Yu Cheng, and Jingjing Liu. Relation-aware graph attention network for visual question answering. *arXiv preprint arXiv:1903.12314*, 2019. 8

[26] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *NeurIPS*, 2019. 2

[27] 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. 1, 2, 8

[28] 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. 2

[29] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014. 6

[30] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018. 8

[31] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *ICLR*, 2017. 2, 4

[32] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015. 4

[33] 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. 8

[34] Jürgen Schmidhuber. On learning how to learn learning strategies. *Citeseer*, 1995. 2, 4

[35] Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. Vl-bert: Pre-training of generic visual-linguistic representations. *arXiv preprint arXiv:1908.08530*, 2019. 2

[36] Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. A corpus for reasoning about natural language grounded in photographs. *ACL*, 2019. 2

[37] Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. In *EMNLP*, 2019. 2, 6, 8

[38] 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. 4

[39] 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. 8

[40] 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. 8

[41] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *CVPR*, 2017. 6

[42] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. In *CVPR*, 2016. 6, 8

[43] 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. 8

[44] Zhou Yu, Jun Yu, Yuhao Cui, Dacheng Tao, and Qi Tian. Deep modular co-attention networks for visual question answering. In *CVPR*, 2019. 2, 6, 7, 8

[45] 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. 2, 8

[46] Rowan Zellers, Yonatan Bisk, Ali Farhadi, and Yejin Choi. From recognition to cognition: Visual commonsense reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6720–6731, 2019. 2

[47] 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. 8

[48] Chen Zhu, Yanpeng Zhao, Shuaiyi Huang, Kewei Tu, and Yi Ma. Structured attentions for visual question answering. In *ICCV*, 2017. 8

[49] Yuke Zhu, Oliver Groth, Michael Bernstein, and Li Fei-Fei. Visual7w: Grounded question answering in images. In *CVPR*, 2016. 8

A. Visual Encoder and Multi-head Attention

The multi-head attention network is illustrated in Figure 8:

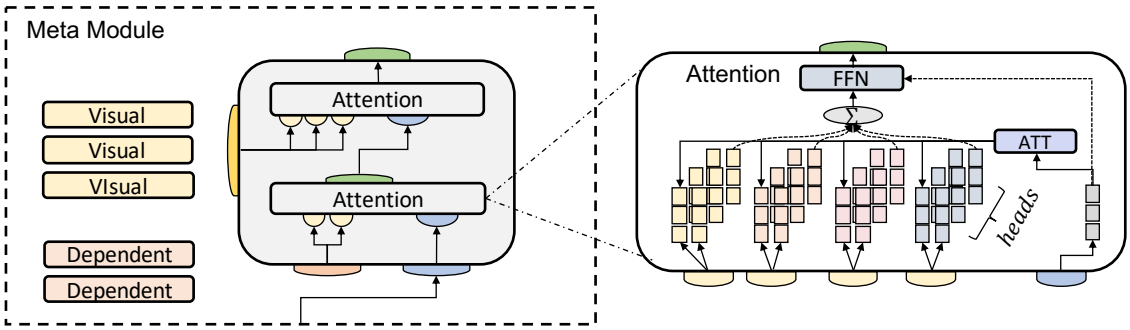

Figure 8. Illustration of the multi-head attention network used in the Meta Module.

B. Recipe Embedding

The recipe embedder is illustrated in Figure 9.

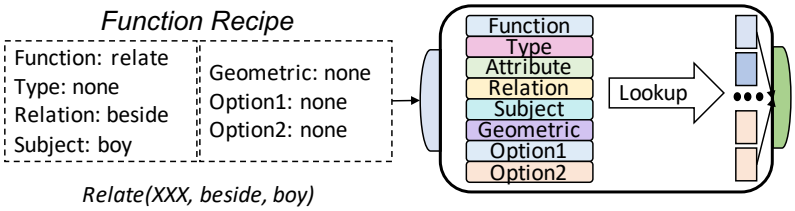

Figure 9. Illustration of the recipe embedder.

C. Implementation

The implementation of the model is demonstrated in Figure 10.

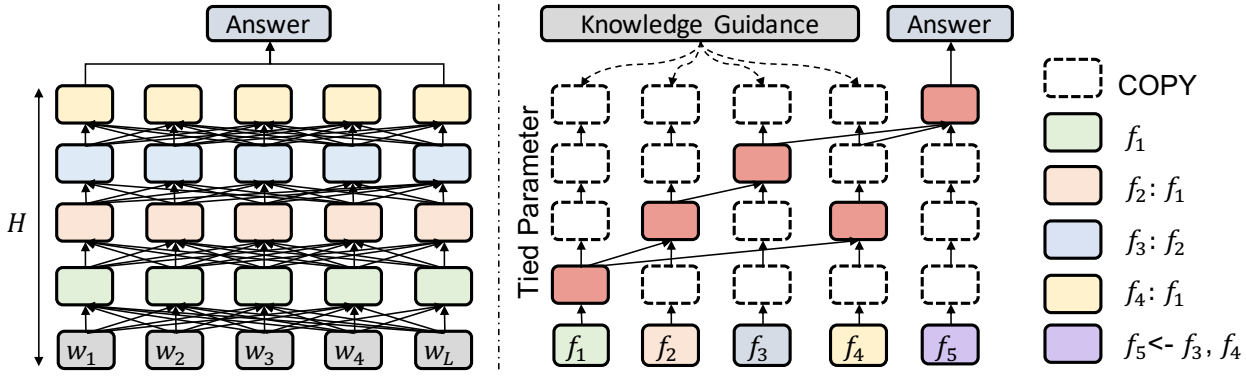

Figure 10. Illustration of the model implementation.

D. Function Statistics

The function statistics is listed in Table 5.

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.

E. Inferential Chains

More inferential chains are visualized in Figure 11 and Figure 12.

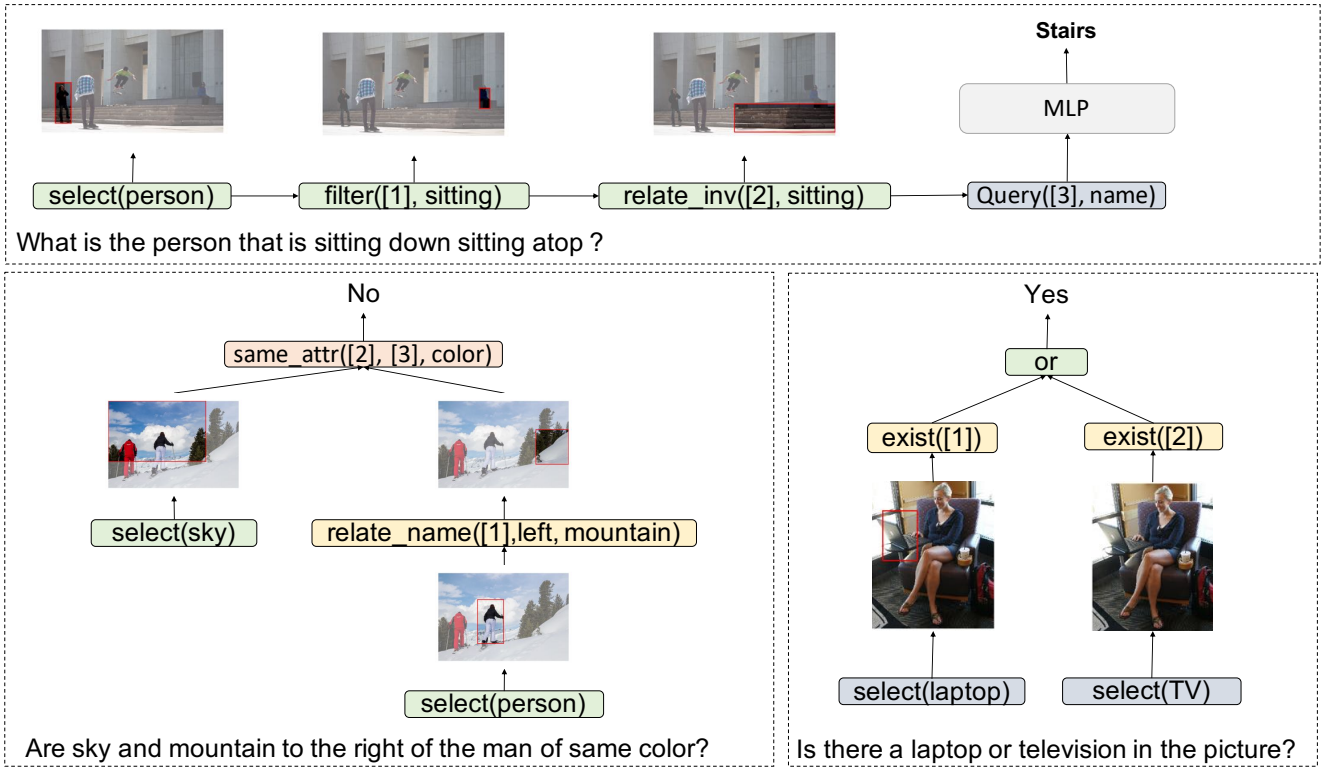

Figure 11. More examples on visualization of the inferential chains learned by our model.

F. Function Description

The detailed function descriptions are provided in Figure 13.

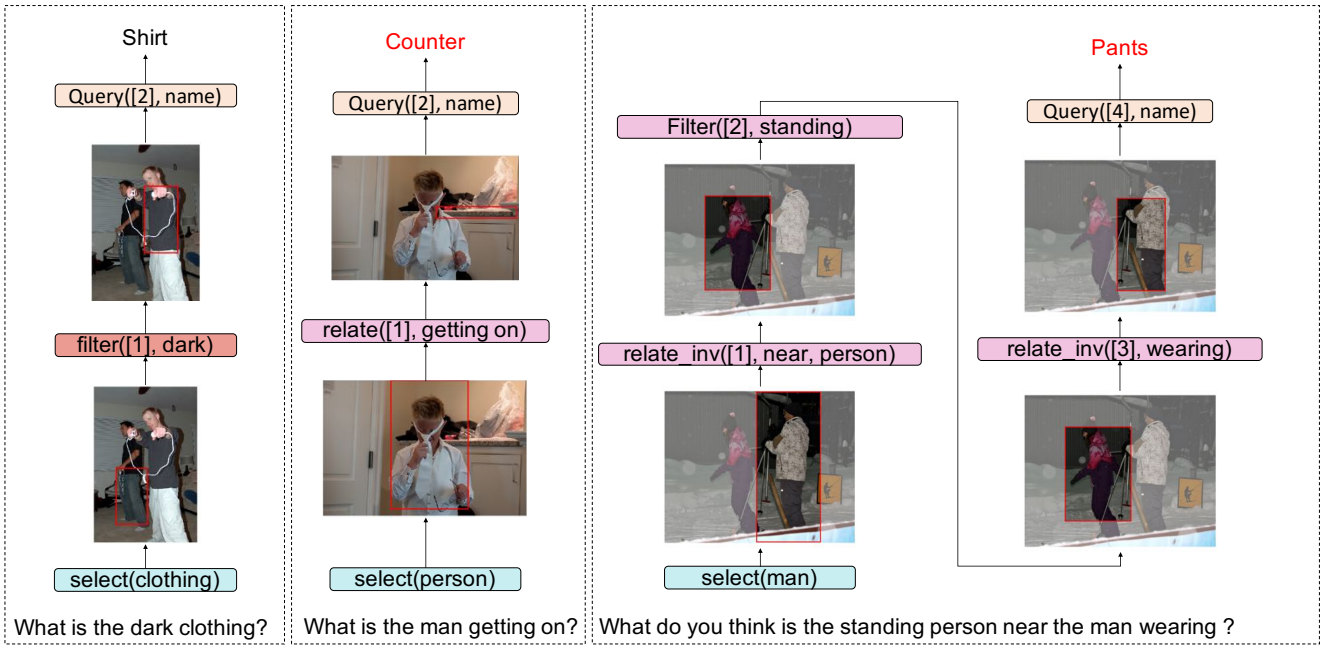

Figure 12. More examples on visualization of the inferential chains learned by our model.

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 13. The function definitions and their corresponding outputs.