Supplementary Materials for Closed Loop Neural-Symbolic Learning via Integrating Neural Perception, Grammar Parsing, and Symbolic Reasoning

1. Posterior Approximation

In Section 3.2.3, we formulate the m-step back search as a Metropolis-Hasting sampler to perform sampling from $p'(z|x,y)$, which is a smoothing of the true posterior distribution $p(z|x,y)$ as shown in Equation 11. Intuitively, as ϵ gets smaller, the distance between two distribution $p'(z|x,y)$ and $p(z|x,y)$ becomes smaller as well. Accordingly, we have the following lemma proved with Equation 9 and Equation 11:

Lemma 1.1. *Given an small ϵ, the Kullback–Leibler divergence of $p'(z|x,y)$ from $p(z|x,y)$ is $O(\epsilon)$.*

Proof. From the definition of Kullback–Leibler divergence, we have:

$$KL(p||p') = \sum_z p(z|x,y) \log \frac{p(z|x,y)}{p'(z|x,y)} \tag{1}$$

$$= \sum_z p(z|x,y) \log[\frac{p(y|z)}{p(y|z)+\epsilon} \cdot \frac{\epsilon + \sum_{z'} p(y|z')p_\theta(z'|x)}{\sum_{z'} p(y|z')p_\theta(z'|x)}] \tag{2}$$

$$= \sum_{z \in Q} \frac{p_\theta(z|x)}{C} \log \frac{C+\epsilon}{(1+\epsilon)C} \tag{3}$$

$$= \log \frac{C+\epsilon}{(1+\epsilon)C} \tag{4}$$

$$= \log(1+\frac{\epsilon}{C}) - \log(1+\epsilon) \tag{5}$$

where $C = \sum_{z'} p(y|z')p_\theta(z'|x) = \sum_{z' \in Q} p_\theta(z'|x)$ is the normalizing constant. With Taylor expansion, we get:

$$\log(1+\frac{\epsilon}{C}) = \frac{\epsilon}{C} + O(\epsilon^2) \tag{6}$$

$$\log(1+\epsilon) = \epsilon + O(\epsilon^2) \tag{7}$$

Then we have:

$$KL(p||p') = (\frac{1}{C} - 1)\epsilon + O(\epsilon^2) = O(\epsilon) \tag{8}$$

□

2. Handwritten Formula Recognition

2.1. Grammar for Math Formulas

For the handwritten formula recognition task, we define the context-free grammar for the mathematical formulas, as shown in Table 1. This grammar considers only simple arithmetic operations over single-digit numbers. We compute the parsed results using a calculator, which is the symbolic reasoning module in this task.

To be noticed, the proposed method can be extended to more complex computations by designing more complicated grammar.

Table 1. The context-free grammar for the mathematical formulas.

G = (V, Σ, R, S)
V= {S, Expression, Term, Factor}
$\Sigma = \{+, -, \times, \div, 0, 1, ..., 9\}$.
S is the start symbol.
R = {
S → Expression
Expression → Term
Expression → Expression + Term
Expression → Expression - Term
Term → Factor
Term → Term × Factor
Term → Term ÷ Factor
Factor → 0|1|2|3...|9 }

2.2. Data Generation

We generate the synthetic dataset based on CROHME 2019 Offline Handwritten Formula Recognition Task[1]. First, we extract all the image patches of symbols from CROHME and only keep ten digits (0∼9) and four basic operators (+,−,×, ÷). We split these images of symbols into a training symbol set (80%) and a testing symbol set (20%). Then we generate formulas by randomly sampling production rules from the predefined grammar. For the training set, we generate 1K formulas with length 1 (1 digit, 0 operator), 1K formulas with length 3 (2 digits, 1 operator), 2K formulas with length 5 (3 digits, 2 operators), and 6K formulas with length 7 (4 digits, 3 operators). For the test set, we generate 200 formulas with length 1, 200 formulas with length 3, 400 formulas with length 5, and 1,200 formulas with length 7. For each formula in the training/test set, we randomly select symbol images from the training/test symbol set. In this way, one symbol image can not exist in both the training set and the test set. Overall, our dataset contains 10K training formulas and 2K test formulas. The generated dataset is also submitted with the code.

[1] https://www.cs.rit.edu/~crohme2019/task.html

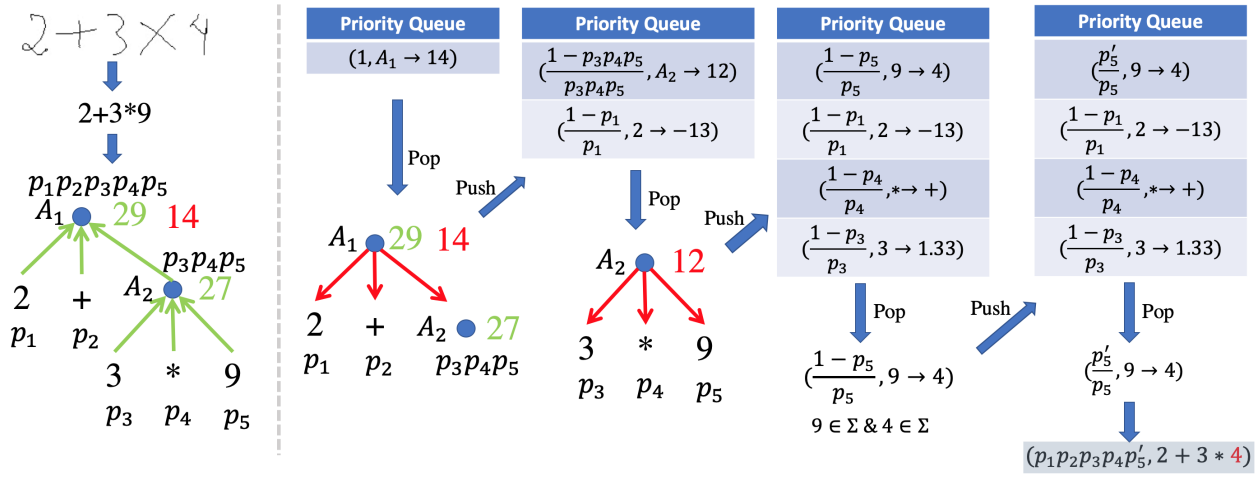

Figure 1. An illustrative example of the 1-BS process. The priority queue ranks the possible corrections to the original results with visiting priority, which reflects the potential of changing the current node or its child nodes to correct the wrong answer.

2.3. Training Details

For the proposed Neural-Grammar-Symbolic models, we use LeNet as the neural perception module and train the models for 100K iterations using the Adam optimizer with a fixed learning rate of 5×10^{-4} and a batch size of 64. For the REINFORCE and reproduced MAPO baselines, we set the reward decay as 0.99. For more details in the implementation and reproduction of the experiment results, please refer to the submitted code.

2.4. Qualitative Examples

Figure 1 shows an illustrative example of the 1-BS process implemented with a priority queue. Figure 2 shows several examples of correcting the wrong predictions using the m-BS algorithms.

3. Neural-Symbolic VQA

3.1. Grammar for CLEVR programs

The grammar model in the neural-symbolic VQA task ensures the generated sequence of function modules can form a valid program, which indicates the inputs and outputs of these modules can be strictly matched. Table 3 groups all function modules by the inputs and output types and Table 2 gives the context-free grammar for the CLEVR programs.

Table 2. The context-free grammar for the CLEVR programs.

G = (V, Σ, R, S)
V = {S, ObjectSet, Concept, Integer, Object}
Σ is the set of all modules as listed in Table 3.
S is the start symbol.
R = {
S → `count` ObjectSet
S → `equal_attribute` Concept Concept
S → `exist` ObjectSet
S → `greater_than` Integer Integer
S → `less_than` Integer Integer
S → `equal_integer` Integer Integer
S → `query_attribute` Object
ObjectSet → `scene`
ObjectSet → `filter_attribute[concept]` ObjectSet
ObjectSet → `intersection` ObjectSet ObjectSet
ObjectSet → `union` ObjectSet ObjectSet
ObjectSet → `relate[RelConcept]` Object
ObjectSet → `same_attribute` Object
Concept → `query_attribute` Object
Integer → `count` ObjectSet
Object → `unique` ObjectSet }

3.2. Implementation Details

The structure of the NGS model is shown in Figure 3. To get the structural scene representations, we train a scene parser following (Yi et al., 2018). Specifically, Mask-RCNN (He et al., 2017) is used to generate segment proposals of all objects in each image. Along with the segmentation mask, the network also predicts the categorical labels of discrete intrinsic attributes such as color, material, size, and shape. The segment for each object is then paired with the original image and sent to a ResNet-34 to extract the spacial attributes such as pose and 3D coordinates. Both networks of the scene parser are trained on 4,000 generated CLEVR images with full annotations. Please refer to (Yi et al., 2018) for more training details of the scene parser.

Instead of the attention-based seq2seq model used by (Yi et al., 2018), we use a Pointer Network as the question parser. Considering the small vocabulary of the CLEVR questions, we can easily build a dictionary to map the keywords in the question to the corresponding modules. Therefore, for each question, we can extract a set of functional modules, and the ground-truth program is a permutation of this set

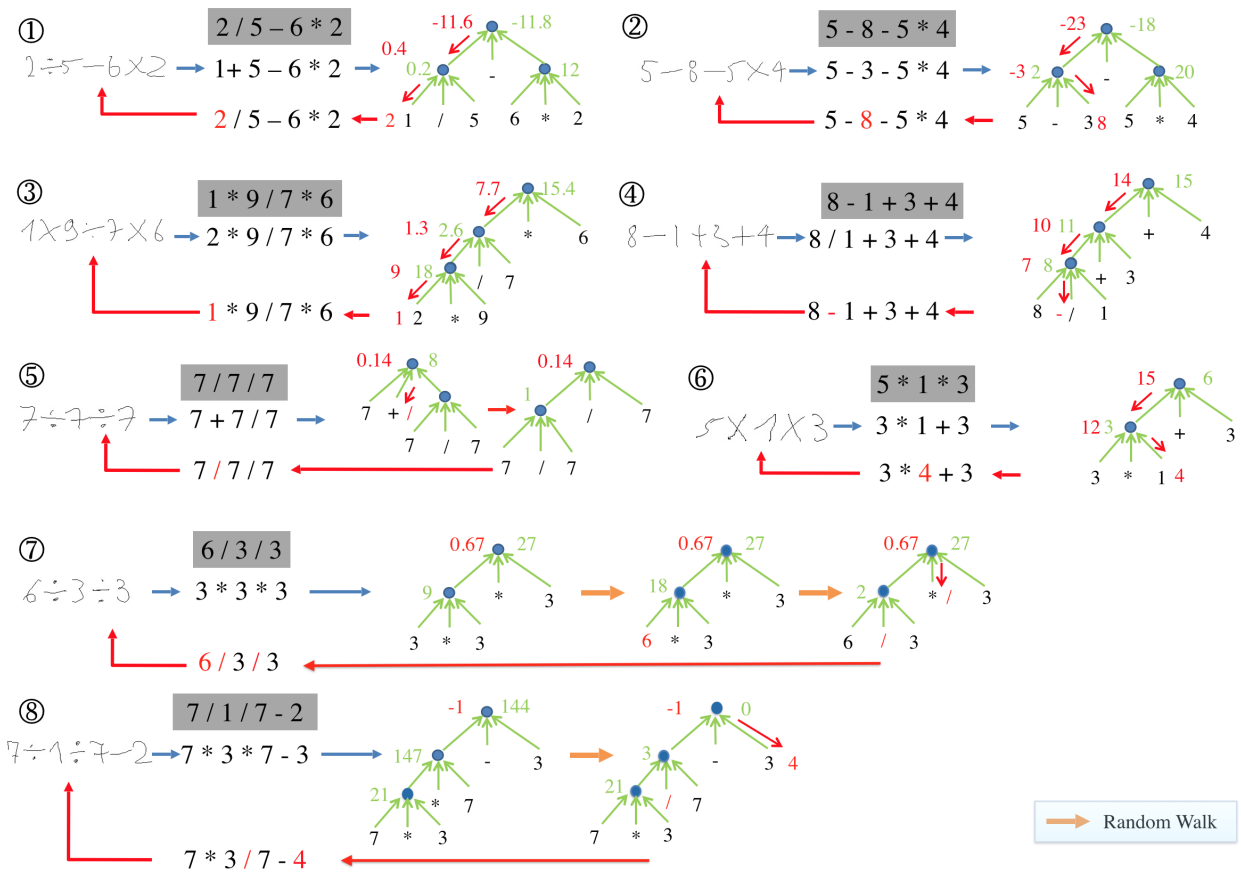

Figure 2. Examples of correcting the wrong predictions using the proposed m-BS algorithm. Some of the wrong predictions are corrected with the randomly sampled random walks noted by the yellow arrows. (6) and (8) are the spurious examples mentioned in Section 4.1.2

of modules. In the Pointer Network, both the encoder and decoder are two-layer LSTMs with 256 hidden units. We set the dimensions of both the encoder and decoder word embedding to 300. The Pointer Network works as the neural perception module in the proposed NGS model. Unlike (Yi et al., 2018), we do not need to pre-train the question parser on a small set of ground-truth question-program pairs.

The symbolic reasoning module in this task executes the generated program on the structural scene representations. The program executor is implemented as a collection of deterministic, generic functions in Python, designed to host all the functional modules in the CLEVR programs. Each function is in one-to-one correspondence with a module from the input program sequence, which has the same representation as in (Johnson et al., 2017; Yi et al., 2018). The execution of a program tree starts from the leaf nodes with `scene` tokens and continues until the root node, which outputs the final answer to the question.

Since the set of the functional modules is given for each question, the 1-step back search algorithm works by *switching* two modules that belong to the same group according to Table 3.

All models are trained with 30K iterations using the Adam optimizer with a fixed learning rate of 1×10^{-5} and a batch size of 64. For the REINFORCE and MAPO baselines, we set the reward decay as 0.99.

3.3. Qualitative Examples

Figure 4 shows several illustrative examples of correcting the wrong programs using the 1-BS model.

References

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. In *ICCV*, 2017.

Johnson, J., Hariharan, B., Van Der Maaten, L., Hoffman, J., Fei-Fei, L., Lawrence Zitnick, C., and Girshick, R. Inferring and executing programs for visual reasoning. In *ICCV*, 2017.

Yi, K., Wu, J., Gan, C., Torralba, A., Kohli, P., and Tenenbaum, J. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *NeurIPS*, 2018.

Table 3. Modules in the CLEVR programs. They are grouped by their inputs and output signatures. Modules listed in the same group (row) can replace each other while keeping the program valid.

Modules	Inputs	Output	Semantics
`scene`	∅	ObjectSet	Return all objects in the scene.
`count`	ObjectSet	Integer	Count the number of objects in the input object set.
`equal_attribute`	Concept, Concept	Bool	Check if two input concepts equal.
`exist`	ObjectSet	Bool	Check if the object set is empty.
`filter_attribute[concept]`	ObjectSet	ObjectSet	Filter out a set of objects having the object-level concept
`intersection`, `union`	ObjectSet, ObjectSet	ObjectSet	Return the intersection or union of two object sets.
`greater_than`, `less_than`, `equal_integer`	Integer, Integer	Bool	Compare two integers.
`query_attribute`	Object	Concept	Query the attribute (e.g., color) of the input object.
`relate[RelConcept]`, `same_attribute`	Object	ObjectSet	Filter out objects with the relational concept or same attribute.
`unique`	ObjectSet	Object	Return the unique object in the object set.

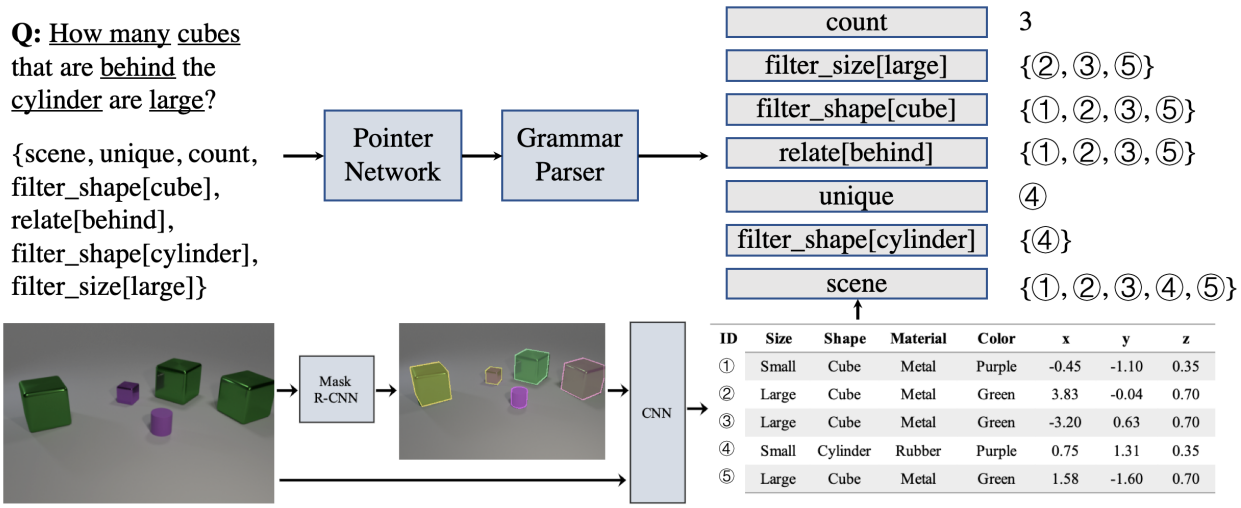

Figure 3. The Neural-Grammar-Symbolic VQA model for the CLEVR dataset.

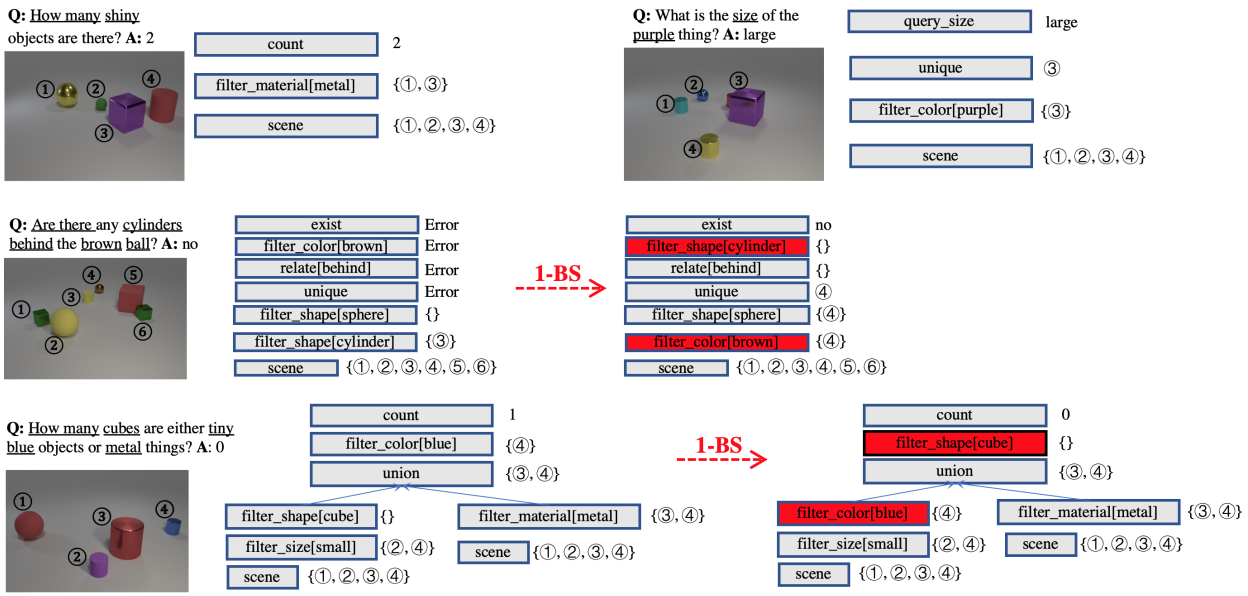

Figure 4. Illustrative examples of correcting the wrong programs using the 1-BS algorithm. * * * denotes the switched modules in the 1-BS algorithm. In the first two simple examples, given the set of the functional modules, only one permutation can form a valid program, and we do not need to use the back search algorithm. In another two examples, 1-BS successfully finds the correct programs.