Using Seq2Seq Models to Assemble Reasoning Modules for VQA

Hersh Sanghvi, Po Yuan Wang
School of Engineering and Applied Sciences
University of Pennsylvania
Philadelphia, Pennsylvania, USA

***Abstract*—Large Visual Question Answering (VQA) models suffer from problems such as overfitting and bias. One approach to combat these problems is to split the task of answering a question into smaller reasoning operations, also referred to as functional programs, so that smaller models can be used for each operation. Recent datasets such as GQA and CLEVR also explicitly annotate training questions with the functional programs necessary to answer them. We use these annotations as ground truth supervision and investigate the use of sequence-to-sequence (Seq2Seq) models to generate functional programs directly from VQA questions. We evaluate a variety of models and compare their performance to a semantic parser which has been used in previous work. We demonstrate that our Seq2Seq models are able to regress functional programs that are close to the ground truth annotations, suggesting that they can be used in future work involving modular VQA.**

I. Introduction

Visual Question Answering (VQA) has gained attention in both the natural language processing and computer vision research communities because it is a difficult problem that involves multimodal reasoning over both language and images. Since the advent of deep learning, notable approaches to solve this problem have ranged from monolithic, end-to-end style approaches where the question and image are encoded by sequential networks and convolutional neural networks (CNNs), to more structured approaches involving structured fusions of both visual and image features.

One problem with these monolithic approaches is that large visual and language models are difficult to train, can incorporate large amounts of biases if the datasets are not balanced properly, and their outputs are not very interpretable. To deal with this, one proposed approach has been to represent VQA problems as functional programs that are sequences of a small, fixed set of visual functions whose inputs depend on the text of the question and the features in the image. These functional programs represent the reasoning process involved in answering the question using the image. With this approach, multiple fixed reasoning components can be reused across different questions, as opposed to simply learning a direct mapping from question features to answer features as the monolithic models do.

One interesting idea is to directly learn how to execute each of the functions as modules, and simply execute the functional program when given a new question and image pair. These fixed reasoning modules can be chosen by the system designer, and if represented by deep networks, can be independently and easily trained in a supervised fashion, or trained end-to-end. However, the problem still remains of how to map from question and image to the exact functional program. In this project, we investigate the use of sequence-to-sequence (Seq2Seq) models to generate functional programs involving visual reasoning modules to solve visual question answering (VQA) problems.

II. Related Work

A. Seq2Seq

Our models are mainly based on Seq to Seq model and attention mechanism. Seq to Seq model was first proposed in [1], which is originally used for end to end language translation. It shows that Deep Neural Nets can be used to perform "End to End" Translation and proposing a 2 LSTM (an "Encoder"- "Decoder") architecture to do Langauge Translation. The encoder takes the input sequence and maps it to a fixed dimension vector while the decoder takes the output vector from the encoder and maps it to the output sequence.

Attention is originally proposed in [2] , the purpose of attention mechanism is to solve the problem in the original Seq to Seq model, which the encoder processes the input sequence and compresses the information into a fixed-length context vector. A critical issue of this fixed-length context vector design is incapability of remembering long sentences. It often forgets the first part once it completes processing the whole input. The attention mechanism was created to improve memorizing long source sequence in neural machine translation (NMT). Instead of building a single context vector out of the encoder's last hidden state, the attention layer create shortcuts between the context vector and the entire source sequence input. The weights of these shortcut connections can be modified for each output element. The alignment between the source and target sequence is also learned and controlled by the context vector.

B. Modular VQA

Modular approaches are designed to solve VQA problems, they explicitly decompose the reasoning process into a chain of subtasks. A well-known Neural Module approach is the Neural Module Network (NMN) [3], which is the model we will be testing and evaluate on GQA, solves the question answering by parsing questions into linguistic substructures and assembling

question-specific deep networks from smaller modules that each solve one subtask. N2NMN(End-To-End Neural Module Network) [4] is similar to the Neural Module Network, but without using external parser. The N2NMN model builds question-specific networks on the fly and executes the model. During the optimization stage, the model uses reinforcement learning to learn the policy for the non-differential portion that can't be optimized directly using back-propagation. Stack Neural Module Networks [5] was later proposed, and is based on the N2NMN, except that the layout policy is a soft distribution on the modules options and select the best possible module base on softmax, instead of making discrete choice like the N2NMN; soft attention distribution are also placed among the text words to extract the textual information for the selected modules to execute on; therefore, the model doesn't need end-to-end reinforcement learning to train the layout policy since it is fully differentiable, which can be trained using propagation. There are more variants for the module approaches, the overall architectures are similar, the main difference are mainly in the layout policy, therefore we think by working on the Neural Module Network, it would give us more insight to the behavior of the parser and how it affects the layout modules selection. In addition, our experiment uses a wider variety of modules than those used in the original NMN paper.

C. Functional Programs in VQA

Functional Programs in Visual Question Answering are recently gaining more attention to deal with more complex multimodal training tasks. The original VQA datasets [6],[7] do not annotate the questions with the functional programs, but recent datasets have included these annotations. CLEVR [8] is one of the datasets that the questions are represented as functional programs that can be run to answer the question, with the goal of enabling detailed analysis of visual reasoning. The information in each image is complete and exclusive so that commonsense knowledge can't increase the chance of answering questions correctly.The dataset minimize question-conditional bias through rejection sampling within families of related questions. The dataset also provides structured ground-truth representations for both images and questions.

In this project, we use GQA dataset [9] to train and evaluate our model. GQA is a new dataset for real-world visual reasoning and question answering, aiming to solve the issues in the VQA datasets. The author have developed a robust question engine that leverages information about objects, attributes and relations provided through Visual Genome Scene Graphs, along with a newly-created extensive linguistic grammar which couples hundreds of structural patterns and detailed lexical semantic resources.

III. Problem Formulation

We formulate this problem as learning to map sequences from one vocabulary to sequences in another, similar to machine translation. Our input vocabulary consists of the English words present in the GQA balanced training dataset. Each of the questions in the GQA dataset is also annotated with the functional program containing the necessary reasoning steps to answer the question. In total, there are 12 unique operators in GQA: `select`, `verify`, `filter`, `relate`, `query`, `exist`, `choose`, `same`, `common`, `different`, `and`, `or`. Some of these operations also take in natural language arguments that are determined by the text of the question. For our initial investigation, we chose to exclude these extra keywords. However, in later models, we also included these arguments (also referred to as "keywords" in this report). The frequencies with which these operators are used in the ground truth functional programs of the GQA training questions are shown in Figure 1.

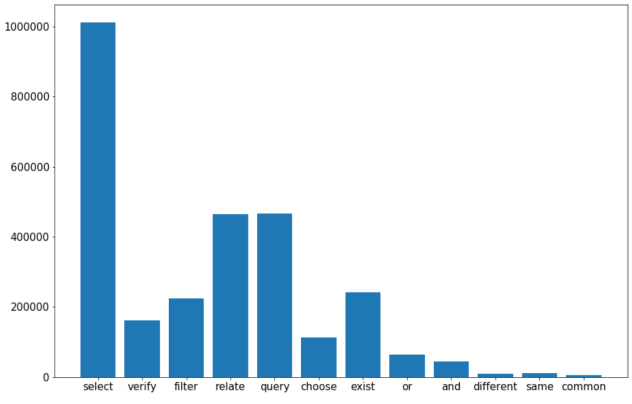

Fig. 1. A histogram showing many times each operator appears in the GQA training data

Because the questions in GQA are already annotated with the necessary functional programs, we can use a simple supervised learning approach where the inputs are the questions, and the labels come from the ground truth functional programs included in GQA.

IV. Models

We implemented a number of models in our project, ranging from a non-learned parser to a more complex attention-based model to convert from the input questions to the output functional programs.

A. Non-learned parser

As a baseline of sorts, we use the semantic parser from [3]. In their method, they first run the Stanford Tree Parser [10] on input questions in order to obtain a typed tree representation of the question, which also tags the words of the questions with their corresponding parts of speech. Based on this tree, they execute a custom parser to generate their "module layouts". As stated in [4], this parser is rigid and often produces layouts that don't entirely make sense for the question. However, we wanted to use this as it would provide a good non-learned baseline to compare against our own approaches.

However, the NMN modules in the original paper do not exactly correspond to the operators used in the GQA dataset. The NMN modules are: `find`, `relate`, `describe`, `and`,

NMN Operator	GQA operator
`find`	`select`
`find`[NN, JJ]	`select, filter`
`find`[NN, VBZ]	`select, relate`
`relate`	`relate`
`is`	`verify`
`describe`	`query`

TABLE I
OUR CONVERSIONS FROM NMN OPERATORS TO GQA OPERATORS

`is`. Because we lacked the expertise to write a semantic parser from scratch that maps from the typed tree to all of the GQA operators, we decided instead to convert from the NMN operators to a roughly equivalent functional program consisting of the GQA operators. These conversions also depend on the parts of speech of the arguments taken in by the NMN operators. The exact conversions are shown in Table I. From our interpretation of their codebase, it is also unclear how the arguments play into their layouts of the modules. Therefore, we omit the keywords in the conversion, and only convert from operators to operators. Note that because there are fewer NMN operators than GQA operators, we only map onto a subset of the GQA operators. Although the conversion is not perfect, it does capture the most frequent operators in the GQA dataset.

B. LSTM models

1) Conventional Seq2Seq: We first implement the basic recurrent Seq2Seq model [1]. This model is composed of two back-to-back recurrent models. One acts as en encoder, mapping the input sequence into a latent space, and the other takes this latent as an input and outputs a sequence from the output vocabulary. We also use LSTMs instead of conventional RNNs due to their superior performance.

First, we show the equations for the encoder. Given a sequence of tokens $X = [x_1, \quad x_2 \quad \dots \quad x_t]$ that represents an input question of length t, we construct the latent vector z as follows:

$$\begin{aligned} e_i &= E_{input}(x_i) \\ h_i, c_i &= \text{LSTM}(e_i, h_{i-1}, c_{i-1}) \\ z &= h_t \end{aligned}$$

Where h_i is the hidden state, c_i is the cell state, and e_i is the embedded token generated by $E_{input} : \mathbb{R} \mapsto \mathbb{R}^d$, an embedding function that maps from the integer tokens to a d dimensional embedding space. These can be either one-hot vectors or more complex word vectors such as GloVe embeddings [**?**]. The hidden and cell vectors h_i, c_i have dimensionalities that are hyperparameters of the model.

Theoretically, latent vector z contains a compressed representation of all of the information in the question needed to form the functional program / module layout. We can simply pass this latent vector into the decoder in order to obtain the functional program representation of the question. The decoder is governed by equations similar to those of the encoder:

$$\begin{aligned} o_0 &= \text{"sos"} \\ h_0 &= z \\ e_i &= E_{output}(o_{i-1}) \\ h_i, c_i &= \text{LSTM}(e_i, h_{i-1}, c_{i-1}) \\ o_i &= f(h_i) \end{aligned}$$

Where "sos" is the start of sequence token, $E_{input} : \mathbb{R} \mapsto \mathbb{R}^{d_o}$ maps from input tokens to an embedding space (not necessarily the same one as the encoder) of dimensionality d_o, and f is a fully-connected linear layer. Also of note is that the decoder does not take in any external inputs besides a "start of sequence", since the ground-truth functional programs are not known ahead of time. For all of our models, because the length of the functional programs are also not known by the decoder, we also rely on them to produce an "end" token at the end of the sequence.

2) Double Decoder Model: One problem with the model described in the previous section is that the arguments for the functional program are drawn from a fundamentally different vocabulary than the actual functional program modules. While the keywords come from a natural language vocabulary similar to that of the questions, the operators are contained in a much smaller vocabulary of size 12. Therefore, it conceptually doesn't make sense to try to represent both the modules and the keywords in the same embedding space. However, if we separate the vocabularies, this means we cannot use a single decoder anymore, since the embedding spaces are different.

To solve this problem, for our second type of model, we use two decoders instead of one. One decoder is responsible for generating the sequence of operators, and the other is responsible for decoding a single keyword input to each operator. We can exploit the structure in the functional programs to run these decoders fully in parallel, since the functional program always has the structure "`operator1`: keyword1 → `operator2`: keyword2...".

These two decoders do not receive any information about the other, since they use separate embeddings which are not necessarily the same dimension. Information sharing between these two parallel decoders is left up to future work, and would require a more sophisticated scheme to bridge between the two different embedding spaces.

3) Context Model: Another problem we sometimes observe in both the basic RNN model and the double decoder model is that the `select` operator is dominant as the first operator in the functional program, and can cause problems in propagating the information forward through the decoder. This problem can manifest as overfitting the functional program because it loses information about the source sequence in the process.

To combat this, we implement a modification to the decoders that allow them to incorporate information about the source sequence across all timesteps. This involves incorporating a context vector into the input to the decoder at each timestep. This idea is similar to the approach presented in [11].

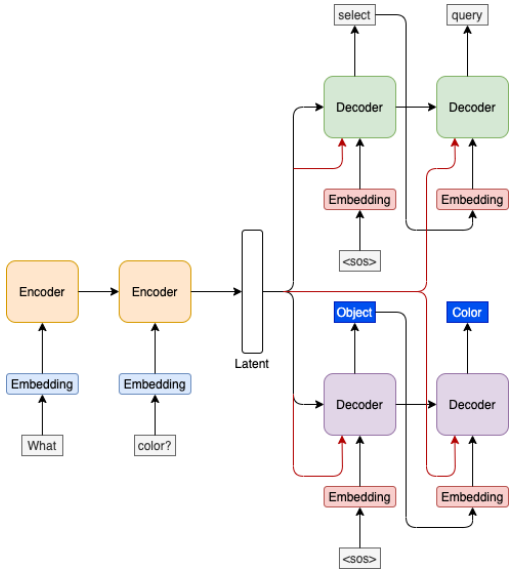

Fig. 2. The double decoder Seq2Seq model with context vectors. The context vectors are the arrows in red. For the normal double decoder model, they are not included. The correct reading of the output of the decoder is "`select`: Object → `query`: Color"

With this modification, the input to the decoder w_i and output o_i at each timestep is now:

$$
\begin{aligned}
e_i &= E_{output}(o_{i-1}) \\
w_i &= [e_i \quad z] \\
h_i, c_i &= \text{LSTM}(w_i, h_{i-1}, c_{i-1}) \\
o_i &= f([e_i \quad h_i \quad z])
\end{aligned}
$$

With this modification, the input w_i is now a concatentation of the embedding and the z vector, which is the final layer from the encoder that now acts as a context vector throughout the decoder. Also, the output o_i is calculated as a function of the concatentation of the embedding, hidden, and context states. Our hypothesis is that including these extra vectors will improve the performance by allowing the decoder to remember the input context at each timestep. We use this context augmentation specifically with our double decoder models, so each parallel decoder receives the same context vector from the encoder.

The double-decoder RNN-based model with context is shown in Figure 2.

C. Attention based models

Transformer has been a popular architecture in the NLP tasks nowadays, the architecture doesn't involve recurrence nor convolutional layers, but only rely on attention, linear layers and layer normalization mechanism. We applied a transformer model with two separate decoders, different from the original transformer architecture in the paper attention is all you need, we used a learned position encoding mechanism instead of a static one, and also fixing the learning rate of the ADAM optimizer; Moreover, we didn't use label smoothing in our task.

The architecture is a Seq to Seq model that involves encoder and decoder stages, The source mask is created by checking where the source sequence is not equal to a pad token. It is 1 if the token is not a pad token and 0 when it is. It is then expanded so it can be broadcast when applying the mask to the energy, in the shape of [batch size, number of heads, sequence length, sequence length]. For the target mask, similar to the source mask, we create a mask for the ¡pad¿ tokens, then we create a "subsequent" mask, which the elements above the diagonal will all be zero and the elements below the diagonal will be set to a value depending on what the input tensor is. In our model, the input tensor will be a tensor filled with ones.

During the encoder part, the source sentence is not compressed into a single context vector; instead, it produces a sequence of context vectors depending on how many tokens there are in the input sequence. Since recurrence layer isn't included in the transformer, a positional embedding layer is applied. The input to this layer is not the token itself but the position in the sequence sentence, we assigned the max vocab the embedding layer 100, which means the sequence sentence can have a max length of 100. The original transformer in the paper uses static embedding, but since BERT uses positional embedding and has achieve good results, we follow the implementation and used the learned positional embedding in this model. The token and positional embeddings are later elementwise summed together to get a vector that contains information about the tokens and also their position with in the sequence.

In the next step, the source sequence and source mask is passed to the encoder layers. The encoder layers first pass the source and mask to the multi-head attention layers and apply residual connection and passed it through the layer normalization layer. The layer-normalization layers normalize the values of the features across the hidden dimension, so each feature has a mean of 0 and a standard deviation of 1. This allows neural networks with a larger number of layers, like the Transformer, to be trained easier. The attention layer in the encoder apply attention over itself, in other words, its performing self-attention during this phase.

In the decoder stage, the decoder takes the encoded representation of the source sentence, and convert it into predicted tokens in the target sentence, in our case, we applied two separate decoder with the same structure, one for predicting the operators and one for predicting the keywords arguments; moreover, the decoder has two kinds of attention layers, a masked multi-head attention layer over the target sequence, and a multi-head attention layer which uses the decoder representation as the query and the encoder representation as the key and value.

In the decoder layers, they use the target sequence mask to prevent the decoder from paying attention to tokens that are "ahead" of the one it is currently processing as it processes all tokens in the target sentence in parallel. In this multi-head attention layer the queries are the decoder representations and

the keys and values are the encoder representations. Here, the source mask is used to prevent the multi-head attention layer from attending to pad tokens within the source sentence. This is then followed by the dropout, residual connection and layer normalization layers.

V. Experiments

For our experiments, we evaluate 6 different methods and models for predicting the module layouts. First, we evaluate the non-learned parser from the NMN paper, with the conversion described in Section 4A. Next, we evaluate the basic Seq2Seq model that only predicts the operators/modules, not the keywords (referred to as "Seq2Seq (Ops)"). We also evaluate a basic Seq2Seq model that jointly predicts operators and keywords with a single decoder ("Seq2Seq (Joint)"). For our double decoder models, we test one with and without context ("DD" and "DD + Context"). Finally, we also test an attention model ("Attention"). For our Seq2Seq (Ops), DD, and DD+Context models, we used pretrained GloVe 6B.100D embeddings [12] for the encoder and keyword decoder. We did not use GloVe embeddings for the Seq2Seq (joint) model or Attention models.

All learned models are trained on the GQA balanced training set, which consists of 900,000 questions. For our quantitative tests, we used the first 10,000 questions in the balanced validation set in GQA. Our methods for calculating our quantitative metrics are discussed below.

A. Scoring Operator Prediction Accuracy

The evaluation metric we used for evaluating the operators is a modified variant of the ROUGE metric [13] : Soft ROUGE. The reason we design the metric based on the original ROUGE is it calculates the longest common subsequences, which values the order and the length of similarites. Since we expect the operators to be concatenated in the correct order so that we could have more confidence that the Neural Modules Network layout can successfully select the right sub modules and execute them, but also the correctness of each operators is also important since every modules have different structures. However, the original ROUGE metric doesn't focus the similarity of different words. Since some operators in GQA are more similar than others (eg. `filter` and `relate` are more similar than `relate` and `verify`), we design a similarity matrix that define the similarity between the operators and assign different score when calculating the longest common subsequences.

B. Scoring Keyword Prediction Accuracy

Because the keywords are drawn from a much larger vocabulary than the operators, using the Soft ROUGE metric will not work for scoring the keywords. In addition, because a sequence of keywords does not follow grammatical conventions, we cannot use conventional scoring methods such as BLEU, which are intended for. Therefore, we adopt a simpler metric of comparing the distances of the predicted keywords from the model and the ground truth keywords in the GloVe embedding space, where distances between the word vectors are meaningful. In cases where the model's predicted sequence of keywords is too short, we pad the sequences with zero vectors, which also serves to penalize predicted sequences with incorrect length.

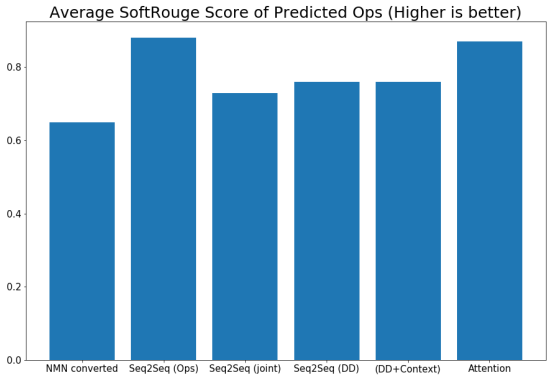

Fig. 3. SoftRouge score of models averaged over test questions. This metric tests operator prediction accuracy. Higher is better.

VI. Results

A. Quantitative Results

Our quantitative results on part of the GQA validation question set are shown in Figures 3 and 4. We can see that for the task of predicting operators, the Seq2Seq (Ops) model has the highest average Soft ROUGE score of 0.88. This is to be expected, since this model has the easiest task of only predicting the operators without keywords. The Attention model performs similarly in this regard, achieving an average score of 0.87. Both double decoder models perform better than the semantic parser, but fall short of the performance demonstrated by the Seq2Seq (Ops) model and Attention models when predicting operators. Predictably, the Seq2Seq (Joint) model exhibits the worst performance in predicting operators, due to the fact that it also must incorporate the keywords into the same decoder vocabulary.

When predicting the keywords, we see that the DD+Context model performs the best, suggesting that incorporating the context vector aids in the prediction of keywords, and the normal DD model performs worse. Somewhat surprisingly, the Attention model actually has a poor performance in this regard. This could be due to the fact that GloVe embeddings were not used when training the Attention Model. As predicted, the Seq2Seq (Joint) model falls far short in performance compared to the double decoder models.

B. Qualitative Results

We also include some qualitative results on five questions selected from the GQA balanced validation set in Tables II, III, IV. These questions were unseen during the training of the models. These questions range from long to short, and require some different reasoning sequences.

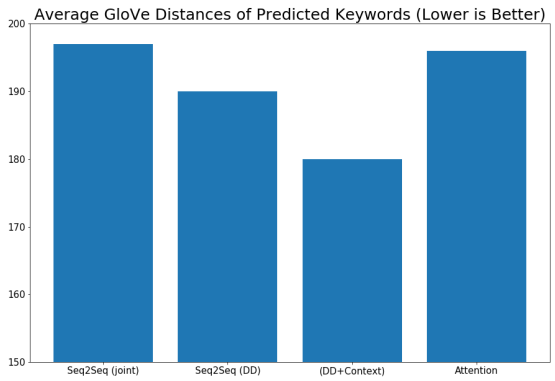

Fig. 4. Average GLoVe distance between predicted keywords and ground truth keywords, per program. Lower is better.

TABLE II
Q: "IS THERE ANY SURFBOARD TO THE RIGHT OF THE MAN THE PEOPLE ARE STANDING BY?"

Model	**Predicted Layout**
Ground truth	select: people → relate:man → relate: surfboard → exist
NMN Parse	find['there', 'by'] → 'find'['is', 'there'] → 'and' → 'is'
Seq2Seq (Ops)	select → relate → relate → exist
Seq2Seq (Joint)	select: people → relate: man → relate: surfboard → exist
DD	select: people → relate: man → relate: surfboards → exist
DD+Context	select: people → relate: man → relate: surfboard → exist
Attention	select: people → relate: man → relate: surfboard → exist

The normal Seq2Seq models tend to struggle with the shorter questions, such as in Table IV, possibly because they overfit to always choosing `select → verify` for these questions. The Attention model has strong performance in predicting operators, but sometimes predicts the incorrect keywords. Meanwhile both double decoder models fall in the middle in terms of performance, but including the context improves the results in terms of predicting both the operators and keywords. The NMN semantic parser produces some sane results in its own vocabulary, but struggles with selecting the correct keywords. We still need to do more investigation to understand the sources of the differences in the performance of the Attention model. However, these are promising results that suggest that Seq2Seq models can be used for a variety of questions to lay out reasoning sequences.

VII. DISCUSSION AND CONCLUSIONS

Our qualitative and quantitative results show promising results; even a simple model can predict layouts that are close to the ground truth programs, although introducing additional complexity into the models does improve the performance. However, our results do come with some caveats. Due to computational restrictions, we were unable to do full hyperparameter tuning on our models. Therefore, our results actually act as a lower bound for the performance achievable with Seq2Seq models. In addition, during training we observed that including pretrained GloVe embeddings improved the performance of the models, suggesting that if we also used the pretrained embeddings for the Attention model, its performance in predicting keywords would exceed those of the recurrent models.

Future work includes using our Seq2Seq models in a full pipeline with the models that are able to execute the functional operators in order to actually answer the questions. In addition, further evaluation on datasets such as the original VQA and CLEVR datasets is necessary so we can understand if our method is able to generalize to datasets with a different distribution of questions.

TABLE III
Q: "DO ALL OF THESE PEOPLE HAVE THE SAME GENDER?"

	Predicted Layout
Ground Truth	select: person → same: gender
NMN Parse	find['people', 'gender'] → find['do', 'people'] → and → is
Seq2Seq (Ops)	select → same
Seq2Seq (Joint)	select: person → same: gender
DD	select: people → select: <None> → different
DD + Context	select: person →same: <None>
Attention	select: person →same: <None>

TABLE IV
Q: "WHO IS WEARING GOGGLES?"

	Predicted Layout
Ground Truth	select: goggles → relate: person →query: name
NMN Parse	find'['is', 'goggles'] →describe['goggles', 'who]
Seq2Seq (Ops)	select → verify
Seq2Seq (Joint)	select: scene → verify: place
DD	select: goggles → filter: rel → verify
DD + Context	select: goggles → relate: <None> → query: name
Attention	select: goggles → relate: person → query: name

REFERENCES

[1] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[3] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, "Neural module networks," 2017.

[4] R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko, "Learning to reason: End-to-end module networks for visual question answering," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[5] R. Hu, J. Andreas, T. Darrell, and K. Saenko, "Explainable neural computation via stack neural module networks," 2019.

[6] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, "VQA: Visual Question Answering," in *International Conference on Computer Vision (ICCV)*, 2015.

[7] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh, "Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[8] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning," 2016.

[9] D. A. Hudson and C. D. Manning, "Gqa: A new dataset for real-world visual reasoning and compositional question answering," 2019.

[10] M.-C. de Marneffe, B. MacCartney, and C. D. Manning, "Generating typed dependency parses from phrase structure trees," in *LREC*, 2006. [Online]. Available: http://nlp.stanford.edu/pubs/LREC06_dependencies.pdf

[11] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[12] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. [Online]. Available: https://www.aclweb.org/anthology/D14-1162

[13] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://www.aclweb.org/anthology/W04-1013