Appendix: Probabilistic Neural-symbolic models for Interpretable Visual Question Answering

1. A variational lower bound on the evidence for sequential latent variable models

Lemma 1. *Given observations $\{\mathbf{x}^n, \mathbf{z}^n\}$ and $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p_\sigma(\mathbf{x}|\mathbf{z})$, let z_t, the token at the t^{th} timestep in a sequence $\mathbf{z}$ be distributed as a categorical with parameters π_t. Let us denote $\Pi = \{\pi_t\}_{t=1}^T$, the joint random variable over all π_t. Then, the following is a lower bound on the joint evidence $\mathcal{L} = \sum_n \log p(\mathbf{z}^n) + \log p_\sigma(\mathbf{x}^n|\mathbf{z}^n)$:*

$$\mathcal{L} \geq \sum_{n=1}^{N} \log q_\phi(\mathbf{z}^n|\mathbf{x}^n) + \log p_\sigma(\mathbf{x}^n|\mathbf{z}^n) - \mathbb{KL}\left(q(\Pi|\mathbf{z}^n, \mathbf{x}^n)||p(\Pi)\right) \quad (1)$$

where $p(\Pi)$ is a distribution over the sampling distributions implied by the prior $p(\mathbf{z})$ and $q(\Pi|\mathbf{x}^n, \mathbf{z}^n) = q(\pi_0)\prod_{t=1}^T q(\pi_t|\mathbf{x}^n, z_0^n, \cdots, z_{t-1}^n)$, where each $q(\pi_t|\cdot)$ is a delta distribution on the probability simplex.

Proof. We provide a proof for the lemma stated in the main paper, which explains how to obtain a lower bound on the evidence $\mathcal{L}$ for the data samples $\{\mathbf{x}^n, \mathbf{z}^n\}_{n=1}^N$.

Let us first focus on the prior on (serialized) programs, $\log p(\mathbf{z})$. This is a sequence model, which factorizes as, $\log p(\mathbf{z}) = \log p(z_1) + \sum_{t=2}^T \log p(z_t|\{z_i\}_{i=1}^{t-1})$, where $p(z_t|\cdot) \sim Cat(\pi_t)$. We learn point estimates of the parameters for the prior (in a regular sequence model), however, the every sequence has an implicit distribution on $p(\pi_t)$. To see this, note that at every timestep (other than $t = 1$), the sampling distribution depends upon $\{z_i\}_{i=1}^T$, which is set of random variables. Let $p(\Pi, \mathbf{z})$ denote the joint distribution of all the sampling distributions across multiple timesteps $\Pi = (\pi_1, \cdots, \pi_T)$ and $\mathbf{z}$, the realized programs from the distribution. Let $\mathcal{G} = \{\pi_i, \mathbf{z}_i\}_{i=1}^T$

Since π_t refers to the sampling distribution for z_t, we immediately notice that $z_t \perp \mathcal{G}/\pi_t|\pi_t$. That is, z_t only depends on the sampling distribution at time t, meaning that $\log p(\mathbf{z}|\Pi) = \sum_{t=1}^T \log p(z_t|\pi_t)$.

With this (stochastic) prior parameterization, we can factorize the full graphical model now as follows: $p(\mathbf{x}, \mathbf{z}, \Pi) = p(\Pi)p(\mathbf{z}|\Pi)p_\sigma(\mathbf{x}|\mathbf{z})$. We treat Π as a latent variable. Then, the marginal likelihood for this model is:

$$\mathcal{L} = \sum_{n=1}^{N} \log p(\mathbf{z}^n) + \log p_\sigma(\mathbf{x}^n|\mathbf{z}^n) \quad (2)$$

Notice that this equals the evidence for the original, graphical model of our interest. Next, we write, for a single data instance:

$$\begin{aligned}\log p(\mathbf{x}, \mathbf{z}) &= \log \int d\Pi p(\mathbf{x}, \mathbf{z}, \Pi) \\ &= \log \int d\Pi p(\Pi)p(\mathbf{z}|\Pi)p_\sigma(\mathbf{x}|\mathbf{z}) \\ &= \log \int d\Pi \frac{p(\Pi)}{q(\Pi|\mathbf{z}, \mathbf{x})} p(\mathbf{z}|\Pi)p_\sigma(\mathbf{x}|\mathbf{z})q(\Pi|\mathbf{z}, \mathbf{x})\end{aligned}$$

where we multiply and divided by a variational approximation $q(\Pi|\mathbf{z}, \mathbf{x})$ and apply Jensen's inequality (due to concavity of log) to get the following variational lower bound:

$$\log p(\mathbf{x}, \mathbf{z}) \geq \int d\Pi q(\Pi|\mathbf{z}, \mathbf{x}) \left[\log p(\mathbf{z}|\Pi) + \log p_\sigma(\mathbf{x}|\mathbf{z})\right] - \mathbb{KL}\left(q(\Pi|\mathbf{z}, \mathbf{x})||p(\Pi)\right) \quad (3)$$

Next, we explain how to parametrize the prior, which is the key assumption for deriving the result. Let $\hat{\pi}_t = f(\mathbf{x}, \{\mathbf{z}_i\}_{i=1}^{t-1})$ be the inferred distribution for $\mathbf{z}_t$. That is, $\mathbf{z}_t = Cat(\hat{\pi}_t)$. Then, let $q_\phi(\pi_t|\mathbf{x}, \{\mathbf{z}_i\}_{i=1}^{t-1}) = \delta(\pi_t - \hat{\pi}_t)$, that is, a Dirac delta function at $\hat{\pi}_t$.

Further, note that we can write the second term $E_{q(\Pi|\mathbf{z},\mathbf{x})}\left[\log p(\mathbf{z}|\Pi)\right]$ as follows:

$$\int d\Pi q(\Pi|\mathbf{z}, \mathbf{x}) \log p(\mathbf{z}|\Pi) \quad (4)$$

$$= \sum_{t=1}^{T} \int d\pi_t q(\pi_t|\{z_i\}_{i=1}^{t-1}, \mathbf{x}) \log p(z_t|\pi_t) \quad (5)$$

Each integral in the sum above can be simplified as follows:

$$\begin{aligned}&\int d\pi q(\pi_t|\{z_i\}_{i=1}^{t-1}, \mathbf{x}) \log p(z_t|\pi_t)\\&= \int d\pi \delta(\pi_t - \hat{\pi}_t) \log p(z_t|\pi_t)\\&= \log p(z_t|\hat{\pi}_t)\\&= \log q_\phi(z_t|\{z_i\}_{i=1}^{t-1}, \mathbf{x}) \ \texttt{[by definition]}\end{aligned}$$

Substituting into Equation (4), we get:

$$\begin{aligned}&\sum_{t=1}^{T} \int d\pi_t q(\pi_t|\{z_i\}_{i=1}^{t-1}, \mathbf{x}) \log p_\sigma(z_t|\pi_t)\\&= \sum_{t=1}^{T} \log q_\phi(z_t|\{z_i\}_{i=1}^{t-1}, \mathbf{x})\\&= \log q_\phi(\mathbf{z}|\mathbf{x})\end{aligned}$$

This gives us the final bound, written over all the observed data points:

$$\mathcal{L} \geq \sum_{n=1}^{N} \log p_\sigma(\mathbf{x}^n|\mathbf{z}^n) + \log q_\phi(\mathbf{z}^n|\mathbf{x}^n) - \mathbb{KL}\left(q(\Pi|\mathbf{x}, \mathbf{z})||p(\Pi)\right) \quad (6)$$

□

2. Justification for α

We optimize the full evidence lower bound $\mathcal{L}_{qc} + \mathcal{U}_{qc}$ using supervised learning for $\mathcal{L}_{qc}$ and REINFORCE for $\mathcal{U}_{qc}$. When using REINFORCE, notice that $\nabla_\phi \mathcal{U}_{qc} = \nabla_\phi \log q(\mathbf{z}|\mathbf{x}^m)[R_{qc} - B]$, where B is a baseline. For the lower bound on $\mathcal{L}_{qc}$, the gradient is $\nabla_\phi \log q(\mathbf{z}^n|\mathbf{x}^n)$. Notice that the scales of the gradients from the two terms are very different, since the order of R_{qc} is around the number of bits in a program, making it dominate in the early stages of training.

3. Justifications for $\beta < 1$

During question coding, we learn a latent representation of programs in presence of a question reconstructor. Our reconstructor is a sequence model, which can maximize the likelihood of reconstructed question without learning meaningful latent representation of programs. Recent theory from Alemi et al. (2018) prescribes changes to the ELBO, setting $\beta < 1$ as a recipe to learn representations which avoid learning degenerate latent representations in the presence of powerful decoders. Essentially, Alemi et al. (2018) identify that up to a constant factor, the negative log-marginal likelihood term D=$-E_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\sigma(\mathbf{x}|\mathbf{z})]$, and the KL divergence term $R = \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}), p(\mathbf{z}))$ bound the mutual information between the data $\mathbf{x}$ and the latent variable $\mathbf{z}$ as follows:

$$H - D \leq I(\mathbf{x}, \mathbf{z}) \leq R \quad (7)$$

where H is the entropy of the data, which is a constant. One can immediately notice that the standard $\mathrm{elbo} = -D - R$. This means that for the same value of the elbo one can get different models which make drastically different usage of the latent variable (based on the achieved values of D and R). Thus, one way to achieve a desired behavior (of say high mutual information $I(\mathbf{x}, \mathbf{z})$), is to set β to a lower value than the standard elbo (c.f. Eqn. 6 in Alemi et al. (2018).) This aligns with our goal in question coding, and so we use $\beta < 1$.

4. Implementation Details for SHAPES

We provide more details on the modeling choices and hyperparameters used to optimize the models in the main paper.

Sequence to Sequence Models: All our sequence to sequence models described in the main paper (Section. 2) are based on LSTM cells with a hidden state of 128 units, single layer depth, and have a word embedding of 32 dimensions for both the question as well as the program vocabulary.

The when sampling from a model, we sample till the maximum sequence length of 15 for questions and 7 for programs.

Image CNN: The SHAPES images are of `30x30` in size and are processed by a two layered convolutional neural network, which in its first layer has a `10x10` filters applied at a stride of 10, and in the second layer, it does `1x1` convolution applied at a stride of 1. Channel widths for both the layers are 64 dimensional.

Moving average baseline: For a reward R, we use an action independent baseline b to reduce variance, which tracks the moving average of the rewards seen so far during training. Concretely, the form for the $q(\mathbf{z}|\mathbf{x})$ network is:

$$\nabla J(\theta) = \mathbf{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}[(R - b_t)\nabla \log q(\mathbf{z}|\mathbf{x})] \quad (8)$$

where, given D as the decay rate for the baseline,

$$b_t = b_{t-1} + D * (R - b_{t-1}) \quad (9)$$

is the update on the baseline performed at every step of training.

In addition, the variational parameters ϕ are also updated via. the path derivative $\nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{x})$.

Training Details: We use the ADAM optimizer with a learning rate of `1e-3`, a minibatch size of 576, and use a moving average baseline for REINFORCE, with a decay factor of 0.99. Typical values of β are set to 0.1, α is set to 100 and γ is chosen on a validation set among (1.0, 10.0, 100.0).

Simulating programs from known syntax: We follow a simple two-stage heuristic procedure for generating a set of samples to train the program prior in our model. Firstly, we build a list of possible future tokens given a current token, and sample a random token from that list, and repeat the procedure for the new token to get a large set of possible valid sequences – we then pass the set of candidate sampled sequences through a second stage of filtering based on constraints from the work of (Hu et al., 2017).

5. Additional Results on SHAPES

Empirical *vs.* syntactic priors. While our default choice of the prior $p(\mathbf{z})$ is from programs simulated from the known syntax, it might not be possible to exhaustively enumerate all valid program strings in general. Here, we consider the special case where we train the prior on a set of *unaligned*, ground truth programs from the dataset. Interestingly, we find that the performance of the question coding stage, especially at reconstructing the questions improves significantly when we have the syntactic prior as opposed to the empirical prior (from 38.39 ± 11.92% to 54.86 ± 6.75% for 5% program supervision). In terms of program prediction accuracy, we also observe marginal improvements in the cases where we have 5% and 10% supervision respectively, from 56.23 ± 2.81% to 65.45 ± 11.88%. When more supervision is available, regularizing with respect to the broader, syntactic prior hurts performance marginally, which makes sense, as one can just treat program supervision as a supervised learning problem in this setting.

Predicate Factoring. We choose to parameterize a `find[green]` operation with its own individual parameters $\theta_{\texttt{find[green]}}$ instead of having a parameterization for $\theta_{\texttt{find}}$ with `green` supplied as an argument to the neural module network. In doing so we followed prior work (Johnson et al., 2017). However, we also note that other work (Hu et al., 2017) has explored the second parameterization. Conceptually, choice of either is orthogonal to the benefits from our probabilistic formulation. However, for completeness we implemented argument based modules on SHAPES – getting similar accuracy as atomic modules (for both Prob-NMN and NMN). For example, at 15% supervision, Prob-NMN gets to a joint training VQA accuracy on (SHAPES-Val) of 97.53 (± 1.23) [Prob-NMN] *vs.* 80.78 (± 11.75) for NMN. Thus, this design choice does not seem to have an impact on the relative performance of Prob-NMN and the NMN baseline.

Effect of the Program Prior $\mathbf{p(z)}$. Next, we explore the impact of regularizing the program posterior to be close to the prior $p(\mathbf{z})$, for different choices of the prior. Firstly, we disable the KL-divergence term by setting $\beta = 0$ in Equation (6), recovering a deterministic version of our model, that still learns to reconstruct the question $\mathbf{x}$ given a sampled program $\mathbf{z}$. Compared to our full model at 10% supervision, the performance on question coding drops to 23.14 ± 6.1% program prediction accuracy (from 60.18 ± 9.56%). Interestingly, the $\beta = 0$ model has a better performance on question reconstruction, which improves from 83.60 ± 5.57 % to 94.3 ± 1.85%. This seems to indicate that, without the KL term, the model focuses solely on reconstruction and fails to learn compositionality in the latent $\mathbf{z}$ space, such that supervised grounding are poorly propagated to unsupervised questions as evidenced by the drop in program prediction accuracy. Thus, the probabilistic model helps better achieve our high-level goal of data-efficient legibility in the reasoning process. In terms of the VQA accuracy on validation (at the end of joint training) we see a drop in performance from 96.86 ± 2.48 to 74.82 ± 17.88.

Effect of Optimizing the True ELBO, $\beta = 1$. Next, we empirically validate the intuition that for the semi-supervised setting, approximate inference $q_\phi(\mathbf{z}|\mathbf{x})$ learned using Equation (1) for sequence models often leads to solutions that do not make meaningful use of the latent variable. For example, we find that at $\beta = 1$ (and 10% supervision), training the true evidence lower bound on $\mathcal{U}_{qc}$ deteriorates the question reconstruction accuracy from 83.60 % to 8.74% causing a large distortion in the original question, as predicted by the results from Alemi et al. (2018). Overall, in this setting the final accuracy on VQA at the end of joint training is 74.45 ± 15.58, a drop of $>$ 20 % accuracy.

6. Coherence and Sensitivity Sampling

Our goal is to sample from the posterior $p(\mathbf{z}|a, \mathbf{i})$, which we accomplish by sampling from the unnormalized joint distribution $p(\mathbf{z}|a, \mathbf{i}) \propto p(\mathbf{z})p(a|\mathbf{z}, \mathbf{i})$. To answer a query conditioned on an answer $\hat{a}$, one can simply filter out samples which have $a = \hat{a}$, and produce the sample programs as the program, and decode the program $\mathbf{x}^i \sim p(\mathbf{x}|\mathbf{z}^i)$ to produce the corresponding question. Note that it is also possible to fit a (more scalable) variational approximation $q_\sigma(\mathbf{z}|a, \mathbf{i})$ having trained the generative model retrospectively, following results from Vedantam et al. (2018) However, for the current purposes we found the above sampling procedure to be sufficient.

7. Implementation Details on CLEVR

CLEVR dataset is an order of magnitude larger than SHAPES, with larger vocabulary, longer sequences and images with higher visual complexity. We take this scale-up into account, and make suitable architectural modifications

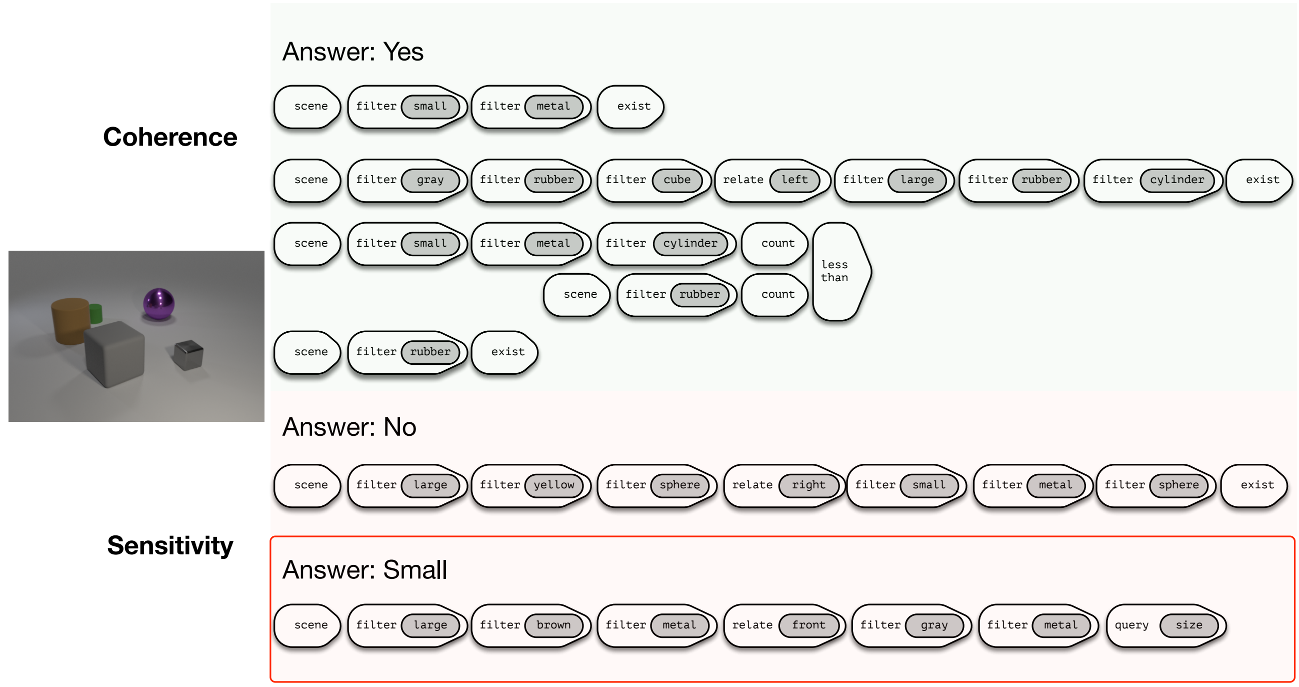

Figure 1. **Coherence and Sensitivity**: Image and a correponding answer are specified, and the model is asked to produce programs it believes should lead to the particular answer for the given image. One can notice that the generated programs are consistent with each other, and evaluate to the specified answer. Results are shown for a Prob-NMN model trained with 0.143% supervision on CLEVR.

in comparison to SHAPES.

Sequence to Sequence Models: The encoders of our sequence-to-sequence models (program generator and question reconstructor) are based on LSTM cells with a hidden state of 256 units, a depth of two layers, and accept input word embeddings of 256 dimensions for both question and program vocabulary. We follow the same architecture for our program prior sequence model. Our decoders are similar to encoders, only with a modification of having single layer depth. When sampling from the model, we sample till the maximum sequence length in observed training data – this results in program sequences of maximum length 28, and question sequences of maximum length 44.

Image CNN: We resize the CLEVR images to `224x224` dimensions and pass them through an ImageNet pre-trained ResNet-101, and obtain features from its ‘third stage’, which results in features of dimensions `1024x14x14`. These features are used as input to the Neural Module Network.

Training Details In order to get stable training with sequence-to-sequence models on CLEVR, we found it useful to average the log-probabilities across multiple timesteps. Without this, in presence of a large variance in sentence lengths, we find that the model focuses on generating the longer sequences, which is usually hard without first understanding the structure in shorter sequences. Averaging (across) the T dimension helps with this. This is also common practice when working with sequence models and has been done in numerous prior works (Vinyals et al., 2015; Johnson et al., 2017; Hu et al., 2017; Lu et al., 2017) *etc.*. With higher linguistic variation in CLEVR questions, learning question reconstruction is difficult at the start of training when the latent program space isn’t learned properly. To make learning easier, we increase the influence of teaching examples by also scaling question reconstruction loss with α in Equation (3). We use the ADAM optimizer with a learning rate of `1e-3` for Question Coding, `1e-4` for Module Training and `1e-5` for Joint Training. We do not introduce any weight decay. We use a batch size of 256 across all stages, and a moving average baseline for REINFORCE with decay factor of 0.99, similar to SHAPES. We set α as 100, β as 0.1 and γ as 10. We validate every 500 iterations for question coding and joint training, and every 2000 iterations for module training. We drop the learning rate to half when our primary metric (either program prediction accuracy or VQA accuracy, in respective stages) does not improve within next 3 validations. We use the code and module design choices of the state of the art TbD nets (Mascharka et al., 2018) to implement the $p_{\theta_{\mathbf{z}}}(a|\mathbf{i})$ mapping in our model.

8. Results on CLEVR-Humans dataset

Johnson et al. (2017) also collect a dataset of questions asked by humans on the images in the CLEVR dataset, which is called CLEVR-humans. CLEVR-humans is out of distribution data for both NMN and Prob-NMN. We train the models on the regular CLEVR dataset (with 0.143 % question-program supervision) and evaluate the checkpoints on the CLEVR-human validation set which contains 7202 questions. We find that the gains on the regular CLEVR setup for Prob-NMN also translate to the CLEVR-humans setting, with Prob-NMN getting to an accuracy of 50.81 ± 1.93 *vs.* 45.45 % for NMN. Both approaches perform better than chance accuracy of 15.66% computed by picking the most frequent answer from the training set. We compute $p(a|\mathbf{x}, \mathbf{i}) \approx \frac{1}{N} \sum_n p(a|\mathbf{z}^n, \mathbf{i})$, where $\mathbf{z}^n \sim q(\mathbf{z}|\mathbf{x})$ for this analysis, drawing $n = 20$ samples. Next we were interested in understanding if modeling the uncertainty via. Prob-NMN helped more with out of distribution questions (such as CLEVR-humans) – this phenomena is also called aleatoric uncertaintly. For this, we varied the number of samples we drew, sweeping from $n = 1$ to $n = 20$. For a single sample, we found that the performance drop for both NMN as well as Prob-NMN was somewhat similar, *i.e.* the NMN performance dropped to 44.98 $\pm$ 2.67 while Prob-NMN dropped to 50.26 $\pm$ 2.01. In general, we do not see a greater benefit of handling aleatoric uncertaintly via. averaging for Prob-NMN over NMN when tested in the out-of-domain CLEVR-humans dataset. Our hypothesis for this is that amortized inference is not handling aleatoric uncertainty as well as we desire in our application.

9. Coherence and Sensitivity results on CLEVR

We repeat the same procedure used for the SHAPES dataset on the CLEVR dataset to analyze the coherence and sensitivity in the reasoning performed by the Prob-NMN model. Given an image and a candidate answer, say "yes" as input, we find that multiple sample programs drawn from the model are coherent with respect to each other. In Figure 1 we show a sorted list of such programs ranked by log-probability of programs, $\mathbf{z}$. Next, when we switch the answer to "no", we find that the program changes in a sensible manner, yeilding something that should correctly evaluate to the specified answer. However, in general we find that the results are not as interpretable for non "yes"/"no" answers. For example, when conditioning on "small", we find that the model makes a mistake of asking for the object in front of "large brown metal" object in stead of "large brown" object. This might be because the proposed model does not take into account question premise (i.e. does not have an arrow from $\mathbf{i}$ to $\mathbf{z}$) meaning that a question about a large brown metal object would never be asked about this image, since it is not present in it.

References

Alemi, A. A., Poole, B., Fischer, I., Dillon, J. V., Saurous, R. A., and Murphy, K. Fixing a broken ELBO. In *ICML*, 2018.

Hu, R., Andreas, J., Rohrbach, M., Darrell, T., and Saenko, K. Learning to reason: End-to-End module networks for visual question answering. In *CVPR*, 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 *Intl. Conf. on Computer Vision*, 2017.

Lu, J., Xiong, C., Parikh, D., and Socher, R. Knowing when to look: Adaptive attention via a visual sentinel for image captioning. In *CVPR*, 2017.

Mascharka, D., Tran, P., Soklaski, R., and Majumdar, A. Transparency by design: Closing the gap between performance and interpretability in visual reasoning. In *CVPR*, 2018.

Vedantam, R., Fischer, I., Huang, J., and Murphy, K. Generative models of visually grounded imagination. In *ICLR*, 2018.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. Show and tell: A neural image caption generator. In *CVPR*, 2015.