Modeling question asking using neural program generation

Ziyun Wang and **Brenden M. Lake**
New York University
Email: ziyunw@nyu.edu, brenden@nyu.edu

Abstract

People ask questions that are far richer, more informative, and more creative than current AI systems. We propose a neuro-symbolic framework for modeling human question asking, which represents questions as formal programs and generates programs with an encoder-decoder based deep neural network. From extensive experiments using an information-search game, we show that our method can ask optimal questions in synthetic settings, and predict which questions humans are likely to ask in unconstrained settings. We also propose a novel grammar-based question generation framework trained with reinforcement learning, which is able to generate creative questions without supervised human data.

1 Introduction

People can ask rich, creative questions to learn efficiently about their environment. Question asking is central to human learning yet it is a tremendous challenge for computational models. There is always an infinite set of possible questions that one can ask, leading to challenges both in representing the space of questions and in searching for the right question to ask.

Machine learning has been used to address aspects of this challenge. Traditional methods have used heuristic rules designed by humans [9, 3], which are usually restricted to a specific domain. Recently, neural network approaches have also been proposed, including retrieval methods which select the best question from past experience [16] and encoder-decoder frameworks which map visual or linguistic inputs to questions [22, 16, 31, 28]. While effective in some settings, these approaches do not consider settings where the questions are asked about partially unobservable states. Furthermore, these methods are heavily data-driven, limiting the diversity of generated questions and requiring large training sets for different goals and contexts. There is still a large gap between how people and machines ask questions.

Recent work has aimed to narrow this gap by taking inspiration from cognitive science. For instance, Lee et al. [12] incorporates aspects of "theory of mind" [17] in question asking by simulating potential answers to the questions. [18] propose a similar method which estimates value of answer information with a Generative Adversarial Network (GAN) [6]. But these approaches relies on imperfect agents for natural language understanding which may lead to error propagation. Related to our approach, Rothe et al. [20] proposed a question-asking framework by modeling questions as symbolic programs, but their algorithm relies on hand-designed program features and requires expensive calculations to ask questions.

We use "neural program generation" to bridge symbolic program generation and deep neural networks, bringing together some of the best qualities of both approaches. Symbolic programs provide a compositional "language of thought" [5] for creatively synthesizing which questions to ask, allowing the model to construct new ideas based on familiar building blocks. Compared to natural language, programs are precise in their semantics, have clearer internal structure, and require a much smaller vocabulary, making them an attractive representation for question answering systems as well [10, 29, 14]. However, there has been much less work using program synthesis for question asking, which requires searching through infinitely many questions (where many questions may be informative) rather than producing a single

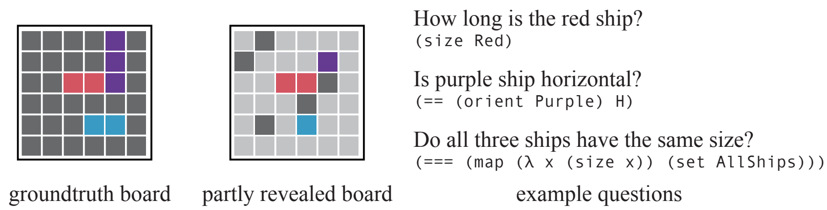


Figure 1: The Battleship task. Blue, red, and purple tiles are ships, dark gray tiles are water, and light gray tiles are hidden. The agent can see a partly revealed board, and should ask a question to seek information about the hidden board. Example questions and translated programs are shown on the right. We recommend viewing the figures in color.

correct answer to a question. Deep neural networks allow for rapid question-synthesis using encoder-decoder modeling, eliminating the need for the expensive symbolic search and feature evaluations in Rothe et al. [20]. Together, the questions can be synthesized quickly and evaluated formally for quality (e.g. the expected information gain), which as we show can be used to train question asking systems using reinforcement learning.

In this paper, we develop a neural program generation model for asking questions in an information-search game, which is similar to "Battleship" and has been studied in previous work [8, 20, 21]. Our model uses a convolutional encoder to represent the game state, and a Transformer decoder [26] for generating questions in a domain specific language (DSL). Importantly, we show that the model can be trained from a small number of human demonstrations of good questions, after pre-training on a large set of automatically generated questions. Our model can also be trained without such demonstrations using reinforcement learning, while still expressing important characteristics of human behavior. We evaluate the model on several aspects of human question asking, including reasoning about optimal questions in synthetic scenarios, density estimation based on free-form question asking, and creative generation of genuinely new questions.

To summarize, our paper makes three main contributions: 1) We propose a neural network for modeling human question-asking behavior, 2) We propose a novel reinforcement learning framework for generating creative human-like questions by exploiting the power of programs, and 3) We evaluate different properties of our methods extensively through three different experiments.

2 Related work

Question generation has attracted attention from the machine learning community. Early research mostly explored rule-based methods which strongly depend on human-designed rules [9, 3]. Recent methods for question generation adopt deep neural networks, especially using the encoder-decoder framework, and can generate questions without hand-crafted rules. These methods are mostly data-driven, which use pattern recognition to map inputs to questions. Researchers have worked on generating questions from different types of inputs such as knowledge base facts [22], pictures [16], and text for reading comprehension [31, 28]. However aspects of human question-asking remain beyond reach, including the goal-directed and flexible qualities that people demonstrate when asking new questions. This issue is partly addressed by some recent papers which draw inspiration from cognitive science. Research from Rothe et al. [20] and Lee et al. [12] generate questions by sampling from a candidate set based on goal-oriented metrics. This paper introduces an approach to question generation that does not require a candidate question set and expensive feature computations at inference time. Moreover, our approach can learn to ask good questions without human examples through reinforcement learning.

Our work also builds on neural network approaches to program synthesis, which have been applied to many different domains [4, 24, 25]. Those approaches often draw inspiration from computer architecture, using neural networks to simulate stacks, memory, and controllers in differentiable form [19, 7]. Other models incorporate Deep Reinforcement Learning (DRL) to optimize the generated programs in a goal oriented environment, such as generating SQL queries which can correctly perform a specific database processing task [32], translating

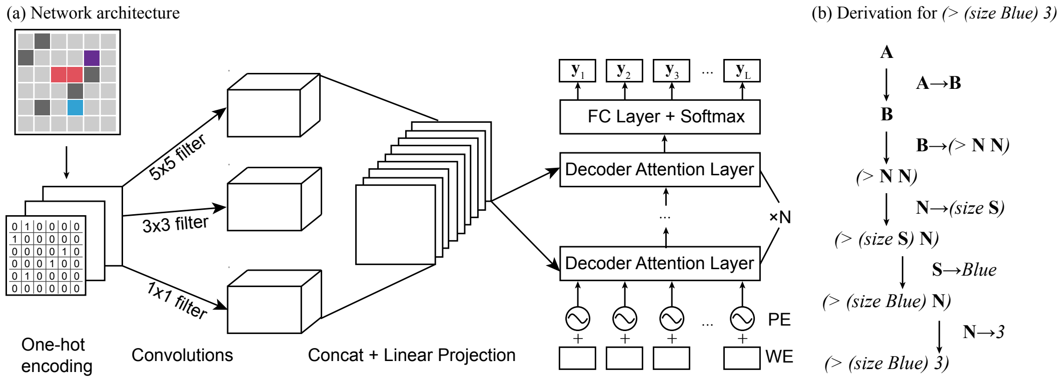


Figure 2: Neural program generation. Figure (a) shows the network architecture. The board is represented as a grid of one-shot vectors and is embedded with a convolutional neural network. The board embedding and a sequence of symbols are inputted to a Transformer decoder [26] to generate output vectors (details in section 4). PE means positional embeddings, and WE means word embeddings. (b) shows the derivation steps for program "`(> (size Blue) 3)`" using a context-free grammar. Non-terminals are shown as bold-faced, and terminals are shown in italic. The production rules used are shown next to each arrow.

strings in Microsoft Excel sheets [4], understanding and constructing 3D scenes [13] and objects [25]. Recent work has also proposed ways to incorporate explicit grammar information into the program synthesis process. Yin and Neubig [30] design a special module to capture the grammar information as a prior, which can be used during generation. Some recent papers [2, 23] encode grammar with neural networks and use DRL to explicitly encourage the generation of semantically correct programs. Our work differs from these in two aspects. First, our goal is to generate informative human-like questions in the new domain instead of simply correct programs. Second, we more deeply integrate grammar information in our framework, which directly generates programs based on the grammar.

3 Battleship Task

In this paper, we work with a task used in previous work for studying human information search [8] as well as question asking [21]. The task is based on an information search game called "Battleship", in which a player aims to resolve the hidden layout of the game board based on the revealed information (Figure 1). There are three ships with different colors (blue, red, and purple) placed on a game board consisting of 6×6 grid of tiles. Each ship can be either horizontal or vertical, and takes 2, 3 or 4 tiles long. All tiles are initially turned over (light grey in Figure 1), and the player can flip one tile at a time to reveal an underlying color (either a ship color, or dark grey for water). The goal of the player is to determine the configuration of the ships (positions, sizes, orientations) in the least number of flips.

In the modified version of this task studied in previous work [20, 21], the player is presented with a partly revealed game board, and is required to ask a natural language question to gain information about the underlying configuration. As shown in Figure 1, the player can only see the partly revealed board, and might ask questions such as "How long is the red ship?" In this paper, we present this task to our computational models, requesting they generate questions about the game board.

The dataset from Rothe et al. [20] consists of a small set of human questions along with their semantic parses, which represent the questions as LISP-like programs that can be executed on the game states to get answers to the questions. The DSL consists of primitives (like numbers, ship colors, etc.) and functions (like arithmetic operators, comparison operators, and other functions related to the game board) from which questions can be composed. Here, we aim to synthesize questions in this "language of thought" of semantic forms, since it captures key notions of compositionality and computability. Figure 1 shows some examples of produced programs. The full DSL grammar can be found in the Appendix.

4 Neural program generation framework

This section introduces our approach to program generation with neural networks and Deep Reinforcement Learning (DRL). We will first introduce the neural architecture, and then explain how it works with different training and inference strategies.

4.1 Network Architecture

The neural network we use is shown in Figure 2(a). It includes a Convolutional Neural Network (CNN) for encoding the input board, and a Transformer [26] decoder for estimating the symbol distribution or selecting actions in different settings.

Encoder. The game board $x \in \{0,1\}^{6\times 6\times 5}$ is a 6x6 grid with five channels, one for each tile color, with the color encoded as a one-hot vector in each grid location. A simple CNN with one layer of filters is used to encode the board. Intuitively, many questions are related to specific positions, thus the position information should be recoverable from the encoding. On the other hand, some features of the board are translation-invariant, such as whether a ship is blocked by another ship. In order to capture the position-sensitive information as well as the translation-invariant patterns, three convolution operations with different filter sizes (1×1, 3×3, and 5×5) are performed in parallel on the same input. The inputs are padded accordingly to make sure the feature maps have the same width and height. Then three feature maps are concatenated together along the dimension of output channels, and passed through a linear projection.

Formally, the outputs of the convolutions $\mathbf{c}$ can be obtained by

$$\mathbf{c} = \mathrm{ReLU}([\mathrm{Conv}_1(x); \mathrm{Conv}_3(x); \mathrm{Conv}_5(x)]) \tag{1}$$

where Conv_k denotes a convolution operation on a $k\times k$ filter, $\mathrm{ReLU}(\cdot)$ means applying a ReLU activation, and $[A;B]$ means the concatenation of matrices A and B. Then $\mathbf{c} \in \mathbb{R}^{6\times 6\times 3C_{out}}$ is projected to the encoder output $\mathbf{e} \in \mathbb{R}^{6\times 6\times M}$ by matrix $W_o^e \in \mathbb{R}^{3C_{out},M}$, where C_{out} is the number of out channels of each convolution, and M is the length of encoded vectors.

Decoder. We use the decoder from the Transformer model [26]. With an input sequence of length L, the decoder computes the hidden states through several stacked Decoder Attention Layers. Each layer is composed by three sub-layers, a self-attention module, an attention over the encoded board, and a fully connected feed-forward network. Residual connections are employed around each sub-layer, followed by a layer normalization [1]. After N layers of attention modules, a final output layer transforms the hidden states to the output vectors $\mathbf{y}_i \in \mathbb{R}^{N_o}$ at every position i from 1 to L, where N_o is the output size. As shown later, the outputs can be interpreted differently in different settings.

Given the output from encoder $\mathbf{e}$, and the hidden representation $\mathbf{h}^{n-1}$ from Decoder Attention Layer $n-1$, each layer computes the hidden representation as

$$\begin{aligned}
\mathbf{g}^n &= \mathrm{LN}(\text{Self-ATT}(\mathbf{h}^{n-1}) + \mathbf{h}^{n-1}) \\
\mathbf{v}^n &= \mathrm{LN}(\mathrm{ATT}(\mathbf{g}^n, \mathbf{e}) + \mathbf{g}^n) \\
\mathbf{h}^n &= \mathrm{LN}(\mathrm{FC}(\mathbf{v}^n) + \mathbf{v}^n)
\end{aligned} \tag{2}$$

where $\mathrm{LN}(\cdot)$ means layer normalization [1], $\mathrm{FC}(\cdot)$ is a fully connected layer, $\mathrm{ATT}(\cdot)$ and $\text{Self-ATT}(\cdot)$ are multi-head attention mechanisms, which computes the attention over the output of encoder $\mathbf{e}$, and the attention over the input $\mathbf{h}^{n-1}$ itself, respectively. They are defined as follows

$$\begin{aligned}
\mathrm{ATT}(\mathbf{g}^n, \mathbf{e}) &= \text{Multi-ATT}(\mathbf{g}^n, \mathbf{e}, \mathbf{e}) \\
\text{Self-ATT}(\mathbf{h}^{n-1}) &= \text{Multi-ATT}(\mathbf{h}^{n-1}, \mathbf{h}^{n-1}, \mathbf{h}^{n-1})
\end{aligned} \tag{3}$$

$\text{Multi-ATT}(\cdot)$ is the multi-head attention mechanism described in the paper by Vaswani et al. [26], which is a concatenation of multiple standard attention mechanisms with inputs projected using different matrices. A multi-head attention with n heads is defined as

$$\begin{aligned}
\text{Multi-ATT}(Q,K,V) = W^o[&\mathrm{Attention}(W_1^Q Q, W_1^K K, W_1^V V); \cdots ; \\
&\mathrm{Attention}(W_n^Q Q, W_n^K K, W_n^V V)]
\end{aligned} \tag{4}$$

where

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{5}$$

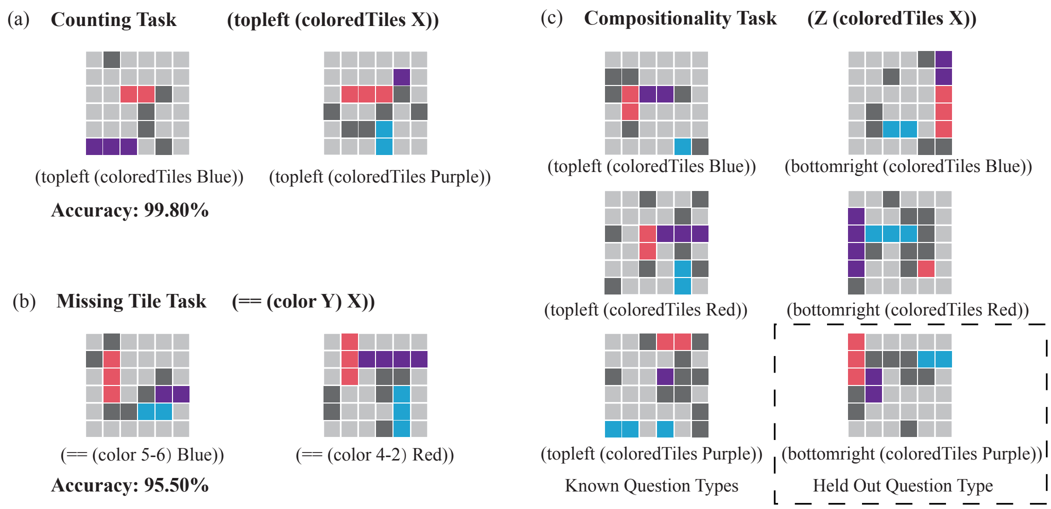


Figure 3: Design of the tasks in experiment 1. The goal of task (a) is to find the color which has the least number of visible tiles; the goal of task (b) to find the location and color of the missing tile; (c) is the compositionality task with 5 questions as known question types, and another one (in dotted box) as held out question type. The format of generated question is shown alongside the title of each task, where X, Y and Z are variables. The accuracy of supervised model for task (a) and (b) are given below each task.

is the scaled dot-product attention operation. Q, K, V are a set of vectors called queries, keys, and values, respectively, and d_k is the dimension of queries and keys.

After N layers, we apply a linear projection and a softmax activation to $\mathbf{h}^N$ to get the output vectors $\mathbf{y}_1, \ldots, \mathbf{y}_L$.

4.2 Training

Our model is compatible with both supervised and reinforcement learning.

4.2.1 Supervised training

In the supervised setting, the goal is to model the distribution of questions present in the training set. Each output $\mathbf{y}_i \in \mathbb{R}^{N_o}$ is a symbol at position i in the program, where N_o is the number of different symbols in the grammar. The model is trained with symbol-level cross entropy loss, and can be used to calculate the log-likelihood of a given sequence, or to generate a question symbol-by-symbol from left to right. Generation works as follows. Suppose at step t, a sequence of length t along with the encoded board is presented to the decoder. The model predicts the vector $\mathbf{y}_t$ which represents the probability of each symbol to be chosen as next. Then we sample a symbol at location $t+1$ and execute the decoder again with the new sequence, until an `<eos>` symbol is generated or a maximum length is reached.

4.2.2 Grammar-enhanced RL

We propose a novel grammar-enhanced RL training procedure for the framework. Figure 2(b) illustrates the process of generating a program from the context-free grammar specified in the DSL we use for this task. Beginning from the start symbol "`A`", at each step a production rule is chosen and applied to one of the non-terminals in the current string. The choice of rule is modeled as a Markov Decision Process, and we solve it with DRL. Each state is a partially derived string passed to the decoder, and we use the first output $\mathbf{y}_1 \in \mathbb{R}^{N_o}$ to represent the probability of selecting each production rule from all possible N_o rules. After the rule is applied, the new string is passed back into the decoder, repeating until only terminals are contained in the sequence. We adopt the leftmost derivation here to avoid the ambiguity of parsing order, so at each step the left-most non-terminal will be replaced.

# of training examples	0	10	50	100	200	400	800
Acc. (%) on held out question type	0.0	2.0	39.0	69.5	81.0	92.0	96.0
Acc. (%) on known question types	96.6	97.3	97.1	96.0	96.3	97.8	96.1
Acc. (%) classify on held out question type	33.0	37.0	49.0	75.5	88.0	94.0	99.5

Table 1: Accuracy on the compositionality task using different numbers of training examples from the held out question type.

5 Experiments

5.1 Reasoning in synthetic settings

In the first experiment, we designed three tasks to evaluate whether the model can learn simple compositional rules and reasoning strategies. These tasks include counting the number of visible ship tiles, locating a missing ship tile, and generalizing both strategies to unseen scenario types. Figure 3 illustrates the three tasks we designed in this experiment by providing examples of each task. The three tasks are defined as follows.

- **Counting task.** Models must select the ship color with the least number of visible tiles on the board. Each board has a unique answer, and models respond by generating a program `(topleft (coloredTiles X))` where `X` is a ship color. 4000 examples are used for training, and another 1000 examples are used for testing.

- **Missing tile task.** Models must select the ship that is missing a tile and identify which tile is missing. All ships are completely revealed except one, which is missing exactly one tile. Models respond by generating `(== (color Y) X)` where `X` is a color and `Y` is a location on the board. The number of training and test examples are the same as the counting task.

- **Compositionality task.** Models must combine both of the above strategies to find the missing tile of the ship with the least visible tiles. Outputs are produced as `(Z (coloredTiles X))` where `X` is a color and `Z` is either `topleft` or `bottomright`. Each board has a unique answer.

  This task further evaluates compositionality by withholding question types from training. With three values for `X` and two for `Z`, there are six possible question types and one is picked as the "held out" type. The other five "known" question types have 800 training examples. For the held out question type, the number of training examples is varied from 0 to 800, to test how much data is needed for generalization. Another 200 new boards of each question type is used for evaluation. More information about the model hyperparameters and training procedures are provided in Appendix A.1.

5.1.1 Results and discussion

We train our model in a fully supervised fashion. The training details of all the experiments are provided in the Appendix, including hyper-parameter settings, computing devices, etc. Accuracy for the counting and missing tile tasks is shown in Figure 3. The full neural program generation model shows strong reasoning abilities, usually asking optimal questions in both the counting and missing tile tasks. We also perform an ablation analysis of the encoder filters of the model, and provide the results in the Appendix.

The results for the compositionality task are summarized in Table 1. When no training data regarding the held out question type is provided, the model cannot generalize to situations systematically different from training data, exactly as pointed out in previous work on the compositional skills of encoder-decoder models [11]. However, when the number of additional training data increases, the model quickly incorporates the new question type while maintaining high accuracy on the familiar question tasks. To further examine the model's ability on the compositionality task, we evaluate another version of the model which replaces the decoder with two linear transformations to directly predict the location and the ship. The results are shown on the last row of Table 1. This model has 33.0% transfer accuracy on compositional scenarios never seen during training. This suggests that the model has the potential to generalize to unseen scenarios if the task can be decomposed to subtasks and combined together.

Model	LL_all	LL_highEnt	LL_lowEnt
Rothe et al. (2017)	1400.06	-	-
full model	**-150.38±0.51**	**-150.92±1.51**	**-156.38±1.92**
-pretrain	-242.53±2.32	-260.84±4.99	-249.98±3.87
-finetune	-415.32±0.95	-443.03±1.33	-409.01±1.21
-encoder	-153.50±0.41	-149.69±0.54	-163.13±0.82

Table 2: Predicting which questions people ask. LL_all, LL_highEnt, and LL_lowEnt shows mean log-likelihood across held out boards. High/low refers to a board's entropy.

5.2 Estimating the distribution of human questions

In this experiment, we examine if neural program generation can capture the distribution of questions that humans ask, using a conditioned language model. We seek to avoid training on a large corpus of human-generated questions; the dataset only offers about 600 human questions [20], and importantly people can ask intelligent questions in a novel domain with little direct training experience. We design a two-step training process; first we pre-train the model on automatically generated questions using an informativeness measure, second we fine-tune the model using only small set of real human questions.

To create pre-training data, we generate a large number of game boards and sample K questions for each board. To generate boards, we uniformly sample the configuration of three ships and cover arbitrary number of tiles, with the restriction that at least one ship tile is observed. With a tractable game model and domain-specific language of questions, we can sample K programs for each board based on the expected information gain (EIG) metric, which quantifies the expected information received by asking the question. EIG is formally defined as the expected reduction in entropy, averaged over possible answers to a question x,

$$\mathrm{EIG}(x) = \mathbb{E}_{d \in A_x}[I(p(h)) - I(p(h|d;x))] \tag{6}$$

where $I(\cdot)$ is the Shannon entropy. The terms $p(h)$ and $p(h|d;x)$ are the prior and posterior distribution of a possible ship configuration h given question x and answer $d \in A_x$. We generate 2,000 boards, and sample 10 questions for each board, with the log-probability of a question determined by its EIG, up to a normalizing constant and with a maximum length of 80 tokens per program. After pre-training, the model is fine-tuned on the corpus of human questions.

5.2.1 Results and discussion

To evaluate the model, we follow the same procedure as Rothe et al. [20], where they run leave-one-out cross-validation on 16 different boards and calculate the sum of log-likelihood of all questions for a board and average across different boards. We evaluate the log-likelihood of reference questions on our full model as well as lesioned variants, including a model without pre-training, a model without fine-tuning, and a model with only a decoder (unconditional language model). A summary of the results is shown in Table 2. We run each experiment 10 times and report the mean values and standard errors. The log-likelihood reported by Rothe et al. [20] is also included for reference [1].

The full model performs the best, out-performing models without pre-training or fine-tuning by a large margin. This demonstrates that (1) the automated pre-training procedure is effective in conveying the task and DSL to the model; (2) the low log-likelihood for the "no finetune" model shows that the distribution of the constructed questions and real human questions are very different, which is consistent with prior work on the informativeness of human questions [21].

The model without an encoder performs surprisingly well, but there are also reasons to think that context is not the most important factor. Some stereotyped patterns of question asking are effective across a wide range of scenarios, especially when little is known about the board (e.g., all games have the same optimal first question when no information is revealed yet). To further examine the role of context, we calculated the entropy of the hypothesis

[1] Rothe et al. [20] mention that the log-likelihood values of their model are approximate and rely on an estimated partition function, which could contribute to the substantial difference compared to our model.

Model	avg. EIG	EIG>0.95	EIG>0	#unique	#unique novel	avg. length
text-based	0.928	62.80%	76.95%	-	-	8.21
supervised	0.972	45.70%	81.80%	183	103	10.98
grammar RL (noSP)	1.760	88.30%	92.40%	224	213	9.97
grammar RL (SP0.02)	**1.766**	**94.90%**	**96.80%**	111	96	6.03
grammar RL (SP0.05)	1.559	91.90%	95.20%	57	47	5.79

Table 3: Analysis of question generation. Our grammar enhanced model is compared with a supervised trained baseline from Experiment 2 and a text-based model. SP means step penalty. The models are compared in terms of average EIG value, the ratio of EIG value greater than 0.95 or 0, number of unique and novel questions generated (by "novel" we mean questions not present in the dataset of human-asked questions). The EIG for the text-based model is calculated based on the program form of the generated questions. The average length is the number of words for text-based model, and number of tokens for others.

space of possible ship locations for each board, and group the top 5 and bottom 5 boards into high and low entropy groups. Then we calculated the average log-likelihood on different entropy groups and list the results in Table 2. When the game entropy is high, questions like "how long is the red ship" are good for almost any board, so the importance of the encoder is reduced. When the game entropy is low, the models with access to the board has substantially higher log-likelihood than the model without the encoder. Also, note that the first experiment would be impossible to perform well without an encoder. Together, this shows the importance of modeling the context-sensitive characteristics of how people ask questions.

5.3 Question generation

In this experiment, we evaluate our reinforcement learning framework on its ability to generate novel questions from scratch, without training on human-generated questions. As described in the framework section, the neural network selects a sequence of grammar-based actions to generate a question, and the model is optimized with REINFORCE [27].

To accomplish this, we use a reward function for training the RL agent that is based on EIG, since it is a good indicator of question informativeness and easy to compute. We give a reward of 1 if the generated question has EIG > 0.95, a reward of 0 for questions with EIG ≤ 0.95, and a reward of -1 for invalid questions (e.g. longer than 80 tokens). We do not directly use the EIG value as our reward because preliminary experiments show that causes the model to choose from a very small set of high EIG questions. Instead, we wish to study how a model, like people, can generate a diverse set of good questions in simple goal-directed tasks. To further encourage more human-like behavior, the reward function includes a step penalty for each action.

5.3.1 Results and discussion

We compare our program-based framework with a simple text-based model, which has the same architecture but is trained with supervision on the text-form questions [20]. We also compare our model with the supervised model from the last experiment. Finally, we compare the RL model with different step penalties. The models are evaluated on 1000 random boards, and generate one question for each board. The results are shown in Table 3.

First, when the text-based model is evaluated on new contexts, 96.3% of the questions it generates were included in the training data. We calculated the EIG of the program form of the text questions, and find that the average EIG and the ratio of EIG>0 is worse than the supervised model trained on programs. Some of these deficiencies are due to the very limited text-based training data, but using programs instead can help overcome these limitations. With the program-based framework, we can sample new boards and questions to create a much larger dataset with executable program representations. This self-supervised training helps to boost performance.

From Table 3, the grammar-enhanced RL model is able to generate more informative and creative questions compared to the alternatives. It can be trained from scratch without examples of human questions, and produces many high EIG questions that are genuinely novel (not present in the human corpus). In contrast, the supervised model can also generate novel questions, although many have limited utility (only 45.70% questions have EIG>0.95). The step penalty helps the RL model to take fewer actions and to generate shorter questions.

human	(size Red)	12.1%
	(size Purple)	9.6%
	(size Blue)	9.1%
	(== (orient Blue) H)	7.0%
	(== (orient Red) H)	4.9%
	(== (orient Purple) H)	4.2%
	(topleft (coloredTiles Red))	3.2%
	(== (size Purple) 4)	1.9%
	(== (size Blue) 3)	1.9%
	(== (size Blue) 4)	1.7%
RL	(size Red)	19.0%
	(size Blue)	18.1%
	(size Purple)	12.3%
	(orient Red)	8.9%
	(orient Blue)	6.9%
	(orient Purple)	4.2%
	(bottomright (coloredTiles Red))	3.5%
	(bottomright (coloredTiles Blue))	3.3%
	(bottomright (coloredTiles Purple))	3.1%
	(setSize (coloredTiles Red))	1.9%

Table 4: Most frequent questions asked by humans and the grammar RL model (SP0.02).

As shown in the table, model with step penalty 0.05 generates questions with on average 5.79 tokens, but with limited diversity. On the other hand, the RL model without a step penalty has high average EIG and the most diverse set of questions; however, it also generates many meaningless questions (only 92.40% have EIG>0).

Another interesting finding is that questions generated by our RL agent are surprisingly consistent with human questions, even though the RL agent is not trained on any human examples. Table 4 lists the top 10 frequent questions asked by humans and our RL model with step penalty 0.2. Both humans and our RL model most frequently ask questions about the size of the ships, followed by questions about ship orientation. A notable difference is that people often ask true/false questions, such as "Is this size of the Blue ship 3?", while the RL model does not since these questions have lower EIG.

We also provide examples in Figure 4 to show the diversity of questions generated by our RL model. Figure 4(a) shows novel questions produced by the model, including clever and human-like questions such as "What is the size of the blue ship plus the purple ship?" or "Where is the bottom right tile of all the blue and purple tiles?" Sometimes it also generates complex-looking questions that can actually be expressed with simpler forms, especially when using map and lambda operators. For instance, the third example in Figure 4(a) is essentially equivalent to "What is the size of the purple ship?" Many additional examples of generated questions are provided in the Appendix.

With the grammar-enhanced framework, we can also guide the model to ask different types of questions, consistent with the goal-directed nature and flexibility of human question asking. The model can be queried for certain types of questions by providing different start conditions to the model. Instead of starting the derivation from the start symbol "`A`", we can start the derivation from an intermediate state such as "`B`" if the model is asked for a true/false question, or a more complicated "`(and B B)`" if the model is asked for a true/false question that uses "and". In Figure 4(b), we show examples where the model is asked to generate four specific types of questions: true/false questions, number questions, location-related questions, and compositional true/false questions. We see that the model can flexibly adapt to new constraints and generate meaningful questions that follow these constraints.

In Figure 4c, we compare arbitrary samples from the model and people in the question-asking task. These examples, along with others in the appendix, again suggest that our model is able to generate clever and human-like questions. There are also meaningful differences; people often ask true/false questions as mentioned before, and people sometimes ask questions with quantifiers such as "any" and "all", which are operationalized in program

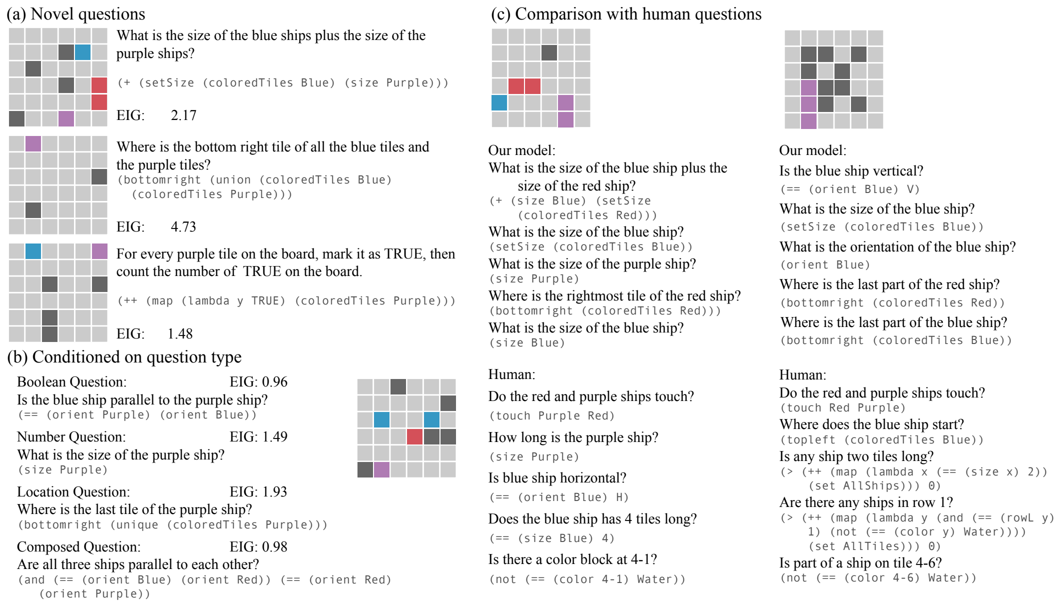


Figure 4: Examples of model-generated questions. The natural language translations of the question programs are provided for interpretation. (a) shows three novel questions generated by the grammar enhanced model, (b) shows an example of how the model generates different type of questions by conditioning the input to the decoder, (c) shows questions generated by our model as well as human question askers.

form with lambda functions. These questions are complicated in representation and not favored by our model.

6 Conclusion

We introduce a neural program generation framework for question asking in partially observable settings, which can generate creative human-like questions based on human demonstrations through supervised learning or without demonstrations through reinforcement learning. Programs provide models with a "machine language of thought" for compositional synthesis, and neural networks provide an efficient means of question generation. We demonstrate the effectiveness of our method through extensive experiments covering a range of human question asking abilities.

The current model is limited in several important ways. It cannot generalize to systematically different scenarios than it was trained on, and it sometimes generates meaningless questions. We plan to further explore the model's compositional abilities in future work. Another promising direction is to train models jointly to both ask and answer questions, which could install a richer sense of the question semantics. In addition, allowing the agent to iteratively ask questions and play the game to completion is another interesting future direction, which could in theory replace the EIG-based reward function with success in the game. We would also like to use our framework in dialog systems and open-ended question asking scenarios, allowing such systems to synthesize more informative and creative questions than current application-driven systems allow.

References

[1] Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[2] Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. In *International Conference on Learning Representations*, 2018.

[3] Yllias Chali and Sadid A Hasan. Towards topic-to-question generation. *Computational Linguistics*, 41(1):1–20, 2015.

[4] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 990–998. JMLR. org, 2017.

[5] Jerry A. Fodor. *The Language of Thought*. Harvard University Press, 1975.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[8] Todd Gureckis and Doug Markant. Active learning strategies in a spatial concept learning game. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 31, 2009.

[9] Michael Heilman and Noah A Smith. Good question! statistical ranking for question generation. In *Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 609–617. Association for Computational Linguistics, 2010.

[10] 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 *Proceedings of the IEEE International Conference on Computer Vision*, pages 2989–2998, 2017.

[11] Brenden M Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, 2018.

[12] Sang-Woo Lee, Youngjoo Heo, and Byoung-Tak Zhang. Answerer in questioner's mind: Information theoretic approach to goal-oriented visual dialog. In *Advances in neural information processing systems*, 2018.

[13] Yunchao Liu, Zheng Wu, Daniel Ritchie, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to describe scenes with programs. In *International Conference on Learning Representations*, 2019.

[14] 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 *International Conference on Learning Representations*, 2019.

[15] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[16] Nasrin Mostafazadeh, Ishan Misra, Jacob Devlin, C. Lawrence Zitnick, Margaret Mitchell, Xiaodong He, and Lucy Vanderwende. Generating natural questions about an image. In *Annual Meeting of the Association for Computational Linguistics*, pages 1802–1813, 2016.

[17] David Premack and Guy Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1(4):515–526, 1978.

[18] Sudha Rao and Hal Daumé III. Answer-based adversarial training for generating clarification questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 143–155, 2019.

[19] Scott Reed and Nando De Freitas. Neural programmer-interpreters. In *International Conference on Learning Representation*, 2016.

[20] Anselm Rothe, Brenden M Lake, and Todd Gureckis. Question asking as program generation. In *Advances in Neural Information Processing Systems*, pages 1046–1055, 2017.

[21] Anselm Rothe, Brenden M Lake, and Todd M Gureckis. Do people ask good questions? *Computational Brain & Behavior*, 1(1):69–89, 2018.

[22] Iulian Vlad Serban, Alberto García-Durán, Caglar Gulcehre, Sungjin Ahn, Sarath Chandar, Aaron Courville, and Yoshua Bengio. Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus. In *Annual Meeting of the Association for Computational Linguistics*, pages 588–598, 2016.

[23] Xujie Si, Yuan Yang, Hanjun Dai, Mayur Naik, and Le Song. Learning a meta-solver for syntax-guided program synthesis. In *International Conference on Learning Representations*, 2019.

[24] Shao-Hua Sun, Hyeonwoo Noh, Sriram Somasundaram, and Joseph Lim. Neural program synthesis from diverse demonstration videos. In *International Conference on Machine Learning*, pages 4797–4806, 2018.

[25] Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. In *International Conference on Learning Representations*, 2019.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[27] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[28] Kaichun Yao, Libo Zhang, Tiejian Luo, Lili Tao, and Yanjun Wu. Teaching machines to ask questions. In *International Joint Conferences on Artificial Intelligence*, pages 4546–4552, 2018.

[29] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *Advances in Neural Information Processing Systems*, pages 1031–1042, 2018.

[30] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In *Annual Meeting of the Association for Computational Linguistics*, pages 440–450, 2017.

[31] Xingdi Yuan, Tong Wang, Caglar Gulcehre, Alessandro Sordoni, Philip Bachman, Sandeep Subramanian, Saizheng Zhang, and Adam Trischler. Machine comprehension by text-to-text neural question generation. In *Workshop on Representation Learning for NLP*, 2017.

[32] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2018.

A Experimental Settings

A.1 Reasoning in Synthetic Settings

In this experiment, we use $C_{out} = 10, L = 50$ for the model encoder. Each word is embedded with 50 dimension vectors in the decoder. The decoder has 2 layers, each multi-head attention module has 4 heads, and $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{50\times 8}, W^O \in \mathbb{R}^{8\times 50}$. The model is trained for 500 epochs using Adam optimizer with initial learning rate at 0.001 and a batch size is set as 32. All the experiments are carried out on a single GTX 1080 Ti Graphics Card, and implemented using PyTorch [2].

A.2 Estimating the Distribution of Human Questions

In this experiment, the model encoder has the same hyper-parameters as in the first experiment. We increase the size of the decoder by setting number of layers to 3, number of heads to 3, and set $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{50\times 16}, W^O \in \mathbb{R}^{16\times 50}$. The model is pretrained for 1000 epochs using Adam optimizer with initial learning rate at 0.001, and is fine-tuned on the human dataset for 15 epochs with the same optimizer and learning rate.

A.3 Question Generation

In this experiment, we set the number of decoder attention heads to 8, set $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{50\times 32}, W^O \in \mathbb{R}^{32\times 50}$, and keep other hyper-parameters the same as the last experiment. The model is optimized by REINFORCE [27] algorithm with initial learning rate 0.0001 and batch size 16. To encourage the exploration of the model, we also apply an ϵ-greedy strategy with ϵ set to 0.25 at the beginning, and gradually decreased to 0.01 as training continues. We also employ an entropy term that encourages the model to explore the action space [15] and ask diverse questions. The model is trained for 500 epochs, within each epoch the model passes $10,000$ different boards.

B Full grammar for Battleship

The grammar is provided in table 5 and 6.

[2]`https://pytorch.org`

Table 5: Part 1 of the grammatical rules. Rules marked with [b] have a reference to the Battleship game board (e.g., during evaluation the function *orient* looks up the orientation of a ship on the game board) while all other rules are domain-general (i.e., can be evaluated without access to a game board).

Answer types	
A → B	*Boolean*
A → N	*Number*
A → C	*Color*
A → O	*Orientation*
A → L	*Location*
Boolean	
B → True	
B → False	
B → (not B)	
B → (and B B)	
B → (or B B)	
B → (== B B)	
B → (== N N)	
B → (== O O)	
B → (== C C)	
B → (=== setN)	*True if all elements in set of numbers are equal*
B → (any setB)	*True if any element in set of booleans is True*
B → (all setB)	*True if all elements in set of booleans are True*
B → (> N N)	
B → (< N N)	
B → (touch S S) [b]	*True if the two ships are touching (diagonal does not count)*
B → (isSubset setL setL)	*True if the first set of locations is subset of the second set of locations*
Numbers	
N → 0	
...	
N → 10	
N → (+ N N)	
N → (+ B B)	
N → (++ setN)	
N → (++ setB)	*Number of True elements in set of booleans*
N → (− N N)	
N → (size S) [b]	*Size of the ship*
N → (row L)	*Row number of location L*
N → (col L)	*Column number of location L*
N → (setSize setL)	*Number of elements in set of locations*
Colors	
C → S	*Ship color*
C → Water	
C → (color L) [b]	*Color at location L*
S → Blue	
S → Red	
S → Purple	
S → x	*Lambda variable for ships*
Orientation	
O → H	*Horizontal*
O → V	*Vertical*
O → (orient S) [b]	*Orientation of the ship S*
Locations	
L → 1-1	*Row 1, column 1*
...	
L → 6-6	
L → y	*Lambda variable for locations*
L → (topleft setL)	*The most left of the most top location in the set of locations*
L → (bottomright setL)	*The most right of the most bottom location in the set of locations*

Table 6: Part 2 of the grammatical rules. See text for details.

Mapping	
setB → (map fyB setL)	*Map a boolean expression onto location set*
setB → (map fxB setS)	*Map a boolean expression onto ship set*
setN → (map fxN setS)	*Map a numerical expression onto ship set*
setL → (map fxL setS)	*Map a location expression onto ship set*
Lambda expressions	
fyB → (λ y B)	*Boolean expression with location variable*
fxB → (λ x B)	*Boolean expression with ship variable*
fxN → (λ x N)	*Numeric expression with ship variable*
fxL → (λ x L)	*Location expression with ship variable*
Sets	
setS → (set AllShips)	*All ships*
setL → (set AllTiles)	*All locations*
setL → (coloredTiles C) [b]	*All locations with color C*
setL → (setDifference setL setL)	*Remove second set from first set*
setL → (union setL setL)	*Combine both sets*
setL → (intersection setL setL)	*Elements that exist in both sets*
setL → (unique setL)	*Unique elements in set*

Table 7: Results of the synthetic reasoning tasks.

(a) Accuracy of different models on the counting and missing tile tasks

Model	Counting	Missing tile
Full model	99.80%	95.50%
3x3 conv only	99.30%	51.50%
1x1 conv only	98.60%	1.90%

(b) Accuracy for selecting the right tile location and color for the missing tile

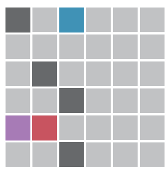
Model	Location acc.	Color acc.
Full model	97.80%	97.60%
3x3 conv only	53.00%	96.20%
1x1 conv only	3.90%	45.50%

C Additional results

We also perform an ablation test on the neural network in the first experiment. Accuracy for the counting and missing tile tasks is summarized in Table 7a for the full model and lesioned variants. The full neural program generation model shows strong reasoning abilities, achieving an accuracy of 99.80% and 95.50% for the counting and missing tile tasks, respectively. The full model is compared to weakened variants with only one filter size in the encoder, either "3x3" and "1x1 conv only," and the performance of the weakened models drop dramatically on the missing tile task.

To better understand the role of different filter sizes, Table 7b breaks down the errors in the missing tile task on whether the question can pick the right ship (color acc.) and whether it can select the right location (location acc.). The 3×3 convolution filters can accurately select the correct color, but often fail to choose the right tile. The model with 1×1 convolution filters has poor performance for both color and location. In the current architecture, predicting the correct location requires precise information that seems to be lost without filters of different sizes.

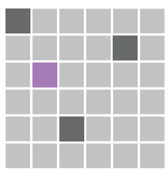
Here we provide more examples of questions generated by our models in the generation experiment (Experiment 3). Figure 5, 6 and 7 provides more examples for the same settings as shown in figure 4 in the main text. Figure 8 shows generated examples of the text-based model.



What is the size of the blue ships plus the red ships?

```
(+ (setSize (coloredTiles Blue)
   (setSize (coloredTiles Red)))
```

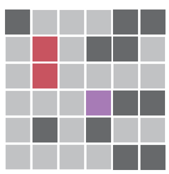
EIG: 2.10



What is the size of the red ship multiplies the number
of ship colors?

```
(++ (map (lambda x (size Red)) (set AllColors)))
```

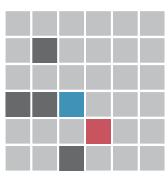
EIG: 1.47



Which row contains the first tile of the blue ship?

```
(rowL (topleft (coloredTiles Blue)))
```

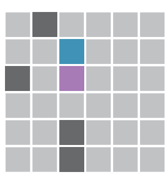
EIG: 2.50



What is the size of blue plus four?

```
(+ (size Blue) 4)
```

EIG: 1.58

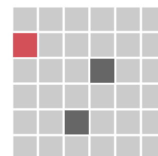


What is the size of the blue ship plus the number of the row
which contains the beginning of the red ship?

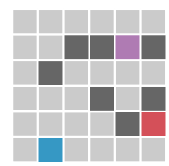
```
(+ (size Red) (colL (topleft (coloredTiles Red))))
```

EIG: 2.74

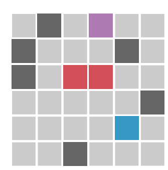
Figure 5: Novel questions generated by the grammar enhanced model.



Boolean Question:
Is the blue ship horizontal? EIG: 1.00
```
(== V (orient Blue))
```
Number Question:
What is the size of the purple ship? EIG: 1.48
```
(size Purple)
```
Location Question: EIG: 1.73
Where is the first tile of all the purple tiles and red tiles?
```
(topleft (union (coloredTiles Purple) (coloredTiles Red)))
```
Composed Question: EIG: 1.00
Is the blue ship horizontal?
```
(and (== H H) (== H (orient Blue)))
```

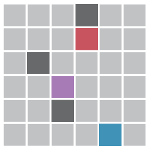


Boolean Question: EIG: 0.95
Is the blue ship longer than the red ship?
```
(> (size Blue) (setSize (coloredTiles Red)))
```
Number Question: EIG: 1.56
What is the size of the blue ship?
```
(size Blue)
```
Location Question: EIG: 1.91
Where is the last tile of the blue ship?
```
(bottomright (coloredTiles Blue))
```
Composed Question: EIG: 0.81
Is the blue ship parallel to the purple ship?
```
(and (== (orient Blue) (orient Purple))
     (== (orient Blue) (orient Blue)))
```



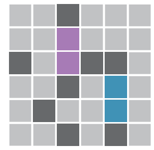
Boolean Question: EIG: 0.00
Is the blue ship parallel to itself?
```
(== (orient Blue) (orient Blue))
```
Number Question: EIG: 1.55
What is the size of the blue ship?
```
(size Blue)
```
Location Question: EIG: 1.50
Where is the last tile of red ship?
```
(bottomright (coloredTiles Red))
```
Composed Question: EIG: 1.00
Are all ships parallel to each other?
```
(and (== (orient Blue) (orient Purple))
     (== (orient Blue) (orient Red)))
```

Figure 6: Generated questions of different types by controlling the start condition.



Our model:
Where is the bottom right of the blue ship?
`(bottomright (coloredTiles Blue))`
What is the size of the blue ship?
`(setSize (coloredTiles Blue))`
What is the orientation of the purple ship?
`(orient Purple)`
Where is the bottom right of the purple ship?
`(bottomright (coloredTiles Purple))`
What is the size of the blue ship?
`(size Blue)`
What is the size of the red ship?
`(setSize (coloredTiles Red))`

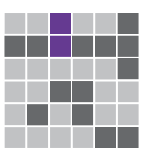
Human:
Are the majority of the ships horizontal or vertical?
```
(> (++ (map (lambda x (== (orient x) H))
   (set AllColors))) 1)
```
Which direction is purple?
`(== (orient Purple) H)`
Is the red ship placed vertically?
`(== (orient Red) H)`
3 tiles is blue ship?
`(== (size Blue) 3)`
Does tile B6 hold a part of the blue ship?
`(== (color 6-2) Blue)`
Where is the end of the blue ship
`(bottomright (coloredTiles Blue))`



Our model:
What is the size of the red ship?
`(size Red)`
Is the red ship in parallel with the blue ship?
`(== (orient Red) (orient Blue))`
What is the size of the red ship plus 0?
`( + 0 ( setSize ( coloredTiles Red) ) )`
What is the size of the red ship?
`( setSize ( coloredTiles Red ) )`
What is the orientation of the red ship?
`( orient Red )`
Where is the bottom right of the red ship?
`( bottomright ( coloredTiles Red ) )`

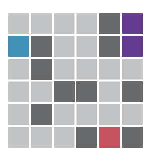
Human:
Is the red ship touching another ship?
`(or (touch Red Blue) (touch Red Purple))`
Is there an item at 6-1 tile?
`(not (== (color 6-1) Water))`
How many tiles is the red ship?
`(size Red)`
Is the red ship touching another ship?
`(or (touch Red Blue) (touch Red Purple))`
Is orange ship 2 tiles long?
`(== (size Red) 2)`
Is any part of the red ship in left half of the grid?
```
(> (++ (map (lambda y (and (< (colL y) 4)
   (== (color y) Red))) (set AllTiles))) 0)
```

Figure 7: Comparisons of questions generated by our model with human questions.



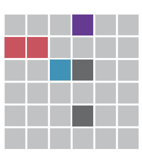
Does the 6th row contain any pieces of the red ship?

EIG: 0.91



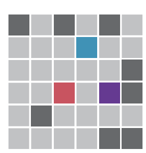
Is the blue ship horizontal or vertical?

EIG: 0.00



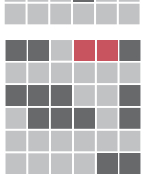
How many blocks is the red ship?

EIG: 1.52



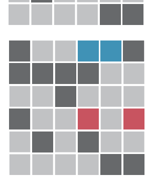
Is the square at b2

Ungrammatical.



Is each ship the same size?

EIG: 0.75



Is any part of the blue ship in the same direction?

Ungrammatical.

Figure 8: Example questions generated by the text-based model.