ℓGym: Natural Language Visual Reasoning with Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

We present ℓGym, a new benchmark for language-conditioned reinforcement learning in visual environments. ℓGym is based on 2,661 human-written natural language statements grounded in an interactive visual environment, and emphasizing compositionality and semantic diversity. We annotate all statements with Python programs representing their meaning. The programs are executable in an interactive visual environment to enable exact reward computation in every possible world state. Each statement is paired with multiple start states and reward functions to form thousands of distinct Markov Decision Processes of varying difficulty. We experiment with ℓGym with different models and learning regimes. Our results and analysis show that while existing methods are able to achieve non-trivial performance, ℓGym forms a challenging open problem.

1 Introduction

Reinforcement learning (RL) with natural language context poses important opportunities and challenges. Language provides an expressive and accessible conduit for task specification, so that RL agents can address a broad set of tasks, rather than learn a single behavior. For natural language processing (NLP), RL is a promising avenue for language use and acquisition through world interaction. However, language is a challenging medium for learning to reason. The RL agent must reason about high-level language concepts, low-level actions, and the relations between them, and it must learn to do so efficiently because language data is inherently limited due to requiring human interaction.

Despite significant interest and promising approaches, it has been challenging to create expressive RL benchmarks with natural language. A core challenge is accurately computing a reward that is dependent on natural language semantics. Existing approaches adopt different strategies to address this issue, such as using synthetic language (Côté et al., 2018; Co-Reyes et al., 2019) or heuristic approximation, for example using demonstration data (Misra et al., 2017). While these approaches open new avenues for research, they either do not explore the full complexity of human language or introduce unexpected artifacts into learning through meaning approximations.

We present ℓGym,[1] a reinforcement learning benchmark for natural language visual reasoning that addresses the above issues. ℓGym implements the standard OpenAI Gym API (Brockman et al., 2016), and aims to bring natural language into the suite of commonly used RL benchmarks.

ℓGym is focused on spatial reasoning environments, where an agent manipulates a visual environment by adding and removing objects conditioned on a natural language statement. The agent's goal is to modify the environment so that a given statement will have a pre-specified truth-value with regard to the environment (i.e., the constraints specified in the language are either satisfied or violated). ℓGym includes 2,661 highly-compositional and semantically-diverse natural language statements from the NLVR corpus (Suhr et al., 2017), that combine with a configurable environment backbone to create thousands of Markov Decision Processes (MDP) of varying complexity, for example with different sizes of the state and action spaces.

A key challenge in constructing ℓGym is accurate reward computation. Because of the flexibility of the environments and language, there are many possible equally correct termination states for each

[1] ℓGym is a temporary placeholder name for submission anonymity.

MDP. Correctly evaluating reward at every possible state requires taking into account the meaning nuances of the highly-compositional language. We address this problem by annotating all statements with executable Python programs representing their meaning, effectively creating a supervised semantic parsing dataset (Zelle & Mooney, 1996; Zettlemoyer & Collins, 2005; Suhr et al., 2018). The programs are executable in a structured representation of the environment, and allow for exact and reliable reward computation at every possible state.

Our experiments with ℓGym show that existing models can provide non-trivial performance given sufficient training time, with multi-modal pre-training providing further benefit. However, our results show that there remains significant room for improvement. For example, on the simplest configuration, our agents can solve 76.54% of the test environments, but performance drops significantly to only 6.09% on the most complex configuration. This is partially due to the challenging exploration problem of the most complex configuration, where performance climbs from 6.09% to 29.95% when we provide additional guidance on when tasks are completed during learning. The ℓGym framework and trained models will be released upon publication.

2 Related Work

There is significant and increasing interest in RL conditioned on natural language. Various strategies are deployed to resolve language semantics for reward computation, mostly by strict control of the language or through approximations.

Potentially the most common approach is to control the language by using synthetic language backed by a formal representation (Narasimhan et al., 2015; Johnson et al., 2017a;b; Côté et al., 2018; Chevalier-Boisvert et al., 2019; Co-Reyes et al., 2019; Jiang et al., 2020). Although synthetic language allows studying the problem of learning high-level concepts, many of the complexities of natural language are stripped away, and such approaches run the risk of reducing the language learning challenge to reverse engineering the hand-crafted generation process.

An alternative that allows for natural language that attempts to retain the control of its semantics is to generate the target sequence of decisions (i.e., task demonstration), and solicit post-hoc language instructions (Shridhar et al., 2020; 2021; Hanjie et al., 2021). While this process uses human-written language, it potentially implicitly retains the regularities of the demonstration generation procedure.

Others have carefully designed the underlying environment to simplify termination state evaluation given demonstrations, for example, with a sparse graph-based structure (Anderson et al., 2018; Chen et al., 2019; Ku et al., 2020). However, recent work shows the potential for evaluation fidelity issues even in these settings (Jain et al., 2019).

We emphasize human-written natural language, and opt to not use underlying hand-crafted procedures as stimuli for the writing. ℓGym prioritizes exact reward computation rather than automated approximations to allow for relatively clean benchmarking of learning methods.

Our annotation of natural language statements with programs is inspired by the annotation of data for supervised learning of semantic parsers (Zelle & Mooney, 1996; Zettlemoyer & Collins, 2005; Suhr et al., 2018). The ontology that we use to define the Python API of our environments is based on the semantic parsing work of Goldman et al. (2018). Robust semantic parsers can assist in automating our annotation process.

3 The ℓGym Benchmark

ℓGym consists of a collection of environments that share a common backbone. The backbone is a 2D plane that is manipulated by placing and removing objects of different types. Each environment instance is a Markov Decision Process (MDP) created by pairing a natural language statement and a target boolean value with a configuration of the shared backbone. The goal of the agent in each environment is to manipulate it by adding and removing objects so that the truth-value of the statement with regard to the environment is the target boolean.

The learning problem ℓGym presents is to induce a policy that generalizes across MDPs. We split the MDPs to training, development, and held-out testing sets. The training environments are to be

Table 1: Data statistics per CMDP configuration and data split. The number of MDPs corresponds to the number of contexts under each CMDP. For `FLIPIT`, "Init." corresponds to the total number of initial states across all MDPs for this CMDP.[3]

	TOWER-SCRATCH	**TOWER-FLIPIT**		**SCATTER-SCRATCH**	**SCATTER-FLIPIT**	
	MDPs	MDPs	Init.	MDPs	MDPs	Init.
Train	989	1,910	5,704	1,241	2,340	6,696
Dev	163	317	676	87	164	313
Test	324	619	1,383	155	285	591
Total	1,476	2,846	7,763	1,483	2,789	7,600

used for parameter estimation, while the two other sets are for testing during development and for final held-out testing to report approach performance.[2]

There are two dimensions of configuration: appearance and starting condition. The appearance determines the state space, transition function, and action space. The starting condition determines the agent's goal. The appearance of the environment can be (a) `TOWER`: the objects include squares only, and they can be stacked into towers only; or `SCATTER`: objects of different types can be freely distributed. `TOWER` gives a more constrained problem with much smaller state and action spaces compared to `SCATTER`.

The starting condition and agent's objective can be: (a) `SCRATCH`: the environment starts without any objects and the goal is to modify it so that the statement's truth-value is `True`; or (b) `FLIPIT`: the environment starts with a set of objects and the agent's goal is to flip the truth-value of the statement. `SCRATCH` generally only requires adding objects, except in cases of correcting for agent's errors, while `FLIPIT` requires both adding and removing, because there are already objects present. The four configurations are `TOWER-SCRATCH`, `TOWER-FLIPIT`, `SCATTER-SCRATCH`, and `SCATTER-FLIPIT`. In our experiments (Section 5), we observe the different configurations provide different levels of difficulty. For example, `SCATTER` configurations are generally much harder than `TOWER`, as expected with the much larger state and action spaces.

Each configuration forms a Contextual Markov Decision Process (CMDP; Hallak et al., 2015). CMDP is an abstraction over a set of Markov Decision Processes (MDPs) to account for a context that remains constant throughout the interaction with an MDP. We set the context to include the statement and the target boolean the interaction is conditioned on. A CMDP is a tuple $(\mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{M}(c))$, where $\mathcal{C}$ is the context space, $\mathcal{S}$ the state space, $\mathcal{A}$ the action space, and $\mathcal{M}$ a function mapping a context $c \in \mathcal{C}$ to an MDP $\mathcal{M}(c) = (\mathcal{S}, \mathcal{A}, T, R^c, \beta^c)$. Here, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a transition function, $R^c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ a reward function, and β^c an initial state distribution. This means a CMDP is a collection of MDPs that share the same state and action spaces. The learning problem is to estimate parameters θ for a policy $\pi_\theta : \mathcal{S} \times \mathcal{C} \to \mathcal{A}$, which takes as input both the current state and the context underlying the MDP.

Table 1 shows the number of MDPs under each configuration. Figure 1 shows example action trajectories in MDPs for each of the four CMDPs. Depending on the context, different types of reasoning are required from the agent. For example, in the second column of Figure 1, the statement *there is no black block as the top of a tower with at most three blocks* requires the agent to reason about negation, soft cardinality, color, and position, while the statement in the third column *there is a box with 2 triangles of same color nearly touching each other* requires a comparison and to reason about several object attributes (shape, color, position). Both require the agent to perform high-level relational reasoning about single objects or sets.

Contexts A context $c \in \mathcal{C}$ is a pair $c = (\bar{x}, b)$, where $\bar{x}$ is a natural language statement and $b \in \{\texttt{True}, \texttt{False}\}$ is a target boolean value for the statement $\bar{x}$ with respect to the state s. The set

[2]We recommend reporting both development and held-out test results in future work for easy comparison.

[3]NLVR includes a total of 18,322 images. This allows expanding the number of initial states to 92,179 initial states through box element permutations. We do not manipulate this property in this work, but future work could take advantage of it.

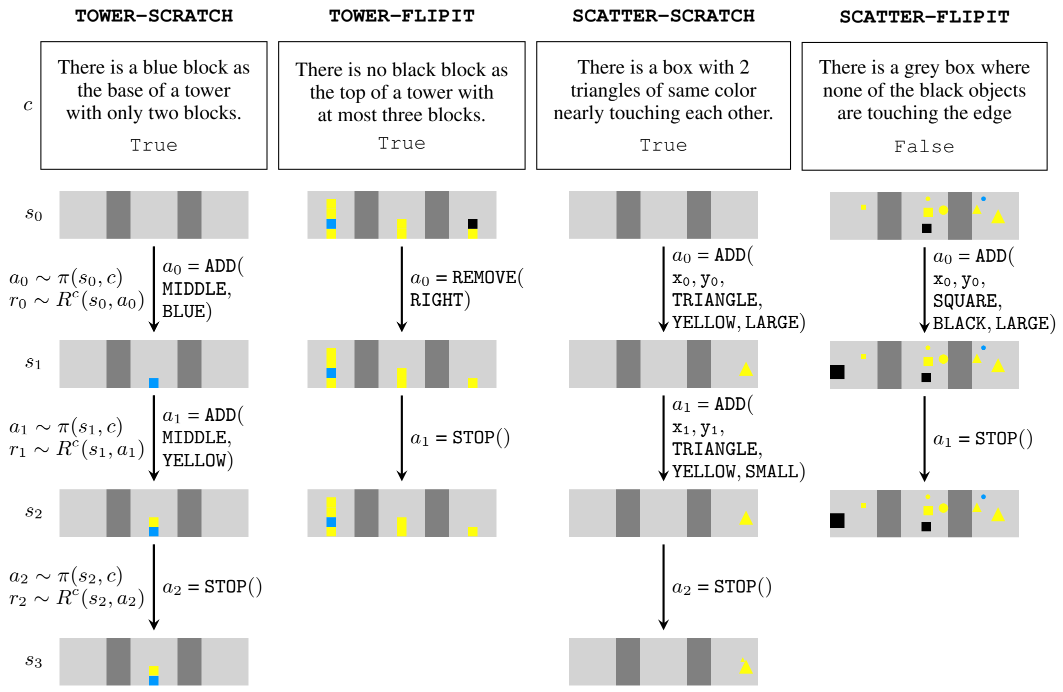

TOWER-SCRATCH
TOWER-FLIPIT
SCATTER-SCRATCH
SCATTER-FLIPIT
c
There is a blue block as the base of a tower with only two blocks.
True
There is no black block as the top of a tower with at most three blocks.
True
There is a box with 2 triangles of same color nearly touching each other.
True
There is a grey box where none of the black objects are touching the edge
False
s_0
s_1
s_2
s_3
$a_0 \sim \pi(s_0, c)$
$r_0 \sim R^c(s_0, a_0)$
a_0 = ADD(MIDDLE, BLUE)
a_0 = REMOVE(RIGHT)
a_0 = ADD(x₀, y₀, TRIANGLE, YELLOW, LARGE)
a_0 = ADD(x₀, y₀, SQUARE, BLACK, LARGE)
$a_1 \sim \pi(s_1, c)$
$r_1 \sim R^c(s_1, a_1)$
a_1 = ADD(MIDDLE, YELLOW)
a_1 = STOP()
a_1 = ADD(x₁, y₁, TRIANGLE, YELLOW, SMALL)
a_1 = STOP()
$a_2 \sim \pi(s_2, c)$
$r_2 \sim R^c(s_2, a_2)$
a_2 = STOP()
a_2 = STOP()

Figure 1: Overview of the four CMDP configurations, with an example for each. Every example is conditioned on a context $c = (\bar{x}, b)$, and starts with a state s_0, sampled from the initial state distribution β^c. For example, for TOWER-SCRATCH (left column), the top image depicts the initial state s_0 with the context c. The context c includes the statement *there is a blue block as the base of a tower with only two blocks* and the target boolean True. The initial state s_0 is an image with three empty light grey box regions separated by dark grey separators. The agent π is presented with (s_0, c), and samples an action $a_0 \sim \pi(s_0, c)$. The environment transitions to the next state s_1, while the context remains the same. This process continues until the termination condition is filled.

of statements is predefined for TOWER and SCATTER based on the NLVR data, but identical across the choice of SCRATCH and FLIPIT. The target boolean value in SCRATCH is always True. In FLIPIT, the target boolean value can be either True or False.

States A state $s \in \mathcal{S}$ is an RGB image. Images in ℓGym are divided into three box regions of identical dimensions by two dark gray separators (Figure 1). The objects in ℓGym have three properties, each can take multiple values: shape (CIRCLE, SQUARE or TRIANGLE), color (BLACK, BLUE, or YELLOW), and size (SMALL, MEDIUM or LARGE). In TOWER, states are constrained to have stacks of up to four SQUAREs of MEDIUM size and any color at the center of each box. SCATTER states support all object shapes, sizes, and colors, and they may be positioned freely. In both conditions, objects cannot cross image boundaries or into the separators. The choice between SCRATCH or FLIPIT does not influence the state space.

Actions and Transitions There are three action types STOP, ADD, and REMOVE. STOP terminates the episode and does not require any parameters. The truth-value of the statement is only evaluated and compared to the target boolean after the STOP action is taken. ADD adds objects to the environment, and REMOVE removes objects.

They take arguments that differ between TOWER and SCATTER:

TOWER: Similar to the state space of TOWER, the actions are also constrained. Both ADD and REMOVE take a position argument, which has three possible values corresponding to the three box regions. Objects are always added or removed at the top of the stack. Adding an object on top of a stack of four objects or removing an object from an empty box are both invalid actions. ADD also takes a color argument. For example, the first action on the left trajectory in Figure 1 is adding a blue square in an empty box. Including `STOP`, there are $1 + (3 + 1) \times 3 = 13$ actions.

SCATTER: Unlike `TOWER`, objects of any type can be placed freely in the box regions. Both `ADD` and `REMOVE` take 2D coordinates that specify the pixel location. Adding an object places it so that its top-left coordinates are the given coordinates. Removing an object will remove the object at the given coordinates. Adding also requires specifying the shape, color, and size. The action is invalid if adding results in objects' overlap or boundary crossing with the separators or image boundaries. Removing from a position that does not include an object is also an invalid action. The native resolution of images in ℓGym is 380×100 pixels. Including `STOP`, there are $1 + (380 \times 100) \times ((3 \times 3 \times 3) + 1) = 1{,}064{,}001$ actions. Because of the extremely large action space, ℓGym also allows acting with a coarser grid system for `SCATTER` that is automatically mapped to the original resolution (Appendix A.1). In our experiments (Section 5), we use a grid of 19×5, giving a total of 2,661 actions. In general, the coarser the grid, the more MDPs are not solvable.

The transition function $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ depends on the choice between `TOWER` and `SCATTER` configurations, because this choice determines the action space. Similar to the action spaces, the transitions in `TOWER` are more constrained compared to `SCATTER`. The transition function does not modify the context, which is fixed for a given MDP.

Reward Function The reward function R^c is computed with respect to the context pair $c = (\bar{x}, b)$, where $\bar{x}$ is a natural language statement and b is the target boolean value. The reward is based on evaluating the truth-value of the natural language statement $\bar{x}$ with respect to a state s, and comparing it to the target boolean b. ℓGym includes an executable evaluation function $\mathcal{E}^{\bar{x}} : \mathcal{S} \times \mathcal{A} \to \{\texttt{True}, \texttt{False}\}$ for every statement $\bar{x}$. Section 4 describes how we create the evaluation functions.

The agent receives a positive reward for terminating the episode using the `STOP` action with the statement evaluation $\mathcal{E}^{\bar{x}}(s)$ equal to the target boolean value b. If the statement boolean value $\mathcal{E}^{\bar{x}}(s)$ does not equal the target boolean b value when taking the `STOP` action, the agent receives a negative reward. If the episode terminated because the current time step t reached the action horizon H or because of an invalid action, the agent also receives a negative reward. Action validity depends on the current state s and on the configuration, because `TOWER` and `SCATTER` have different action spaces. For example, in `TOWER`, adding an object to a box (e.g., `ADD(MIDDLE, BLUE)`) is only valid if the box has less than four objects, because towers have a maximum height of four. There is also a verbosity penalty of δ for every other action. Formally, the reward is:

$$R^c(s,a) = \begin{cases} 1.0 & a = \texttt{STOP} \wedge \mathcal{E}^{\bar{x}}(s) = b \\ -1.0 & a = \texttt{STOP} \wedge \mathcal{E}^{\bar{x}}(s) \neq b \\ -1.0 & (a \text{ is invalid in } s) \vee (t = H) \\ -\delta & \text{otherwise} \end{cases} .$$

Initial State Distribution The initial state distribution β^c is parameterized by the context $c \in \mathcal{C}$, which is different between `SCRATCH` and `FLIPIT`. In `SCRATCH`, the agent modifies an empty environment to satisfy the truth-condition of the statement $\bar{x}$ in the context c, so the initial state s_0 is always an empty image. The set of initial states β^c for every context $c \in \mathcal{C}$ is the set of images associated with the statement $\bar{x}$ in the NLVR data. In practice, for `FLIPIT`, this set includes between 1 to 43 images. Table 1 shows the total number of initial states in each configuration.

4 The ℓGym Data

The data used for ℓGym is based on the NLVR corpus (Suhr et al., 2017). The NLVR data was initially collected as a supervised learning benchmark. We formalize an interactive task on top of the NLVR data and collect additional annotations for reward computation.

4.1 Background: the NLVR Corpus

NLVR includes human-written natural language statements paired with synthetic images. Each pair is annotated with the boolean truth-value of the statement with regard to the image (i.e., `True` if the statement is true with regard to the image, or `False` otherwise). The images are designed to support complex reasoning, including about spatial and set relations. The original learning task posed by NLVR is to classify statement-image pairs as `True` to indicate the statement is true with regard to the image, or `False` otherwise. Various approaches were developed to address the NLVR challenge (Suhr et al., 2017; Tan & Bansal, 2018; Goldman et al., 2018; Pavez et al., 2018; Yao et al., 2018; Hudson & Manning, 2018; Perez et al., 2018; Dasigi et al.; Zheng et al., 2020; Gupta et al., 2021), and a separate version using photos was also released (Suhr et al., 2019).[4]

Qualitative analysis of the data (Table 2 in Suhr et al. (2017)) shows a more diverse representation of semantic and compositional phenomena compared to related corpora (Antol et al., 2015), including requiring joint visual-linguistic reasoning about spatial relations, quantities, and sets of objects. NLVR also provides an underlying structured representation for every image, which supports easy manipulation of images. The combination of simple interface for image manipulation with complex reasoning via natural language makes NLVR ideal to support an interactive benchmark environment.

The original dataset has four splits for training, development, public testing, and hidden testing. We follow the original splits for the training and development sets. Following the rest public release of the hidden testing set, we merge the public and hidden testing sets into a single public test split.

4.2 Annotations for Reward Computation

The NLVR annotations include the truth-value of each statement with regard to the images paired with it in the data. Once we manipulate the image (i.e., change the state in our interactive environment), the truth-value annotation does hold. A key challenge for creating an interactive environment using the NLVR data is the need for an accurate evaluation of the natural language statement for *every* possible state (i.e., image), as required for reward computation (Section 3).

We address this challenge by annotating each statement $\bar{x}$ with an executable boolean Python program representing its meaning, $\mathcal{E}^{\bar{x}}$ (Section 3). The Python program operates on the underlying structured representation. It returns `True` for every image that satisfies the constraints specified in the corresponding statement, and `False` otherwise. In general, there are many states that satisfy any given statement, many more than provided with the original NLVR images.

The programs are written using an API defined over the structured representations. We base the API design on the logical ontology designed for NLVR's structured representations by Goldman et al. (2018), which we extend to include a total of 66 functions. Figure 4 in Appendix B shows two examples of logical forms paired with a corresponding image.

We use the freelancing platform Upwork[5] for annotation. We recruit three annotators based on preliminary screening of their fluency in English and competency in Python. We de-duplicate the naturally occurring sentences in the data, collect 2,666 annotations at a total cost of \$3,756, and keep 2,661 valid annotations.

All the sentences in the dataset are randomly distributed to the annotators, each with an example image. Every sentence is annotated with a logical form by one annotator. Each logical form is evaluated against a corresponding hidden validation set, and must pass all the tests. Appendix B.2 describes our annotation and validation process.

5 Experiments

5.1 Methods

We experiment with each of the four CMDPs separately, training on the training data and testing on the development and held-out test splits. We sample a validation set from the training data for model selection. For `SCATTER` we use a grid of 19×5 (Section 3). Each grid cell is of size 20×20 pixels. We set the action horizon $H = 12$. Appendix A.1 and Appendix C provide more details about our setup.

[4]We do no use the photographic NLVR2 in this work.

[5]`https://www.upwork.com`

We use PPO (Schulman et al., 2017) for parameter estimation,[6] with a separate network as a critic. The critic network is identical to the policy, except that we add a tanh activation for the value output. Because of the large action space, especially for SCATTER, the agent rarely observes positive reward, which requires taking a STOP action at an appropriate state. We design a simple variant of PPO called PPO+SF (PPO with stop forcing) to address this issue. PPO+SF is identical to PPO, except that during training, we mask all actions except STOP when the agent reaches a state where selecting STOP will give a positive reward. This modification is present only during training. All testing is done under the same conditions, without stop forcing.

We experiment with two models:

C+BERT We process the statement $\bar{x}$ using BERT (Devlin et al.), and do mean pooling across all layers and tokens to get the statement representation. We use a three-layer CNN (Fukushima & Miyake, 1982) to embed the image of the current state s. We concatenate the statement representation, image representation, and an embedding for the target boolean b, and process the vector through a multi-layer perceptron (MLP) to compute the action distribution.

ViLT ViLT is a pretrained multi-modal Transformer that jointly processes text and image inputs (Kim et al., 2021). We create a sequence of tokens by concatenating the statement, a token for the target boolean, and image patches, separated by special tokens. The image patches are the same size as the 19×5 grid cells, including in TOWER, where the action space does not use a grid.

5.2 Results and Quantitative Analysis

Table 2 shows development and test set accuracies for all CMDPs. Figure 2 shows development set accuracies for FLIPIT CMDPs broken down by the target boolean, and Figure 3 shows development rollout statistics. ViLT outperforms C+BERT, except on SCATTER-SCRATCH. This is relatively expected given the joint reasoning architecture and multi-modal pre-training of ViLT.

Generally, policies do better on FLIPIT examples with a False target boolean, except when learning largely does not work (Figure 2). The set of states that invalidate a statement is usually larger than the set that validates it, and it generally requires fewer actions to invalidate a statement.

We observe more rollouts that are terminated by reaching the action horizon H (i.e., without STOP) on TOWER CMDPs compared to SCATTER (Figure 3, left). This difference is partially explained by a higher rate of invalid actions in SCATTER (Figure 3, center left), which immediately terminate the rollout. In general, ViLT has a higher rate of invalid actions, except on SCATTER-FLIPIT with PPO, where overall performance is extremely low. We also see more non-stopped rollouts for TOWER-FLIPIT when training with PPO+SF. These rollouts often correspond to the model getting stuck in add-remove loops. There is no consistent difference in the length of rollouts between the two models and agents (Figure 3, center right).

In general, we observe no significant effective learning of using REMOVE actions. TOWER-FLIPIT is an exception with REMOVE dominating the rollouts, potentially because removing objects generally provides a more efficient path to flip the boolean value. While PPO policies generate REMOVE actions for SCATTER CMDPs, the much higher performance of PPO+SF policies indicates that the use of these actions is ineffective.

5.3 Qualitative Analysis

We sample 50 development examples for each CMDP, and annotate them with expert[7] trajectories to estimate the expert reward (Table 3). For each model, we compute the mean reward on all development examples. The analysis indicates that there is significant room to improve model performance and efficiency. For example, on TOWER-FLIPIT, the estimated expert mean reward is 0.83, while ViLT trained with PPO, the best of our models, gets a reward of 0.22, or 0.83 for the subset that is completed successfully.

[6]We use the PPO implementation of Kostrikov (2018).
[7]The expert is an author of this paper.

Table 2: Accuracies for all the four CMDP. Evaluation is always without stop forcing.

		TOWER-SCRATCH		TOWER-FLIPIT		SCATTER-SCRATCH		SCATTER-FLIPIT	
		Dev	Test	Dev	Test	Dev	Test	Dev	Test
PPO	C+BERT	71.78	63.27	35.95	34.78	**39.08**	**48.39**	0.00	0.00
	ViLT	**81.60**	**76.54**	**67.60**	**65.80**	35.63	41.29	**3.51**	**6.09**
PPO+SF	C+BERT	80.98	78.70	27.22	26.75	**70.12**	**74.84**	8.31	8.46
	ViLT	**84.05**	**82.41**	**65.09**	**62.91**	64.37	70.97	**27.48**	**29.95**

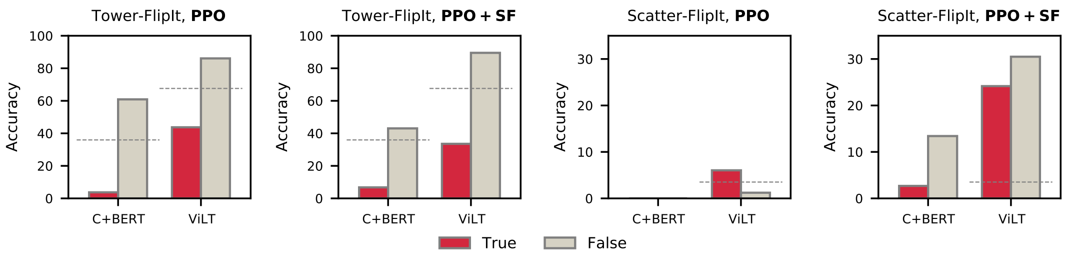


Figure 2: Development set accuracies for FLIPIT CMDPs, reported according to the value of the context target boolean. **Red**: target is True. **Gray**: target is False. **Dashed gray line**: accuracy on the full development set.

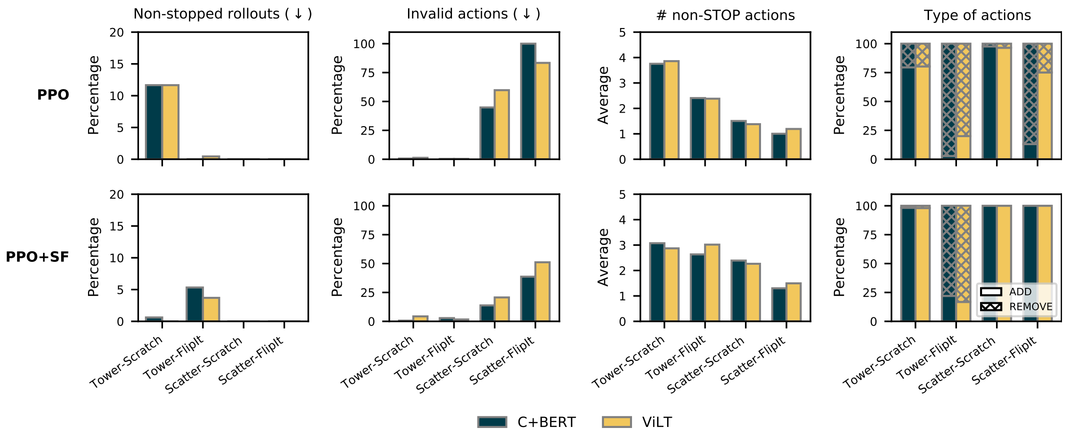


Figure 3: Development rollout statistics, from left to right: percentage of rollouts that reach the action horizon without a STOP action; percentage of rollouts with an invalid action; mean actions per rollout; percentage of ADD/REMOVE actions. ↓ shows the preferred direction.

Suhr et al. (2017) released a set of 200 development examples annotated for semantic phenomena. Table 4 shows the performance of policies on the different CMDPs trained with PPO on this data. We only include categories with more than 10 instances across all CMDPs. Appendix D.2 includes the complete tables with examples, including for PPO+SF. In most cases, we observe the two models to follow similar trends with respect to the categories on which they perform above and below average performance. A notable exception is on coordination, where ViLT performs above general performance on 2/4 cases, and C+BERT always performs below. A notable difference between TOWER and SCATTER is on soft cardinality (e.g., ... *at least two* ...), where we observe above average performance on TOWER and below average on SCATTER.

We diagnose the errors that PPO makes by sampling 50 erroneous development examples for each of the two SCATTER CMDPs. In SCATTER-SCRATCH with C+BERT, 76% of the errors are due to invalid actions, and 24% due to early termination. Among the invalid actions, 58% are due to trying to put an item that cannot fit in the box, 24% are due to trying to perform an action on a

Table 3: Mean development set rewards. Expert reward are estimated on 50 examples.

		TOWER-SCRATCH	TOWER-FLIPIT	SCATTER-SCRATCH	SCATTER-FLIPIT
Expert		0.79	0.83	0.79	0.89
PPO	C+BERT	0.27	-0.42	-0.27	-1.00
	ViLT	0.45	0.22	-0.33	-0.95
PPO+SF	C+BERT	0.42	-0.57	0.26	-0.86
	ViLT	0.49	0.13	0.16	-0.50

Table 4: Performance on a set of development examples annotated for semantic categories by Suhr et al. (2017) for both models (C+BERT | ViLT) when trained with PPO. Dev performance refers to the performance on the respective full development set. Results outperforming the average performance are in bold.

	TOWER-SCRATCH			TOWER-FLIPIT			SCATTER-SCRATCH			SCATTER-FLIPIT		
	Total	Correct %		Total	Correct %		Total	Correct %		Total	Correct %	
Cardinality (hard)	98	67.3	78.6	480	**36.7**	**70.8**	35	31.4	**37.1**	119	0.0	**5.0**
Cardinality (soft)	21	**81.0**	**85.7**	82	**43.9**	65.9	11	9.1	18.2	42	0.0	2.4
Existential	122	**73.8**	**83.6**	577	**37.6**	**70.7**	55	36.4	32.7	192	0.0	**3.6**
Coordination	19	31.6	52.6	86	29.1	**70.9**	15	20.0	**40.0**	55	0.0	0.0
Spatial Relations	93	**77.4**	**87.1**	438	**39.7**	**69.2**	39	**41.0**	30.8	128	0.0	**5.5**
Presupposition	17	47.1	58.8	74	35.1	**68.9**	22	**40.9**	**40.9**	78	0.0	**5.1**
Dev performance		71.78	81.60		35.95	67.60		39.08	35.63		0.00	3.51

separator, and 18% due to trying to remove an object from a position that does not include an object. For other configurations, the causes of errors are similar, and details about the distribution are in Appendix D.1.

6 CONCLUSION

We introduce ℓGym, a reinforcement learning benchmark that focuses on natural language visual reasoning. ℓGym is designed to be accessible for RL researchers, while still displaying the reasoning richness of natural language. It is relatively easy to deploy using the standard OpenAI Gym API (Brockman et al., 2016), and has light compute requirements. Our data annotation approach allows including expressive and diverse natural language, while still providing accurate and automatic reward computation. This allows ℓGym to balance between posing a challenging research problem and avoiding engineering challenges or approximation costs. Our strong baselines illustrate the range of challenges ℓGym presents, showing that existing methods can achieve non-trivial performance, but that there remains significant progress to be made. Our analysis lays out the framework for studying and reporting these future results.

ℓGym also holds significant potential beyond RL. Our annotations form a new semantic parsing corpus with annotated executable meaning representations. The semantic diversity of the data, its executability, and the focus on visual reasoning make it a unique asset in the landscape of corpora for semantic parsing. ℓGym also holds potential for program synthesis, where there is recent focus on using language to guide synthesis from examples (Wong et al., 2021), data that ℓGym provides.

REFERENCES

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3674–3683, 2018.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: Visual question answering. In *IEEE International Conference on Computer Vision*, pp. 2425–2433, 2015.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Howard Chen, Alane Suhr, Dipendra Misra, Noah Snavely, and Yoav Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12538–12547, 2019.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *7th International Conference on Learning Representations, ICLR*, 2019.

John D. Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, John DeNero, P. Abbeel, and Sergey Levine. Guiding policies with language via meta-learning. In *7th International Conference on Learning Representations, ICLR*, 2019.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben A. Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew J. Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. In *CGW@IJCAI*, 2018.

Pradeep Dasigi, Matt Gardner, Shikhar Murty, Luke Zettlemoyer, and Eduard Hovy. Iterative search for weakly supervised semantic parsing. 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)*, pp. 2669–2680.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. 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)*, pp. 4171–4186.

Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pp. 267–285. Springer, 1982.

Omer Goldman, Veronica Latcinnik, Ehud Nave, Amir Globerson, and Jonathan Berant. Weakly supervised semantic parsing with abstract examples. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1809–1819, 2018.

Nitish Gupta, Sameer Singh, and Matt Gardner. Enforcing consistency in weakly supervised semantic parsing. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 168–174, 2021.

Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

Austin W Hanjie, Victor Y Zhong, and Karthik Narasimhan. Grounding language to entities and dynamics for generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 4051–4062. PMLR, 2021.

Drew A. Hudson and Christopher D. Manning. Compositional attention networks for machine reasoning. In *6th International Conference on Learning Representations, ICLR*, 2018.

Vihan Jain, Gabriel Magalhaes, Alexander Ku, Ashish Vaswani, Eugene Ie, and Jason Baldridge. Stay on the path: Instruction fidelity in vision-and-language navigation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 1862–1872, July 2019.

Minqi Jiang, Jelena Luketina, Nantas Nardelli, Pasquale Minervini, Philip HS Torr, Shimon Whiteson, and Tim Rocktäschel. Wordcraft: An environment for benchmarking commonsense agents. *arXiv preprint arXiv:2007.09185*, 2020.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2901–2910, 2017a.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. Inferring and executing programs for visual reasoning. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 3008–3017, 2017b.

Wonjae Kim, Bokyung Son, and Ildoo Kim. Vilt: Vision-and-language transformer without convolution or region supervision. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 5583–5594, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR*, 2015.

Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms, 2018.

Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4392–4412, 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR*, 2019.

Dipendra Kumar Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP7*, pp. 1004–1015, 2017.

Karthik Narasimhan, Tejas D. Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pp. 1–11, 2015.

Juan Pavez, Héctor Allende, and Héctor Allende-Cid. Working memory networks: Augmenting memory networks with a relational reasoning module. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1000–1009, 2018.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *9th International Conference on Learning Representations, ICLR*, 2021.

Alane Suhr, Mike Lewis, James Yeh, and Yoav Artzi. A corpus of natural language for visual reasoning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 217–223, 2017.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2238–2249, 2018.

Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. A corpus for reasoning about natural language grounded in photographs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 6418–6428, 2019.

Hao Tan and Mohit Bansal. Object ordering with bidirectional matchings for visual reasoning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 444–451, 2018.

Catherine Wong, Kevin Ellis, Joshua B. Tenenbaum, and Jacob Andreas. Leveraging language to learn program abstractions and search heuristics. In *Proceedings of the 38th International Conference on Machine Learning, ICML*, pp. 11193–11204, 2021.

Yiqun Yao, Jiaming Xu, Feng Wang, and Bo Xu. Cascaded mutual modulation for visual reasoning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 975–980, 2018.

J.M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*, 1996.

Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2005.

Wenbo Zheng, Lan Yan, Chao Gou, and Fei-Yue Wang. Webly supervised knowledge embedding model for visual reasoning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12442–12451, 2020.

A Additional ℓGym Design Details

A.1 SCATTER Grid Details

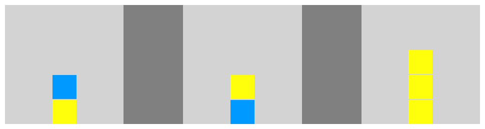
In this configuration of SCATTER, there can be objects smaller than an individual cell. Therefore, we can add multiple items in a cell if there is enough space, and when two objects are close enough (their closest distance is smaller than ϵ), we *stick* them by making them touch.

For ADD actions, we search for a pixel in the grid box where we can add the object starting from the upper left corner. For REMOVE, we remove the largest element overlapping with the given grid cell.

B Additional Data Details

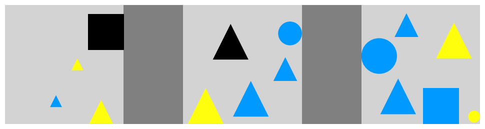
B.1 Annotation Examples

Figure 4 shows two examples of text statement with their respective executable boolean Python program.



There are two towers with the same height but their base is not the same in color.

```
exist(filter_obj(all_boxes, lambda x:  x.is_tower() and exist(filter_obj(all_boxes,
lambda y:  y.is_tower() and count(x.all_items_in_box()) == count(y.all_items_in_box())
and get_set_colors(filter_obj(y.all_items_in_box(), is_bottom)) !=
get_set_colors(filter_obj(x.all_items_in_box(), is_bottom)))))
```



There is a box with all 3 different colors and a black triangle touching the wall with its top.

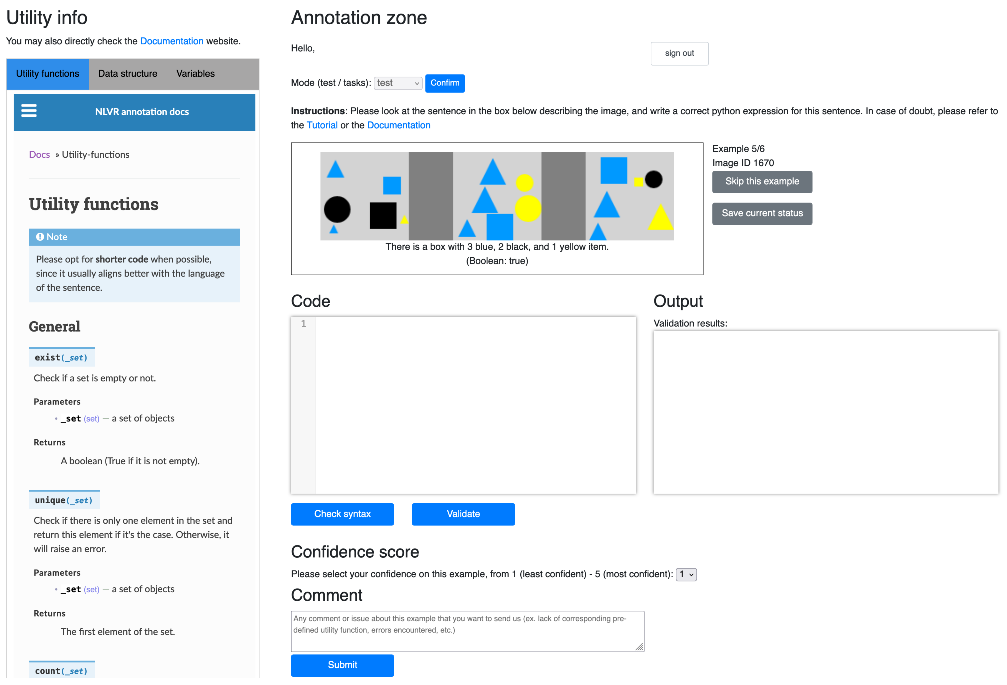
```
exist(filter_obj(all_boxes, lambda x:  count(get_set_colors(x.all_items_in_box())) == 3
and exist(filter_obj(x.all_items_in_box(), lambda y:  is_black(y) and is_triangle(y)
and is_touching_wall(y, Side.TOP)))))
```

Figure 4: Example sentences, a corresponding logical form annotation under the format of a Python program, and an example image from the dataset. The sentence and logical form are `True` for the top statement, and `False` for the bottom statement.

B.2 Annotation and Validation Process

We provide the annotators with a web-based annotation interface (Figure 5), a tutorial and an application programming interface (API) presenting a set of functions, classes and objects that they can use for annotation. We ask the annotators to prioritize the faithfulness of the program to the natural language sentence and to prefer shorter annotations. We also provide them with examples of spurious logical forms and ask them to avoid such expressions. Annotators can raise questions in case of doubts.

For every sentence, annotators are provided with an example image and an associated boolean value. The program written is executed online, and the annotators can only submit their annotation after passing the online Python syntax checker, the visible example and all the hidden validation examples. The annotators can assign a confidence score to their annotation and provide a comment. Annotators can also skip examples in case of doubt. When skipping, they need to explicitly provide the reason. We assess the annotations by batch, then randomly redistribute the skipped examples or examples with problematic annotations to the annotators after the questions have been solved. An iterative communication has been followed throughout the annotation process.



Utility info

You may also directly check the Documentation website.

Utility functions

Note

Please opt for **shorter code** when possible, since it usually aligns better with the language of the sentence.

General

`exist(_set)`

Check if a set is empty or not.

Parameters
- `_set` (set) — a set of objects

Returns

A boolean (True if it is not empty).

`unique(_set)`

Check if there is only one element in the set and return this element if it's the case. Otherwise, it will raise an error.

Parameters
- `_set` (set) — a set of objects

Returns

The first element of the set.

`count(_set)`

Annotation zone

Hello,

Mode (test / tasks): test

Instructions: Please look at the sentence in the box below describing the image, and write a correct python expression for this sentence. In case of doubt, please refer to the Tutorial or the Documentation

There is a box with 3 blue, 2 black, and 1 yellow item.
(Boolean: true)

Example 5/6
Image ID 1670

Code

Output

Validation results:

Confidence score

Please select your confidence on this example, from 1 (least confident) - 5 (most confident): 1

Comment

Any comment or issue about this example that you want to send us (ex. lack of corresponding pre-defined utility function, errors encountered, etc.)

Figure 5: Annotation interface for collecting the Python program annotations in ℓGym, as seen by the annotators.

C EXPERIMENTAL SETUP DETAILS

C.1 LEARNING DETAILS

Hyperparameters For our experiments with C+BERT, we use Adam (Kingma & Ba, 2015) for optimization with a learning rate of 3e-4, except on `TOWER-FLIPIT` trained with PPO+SF and on `SCATTER-FLIPIT`, where we use 3e-5.

For ViLT's optimization, we use AdamW (Loshchilov & Hutter, 2019) with a cosine scheduler and a base learning rate of 3e-5 for all experiments, except on `SCATTER-SCRATCH` trained with PPO+SF where we use 3e-4. The learning rate is warmed up for 1% of the maximal total training steps.

For all our experiments, entropy is set to 0.3. The number of steps per batch is 2,048 and a mini-batch size of 32 is used. We use patience for early stopping, and stop after 4 million steps regardless of patience.

PPO+SF Setup Details PPO+SF is a simple variant of PPO, which applies masking to all the actions except for `STOP` when the agent reaches a goal state, in which it will receive a positive reward if it selects `STOP`. To account for issues where the `STOP` action has very low probability and the gradients potentially exploding creating instability, we clip the PPO ratio. Formally, the original PPO objective can be defined as:

$$L(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right] \tag{1}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$, $\hat{A}$ is the advantage function and ϵ is a hyperparameter (see Schulman et al. (2017) for more details). For PPO+SF we clip the ratio term $r_t(\theta)$, as follows:

$$\hat{r}_t(\theta) = \min\Big(r_t(\theta), M \Big) \tag{2}$$

where M is a threshold bounding the ratio. We use $\hat{r}_t(\theta)$ in place of $r_t(\theta)$ for our experiments.

C.2 Inference Details

During inference, the agent π is presented with a given state s_t and an action $a \sim \pi(\cdot|s_t, c)$. The action space consist of three actions `STOP`, `ADD` and `REMOVE`. These actions also take a set of arguments. Some actions take positional arguments, for example in `TOWER`, `ADD`(position, color) has two arguments, `REMOVE`(position) has one argument, where `STOP`() has none.

Since not all actions arguments are used in `REMOVE` and `STOP`, we marginalize over these unspecified action arguments.

D Additional Results and Analysis

D.1 Additional Error Analysis

What types of mistakes does PPO models make? We provide detailed error analysis for the configurations on `SCATTER-FLIPIT` and `SCATTER-SCRATCH` CMDPs trained with PPO, following Section 5.3. For each configuration, we analyze 50 erroneous development examples.

In `SCATTER-SCRATCH` trained with PPO, for ViLT, all mistakes are due to invalid actions. Among the invalid actions, 90% are due to trying to perform an action on a separator, and 10% due to trying to remove an object from a position that does not include an object.

In `SCATTER-FLIPIT` trained with PPO, all the mistakes of C+BERT are also due to invalid actions. Among the invalid actions, 84% are due to trying to remove an object from a position that does not include an object, 8% are due to trying to perform an action on a separator, 4% are due to trying to put an item that cannot fit in the box, and 4% are due to trying to add an object on top of an existing one.

For ViLT, 90% of the mistakes are due to invalid actions, and 10% due to early termination. Among the invalid actions, 44% are due to trying to remove an object from a position that does not include an object, 34% due to trying to perform an action on a separator, 10% due to trying to add an object on top of an existing one, and 2% due to trying to put an item that cannot fit in the box.

D.2 Analysis by Semantic and Syntactic Categories

Table 5 and Table 6 show the complete tables of the performance of both PPO and PPO+SF policies on all four CMDPs, on a set of development examples annotated by Suhr et al. (2017), per semantic and syntactic categories. For each category, an example sentence is provided.

Table 5: Performance on a set of development examples annotated for semantic and syntactic categories by Suhr et al. (2017) for both models (C+BERT | ViLT) when trained with PPO. Dev performance refers to the performance on the respective full development set. Results outperforming the average performance are in bold.

	`TOWER-SCRATCH`			`TOWER-FLIPIT`			`SCATTER-SCRATCH`			`SCATTER-FLIPIT`			
	Total	Correct %		Total	Correct %		Total	Correct %		Total	Correct %		Example
Semantics													
Cardinality (hard)	98	67.3	78.6	480	**36.7**	**70.8**	35	31.4	**37.1**	119	0.0	**5.0**	*There are **exactly four objects** not touching any edge*
Cardinality (soft)	21	**81.0**	**85.7**	82	**43.9**	65.9	11	9.1	18.2	42	0.0	2.4	*There is a box with **at least one** square and **at least three** triangles.*
Existential	122	**73.8**	**83.6**	577	**37.6**	**70.7**	55	36.4	32.7	192	0.0	**3.6**	***There is a tower** with yellow base.*
Universal	7	14.3	14.3	28	32.1	50.0	9	**44.4**	**55.6**	36	0.0	0.0	*There is a black item in **every box**.*
Coordination	19	31.6	52.6	86	29.1	**70.9**	15	20.0	**40.0**	55	0.0	0.0	*There are 2 blue circles **and** 1 blue triangle*
Coreference	3	0.0	33.3	10	30.0	30.0	3	0.0	33.3	9	0.0	0.0	*There is a blue triangle touching the wall with **its** side.*
Spatial Relations	93	**77.4**	**87.1**	438	**39.7**	**69.2**	39	**41.0**	30.8	128	0.0	**5.5**	*there is one tower with a yellow block **above** a yellow block*
Comparative	5	40.0	20.0	20	20.0	30.0	1	**100.0**	**100.0**	4	0.0	0.0	*There is a box with multiple items and only one item **has a different color.***
Presupposition	17	47.1	58.8	74	35.1	**68.9**	22	**40.9**	**40.9**	78	0.0	**5.1**	*There is a box with seven items and **the three black items** are the same in shape.*
Negation	4	**75.0**	75.0	15	13.3	26.7	15	33.3	28.6	53	0.0	0.0	*there is exactly one black triangle **not touching** the edge*
Syntax													
Coordination	4	25.0	**100.0**	14	21.4	**71.4**	5	20.0	20.0	20	0.0	0.0	*There is a box with at least one square **and** at least three triangles.*
PP Attachment	44	**84.1**	**93.2**	215	**42.3**	**72.1**	2	0.0	33.3	7	0.0	**14.3**	*There is a black block on a black block as the base of a tower **with** three blocks.*
Dev performance		71.78	81.60		35.95	67.60		39.08	35.63		0.00	3.51	

Table 6: Performance on a set of development examples annotated for semantic and syntactic categories by Suhr et al. (2017) for both models (C+BERT | ViLT) when trained with PPO+SF. Dev performance refers to the performance on the respective full development set. Results outperforming the average performance are in bold.

	`TOWER-SCRATCH`			`TOWER-FLIPIT`			`SCATTER-SCRATCH`			`SCATTER-FLIPIT`			
	Total	Correct %		Total	Correct %		Total	Correct %		Total	Correct %		Example
Semantics													
Cardinality (hard)	98	**83.7**	**84.7**	480	24.6	62.7	35	68.6	60.0	119	6.7	22.7	*There are* ***exactly four objects*** *not touching any edge*
Cardinality (soft)	21	**81.0**	81.0	82	**29.3**	63.4	11	27.3	27.3	42	7.1	26.2	*There is a box with* ***at least one*** *square and* ***at least three*** *triangles.*
Existential	122	**86.9**	**82.8**	577	25.8	64.1	55	**70.9**	**67.3**	192	5.2	27.1	***There is a tower*** *with yellow base.*
Universal	7	42.9	**100.0**	28	14.3	57.1	9	44.4	44.4	36	5.6	16.7	*There is a black item in* ***every box****.*
Coordination	19	63.2	**84.2**	86	**34.9**	60.5	15	33.3	33.3	55	3.6	14.5	*There are 2 blue circles* ***and*** *1 blue triangle*
Coreference	3	33.3	**66.7**	10	20.0	40.0	3	33.3	**66.7**	9	0.0	**33.3**	*There is a blue triangle touching the wall with* ***its*** *side.*
Spatial Relations	93	**83.9**	84.0	438	21.0	60.3	39	82.1	**76.9**	128	3.9	**34.4**	*there is one tower with a yellow block* ***above*** *a yellow block*
Comparative	5	40.0	80.0	20	0.0	40.0	1	**100.0**	**100.0**	4	0.0	25.0	*There is a box with multiple items and only one item* ***has a different color.***
Presupposition	17	58.8	**94.1**	74	18.9	58.1	22	68.2	**72.7**	78	7.7	**28.2**	*There is a box with seven items and* ***the three black items*** *are the same in shape.*
Negation	4	75.0	75.0	15	13.3	53.3	15	**80.0**	**66.7**	53	3.8	**28.8**	*there is exactly one black triangle* ***not touching*** *the edge*
Syntax													
Coordination	4	75.0	75.0	14	**28.6**	57.1	5	40.0	40.0	20	0.0	10.0	*There is a box with at least one square* ***and*** *at least three triangles.*
PP Attachment	44	**95.5**	**84.1**	215	22.8	60.0	2	0.0	0.0	7	0.0	12.5	*There is a black block on a black block as the base of a tower* ***with*** *three blocks.*
Dev performance		80.98	84.05		27.22	65.09		70.12	64.37		8.31	27.48	

D.3 Analysis of Positional Bias on SCATTER

Figure 6 and Figure 7 show the positional bias on the development set of `SCATTER-SCRATCH` and `SCATTER-FLIPIT`, for both PPO and PPO+SF algorithms, and for both C+BERT and ViLT models.

For all algorithm-model pairs on both CMDPs, we can see that actions tend to be biased towards a set of positions. When there is a bias towards positions at the bottom of the image, for instance for ViLT trained with PPO on `SCATTER-FLIPIT` (Figure 7, upper right), the agent would hit an invalid action if it tries to put a large item in a bottom cell that cannot fit into it. For ViLT, we can see that training with PPO+SF (Figure 6, bottom right and Figure 7, bottom right) helps to make more varied choices of positions on both CMDPs.

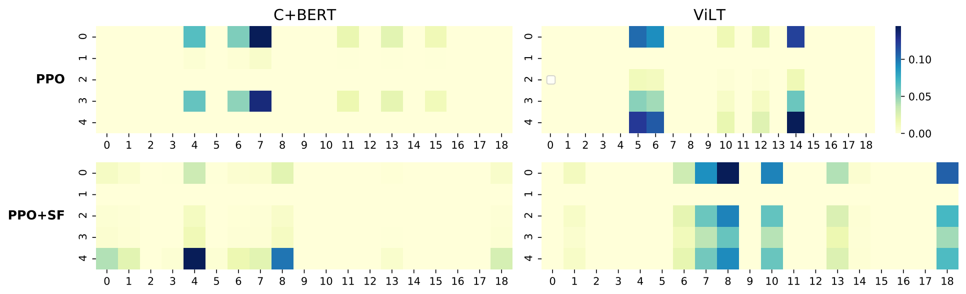


Figure 6: Frequency of choosing the cell (x, y) for `SCATTER-SCRATCH`, on the development set. x-axis and y-axis respectively show the x and y position of the cells in the `SCATTER` grid.

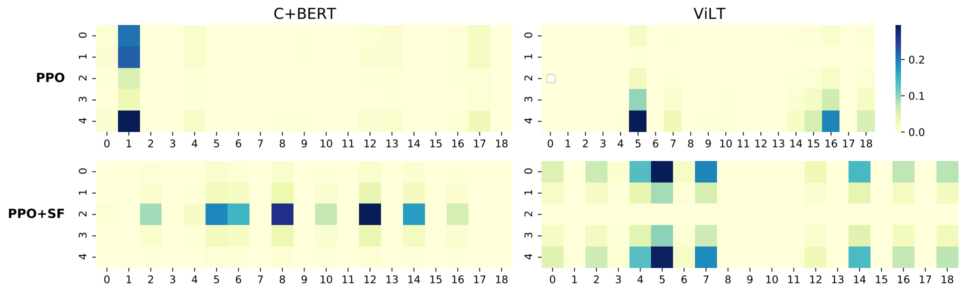


Figure 7: Frequency of choosing the cell (x, y) for `SCATTER-FLIPIT`, on the development set. x-axis and y-axis respectively show the x and y position of the cells in the `SCATTER` grid.