Language Inference for Reward Learning

Xiang Fu
MIT CSAIL
Cambridge, MA 02139
xiangfu@csail.mit.edu

Tao Chen
MIT CSAIL
Cambridge, MA 02139
taochen@csail.mit.edu

Pulkit Agrawal
MIT CSAIL
Cambridge, MA 02139
pulkitag@mit.edu

Tommi S. Jaakkola
MIT CSAIL
Cambridge, MA 02139
tommi@csail.mit.edu

Abstract

Inferring the demonstrator's intent from a set of demonstrations is the central problem in inverse reinforcement learning. The fundamental challenge stems from the existence of multiple reward functions that explain the same set of demonstrations. We make an observation to address this problem: structure in reward functions for many real-world tasks is well captured by formal language. We use this observation to construct an efficient reward-learning algorithm based on regular expressions called *Regular Expression Reward Learning* (RERL). We demonstrate that RERL can learn intent with very few demonstrations, and outperforms existing methods for reward inference on a suite of long-horizon tasks.

1 Introduction

Demonstrations are a natural and powerful way to communicate tasks to an agent. However, directly mimicking demonstrations often results in poor performance due to factors such as differences in agent's and demonstrator's environment. These issues can be overcome by first inferring the intent of the demonstrator (i.e., the reward function) and then determining a policy in the agent's environment that is consistent with the inferred intent. This approach, known as inverse reinforcement learning (IRL) [1, 2] has been widely studied.

Despite the success of IRL in domains such as simulated locomotion, one major challenge in inverse reinforcement learning is that a set of few demonstrations are well explained by many reward functions (i.e., the *reward ambiguity problem*) [3]. As an example, consider a demonstration involving picking up a blue ball, a red box, and then a green star. Just from this demonstration, it is unclear whether the order of picking up the ball, box, and star is important or if the demonstrator's intent was just to pick up these objects in any order?

The previous discussion suggests that a model for inferring intent should be expressive and capable of simultaneously representing a diverse set of plausible reward functions (i.e., capture multi-modality). An additional desideratum is a mechanism to efficiently prune the space of reward functions, either by querying the demonstrator or by incorporating good priors. Past works such as [4] have used deep neural networks to represent the reward function. While deep neural networks (DNNs) are universal function approximators and, therefore, very expressive, they suffer from the curse of data inefficiency. Furthermore, because DNNs lack inductive biases that are useful when working with natural data and tasks, the inferred reward function does not generalize to environments beyond the training set.

In this work, we advocate using stronger but very general priors to infer reward functions for the family of tasks humans typically engage in. Some such priors are: humans think about the world

Biological and Artificial RL Workshop at NeurIPS 2020. https://sites.google.com/view/biologicalandartificialrl

in terms of objects, relationships between objects [5, 6], and often make use of recursion [7]. To incorporate such general priors, we propose to use formal language and, more specifically, regular expressions for inferring reward functions. We call this scheme of inferring reward functions as *regular expression reward learning* (RERL). We show that RERL outperforms alternative approaches based on deep learning on a suite of long-horizon sequence-dependent reward inference tasks. More importantly, due to the built-in inductive biases, RERL generalizes to previously unseen environments.

2 Regular Expression Reward Learning (RERL)

We can represent the task by a language pattern ζ (*e.g.*, 'trajectories where the agent visit s_{target} twice') and the reward r can then be defined by the degree of match M between the current trajectory τ and the language pattern ζ, *i.e.*, $r_\zeta(\tau) = M(\zeta, \tau)$. In essence, we can treat state-action trajectories as strings. The task of reward inference then reduces to finding a common pattern that distinguishes the demonstrations.

Regular expression (RE) is a standard tool for describing string patterns. Regular expression is sufficient for representing any regular language. We consider regular expressions because regular expression operators (?, * , |, etc.) can easily describe complex patterns such as options and recurrence. For example, 'A?' represents that A is optional; 'A|B' means either A or B is acceptable; 'A*' denotes A may be repeated. We call a regular expression *satisfying* if it accepts all the positive demonstrated strings and rejects all the negative strings.

We consider the following setup: A demonstrator provides multiple demonstrations of how to perform (i.e., positive) and how not to perform (i.e., negative) task demonstrations. From these demonstrations, using regular expressions based inference scheme the agent determines the reward function (i.e., the demonstrator's intent). While in general, many language patterns maybe consistent with the demonstrations, simple ones are preferred based on the principle of Occam's Razor [8]. In this work, we define a cost on the regular expression as a surrogate for measuring the pattern's simplicity. Each symbol and operator in the regular expression has a cost c_i. A sequence has a total cost $c_\zeta = \sum_0^{|\zeta|} c_i$. We sample regular expressions with lower costs more often than ones with higher costs. For this we use a simple mapping function $f(c) = \exp(-c)$ to construct the sampling distribution: $q(\zeta) \propto \exp(-c_\zeta)$.

We define the reward function based on two principles: (1): Reward depends on the regular expressions that accept all positive examples and reject all negative examples. (2): Among all the regular expressions that satisfy (1), one with lower cost should have a greater influence on reward estimation. With these principles, we define the reward function as follows:

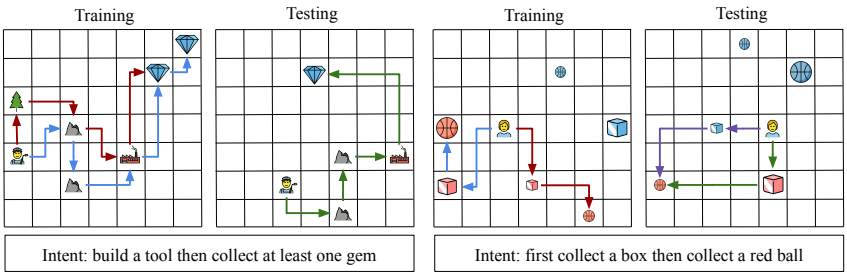
$$r(\tau) = \mathbb{E}_{\zeta \in Z_S} M(\zeta, \tau) = \sum_{\zeta \in Z_S} \frac{\exp(-c_\zeta)}{\sum_{\zeta \in Z_S} \exp(-c_\zeta)} M(\zeta, \tau) \tag{1}$$

Where c_ζ is the cost of the regular expression ζ, $M(\zeta, \tau) = 1$ if the string τ matches the regular expression ζ and $M(\zeta, \tau) = 0$ otherwise. Z_S is a set of satisfying regular expressions. In practice, we search for K lowest cost expressions using the language inference engine and add them to Z_S as an approximation of the true posterior. In our experiments with sparse binary reward, $K = 1$ is sufficient, which means the reward is estimated through the regular expression with the lowest cost. In our implementation, we use an efficient search-based algorithm from [9] to search for the lowest cost satisfying regular expression.

RERL can represent a variety of natural patterns with a small number of bits, and infer a reward function via regular expression with a small number of examples. RERL is also more interpretable than a parameterized reward function. For general IRL algorithms, one needs to train an RL policy on the learned reward function to act in a new task. With RERL, one can also use a regular expression interpreter and perform a new task by producing trajectories that match the regular expression.

3 Environments and evaluation

In this section, we evaluate our methods for reward inference tasks, in three long-horizon environments where the reward function is non-Markovian. Besides these environments, we also illustrate the effectiveness of language inference in a non-sequential, repeated game setting using contextual bandits [10] in Appendix A.



(a) (b)

Figure 1: Non-Markovian reward environments. The true intents of the demonstrator are shown in the text box below the environment figures. **(a)**: The crafting environment. In the training environment, both the red and the blue demonstrations are valid for the intent. In the example testing environment, the only rewarding trajectory is the green trajectory. **(b)**: The collecting environment. In the training environment, both the red and the blue demonstrations are valid for the intent. While in the testing environment with different objects, the rewarding trajectories are shown in green and purple. More details on the environments are included in Appendix B. Emojis are from OpenMoji.

3.1 Non-Markovian 2D Environments

Environment I - Crafting: We first consider a crafting environment modified from [11]. The crafting environment is a 2D world with objects (wood, iron, gem, crafting station) that the agent can interact with. To collect any gem (`G`), the agent needs to craft either a hammer or a pickaxe as the tool. The number of materials varies across environments. If there are wood (`W`) and iron (`I`), the agent can make a pickaxe. If there are only irons, the agent makes a hammer. The agent collects the raw materials (wood or iron), goes to the crafting station (`S`) to make the tool, and then collects the gems.

The demonstrator will give several positive/negative demonstrations with an intent in mind. Each demonstration is represented by an interaction sequence (trajectory) τ. For example, the demonstrator in Figure 1a has the intent $\zeta = \texttt{(WI|II)SGG*}$ to build a tool and then collect at least one gem. He gives the blue and red positive demonstrations. $\tau_{\text{red}} = \texttt{WISG}$, $\tau_{\text{blue}} = \texttt{IISGG}$. τ_{red} represents an interaction sequence where the demonstrator collects one piece of wood (`W`) and one iron (`I`), builds a pickaxe in the crafting station (`S`), and then collects the gems (`G`). τ_{blue} means that the demonstrator first collects two irons and builds a hammer, then moves to collect the two gems.

Environment II - Object Collection from Feature Observations: In the crafting environment described above, every object is represented by a specific symbol. However, in the real world such symbolic representation must be inferred from sensory inputs or feature descriptions. In order to investigate if our method can also work with features we constructed another environment where symbolic information is not directly available. The details of the environment are as following: each object has three binary features: (a) shape (`ball(B)` or `cube(C)`); (b) size (`large(0)` or `small(1)`); and (c) color (`red(0)` or `blue(1)`). A teacher demonstrates several trajectories of object collection (Figure 1b) that reflects her goal. Similar to the crafting environment, these demonstrations can be represented as strings, where each element in the string denotes the presence/absence of a feature. Because each object has three features it can be represented by a string of length 3. A trajectory is a concatenation of the strings of all objects that the agent interacts with. For example, the string `01C` means a large blue cube. Figure 1b illustrates more examples: the blue-colored trajectory can be represented as $\tau_{\text{blue}} = \texttt{01C} \oplus \texttt{00B} = \texttt{01C00B}$. The red trajectory is $\tau_{\text{red}} = \texttt{10C10B}$. Both trajectories can be captured by the intent ζ in the form of a regular expression `(0+1)(0+1)C(0+1)0B`. Note that we do not assume that trajectories are of fixed length.

3.1.1 Evaluation

We aim to learn reward prediction functions that can identify all rewarding trajectories in a testing environment. In general, there are many non-rewarding (negative) trajectories and a few rewarding (positive) trajectories. Therefore, a better evaluation metric than the prediction accuracy is the precision-recall curve or the area under the precision-recall curve (AUC). In the test time, we enumerate all possible trajectories in testing environments and rank all the trajectories by the model's

predicted probability $p(r = 1|\tau_{\text{test}})$ that a trajectory τ_{test} match the intent. With this ranking, we can compute the precision-recall curves and AUC for a model.

RERL uses the language inference engine to find the lowest cost regular expression $\hat{\zeta}$ that accepts all positive demonstrations and rejects all negative demonstrations. RERL then score a testing trajectory τ_{test} by matching it to $\hat{\zeta}$. The score $h_{\text{RERL}}(\tau_{\text{test}}) = 1$ if τ_{test} satisfies $\hat{\zeta}$ and $h_{\text{RERL}}(\tau_{\text{test}}) = 0$ otherwise. Note that this score is exactly RERL's estimate of the probability that the trajectory matches the intent: $h_{\text{RERL}}(\tau_{\text{test}}) = p_{\hat{\zeta}}(r = 1|\tau_{\text{test}})$. RERL rank all trajectories based on the predicted score $h_{\text{RERL}}(\tau_{\text{test}})$. We compare RERL with two baseline models based on the LSTM [12]:

(a) **LSTM-LM**: Given positive and negative trajectory-reward pairs $(\tau^+, r^+), (\tau^-, r^-)$, we model the distribution of the trajectories given the rewards $p(\tau|r)$. we denote s_{-1} = START which represents the start of a sequence, and s_{m+1} = EOS which is the token of the end of sequence. The auto-regressive LSTM model parametrized by θ_s is trained to maximize the log-likelihood of the demonstrated trajectories conditioned on the rewards: $\sum_{(\tau,r)\sim\mathcal{D}} \log p_{\theta_s}(\tau|r) = \sum_{(\tau,r)\sim\mathcal{D}} \sum_{i=0}^{m_\tau} \log p_{\theta_s}(s_{i+1}|s_{-1}, s_0, ..., s_i, r)$. For a testing sequence τ_{test}, $p_{\theta_s}(r = 1|\tau_{\text{test}})$ is not obtainable from LSTM-LM. Instead LSTM-LM can output a score: $h_{\text{LSTM-LM}}(\tau_{\text{test}}) = \log p_{\theta_s}(\tau_{\text{test}}|1) - \log p_{\theta_s}(\tau_{\text{test}}|0)$. This is a valid score because it can be shown that $p_{\theta_s}(r = 1|\tau_{\text{test}})$ is strictly monotonically increasing when $h_{\text{LSTM-LM}}(\tau_{\text{test}})$ increases. LSTM-LM ranks all trajectories according to this score, and the ranking result will be the same as if LSTM-LM ranks all trajectories according to $p_{\theta_s}(r = 1|\tau_{\text{test}})$.

(b) **LSTM-CLS**: We also use a binary LSTM classifier, parameterized by θ_c, which takes as input the entire sequence τ and predicts the reward $r \in \{0, 1\}$. It scores a testing sequence with $h_{\text{LSTM-CLS}}(\tau_{\text{test}}) = p_{\theta_c}(r = 1|\tau_{\text{test}})$, which is exactly τ_{test}'s probability of matching the intent. LSTM-CLS use the score $h_{\text{LSTM-CLS}}(\tau_{\text{test}})$ to rank all trajectories.

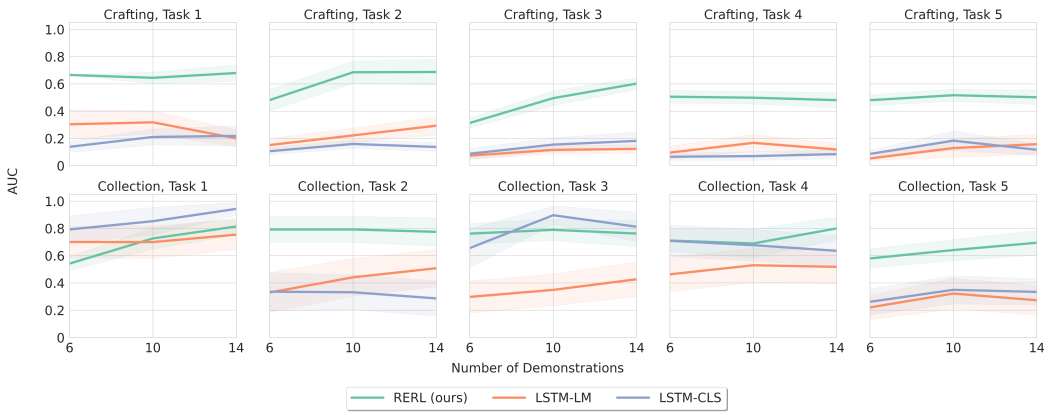


Figure 2: AUC of each model on the five tasks in each 2D environment with $6, 10, 14$ training demonstrations. RERL outperforms baseline models in most tasks.

We evaluate our method on five natural tasks (Table 3 in Appendix B) in each of the two environments described above. For each experimental run, we randomly sample one training environment and multiple testing environments with a different number of objects and layouts. During training, we randomly sample N positive and N negative demonstrations from all possible trajectories in the training environment. In the test time, we enumerate all possible trajectories in testing environments and rank all trajectories using each model as described above. More details on this section's experiments are in Appendix B.4.

Performance: Figure 2 shows the AUC for each model with $2N = 6, 10, 14$ demonstrations. Our model outperforms both baseline models in most tasks. We also observe that for most tasks, adding additional demonstrations only helps marginally, which motivates the experiments in the next section.

Data efficiency: We explore how many demonstrations are needed for the baseline models to catch up with RERL in tasks where they perform poorly. Figure 3 shows the AUC for each model when we increase the number of training demonstrations significantly. In more than half of the tasks, baseline models do not perform better than RERL, even with a double amount of RERL's training data.

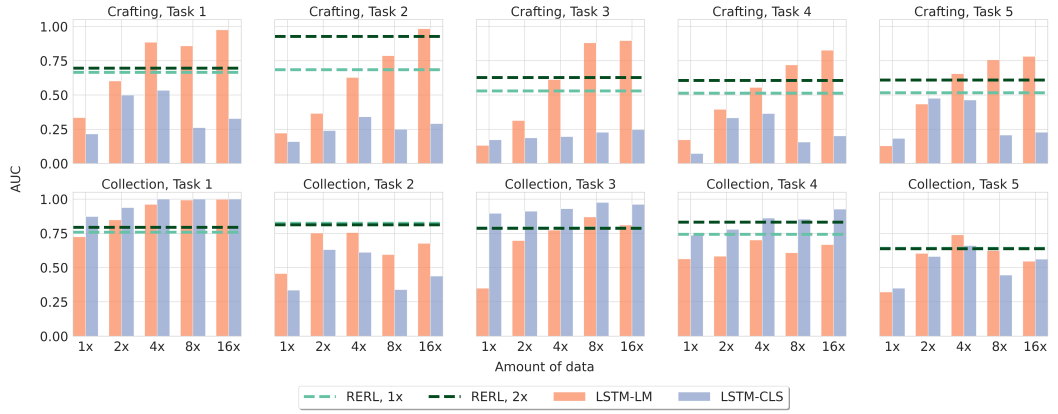


Figure 3: AUC when the LSTM models are given significantly more data on the five tasks in each 2D environment. Kx on the x-axis means that the number of demonstrations used for training is $10 \times K$. The two dashed lines are the performance of RERL with 1x data and 2x data.

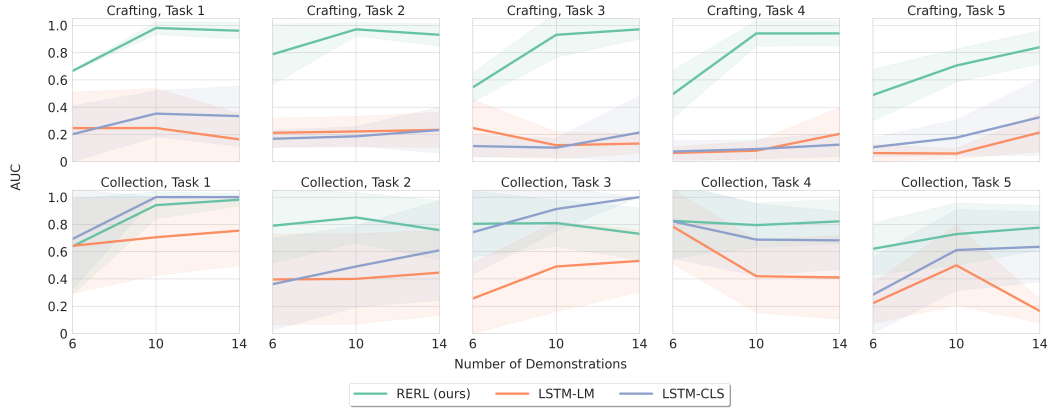


Figure 4: AUC with the interactive data acquirement procedure on the five tasks in each 2D environment. The x-axis reflects the number of training examples used in the three testing environment.

3.1.2 Interaction with the demonstrator

We hypothesize that the quality of the demonstrations is crucial for models to infer a generalizable reward. Therefore, we consider an interactive learning setting. After each evaluation in a testing environment, the agent can query the reward information of w trajectories in this environment. These trajectories will be added to the training set. The agent uses the expanded training set to re-fit the model. Then the agent is evaluated again in a new environment. More details on how we implement this are included in Appendix B.5.

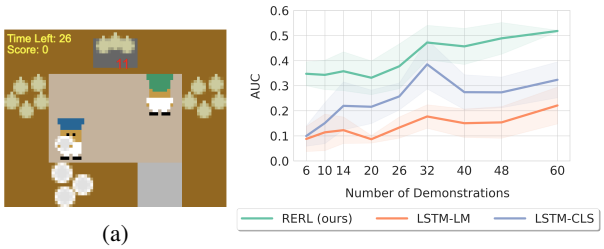
Figure 4 shows the AUC of interactive learning. For the crafting tasks, the benefit of using interactive learning is salient. While we see in Figure 2 that the performance only has a modest increase given more randomly sampled labeled trajectories, Figure 4 shows that the performance of RERL improves significantly when the trajectories are proposed and labeled. For the object collection environment where features of the objects matter, we do not observe an advantage of proposing and querying trajectories over randomly sampling trajectories, because the total number of trajectories is small and the search space of the reward function is much larger in the object collection environments.

3.2 Environment III - The Overcooked environment

Finally, to further test our method's robustness against sub-optimal demonstrations and task objective not specified by a regular expression, we evaluate on a challenging video game environment, Overcooked [13]. Here, two agents need to follow a coordinated procedure to serve a dish: (1) move

three onions (O) to the pot (P), (2) take the dish (D) to the pot to hold the food, (3) serve it at the serving station (S). We represent trajectories by the object interaction sequence for both agents. For example, an interaction sequence OPOPOPDPS means that the two agents need to move three onions to the pot three times repeatedly, then take a dish to the pot to hold the food, and finally serve it at the serving station. Two agents can collaborate on the task in different ways, and hence, there are many rewarding trajectories and sub-optimal trajectories. For instance, one agent may take the dish at the beginning, and the other agent would have to fetch all three onions (DOPOPOPPS); or agents may take more onions than needed (OPOPOPODPS), etc.

To collect positive trajectories, we use behavior-cloning (BC) to train a policy that imitates human-play data, and use the learned policy to generate positive demonstrations. As there exists noise in the human-play data, the learned policy also generates failed trajectories, which we use as negative demonstrations. We also use a random policy to generate more negative demonstrations. The training dataset is split into a training set and a testing set. To test the out-of-distribution generalization, we augment the testing set with a set of positive and negative trajectories from a heuristic game-play policy provided in the environment. Figure 5 shows that RERL consistently outperforms baseline models given different numbers of demonstrations, including the case when more than 50 demonstrations are given.



(a) (b)

Figure 5: **(a)**: The Overcooked environment. **(b)**: AUC of each model on the Overcooked task with varying number of training demonstrations. RERL outperforms LSTM models at all numbers of demonstration.

4 Related work

Inverse reinforcement learning: Most of the existing IRL approaches focus on learning Markovian rewards [1, 2, 14, 15, 3, 4] from optimal or near-optimal demonstrations. However, such IRL algorithms are usually not directly applicable to non-Markovian settings [16, 17], which makes the space of reward functions much larger and thus the inference problem much more challenging.

Symbolic hierarchical reinforcement learning: These methods [18, 19] either pre-define or learn a set of symbols with a low-level policy for each symbol, and learn a high-level policy to compose these symbols for solving a variety of long-horizon tasks, including robotic imitation, games, combinatorial problems [11, 20–24], etc. The problem setting of these methods is highly compatible with our framework, given the symbolic nature of our model.

Program synthesis: Neural program synthesis is gaining increasing attention [24–30]. Our framework can be integrated with these methods to generalize to other language patterns and improve language inference efficiency.

5 Discussion

A typical problem in applying methods that use language for inference in the real world is that they assume access to symbols. The use of symbolic representation is a two-edge sword: it allows very efficient learning at the cost of not operating directly from sensory inputs. We partially address this limitation by showing our method can work from binary features. We believe our contributions are complementary to the rapidly advancing body of work looking at inferring symbolic representations from images [31]. With advances in this direction, which are complementary to our work, we hope that our approach can be applied to even more complex problems in the future.

References

[1] Stuart Russell. Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 101–103. ACM Press, 1998. 1, 6

[2] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann, 2000. 1, 6

[3] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rkHywl-A-. 1, 6

[4] Chelsea Finn, Paul F. Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *CoRR*, abs/1611.03852, 2016. URL http://arxiv.org/abs/1611.03852. 1, 6

[5] Elizabeth S. Spelke and Katherine D. Kinzler. Core knowledge. *Developmental Science*, 10 (1):89–96, 2007. doi: 10.1111/j.1467-7687.2007.00569.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-7687.2007.00569.x. 2

[6] Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Alyosha A. Efros, and Thomas L. Griffiths. Investigating human priors for playing video games. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJov3dxDM. 2

[7] Giuseppe Vicari and Mauro Adenzato. Is recursion language-specific? evidence of recursive mechanisms in the structure of intentional action. *Consciousness and Cognition*, 26:169 – 188, 2014. ISSN 1053-8100. doi: https://doi.org/10.1016/j.concog.2014.03.010. URL http://www.sciencedirect.com/science/article/pii/S1053810014000555. 2

[8] Carl Edward Rasmussen and Zoubin Ghahramani. Occam's razor. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, page 276–282, Cambridge, MA, USA, 2000. MIT Press. 2

[9] Mina Lee, Sunbeom So, and Hakjoo Oh. Synthesizing regular expressions from examples for introductory automata assignments. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, GPCE 2016, page 70–80, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450344463. doi: 10.1145/2993236.2993244. URL https://doi.org/10.1145/2993236.2993244. 2, 9, 10

[10] John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in neural information processing systems*, pages 817–824, 2008. 2, 9

[11] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 166–175, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/andreas17a.html. 3, 6

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735. 4

[13] Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. In *Advances in Neural Information Processing Systems*, pages 5174–5185, 2019. 5

[14] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 1, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138385. doi: 10.1145/1015330.1015430. URL https://doi.org/10.1145/1015330.1015430. 6

[15] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, page 1433–1438. AAAI Press, 2008. ISBN 9781577353683. 6

[16] Tommi Jaakkola, Satinder Singh, and Michael Jordan. Reinforcement learning algorithm for partially observable markov decision problems. *Advances in Neural Information Processing Systems*, 7, 11 1999. 6

[17] Steven D. Whitehead and Long-Ji Lin. Reinforcement learning of non-markov decision processes. *Artificial Intelligence*, 73(1):271 – 306, 1995. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(94)00012-P. URL http://www.sciencedirect.com/science/article/pii/000437029400012P. Computational Research on Interaction and Agency, Part 2. 6

[18] Ronald Parr and Stuart J. Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems 10, [NIPS Conference, Denver, Colorado, USA, 1997]*, pages 1043–1049. The MIT Press, 1997. URL http://papers.nips.cc/paper/1384-reinforcement-learning-with-hierarchies-of-machines. 6

[19] David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In Rina Dechter, Michael J. Kearns, and Richard S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, pages 119–125. AAAI Press / The MIT Press, 2002. URL http://www.aaai.org/Library/AAAI/2002/aaai02-019.php. 6

[20] Thomas Pierrot, Guillaume Ligner, Scott E. Reed, Olivier Sigaud, Nicolas Perrin, Alexandre Laterre, David Kas, Karim Beguir, and Nando de Freitas. Learning compositional neural programs with recursive tree search and planning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 14646–14656, 2019. URL http://papers.nips.cc/paper/9608-learning-compositional-neural-programs-with-recursive-tree-search-and-planning. 6

[21] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 1–8. IEEE, 2018. doi: 10.1109/ICRA.2018.8460689. URL https://doi.org/10.1109/ICRA.2018.8460689.

[22] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 8565–8574. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00876. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Huang_Neural_Task_Graphs_Generalizing_to_Unseen_Tasks_From_a_Single_CVPR_2019_paper.html.

[23] De-An Huang, Danfei Xu, Yuke Zhu, Animesh Garg, Silvio Savarese, Li Fei-Fei, and Juan Carlos Niebles. Continuous relaxation of symbolic planner for one-shot imitation learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*, pages 2635–2642. IEEE, 2019. doi: 10.1109/IROS40897.2019.8967761. URL https://doi.org/10.1109/IROS40897.2019.8967761.

[24] Shao-Hua Sun, Hyeonwoo Noh, Sriram Somasundaram, and Joseph Lim. Neural program synthesis from diverse demonstration videos. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4790–4799, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/sun18a.html. 6

[25] Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=ByldLrqlx.

[26] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rJ0JwFcex.

[27] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 440–450. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1041. URL https://doi.org/10.18653/v1/P17-1041.

[28] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy I/O. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 990–998. PMLR, 2017. URL http://proceedings.mlr.press/v70/devlin17a.html.

[29] Rudy Bunel, Matthew J. Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=H1Xw62kRZ.

[30] Ashwin Kalyan, Abhishek Mohta, Alex Polozov, Dhruv Batra, Prateek Jain, and Sumit Gulwani. Neural-guided deductive search for real-time program synthesis from examples. In *6th International Conference on Learning Representations (ICLR)*, January 2018. URL https://www.microsoft.com/en-us/research/publication/neural-guided-deductive-search-real-time-program-synthesis-examples/. 6

[31] 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. *arXiv preprint arXiv:1904.12584*, 2019. 6

[32] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540. 11

[33] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018. 11

Appendix A Contextual bandit experiment

We illustrate the use of language inference in a non-sequential, repeated game setting using contextual bandits [10]. Regular expressions are used to infer how the context drives action-dependent rewards specifically. Suppose there are k arms. Each arm has an underlying regular expression (for example, `1?(01)*0?`, which means 0 and 1 alternate.) that determines the reward of pulling the arm. The action space is $\mathcal{A} = \{1, 2, \ldots, k\}$. At round t, the agent is given a context s_t, which is a binary string (for example, `010`). The agent decides an arm a_t to pull based on s_t. If s_t satisfies the underlying regular expression for arm a_t, the agent gets $+1$ reward. Otherwise, the agent gets 0 reward. The goal of the agent is to maximize the cumulative total reward over a finite horizon T. In our experiments, we take 12 regular expressions (in appendix) from [9]. Each regular expression has a few positive and negative example strings. The context set $\mathcal{S}$ is the union of all these examples. The context s_t is sampled from $\mathcal{S}$. In every experiment, we randomly sample k out of 12 regular expressions and assign one to each arm.

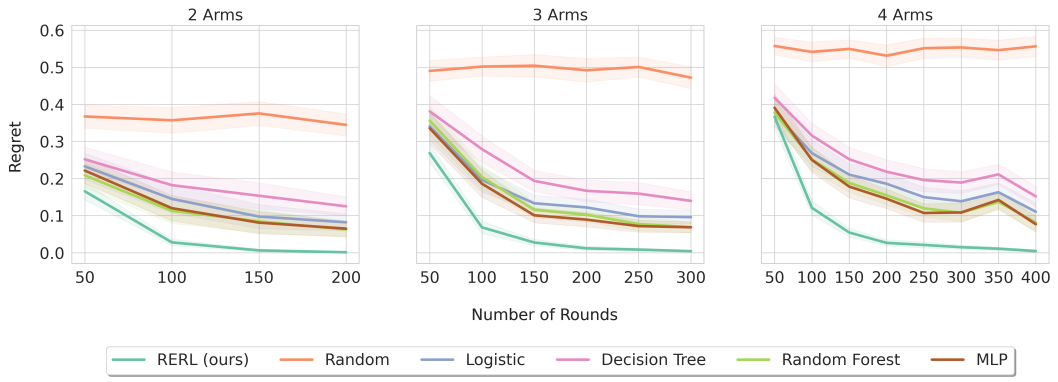


Figure 6: Regret curves of the contextual bandit problem. RERL learns the fastest and achieves the lowest regret compared to all the baselines. We report the average regret curves over 75 runs. The shaded area represents 95% confidence level.

RERL estimates the regular expression $\hat{\zeta}_{a_t}$ that satisfies all the contexts $\mathcal{D}_{a_t} = \{(s_i, r_i) | a_i = a_t, i = 0, 1, ..., t\}$ for arm a_t using the language inference engine. With the inferred regular expression for each arm, RERL can predict the reward via $r(s_j, a_j) = M(\hat{\zeta}_{a_j}, s_j)$ where $M = 1$ if s_j satisfies $\hat{\zeta}_{a_j}$ and $M = 0$ otherwise. We consider 4 binary classification models as baselines: logistic regression, decision tree, random forest, and multi-layer perceptron (MLP). Each arm a has its own classifier f_a. For all the baselines, we re-train the reward prediction models f_{a_t} by $\min_{(s_i,r_i)\in\mathcal{D}_{a_t}} \|r_i - f_{a_t}(s_i)\|_2^2$ after round t. We pad each context string s_t to the same length and use the one-hot encoding method to represent every symbol (`0`, `1`, `padding`) in the context for the models that require a fixed-length input. As a reference, we also compare all methods to a random policy. We use the ϵ-greedy exploration strategy for all the models, with $\epsilon = 0.1$. More experiment details are included in the appendix.

We use the averaged regret over a period of m rounds: $R = \frac{1}{m}\sum_{t=mn}^{m(n+1)} \left[\max_{a_t \in \mathcal{A}} r(s_t, a_t) - r(s_t, a_t)\right], n \in \mathbb{Z}^+$ to evaluate the performance of all methods. Figure 6 shows the regret curves when $m = 50$. In all experiments, RERL consistently outperforms all the baselines in all periods.

The regular expressions used in the contextual bandit problem (Table 1) are adopted from [9]. In our experiments, the regular inference module will be called for hundreds of times per rollout. Therefore we remove the regular expressions that take too long to infer (more than 10 seconds) as indicated in Table 3 of [9], so we can run a large number of rollouts in a shorter time. We also remove the regular expressions that are too easy to satisfy. For example, the regular expression `(0|1)(0|1)(0|1)*`. This expression will accept all strings other than `0` and `1`. If an arm is assigned with this expression, almost all context strings can satisfy this regular expression. So the agent can always pull this arm at any round and achieve very low regret, which makes the contextual bandit problem trivial to solve. There are also several regular expressions in [9] that use the special symbol `X`, which represents 'either 0 or 1'. We remove these expressions as they have unproportional number of examples compared to other regular expressions. After removing the expressions mentioned above from Table 3 of [9], we obtain the expressions in Table 1. The cost of regular expressions we used for this experiment is shown in Table 2. These costs are unchanged from those used in [9].

We run experiments with $k = 2, 3, 4$ arms. For a contextual bandit with k arms, we run the experiment with horizon $T = 100 \times k$.

Appendix B Non-markovian environment experiment

B.1 Ablation: non-structured tasks

The tasks described in Table 3 are structural and can be represented with relatively compact regular expressions. We want to further test the performance of RERL on non-structured tasks in the crafting environment (Table 5 in Appendix B.5) where using regular expressions to describe the tasks is

Table 1: Regular expressions used in the contextual bandit experiment.

Regex	description	#pos	#neg
`0?(1(10)?)*`	There are at least two occurences of 1.	7	7
`1?(01)*0?`	0 and 1 alternate.	9	8
`000*0(11)*`	(0^n1^m) where $n \geq 3$ and m is even.	6	5
`(1?0)*1?1?(01?)*`	Has at most one pair of consecutive 1s.	5	5
`0?((01)?10?)*`	Every pair of consecutive 0s appears before any pair of adjacent 1s.	9	9
`(1|01*01*0)*`	Number of zeros divisible by 3.	8	7
`(1|01*0)*`	Even number of zeros.	7	7
`1*0?1*0?1*`	Have at most two zeros.	8	7
`(0?1)*`	Each 0 is followed by at least one 1.	7	6
`(1|011*01)*`	Even number of 0's and each 0 is followed by at least one 1.	6	9
`0*(01?|100*)`	Contains at least one 0 and at most one 1.	6	9
`(0?1)*00(10?)*`	Has exactly one pair of consecutive 0s.	4	4

Table 2: Cost for regular expressions used in the contextual bandit experiments.

Symbol or Operator	Cost
Alphabet	20
Concatenation (e_1e_2)	cost(e_1) +cost(e_2) + 5
Or $(e_1\|e_2)$	cost(e_1) + cost(e_2) + 30
Kleene star $(e*)$	cost(e) + 20
Zero or one $(e?)$	cost(e) + 20

not efficient. Figure 7 shows that even though the performance of all the methods drop, RERL outperforms all the baseline models.

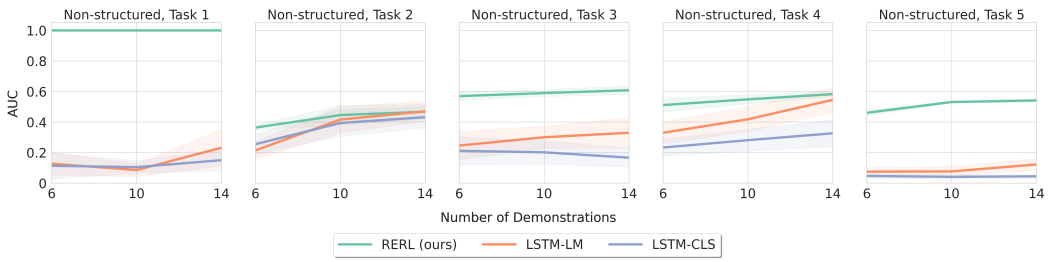


Figure 7: AUC for five tasks that do not have a regular structure in the crafting environment. RERL outperforms baseline models.

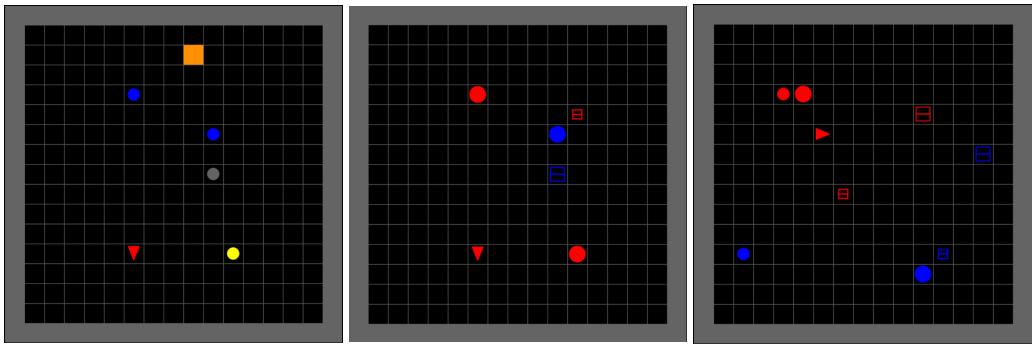
B.2 Environment implementation

We build the two 2D environments using Gym-MiniGrid [32, 33]. Each environment is a grid of size 17×17. Each grid is empty or contains at most one object. There are seven possible actions: move forward, turn left, turn right, pick up, drop off, toggle, and done. Each object in the environment has two features: color and size. We focus on the high-level demonstrator intent inference. The low-level agent movement is achieved using the A* path planning algorithm.

Environments are shown in Figure 8. The red triangle represents the agent. In the crafting environment shown in 8a, a yellow ball represents a wood, a grey ball represents an iron, and a blue ball represents

Table 3: Tasks used in the 2D environments.

Crafting	
Regular Expression	Description
`ISW*S`	iron → station → some wood → station
`(W|I)G*(W|S)`	iron or wood → some gem → wood or station
`(W|I)(W|I)SG*`	build a tool (two wood or iron → station) → some gem
`WI(G|S)W*S`	wood → iron → gem or station → some wood → station
`(WI|IW)SGG*`	build a pickaxe (wood and iron → station) → at least one gem
Featured Object Collection	
Regular Expression	Description
`(O|1)(O|1)(B|C)`	collect only one object of any feature or kind
`(O1(B|C))*`	collect any number of large blue objects
`((O|1)(O|1)B)*`	collect any number of balls
`O(O|1|B|C)*OC`	first collect a large object and collect a red box at last
`(O|1)*C(1OB)*`	first collect a box, then collect any number of small red balls



(a) (b) (c)

Figure 8: Visualization of the actual implementation of our crafting and object collection environments. **(a)**: An example of the crafting environment. **(b)**: An example of the featured object collection environment. **(c)**: All possible objects in the featured object collection environment.

a gem. The orange tile represents the crafting station. In the featured object collection environment, each object has three binary features: size (large or small), color (red or blue), and type (ball or box). Hence, in total, we have 8 possible different objects, as shown in 8c.

The environment is randomly initialized for each training and testing experiment. For the crafting environment, the station always exists, and the number of wood/iron/gem is either 1 or 2 with the same probabilities. For the featured object collection environment, we sample 5 objects uniformly at random from the 8 possible objects with replacement. All objects are placed at random locations on the map without any overlap. An example of the crafting environment is shown in Figure 8a. An example of the featured object collection environment is shown in Figure 8b.

B.3 Model details

In all our experiments, we implement the LSTM-LM model with a one-layer LSTM with 8 hidden units, and we implement the LSTM-CLS model with a bi-directional one-layer LSTM with 8 hidden units, and a fully-connected layer of 16×2 for classification. We train the LSTM model for 6000 iterations at the beginning, and 6000 more iterations every time after we acquire more data in the interactive learning setting described in the section 'Interaction with the demonstrator'. The LSTM-LM model is trained using SGD with learning rate 0.01 and momentum 0.9. The LSTM-CLS model is trained using SGD with learning rate 0.001 and momentum 0.9. The hyperparameters are

well-tuned for each model. The LSTM models are small as the amount of training data and the data dimension are small.

The cost of regular expressions we used for experiments in the non-Markovian environments is shown in Table 4. We did not find the need to tune these cost numbers; we mainly raise the cost for the '?' symbol since it is not necessary for the tasks we designed.

Table 4: Cost for regular expressions used in the non-Markovian environment experiments.

Symbol or Operator	Cost
Alphabet	10
Concatenation (e_1e_2)	cost(e_1) +cost(e_2) + 5
Or ($e_1\|e_2$)	cost(e_1) + cost(e_2) + 20
Kleene star ($e*$)	cost(e) + 20
Zero or one ($e?$)	cost(e) + 100

B.4 Experiment details

As described in Section 3.1.1, we would like to sample N positive examples and N negative examples in the training environment. However: (1) sometimes there does not exist any positive demonstration in the sampled training environment. (*e.g.*, in the featured object collection environment, for the intent 'pick any ball', it may happen that a sampled training environment does not have any ball.) In this case, we keep resampling the training environment until there is at least one positive demonstration. (2) sometimes there are $0 < N' < N$ possible positive trajectories in total in the training environment. In this case we will have the N' trajectories as positive demonstrations and sample $2N - N'$ negative demonstrations. So the total number of demonstrations is always $2N$. Note that our tasks are designed to have multiple rewarding trajectories in general, and all models always receive the same set of training demonstrations.

In our experiments, we use a relatively small number of objects in the environment so that we can enumerate all possible trajectories. For environments with more objects and features, exhaustive enumeration may be computationally expensive or even impossible. In such cases, we can train a policy that proposes good candidates with reinforcement learning algorithms by maximizing the learned reward for each model, then evaluate the reinforcement learning policy learned from each model. Note that for RERL, it is also possible to obtain a policy through a parsing module that can interpret the inferred regular expression. For LSTM-LM, it is possible to sample trajectories conditioned on the reward being 1.

For each task, we sample 8 random training environments with randomly sampled $2N$ demonstrations. We run experiments for $N = 3, 5, 7$. For each training environment, we sample 4 testing environments to evaluate the learned reward. Therefore for each task, we run 32 sampled configurations and report the averaged performance with a 95% confidence interval.

For the experiment where LSTM models take a much larger training set, It is possible that $2N$ is larger than the number of all possible trajectories in the training environment. For example, in the featured object collection environment, we may sample the same object five times, and there will be only five possible trajectories. In this case, we will use all possible trajectories as the training set for that run.

B.5 Experiment Details for the Interactive Setting

We start with $2N = 6$ training demonstrations and let each model propose and query $w = 4$ more trajectories in a testing environment. We repeat the interactive learning process twice. So the size of the training set is $6, 10, 14$, respectively, for the first, second, and third testing environment. For all models, the probability of a trajectory τ being proposed and queried by the agent is proportional to the agent's predicted reward of τ.

For all models, the probability of proposing a trajectory τ for getting the ground-truth reward is proportional to the model predicted score of τ. For LSTM-LM, we use a sigmoid function over the

Table 5: Non-structured tasks in the crafting environment used in our experiments. Regular expressions are not efficient in representing these intents.

Non-structured Tasks	
Regular Expression	Description
`WISG`	wood $\rightarrow$ iron $\rightarrow$ station $\rightarrow$ gem
`(W|I|S|G)(W|I|S|G)`	interact with 2 arbitrary objects
`WIS|WSI|IWS|ISW|SWI|SIW`	interact with wood, iron, station once, order doesn't matter
`(W|I|S)(W|I|S)`	interact with wood or iron or station twice
`WISG|GSIW|WGSI|WSIG`	4 length 4 sequences with no explicit structure

score to normalize the score to the range of 0 to 1 in this process. A model only proposes trajectories that are not already in its training demonstrations.

B.6 Experiment Details for the Overcooked Environment

For the Overcooked experiments, we use the official implementation of the environment and collect trajectories from the environment with a behavior cloning (BC) policy and a pre-defined heuristic policy, all open-sourced by the original authors. The behavior cloning policy is learned over crowd-sourced human play data. The positive trajectories are the ones that serve the dish within 50 timesteps. The model details, including hyperprameters, cost for regular expressions, etc. are all the same as those in the 2D environments.