Demonstration-efficient Inverse Reinforcement Learning in Procedurally Generated Environments – Supplementary Material

Alessandro Sestini,[1] **Alexander Kuhnle,**[2] **Andrew D. Bagdanov**[1]
[1]Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Firenze, Florence, Italy
[2]Department of Computer Science and Technology, University of Cambridge, United Kingdom
{alessandro.sestini, andrew.bagdanov}@unifi.it, alexander.kuhnle@cantab.net

1 Implementation details

In this section we give additional details on the network architectures used for DE-AIRL on the Minigrid and DeepCrawl environments.

Network Structures

Fu, Luo, and Levine use a multilayer perceptron for reward and policy models, however we use Convolutional Neural Networks (CNNs) like Tucker, Gleave, and Russell. Moreover, we use Proximal Policy Optimization (PPO) (Schulman et al. 2017) instead of Trust-Region Policy Optimization (TRPO) (Schulman et al. 2015) as in the original paper.

- **Minigrid.** The policy architecture consists of two branches. The first branch takes the global view of the 15×15 grid, and each tile is represented by a categorical value that describes the type of element in that tile. This input is fed to an embedding layer and then to a convolutional layer with 3×3 filters and 32 channels. The second branch is like the first, but receives as input the 7×7 categorical local view of what the agent sees in front of it. The outputs of the convolutional layers are flattened and concatenated together before being passed through a fully-connected layer of size 256. The last layer is a fully connected layer of size 4 that represents the probability distribution over actions.

  The reward model and the shaping term ϕ_ω have the same architecture. Unlike the policy network, they take only the global categorical map and pass it through an embedding layer, two convolutional layers with 3×3 filters and 32 channels followed by a maxpool, and then two fully-connected layers of size 32 and a final fully-connected layer with a single output. All other layers except the last one use leaky-ReLu activations.
- **Potions and Maze.** The convolutional structure of the policy of the Potions and Maze tasks are the same of Sestini, Kuhnle, and Bagdanov without the *"property module"* and the LSTM layer. The reward model takes as input only the global view, then it is followed by a convolutional layer with 1×1 filters and size 32, by two convolutional layers with 3×3 filters and 32 filters, two fully-connected layers of size 32, and a final fully-connected layer with a single output and no activation. The shaping term ϕ_ω shares the same architecture. We used leaky ReLu instead of simple ReLu as used in DCGAN (Radford, Metz, and Chintala 2016).
- **Ranged Attacks.** In this case the policy has the complete structure of Sestini, Kuhnle, and Bagdanov without LSTM, and the reward model is the same of the previous tasks with the addition of other two input branches that take as input two lists of properties of the agent and the enemy. Both are followed by embedding layers and two fully connected layers of size 32. The resulting outputs are concatenated together with the flattened result of the convolutional layer of the first branch. This vector is then passed to the same 3 fully connected layers of the potion task. The shaping term shares the same architecture.

Table 1: Hyper-parameters for all the tasks. Most of the values were chosen after many preliminary experiments made with different configurations

Parameter	Minigrid	Potions	Maze	Ranged Attack
lr_{policy}	$5e^{-5}$	$5e^{-5}$	$5e^{-6}$	$5e^{-5}$
lr_{reward}	$5e^{-6}$	$5e^{-4}$	$5e^{-4}$	$5e^{-4}$
$\text{lr}_{baseline}$	$5e^{-4}$	$5e^{-4}$	$5e^{-4}$	$5e^{-4}$
entropy coefficient	0.5	0.1	0.1	0.1
exploration rate	0.5	0.2	0.2	0.2
K	3	3	5	3
γ	0.9	0.9	0.9	0.9
max timesteps	30	20	20	20
std_{reward}	0.05	0.05	0.05	0.05

Hyperparameters

In table 1 we detail the hyperparameters used for all tasks for both policy and reward optimization.

2 Effects of modifications to AIRL

In figure 1 we give an ablation study on both SeedEnv and ProcEnv for the modifications to AIRL proposed in section 4 of the main text. The plots show how the use of both reward

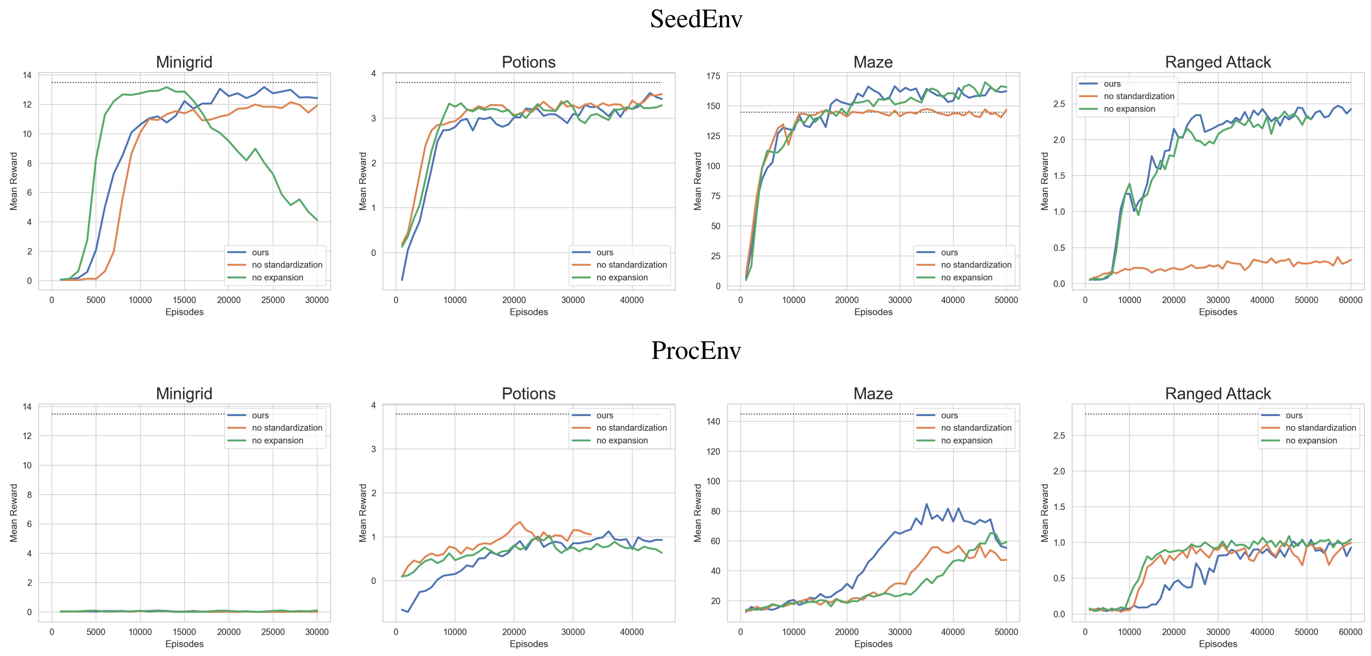

Figure 1: Ablation study of the modifications described in section 4 of the main text. The first row represents training in a SeedEnv, while the last row represents training in a ProcEnv. For all the DeepCrawl tasks we used 20 seed levels and 20 demonstrations, while for Minigrid we used 40 seed levels and 40 demonstrations.

standardization and policy dataset expansion yield more stable and better results for the majority of the tasks on both the environment types.

3 Additional experimental results

In figure 2 we summarize all the experimental results described in section 6 of the main text. Included are the performance of our demonstration-efficient AIRL for all tasks, the evolution of discriminator losses, and plots showing the importance of using a disentangling reward function.

References

Fu, J.; Luo, K.; and Levine, S. 2018. Learning Robust Rewards with Adverserial Inverse Reinforcement Learning. In *International Conference on Learning Representations*.

Radford, A.; Metz, L.; and Chintala, S. 2016. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations*.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *International Conference on Machine Learning*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* .

Sestini, A.; Kuhnle, A.; and Bagdanov, A. D. 2019. DeepCrawl: Deep Reinforcement Learning for Turn Based Strategy Games. In *Proceedings of AIIDE Workshop on Experimental AI in Games*.

Tucker, A.; Gleave, A.; and Russell, S. 2018. Inverse reinforcement learning for video games. In *Proceedings of NIPS Workshop on Deep Reinforcement Learning*.

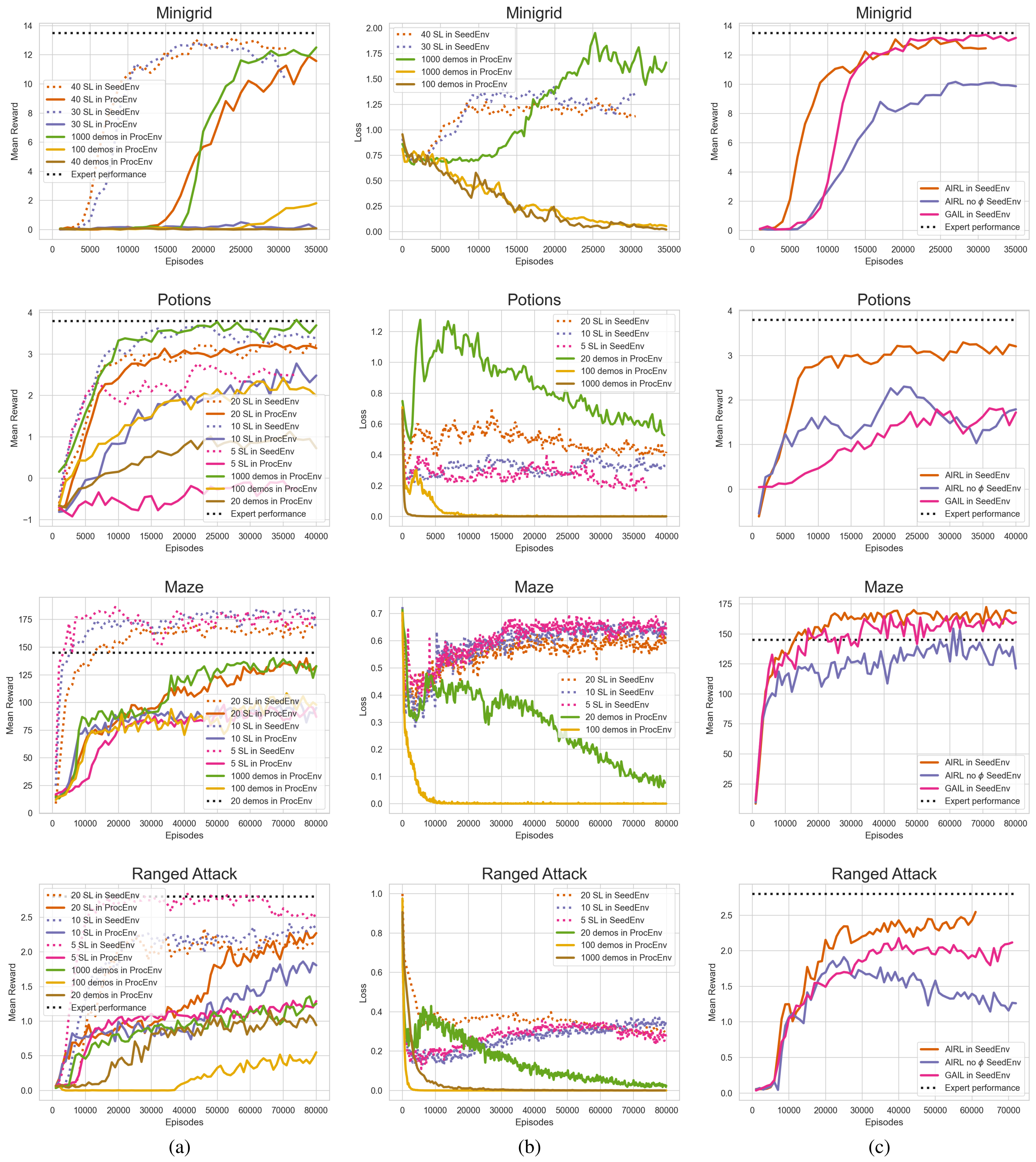
Figure 2: Summary of experimental results. Column (a): reward evolution in SeedEnv and ProcEnv with different numbers of seed levels, and naive AIRL on ProcEnv. Column (b): the evolution of the loss function. Column (c): the training of AIRL without the shaping term and GAIL, both in the SeedEnv with 20 seed levels for DeepCrawl and 40 seed levels for Minigrid. The dotted horizontal line refers to expert performance in the ProcEnv.