Reinforcement Learning reproducibility challenge

Generative Adversarial Imitation Learning

Massimiliano Nigro
massimiliano.nigro@polimi.it

Luca Subitoni
luca.subitoni@polimi.it

Reproducibility Summary

Scope of Reproducibility

The claims of the original paper [1] we are trying to reproduce are: (1) Does GAIL perform better than Behavioral Cloning across the different implemented environments? (2) Does Generative Adversarial Imitation Learning (GAIL) perform better than Behavioral Cloning across different dataset sizes?

Methodology

We initially attempted to use the authors' original code on Google Colab, but we encountered several issues. Therefore we re-implemented the pipeline described in the original paper, trying to match the original implementation as closely as we could. We trained expert policies in each environment using the Trust Region Policy Optimization (TRPO) algorithm and we focused on comparing Behavioral Cloning and GAIL performances.

The code was run on Google Colab's CPU; the use of Google Colab's GPU did not improve substantially the execution speed, likely due to the small size of the implemented neural networks. The total time budget for running the experiments was approximately 200 hours.

Results

We were generally able to reproduce the results of the paper, as GAIL generally outperformed Behavioral Cloning in Walker2D, Hopper, and CartPole, approaching expert-level performance. However, in Acrobot, MountainCar, and Reacher, GAIL's performance was inferior to Behavioral Cloning, and in Ant, it achieved 30-80% of the expert's performance but did not match Behavioral Cloning. Additionally, the impact of dataset size varied across environments; while it had little effect on GAIL's performance in Cartpole and Hopper, performance in Walker2D and Ant environments unexpectedly showed a slight decrease with larger datasets.

What was easy

We used existing implementations of TRPO, Behavioral Cloning, and GAIL. The Appendix from the original GAIL paper was detailed and contained most important hyperparameters for running the experiments.

What was difficult

Running the original authors' code on Google Colab proved unfeasible. After having re-implemented the pipeline described in the paper, the experiments required substantial computational resources. We encountered challenges locating open-source implementations of GTAL and FEM algorithms.

GitHub project repository: https://github.com/lucasubitoni/Reinforcement_Learning_Project_PhD_course

1 Introduction

In imitation learning, an agent seeks to learn the optimal policy by following a set of expert demonstrations. We focus on a specific scenario where the agent cannot query the expert for additional data during training and does not receive any reinforcement signal. A significant algorithm in this context is Generative Adversarial Imitation Learning (GAIL). GAIL leverages generative adversarial training to accurately fit the distributions of states and actions that define expert behavior. Unlike its predecessors, Behavioral Cloning and Inverse Reinforcement Learning, GAIL can efficiently learn a policy from expert demonstrations without requiring large amounts of data and is able to scale well to large environments. As a part of the assignment for the Reinforcement Learning P.h.D. course, we aim to replicate the work done by Ho. et al. [1]

2 Scope of reproducibility

In their work, Ho et al. introduced the algorithm and compared its performance to Behavioral Cloning (BC), the Feature Expectation Matching (FEM) algorithm, and the Game Theoretic Apprenticeship Learning (GTAL) algorithm across nine physics-based control tasks. These tasks ranged from low-dimensional control tasks from classic reinforcement learning literature (such as Cartpole, Acrobot, and Mountain Car) to challenging high-dimensional tasks (including Ant, HalfCheetah, Humanoid, Hopper, and Walker2D), each using various dataset sizes. In our study, we limited our comparison to the GAIL algorithm and Behavioral Cloning due to the lack of open-source implementations for the FEM and GTAL algorithms. The key questions we focused on were:

- Does GAIL perform better than Behavioral Cloning in these nine environments?
- Does GAIL perform better than Behavioral Cloning across different dataset sizes?

3 Methodology

3.1 Approach

We initially tried to use the official code repository for the Generative Adversarial Imitation Learning (GAIL) paper by Jonathan Ho and Stefano Ermon (www.github.com/openai/imitation). However, we encountered several issues when running the code on Google Colab. The original code was written in Python 2, which we installed on Colab, but it also required a specific bash command, `qsub`, to submit the multiple jobs needed for execution. Unfortunately, `qsub` was not functioning on Google Colab and multiple errors were encountered when running the scripts separately.

As a result, we decided to implement the pipeline described in the original GAIL paper from scratch, trying to match the original implementation as closely as we could. For this, we primarily relied on the following Python libraries: Stable Baselines3, Stable Baselines3 Contrib, Gymnasium, Mujoco, PyTorch, and Imitation.

The code for both the approaches described above is available on the GitHub project repository:
https://github.com/lucasubitoni/Reinforcement_Learning_Project_PhD_course

3.2 Computational resources

The code was executed on Google Colab using the CPU (Intel Xeon CPU @ 2.20GHz). Additionally, we attempted to use Google Colab's GPU (Nvidia Tesla T4); however, the speed of the code execution did not substantially improve, likely due to the small size of the implemented neural networks.

3.3 Description of the environments

The environments used to train and evaluate the Reinforcement Learning (RL) agents are: CartPole, Acrobot, MountainCar, HalfCheetah, Hopper, Walker2D, Ant, Humanoid, and Reacher. The original version of each environment is detailed in Appendix A of the GAIL paper. However, some of these environment versions have been deprecated, so we used the closest available ones. The differences (if any) between the original version of each environment and our implemented version are outlined in Table 1. Additionally, we vectorized each environment to speed up the training process.

Environment	Original version	Our version	Differences
CartPole	v0	v1	Increased `max_episode_steps` (from 200 to 500). Increased `reward_threshold` (from 195.0 to 475.0)
Acrobot	v0	v1	Provides sine and cosine of each angle instead of direct angles. Increased `max_episode_steps` (from 200 to 500).
MountainCar	v0	v0	-
HalfCheetah	v1	v2	No differences, only provides compatibility with newer MuJoCo simulator
Hopper	v1	v2	No differences, only provides compatibility with newer MuJoCo simulator
Walker2d	v1	v2	No differences, only provides compatibility with newer MuJoCo simulator
Ant	v1	v2	No differences, only provides compatibility with newer MuJoCo simulator
Humanoid	v1	v2	No differences, only provides compatibility with newer MuJoCo simulator
Reacher	v1	v2	No differences, only provides compatibility with newer MuJoCo simulator

Table 1: Comparison between the environment versions specified in the original GAIL paper and those implemented in this project, reflecting changes due to deprecation of the original versions. Additionally, a description of the differences is provided.

3.4 Training the expert policy

As specified in the original GAIL paper, we trained expert policies in each environment using the Trust Region Policy Optimization (TRPO) algorithm to generate expert trajectories and establish reference performances for comparing the imitation learning algorithms. Specifically, each expert policy network consisted in a fully connected neural network with 2 layers of 100 neurons each, separated by tanh activation functions, consistent with the architecture described in the original paper.

We trained the experts until they achieved the average reward reported in Appendix A of the original paper, employing early stopping once the target value was surpassed. In addition, a maximum threshold of 5 million iterations was imposed. For algorithms that presented significant training challenges (e.g., Mountain Car), we either sourced expert datasets from HuggingFace or relied on experts trained by ourselves with lower performance. Additional details relative to these training difficulties are discussed below.

After each expert policy network was trained, model evaluations were conducted by computing the mean reward over 50 trajectories, as specified in the original paper. During evaluation, we set the expert policy to behave deterministically (i.e., random exploration disabled), along with a γ discount factor of 1, which was not specified in the original paper.

All the training reward curves relative to our expert algorithms in each considered environment are reported in the Results section of this report (Figure 1).

3.4.1 Training difficulties

Four environments posed significant challenges in training the expert policy network:

1. **Mountain Car**: the TRPO algorithm consistently yielded a mean reward of -200, indicating no learning progress. We suppose that this issue may be attributed to the sparse reward structure of this environment, where the agent only receives a penalty of -1 for each time step it fails to reach the goal of climbing the mountain.

Consequently, if the agent never reaches the goal it never receives the necessary reinforcement signals needed for learning. In this case, we relied on expert trajectories sourced from HuggingFace.

2. **Half Cheetah**: the agent trained using the TRPO algorithm struggled with frequent flipping and became trapped in local reward maxima, preventing it from achieving satisfactory performance. We solved this issue by training the agent multiple times with different random seeds until it did not flip over and it achieved expert-level performance (i.e., average reward comparable to the one reported in the original GAIL paper).

3. **Walker2d**: the agent we trained using the TRPO algorithm struggled to match the expert performances reported in the original paper, even after trying different random seeds. However, the agent's performance was still satisfactory relative to the one reported in the original paper. Therefore, we opted to use our trained expert, despite it being slightly below par.

4. **Humanoid**: we were unable to train an expert that achieved satisfactory performance, and to the best of our knowledge, we did not find expert datasets available relative to the v2 environment version. Additionally, in the imitation learning section, the humanoid expert in the original paper was trained for significantly more iterations compared to other algorithms, which would pose additional challenges when running the imitation learning algorithms. Hence, due to time and computational constraints, we did not further explore this environment.

3.5 Imitation learning

In the original paper, the authors implemented Behavioral Cloning (BC), Feature Expectation Matching (FEM), Game-Theoretic Apprenticeship Learning (GTAL) and compared their results with their proposed algorithm, i.e., Generative Adversarial Imitation Learning (GAIL). Due to time constraints and, to the best of our knowledge, a lack of easy-to-use libraries compatible with the implemented environments, we focused solely on Behavioral Cloning and GAIL.

We trained each imitation learning algorithm using the trajectories generated by the trained expert policies and evaluated their performances against the expert policy itself and a random policy (i.e., a policy which takes a random action according to a uniform distribution in the action space). The policies trained via the imitation learning algorithms were evaluated by computing the mean reward over 50 different trajectories (as reported in the paper) using a γ discount factor of 1. During evaluation we set each policy to behave deterministically.

The performance of each policy trained via the implemented imitation learning algorithms was then measured by the rescaled mean reward achieved by that policy. This rescaling procedure normalizes the expert policy's score to 1, while the random policy's score is set to 0, allowing comparative analysis across the different environments.

Following the methodology of the original GAIL paper, each imitation learning algorithm was trained and evaluated multiple times, varying the number of expert trajectories in the training dataset for each run (as shown on the horizontal axis of Figure 2). To ensure robust evaluation, each training session with a specific number of trajectories was repeated 5-7 times in the original paper. However, due to time and computational constraints, we limited our approach to conducting 3 re-runs per set of training trajectories for each of the implemented environments.

3.5.1 Behavioral Cloning

The Behavioral Cloning (BC) algorithm follows a supervised learning approach using state-action pairs from expert demonstrations.

We replicated the architecture described in the original GAIL paper which consists of a fully connected neural network with two hidden layers, each containing 100 neurons each. These layers are then connected by a tanh activation functions. As outlined in the original paper, we partitioned each dataset into a training set and a validation set using a 70%-30% split, respectively. The batch size for the training procedure was set to 128, except for Acrobot and MountainCar for which was set to 32, due to the limited dataset size. The neural network training was was halted as soon as the loss stopped decreasing on the validation set (i.e., early stopping, no patience). The original paper did not specify the exact loss function used. Hence, we chose to use the default loss function provided by the Imitation Python library for our implementation.

3.5.2 Generative Adversarial Imitation Learning

The Generative Adversarial Imitation Learning (GAIL) algorithm uses expert trajectories to learn a policy that imitates the expert's behavior. It achieves this by training a discriminator network to distinguish between the state-action distribution (more specifically, the occupancy measure, which can be interpreted as the distribution of state-action pairs that an agent encounters when navigating the environment with a given policy) of the expert and of the generator network. Simultaneously, it trains the learner's policy to generate distributions that confuse the discriminator. When the discriminator can no longer distinguish between the data generated by the generator network and by the expert, it indicates that the generator's policy has successfully matched the expert's behavior. This adversarial process allows the GAIL algorithm to learn a policy directly from expert demonstrations without needing to first derive a reward function.

We replicated the architectures described in the original GAIL paper, which consist for both the generator and the discriminator in a fully connected neural network with two hidden layers of 100 neurons each, connected by tanh activation functions. Due to its computational intensity, we trained the models using only 1/5th of the total iterations specified in Appendix A of the original paper.

4 Results

In this section, we discuss the evaluation results of GAIL compared to Behavioral Cloning across various environments and different numbers of trajectories in the dataset.

4.1 Results across the environments

The results shown in Figure 2 and Table 1 show that GAIL consistently outperformed Behavioral Cloning in the Walker2D, Hopper, and CartPole environments, with its performance often approaching the one of the expert. Conversely, in the Acrobot, MountainCar, and Reacher environments, GAIL's performance was at or below the level of a random model, while Behavioral Cloning performed similarly to the expert. In the Ant environment, GAIL showed better results than the random model, achieving around 30-80% of the expert's performance. However, it did not reach the level of Behavioral Cloning, which was close to the expert's performance. Finally, in the HalfCheetah environment, GAIL achieved around 80-90% of the expert's performance but was still outperformed by Behavioral Cloning. Our results confirm the findings of the original GAIL paper in the Walker2D, Hopper, and Cartpole environments. However, they differ in the HalfCheetah, Reacher, Acrobot, MountainCar, and Ant environments, where GAIL was outperformed by Behavioral Cloning.

4.2 Results across the different dataset sizes

The number of trajectories in the dataset had varying impacts on GAIL's performance across different environments.

Consistent with the original paper, the number of trajectories did not impact GAIL's performance in the MountainCar, Acrobot, Cartpole, and Hopper environments. In MountainCar and Acrobot, GAIL never surpassed the performance of a random model, regardless of the number of trajectories. Similarly, in Cartpole and Hopper, GAIL's performance consistently matched the expert level across all trajectory counts.
However, in contrast to the original paper's results, GAIL's performance in the Walker2D and Ant environments decreased when the number of trajectories reached 25, falling below the performance achieved with fewer trajectories. In the HalfCheetah environment, the number of trajectories had minimal impact on GAIL's performance, similar to the original paper. In Reacher, GAIL performed worst with 11 trajectories compared to better performances with 4 and 18 trajectories.

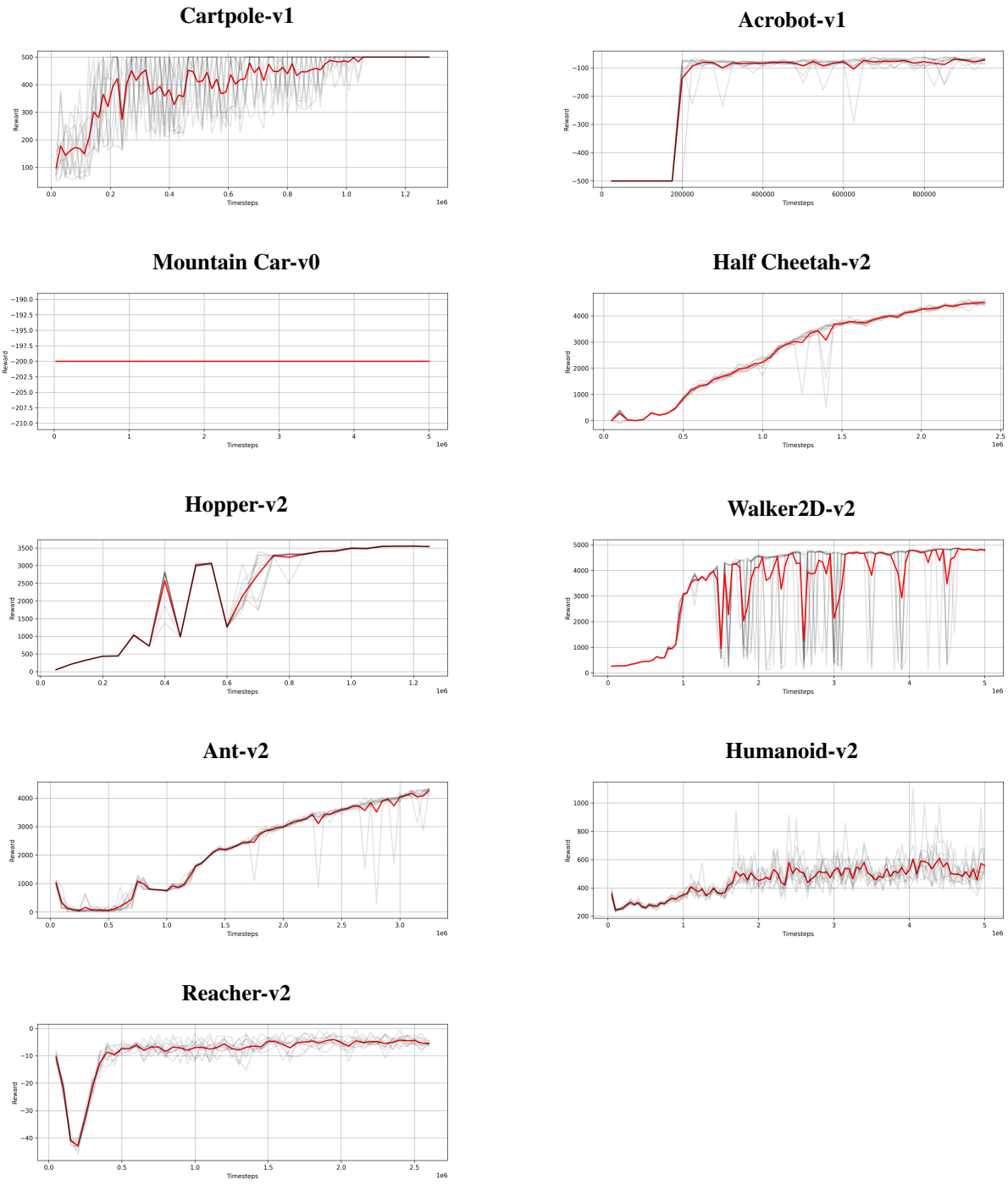

Figure 1: Expert training reward curves using Trust Region Policy Optimization (TRPO). The policy was continuously evaluated during training using 10 episodes (gray lines). The average reward across the 10 episodes is plotted in red. The maximum number of training timesteps was set to $5 \cdot 10^6$. Training was halted if the average performance was equal or greater than the one reported in the paper.

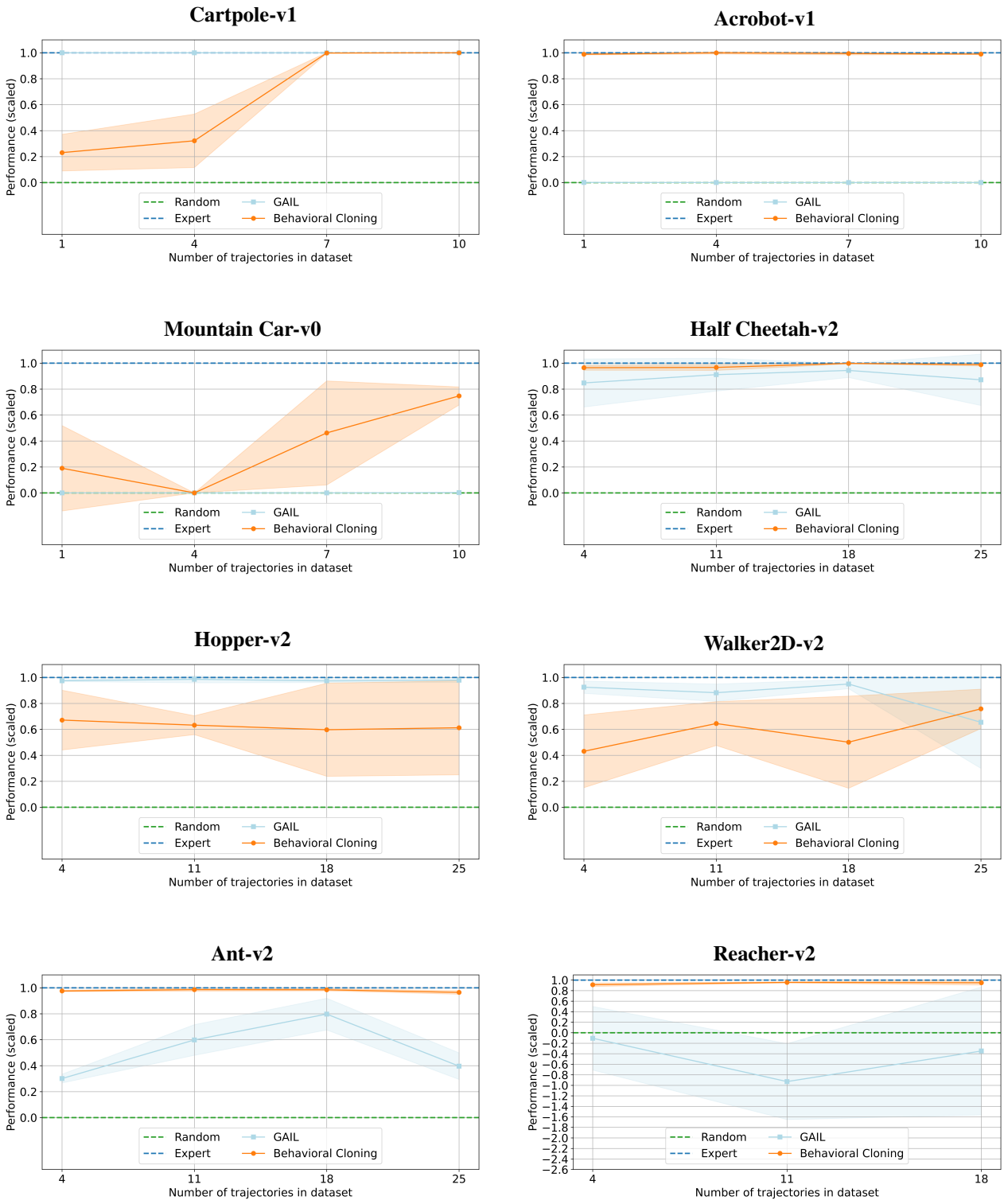

Figure 2: Results on the Classic Control and MuJoCo environments. The performance of the RL algorithms was rescaled between 0 (random policy) and 1 (expert policy). Humanoid v2 not included, as specified in Subsection 3.3.

Env.	Expert	Random	N° runs	N° trajs.	Behavioral Cloning	GAIL
Cartpole	500.0 ± 0.0	19.8 ± 1.42	3	1	130.86 ± 67.67	500.0 ± 0.0
				4	174.66 ± 98.80	500.0 ± 0.0
				7	499.12 ± 1.51	500.0 ± 0.0
				10	500.0 ± 0.0	500.0 ± 0.0
Acrobot	-76.72 ± 1.86	-500.0 ± 0.0	3	1	-81.54 ± 2.09	-499.56 ± 0.75
				4	-76.92 ± 3.66	-500.00 ± 0.00
				7	-79.14 ± 4.43	-500.00 ± 0.00
				10	-80.99 ± 1.04	-500.00 ± 0.00
Mountain Car	-98.877 ± 0.00*	-200.0 ± 0.0	3	1	-180.78 ± 33.29	-200.00 ± 0.00
				4	-200.00 ± 0.00	-200.00 ± 0.00
				7	-153.23 ± 40.55	-200.00 ± 0.00
				10	-124.38 ± 7.03	-199.77 ± 0.39
Half Cheetah	4510.47 ± 9.76	-278.31 ± 13.48	3	4	4341.45 ± 90.50	3780.26 ± 887.39
				11	4352.66 ± 109.63	4085.30 ± 605.67
				18	4501.26 ± 19.96	4241.26 ± 259.22
				25	4453.73 ± 49.01	3895.49 ± 947.04
Hopper	3616.37 ± 1.82	16.16 ± 2.03	3	4	2434.09 ± 829.27	3525.79 ± 21.04
				11	2293.38 ± 262.35	3565.22 ± 96.90
				18	2166.36 ± 1291.97	3521.46 ± 46.46
				25	2220.92 ± 1304.74	3541.90 ± 64.20
Walker2D	4811.04 ± 3.68	1.77 ± 0.77	3	4	2075.11 ± 1348.34	4450.46 ± 226.35
				11	3103.76 ± 811.85	4246.48 ± 321.66
				18	2412.50 ± 1707.60	4567.95 ± 171.64
				25	3649.00 ± 729.20	3149.38 ± 1709.03
Ant	4304.50 ± 24.88	-62.30 ± 15.62	3	4	4198.22 ± 24.22	1250.79 ± 147.13
				11	4243.65 ± 49.40	2549.85 ± 515.62
				18	4238.19 ± 35.13	3420.08 ± 532.84
				25	4150.89 ± 61.24	1668.49 ± 447.23
Humanoid	519.70 ± 7.32	126.85 ± 5.39	-	80	-	-
				160	-	-
				240	-	-
Reacher	-5.13 ± 0.18	-42.19 ± 0.65	3	4	-8.37 ± 1.22	-46.08 ± 22.39
				11	-6.72 ± 0.41	-76.62 ± 26.73
				18	-6.98 ± 1.62	-55.0 ± 45.05

Table 2: Summary results. Mean and standard deviations are computed over 50 environment simulations, as specified in the original GAIL paper. *mean reward of the downloaded trajectories reported on Hugging Face; standard deviation was not reported on Hugging Face.

5 Discussion

We validated the authors' results in the CartPole environment for both Behavioral Cloning and GAIL. In the Hopper and Walker2D environments, GAIL produced results very similar to those reported by the authors. However, we obtained substantially different results relative to the Acrobot, MountainCar, and Reacher environments.

We first emphasize that our reduced number of training iterations, due to computational and time constraints, may explain why GAIL performs worse for us when compared to the original results. Moreover, we hypothesize that the worse performance relative to the MountainCar, Acrobot and Reacher environments could be due to the limited number of state-action pairs in the trajectories of the experts' datasets. Indeed, these environments feature terminal states that the expert policies quickly reach, resulting in trajectories with fewer distinct state-action pairs. We observed a similar trend in the original GAIL paper, where imitation learning algorithms exhibited either an increased variance or a lower average performance in such environments. Furthermore, we acknowledge that in environments such as Reacher, our results show significant variance (i.e., some trials closely match expert performance while others resemble the performance of a random policy). This could be due to having conducted fewer re-runs compared to the original study, thus widening the variance. Lastly, another key difference we discovered in comparing GAIL and Behavioral Cloning is that our Behavioral Cloning algorithm performs substantially better than the version described in the original GAIL paper. This difference could be due to us using a different implementation (e.g., different loss function) relative to the one used by the authors of the original paper.

5.1 What was easy

- We were able to use pre-existing implementations of TRPO, Behavioral Cloning, and GAIL.
- The appendix included in the original GAIL paper provided valuable information, since we were able to find most of the hyperparameters for running the algorithms.

5.2 What was difficult

- We were not able to run the original authors' code on Google Colab.
- Running the experiments required significant computational time.
- To the best of our knowledge, we did not find an open-source implementation for GTAL and FEM algorithms.

References

[1] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.