

Self-Supervised Adversarial Imitation Learning

1st Juarez Monteiro¹ 

Pontifícia Universidade Católica do RS

Porto Alegre, Brazil

juarez.santos@acad.pucrs.br

2nd Nathan Gavenski¹ 

King's College London

London, England

nathan.schneider_gavenski@kcl.ac.uk


3rd Felipe Meneguzzi 

University of Aberdeen

Pontifícia Universidade Católica do RS

Aberdeen, Scotland

felipe.meneguzzi@abdn.ac.uk

4th Rodrigo C. Barros 

Pontifícia Universidade Católica do RS

Porto Alegre, Brazil

rodrigo.barros@pucrs.br

Abstract—Behavioural cloning is an imitation learning technique that teaches an agent how to behave via expert demonstrations. Recent approaches use self-supervision of fully-observable unlabelled snapshots of the states to decode state pairs into actions. However, the iterative learning scheme employed by these techniques is prone to get trapped into bad local minima. Previous work uses goal-aware strategies to solve this issue. However, this requires manual intervention to verify whether an agent has reached its goal. We address this limitation by incorporating a discriminator into the original framework, offering two key advantages and directly solving a learning problem previous work had. First, it disposes of the manual intervention requirement. Second, it helps in learning by guiding function approximation based on the state transition of the expert’s trajectories. Third, the discriminator solves a learning issue commonly present in the policy model, which is to sometimes perform a ‘no action’ within the environment until the agent finally halts.

Index Terms—Imitation Learning, Adversarial Learning, Learning from Observation, Self-Supervised Learning

I. INTRODUCTION

Learning by observing is an intrinsic human ability that we have been able to rely on since childhood. We can learn tasks by watching a video teaching us how to cook or how to play a specific video game. Learning from demonstrations allows humans to learn tasks from proficient sources and apply the newly-acquired knowledge in different domains, similar tasks, or after adapting it to their own reality, *i.e.*, different body sizes and proportions. Occasionally, learning a task by observing a specialist can be difficult. We can watch tennis players performing their best moves, but it is not a simple task for us to break down their actions into straightforward instructions to learn them properly.

In Machine Learning (ML), we refer to the technique of learning from a teacher as Imitation Learning (IL). It consists of an agent learning from the actions of a known teacher in order to solve a given task [1]. The learning agent must be able to achieve the goal or conclude the task which it was trained for. Recent approaches try to approximate the human learning experience by exploring a strategy where the agent needs no

explicit label of the actions performed by the teacher to imitate them. This strategy of learning without explicitly receiving the teacher’s actions (labels) or learning by observing is called in the literature *Learning from Observation* (LfO) [2]–[5].

LfO emerges with improvements in efficiency [6] and generalisation [4], overcoming the need for fine-grained information found in annotated trajectories or in complex reward functions. Since they are more effective, LfO methods require fewer teacher snapshots, which helps mitigate the lack-of-labels problem in the available training data. The fact that we can use LfO with smaller amounts of (unsupervised) data gives us the opportunity to explore problems where data is scarce or costly to collect, *e.g.*, autonomous vehicles [7]. The existing methods for LfO often rely on learning how to map the state-action transition in a self-supervised manner. This, in turn, requires access to the test environment and multiple deliberations from the agent, so we properly map the state-action transitions.

LfO strategies are frequently benchmarked by measuring *performance* and *efficiency* following a specific formalisation [3]–[5]. One can also evaluate imitation by comparing the trajectories that both the agent and the teacher have taken in order to solve a given task. Both perspectives have their issues. With the traditional measures, an agent can diverge from the teacher and still achieve the same reward, even though it is performing in a completely different way from what was expected. Indeed, sparse rewards in an environment make identifying proficient behaviour non-trivial. By only comparing rewards and not trajectories or intent, an agent might reach the same reward as its proficient counterpart, though perhaps missing part of the desired behaviour that it should account for. We can derive a similar example for the second perspective. Suppose the agent is acting within a maze environment, where the agent and the teacher follow practically the same trajectory, but in the end, the agent does not achieve a final state, resulting in a totally different reward.

II. RELATED WORK

Behavioural Cloning (BC) [8] is one of the most straightforward techniques for Imitation Learning [9]. BC uses teacher

¹These authors contributed equally to the work.

trajectories containing the state s_t and the action a_t of a given task at time t to create a policy $\pi = P(a | s_t)$. BC presents consistent results in the aspect of episodic rewards by training a policy in a supervised manner. However, it comes with a high cost for wide state-space scenarios, requiring sufficient trajectories to make the policy generalise for unknown states.

To solve this issue, recent approaches in IL [3]–[5], [10] employ strategies that do not require labelled data provided by a teacher. Torabi *et al.* [3] designed a LfO model-based strategy called Behavioural Cloning from Observation (BCO), which learns to imitate using a self-supervised strategy that does not require a teacher to provide annotated data. BCO starts by learning the state-action transition to build a predictor capable of guessing what action occurred in a given pair of states S_t and S_{t+1} . It then uses such a model to label the teacher trajectories. Next, all automatically-labelled data is used to train a policy in a supervised fashion. Even though the authors present better results than supervised methods that use the teacher’s original labels, the approach lacks a proper exploration technique, leading to a situation in which it repeatedly finds itself stuck in endless states.

Imitating Unknown Policies via Exploration (IUPE) [5] is an approach that uses sampling and exploration mechanisms to solve the efficiency issue. After each interaction, the inverse dynamics model \mathcal{M} learns the most likely action given a state-transition in a supervised fashion, and then it trains a policy with the pseudo-labelled (\mathcal{M} ’s outputs) as teacher’s actions. IUPE weighs the random and policy samples to create a dataset similar to the teacher’s transitions. Finally, its exploration mechanism uses the softmax distribution over its outputs to perform weighed sampling over all actions from both models. By avoiding the usage of its *maximum a posteriori* (MAP) estimation, it creates a stochastic policy that dynamically changes its exploration ratio according to each model’s output. The downside of using IUPE is its need for hand-crafted goal-aware functions, which require prior domain knowledge, to retrieve intermediate samples capable of approximating the initial random state-action pairs to the proficient ones.

Ho and Ermon [11] propose Generative Adversarial Imitation Learning (GAIL), an approach that uses adversarial training to solve that same problem. GAIL requires a smaller teacher dataset compared to other approaches. Nevertheless, the method needs extensive interactions with the environment. Torabi *et al.* [10] find inspiration on GAIL to design a Generative Adversarial Network (GAN) [12] named Generative Adversarial Imitation from Observation (GAIfO). GAIfO tries to learn a policy by creating a mechanism to distinguish if the source of the data is from a teacher or provided by the model. By training the model to understand what is the next state S_{t+1} from a given state-action pair S_t and a , GAIfO produces a policy that has similar behaviour to a teacher. Although GAIfO has yielded significantly better results than GAIL, by reducing the number of samples necessary to train a policy, it falls in the same issue as GAIL, where the number of interactions with the environment is a bottleneck.

III. PROBLEM FORMULATION

We formalise Imitation Learning assuming an environment defined by a Markov Decision Process (MDP), which is represented by a five-tuple $M = \{S, A, T, r, \gamma\}$ [13, Ch 3.], where S is the state-space, A is the action space, T is the transition model, r is the immediate reward function, and γ is the discount factor. Solving an MDP yields a stochastic policy $\pi(a|s)$ with a probability distribution over actions for an agent in state s that needs to take a given action a . *Imitation from observation* (IfO) [3] aims to learn the inverse dynamics of the agent, $\mathcal{M}_a^{s_t, s_{t+1}} = P(a|s_t, s_{t+1})$, *i.e.*, the probability distribution of each action a when the agent transitions from state s_t to s_{t+1} . While we assume the environment is an MDP, in imitation learning the agent has no access to an explicit reward signal. The actions performed by the teacher are unknown, so we want to find an imitation policy from a set of state-only demonstrations of the teacher $\mathcal{T} = \{\zeta_1, \zeta_2, \dots, \zeta_N\}$, where ζ is a state-only trajectory $\{s_0, s_1, \dots, s_N\}$.

A classic self-supervised IL approach in the LfO area focuses on using two models to learn to imitate. The models are the Inverse Dynamic Model (\mathcal{M}) and the policy model π_θ . The Inverse Dynamic Model is responsible for learning to predict $P(a | s_t, s_{t+1})$, which action caused the transition between a given pair of states (s_t, s_{t+1}) . After training the (\mathcal{M}) model, it is now possible to predict the most possible action taken by the teachers in their collected trajectories that will be used in the policy π_θ training [3], [4]. Using the pseudo-labels \hat{a} to replace the potential actions that the teachers might have taken builds a natural exploration for LfO methods which helps the agent acquire the capability of generalisation by accessing different states from the original trajectory and acquiring knowledge about state transitions [14]. A learner can iterate over this self-supervised pipeline to reduce the error coming from both \mathcal{M} and π_θ models.

LfO strategies offer a more data-efficient approach to imitation learning, since they can learn similar policies to other methods that require labelled data. Specifically, we consider policies to be similar not only in terms of similar returns, but also policies that generate similar trajectories in the state space. This is in contrast to the literature in imitation learning, which often measures similarities by looking at the returns alone for any given task [3]–[5]. However, just because an agent achieves similar returns in a given task, this does not mean the agent is actually imitating observed behaviour if the policies generate radically different trajectories.

IV. SELF-SUPERVISED ADVERSARIAL IMITATION LEARNING

In this paper, we create the Self-supervised Adversarial Imitation Learning (SAIL), an IL method that interleaves self-supervised and adversarial learning to create a policy based on observation of a teacher without any use of labelled snapshots. SAIL uses an exploration mechanism based on previous work [5], [15] to explore when it is unsure of the teacher’s actions, and a discriminator model to classify whether the policy trajectory is similar to a teachers’ one. SAIL comprises

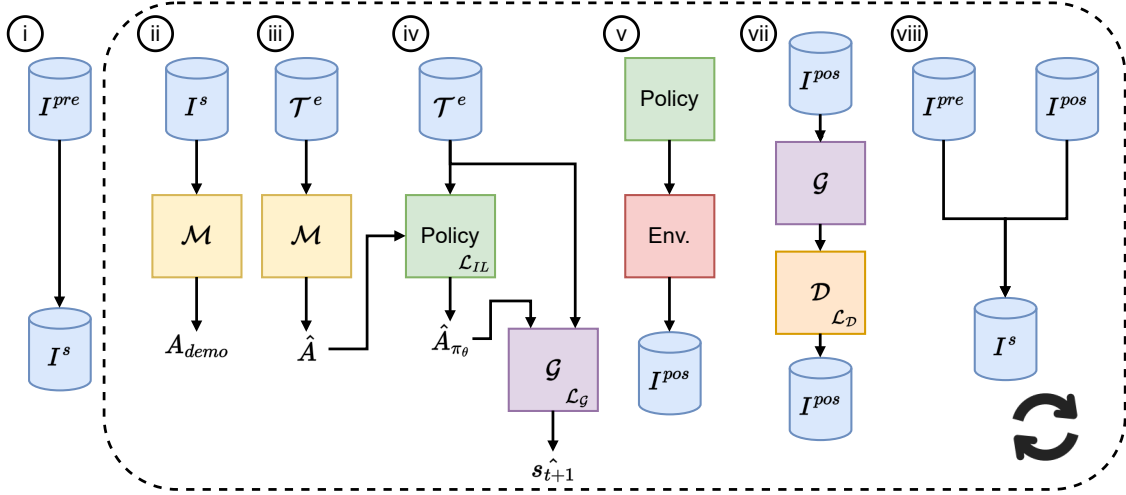


Fig. 1: Self-supervised Adversarial Imitation Learning (SAIL) training pipeline proposed in this paper. All models are initialised with random weights, and the agent interacts with the environments to collect data, so \mathcal{M} learns state transitions. Afterwards, given state-only demonstrations, the \mathcal{M} creates self-supervised labels to use with its policy in a supervised manner. The updated policy then interacts with the environment and collects new samples to be used in a new iteration by the state transition model. Unlike previous work, SAIL uses a discriminator to classify whether samples should be appended to the initial samples and a generative model to update its policy to act more like its observation samples. SAIL repeats steps (ii) from (viii) for a specific amount of epochs or until both models stop improving.

four different models: (i) \mathcal{M} , which predicts an action given a state transition $P(\hat{a} | s_t, s_{t+1})$; (ii) a policy model π_θ that uses the self-supervised labels \hat{a} to mimic a teacher, given a state $P(\hat{a} | s_t)$; (iii) a generative model \mathcal{G} conditioned by prediction from π_θ and the current state $P(s_{t+1} | s_t, \hat{a})$; and (iv) a discriminator model \mathcal{D} to discriminate π_e and π_θ , creating better samples for \mathcal{M} and updating weights θ when the policy is not similar to its proficient counterpart.

Algorithm 1 provides an overview of the learning procedure of SAIL. We first initialise all four models with random weights (Line 1-4). With π_θ randomly initialised, we can use this model as a random agent and collect all samples required (Line 5). Considering that IL methods have no knowledge about optimal or the teacher’s action distributions, these ran-

dom samples help \mathcal{M} to classify each state transition without any biases since the randomly generated ones are equally distributed. Once all samples consisting of (s_t, a, s_{t+1}) tuples are appended to I^s , SAIL trains its inverse dynamic model in a supervised manner (Function SUPERVISED in Line 7). With its updated weights θ , \mathcal{M} predicts all pseudo-labels \hat{A} to all teacher’s transitions in \mathcal{T}^e (Line 8). As the model’s weights might not be optimal in each iteration, SAIL implements an exploration mechanism that allows for \mathcal{M}_θ to deviate from its MAP by sampling from its predictions using softmax distributions of the same output as weights. This mechanism allows SAIL to dynamically explore other labels when unsure (with more uniform distributed MAP values) and exploit once its MAPS values are farther apart. After creating all self-supervised labels, we train π_θ using a behavioural cloning approach (Function BEHAVIOURALCLONING in Line 9). Unlike other behavioural cloning approaches, SAIL also uses a generative model \mathcal{G} to predict its next state, conditioned by the action predicted by π_θ . Hence, it also updates \mathcal{G} during π_θ ’s training (we further explain the learning process of the generative model in Section IV-B). Afterwards, SAIL uses π_θ to create new samples that might help \mathcal{M} better approximate the unknown ground-truth actions from \mathcal{T}^e (Line 10). Finally, SAIL appends to I^s all samples that \mathcal{D} could not differentiate between \mathcal{T}^e and $\mathcal{G}(I^{pos})$ (Line 12). We aim to discard trajectories that could result in \mathcal{M} getting stuck in bad local minima and update π_θ to correct some behaviours that \mathcal{D} uses to differentiate between teacher and student. We better explain how \mathcal{D} benefits SAIL in Section IV-A.

Algorithm 1 SAIL

- 1: Initialise model \mathcal{M}_θ as a random approximator
 - 2: Initialise policy π_θ with random weights
 - 3: Initialise generative model \mathcal{G} with random weights
 - 4: Initialise discriminator \mathcal{D} with random weights
 - 5: Generate I^s using π_θ
 - 6: **for** $i \leftarrow 1$ to epochs **do**
 - 7: Improve \mathcal{M}_θ by SUPERVISED(I^s)
 - 8: Use \mathcal{M}_θ with \mathcal{T}^e to predict \hat{A}
 - 9: Improve π_θ and \mathcal{G} by BEHAVIOURALCLONING(\mathcal{T}^e, \hat{A})
 - 10: Use π_θ to solve environments E
 - 11: Append samples $I^{pos} \leftarrow (s_t, \tilde{a}_t, s_{t+1})$
 - 12: Append $I^s \leftarrow \forall i \in I^{pos} | \mathcal{D}(\mathcal{G}(I_i^{pos})) = 1$
-

A. Goal-aware function

Developing a goal-aware function can be a complex task. Environments in the agent literature have different meanings for what a goal is. Environments typically have one of two different types of tasks: 1) maintenance; or 2) achievement [16, Chapter 2]. Environments with an achievement task define a clear end goal, such as MountainCar [17] – where an agent has to reach a flag located on top of a mountain with an accumulated reward ≥ -110 . Since agents in an IL context have no access to the reward signal, we must consider the number of steps an agent performs before reaching its objective. As environments grow in complexity, such a function will become harder to encode. By contrast, maintenance task environments usually define a set of states that an agent should not reach. For example, Ant [18], where an agent walks as far as possible without reaching angles that it classifies as ‘falling’. While others, such as CartPole [19], define a stopping criterion, *e.g.*, when its pole reaches a certain angle, and an optimal threshold, *e.g.*, maintain its task for 195 consecutive steps. Thus, encoding a goal-aware function creates a degree of unwanted complexity in a learning algorithm.

By only using samples that reach a goal, SAIL acquires examples that have ‘some’ degree of optimality, and approximates I^s samples from \mathcal{T}^e [5]. Nevertheless, classifying whether samples are close to \mathcal{T}^e might be difficult. First, defining what indicates a sample being close to a proficient teacher is hard. If we consider a stationary agent (which SAIL is), we might discard samples that allow \mathcal{M} to accurately predict transitions due to their distance to \mathcal{T}^e states alone. Therefore, to achieve better policies, SAIL needs a goal-aware function that allows for \mathcal{M} to deal with sub-optimal samples.

To remove the usage of hand-crafted goal-aware functions, SAIL uses a discriminator \mathcal{D} to discriminate between π_θ and π_e . By doing so, SAIL eliminates all human intervention and gains a non-greedy sampling mechanism by using a model to classify which agent created a trajectory. Moreover, since \mathcal{D} starts with random weights (Line 4), it allows samples that did not reach a ‘goal’, *i.e.*, sub-optimal samples, to be appended to I^s . However, considering that SAIL works under LFO constraints (not having access to π_e actions), there is a need to create a mechanism that can discriminate between teachers’ and students’ state-only trajectories. Using state-only trajectories from π_θ and π_e , and an adversarial learning approach (Equation 1), SAIL allows π_θ via \mathcal{G} to be updated from its gradient flow from \mathcal{D} .

$$\begin{aligned} \min_{\mathcal{M} \cup \pi} \max_{\mathcal{D}} \text{SAIL}(\mathcal{M}, \pi, \mathcal{G}, \mathcal{D}) = \\ \mathbb{E}_{r \sim R(\pi_e, Env)} [\log(\mathcal{D}(r))] + \\ \mathbb{E}_{r' \sim R(\pi_\theta, Env)} [\log(1 - \mathcal{D}(\mathcal{G}(r', \pi_\theta(r'))))] \end{aligned} \quad (1)$$

Considering that SAIL uses a few samples (in the form of $\mathcal{T}^e \cup \mathcal{T}^\pi$), it is important to avoid \mathcal{D} overfitting. For that, we record all π_θ trajectories in a replay buffer RB and only sample a few trajectories from both sets at each iteration, $RB_{(s_t, \dots, s_{t+n})} \sim \mathcal{T}^e \cup \mathcal{T}^\pi$. By only using a sub-set from each sample pool of

trajectories, SAIL avoids overfitting \mathcal{D} during first iterations (where π_θ ’s trajectories are considerably different).

B. Generative model

SAIL uses a generative model in two different steps. Firstly during Function BEHAVIORALCLONING; and afterwards, when selecting which samples should be appended to I^s . Although SAIL adds a new model to its pipeline, in these situations, it benefits in two folds: (i) intrinsically encodes environments physics in π_θ ; and (ii) it updates π_θ when using \mathcal{D} via its gradient flow. Which we believe far surpasses the overhead cost created by using a generative model.

The first benefit results from SAIL updating \mathcal{G} weights using Equation 2. Thus, when using the generative model in Line 9, SAIL allows \mathcal{L}_G to update π_θ to create actions that would correctly condition \mathcal{G} to generate correct state transitions and equal to those observed. Moreover, by learning how to properly decode from $(s_t, \tilde{a}_t) \mapsto s_{t+1}$, the generative model becomes a forward dynamics model, which helps π_θ encode some of the environments’ dynamics, *e.g.*, physics. We hypothesise that it is also possible only to update \mathcal{G} weights when π_θ correctly predicts a self-supervised label. However, this creates two different problems. The first problem is that not always \mathcal{M} will be correct. Thus, π_θ being correct about \hat{a} might not be indicative of how accurate it is in conditioning \mathcal{G} . The second problem originates from the fact that \mathcal{M} might stop predicting some actions when being stuck in local minima [5]. In this scenario, \mathcal{G} will not update for state transitions for the action not being predicted. Therefore, \mathcal{G} will update π_θ fewer times, resulting in less exploration, since updating π_θ weights with two different objectives can also help it to no be stuck at local minima.

$$\mathcal{L}_G = -\frac{1}{N} \left[\sum_{i=1}^N s_{i+1} \cdot \log(\mathcal{G}(s_i, \pi_\theta(s_i))) \right] \quad (2)$$

The second benefit originates from the fact that SAIL has an adversarial training mechanism in its pipeline. Hence, \mathcal{D} directly updates π_θ weights via gradient flow when it correctly discriminates between teacher and student (Equation 1). This behaviour is beneficial because updating π_θ through \mathcal{D} allows for the agent to have a direct temporal signal, *i.e.*, where it deviates from its teachers’ observations. Consequently, since SAIL maintains all original behavioural cloning techniques, it creates agents that mimic teachers’ trajectories more accurately while not losing performance.

Both generative update moments allow SAIL to have more precise trajectories (further explored in Section V), as well as more trajectories that are similar to their source (discussed in Section VI-B). We believe having trajectories closer to the teachers is beneficial since it avoids unwanted biases due to previous methods only using the performance metrics, which does not carry behaviour meaning [20].

V. EXPERIMENTAL RESULTS

We tested SAIL and all baselines described in Section II with four different environments: (i) CartPole; (ii) Mountain-

Car; (iii) Acrobot; and (iv) LunarLander. We use OpenAI Gym [21] versions for all environments. Figure ?? illustrates a single frame for each of these environments, while Section V-A gives a brief description of all environments and SAIL neural network topology.

A. Implementation and Metrics

We follow Gavenski *et al.* [5] implementation for our agents. Therefore, π_θ is a Multi-Layer Perceptron (MLP) model with 2 hidden layers with 32 neurons and 2 self-attention modules after each layer. \mathcal{M} is an MLP with 2 hidden layers with 32 neurons, 2 self-attention modules and 2 Layer Normalisation layers. \mathcal{G} is an MLP with 2 hidden layers with $2 \times (|s| + 1)$ neurons, where $|s|$ is the size of the environment state vector, and no self-attention or normalisation layers. \mathcal{D} is a Long Short Term Memory [22] with 2 layers, 32 neurons each, and dropout of 50%. The official SAIL implementation can be found at: <https://github.com/NathanGavenski/SAIL>.

To measure our experiments, we will use two main metrics: Average Episodic Reward (AER), and *Performance* (\mathcal{P}) [11] metrics. AER is the average of all accumulated rewards for a consecutive amount of tries in each environment. Performance is returned by calculating the average reward for each run scaled to be within $[0, 1]$, where zero is a behaviour compatible with a random policy (π_ε) reward, and one a behaviour compatible with the teacher (π_ε).

$$\mathbb{P} = \frac{\sum_{i=1}^E \frac{\pi_\phi(e_i) - \pi_\varepsilon(e_i)}{\pi_\varepsilon(e_i) - \pi_\varepsilon(e_i)}}{E} \quad (3)$$

It is possible for a model to achieve scores < 0 if it has the worst performance than a random policy and > 1 if the model can perform better than its teacher. We do not use accuracy for evaluation since achieving high accuracy in Imitation Learning tasks does not guarantee good results in solving a task.

We now briefly describe all environments used in this work.

- **CartPole-v1** is an environment where an agent moves a car sideways, applying force to a single pole. The goal is to prevent the pole from falling over. The space state has four dimensions: *car position*, *car velocity*, *pole angle*, and *pole velocity* at tips. The agent receives a single reward point every time the pole remains upright. Barto [19] describes solving CartPole as getting an average reward of 195 over 100 consecutive trials.
- **MountainCar-v0** environment consists of a car situated in a valley. The agent needs to learn to leverage potential energy by driving up the opposite hill until completing the goal. The state-space has two continuous attributes: *velocity* and *position* and three discrete action spaces: *left*, *neutral*, and *right*. A reward of -1 is provided for every time step until the goal position of 0.5 is reached. The first state starts in a random position with no velocity. Moore [17] defines solving MountainCar as getting an average reward of -110 over 100 consecutive trials.
- **Acrobot-v1**, based on Sutton’s work [23], is an environment where an agent has two joints and two links. The

joint between the two links is actuated. The state space consists of: $\{\cos \theta_1, \sin \theta_1, \cos \theta_2, \sin \theta_2, \theta_1, \theta_2\}$, and the action space consists of the 3 possible forces. The goal is to move the end of the lower link up to a given height. Although Acrobot is an achievement task environment, it does not have a specified reward threshold.

- **LunarLander-v2**, created by Klimov [21], is an environment where an agent needs to land on the moon under low gravity conditions. The state space is continuous, and the action space is discrete. There are four actions: *do nothing*, *move left*, *right*, and *reduce the falling velocity*. All actions have a reward of -1 , except for *do nothing* state, which is -0.3 . A positive value is returned when the agent moves in the right direction (always at 0, 0 coordinates). LunarLander-v2 is solved when the agent receives a reward of 200 over 100 constitutive trials.

B. Results

Table I shows results for all baselines and SAIL in four different environments. SAIL’s performance is closer to the teacher’s reward in almost all environments (CartPole, MountainCar and Acrobot), and it performs worst in the LunarLander environment, resulting in the best algorithm throughout all environments. When comparing SAIL to other methods, we observe it yields a lower standard deviation (≤ 1) in the first three environments, with a higher deviation for the LunarLander environment (5.63). We believe these lower deviations are due to the gradient flow from \mathcal{D} into π_θ discussed in Section IV-A, and \mathcal{G} intrinsically encoding each environment physics into π_θ without a direct signal (Section IV-B). By properly encoding physics in its policy, SAIL achieves a behaviour that helps its agent yield similar results to the teacher since it has knowledge on how s_{t+1} should be given a and s_t . Moreover, \mathcal{D} allows π_θ to have a temporal signal in its trajectory, helping to correct any unwanted/divergent behaviour that helps \mathcal{D} discriminate against teacher and student.

Conversely, SAIL does not achieve the best results for Acrobot and LunarLander. For the Acrobot environment, SAIL achieves an accumulated reward lower than IUPE (3.19 lower). However, its standard deviation is even lower than behavioural cloning, which had labelled snapshots during its training. We believe this is the case for this environment because Acrobot rewards hectic behaviour from the agent, which IUPE deeply beneficiates given its exploration mechanism, while \mathcal{D} incentives SAIL to have more consistent trajectories. We hypothesise that for cases where SAIL has a higher exploration ratio, reducing the gradient from its adversarial phase would be beneficial, avoiding it getting into an exploitative phase too soon. As for the LunarLander environment, SAIL achieves a similar result to GAIfO, which had 16.38 more accumulated reward, but with 24.32 more deviation points. We believe this is the case for SAIL due to it learning the proficient behaviour much more closely than the other IL counterparts. If we compare BC’s performance, we observe that by having the finer-grained information of all actions, the results dramatically changed – which we

TABLE I: SAIL and baselines results for all environments.

Algorithm	Metric	CartPole	MountainCar	Acrobot	LunarLander
Random	AER	21, 92±	-200 ± 0	-499.36±	-170.47±
	\mathcal{P}	0	0	0	0
Expert	AER	500 ± 0	-98.03 ± 8.17	-74.85 ± 8.61	256.79 ± 21.38
	\mathcal{P}	1	1	1	1
BC	AER	218.53 ± 160.71	-102.06 ± 4.23	-80.21 ± 3.61	63.05 ± 79.50
	\mathcal{P}	0.37	0.97	0.99	0.63
GAIL	AER	302.03 ± 158.96	-200 ± 0	-274.27 ± 116.85	120.21 ± 28.03
	\mathcal{P}	0.41	0	0.54	0.66
GAIfO	AER	500 ± 0	-200 ± 0	-128.20 ± 15.88	200 ± 29.95
	\mathcal{P}	1	0	0.85	0.86
IUPE	AER	500 ± 0	-166.97 ± 18.34	-75.65 ± 12.85	-81.34 ± 74.5
	\mathcal{P}	1	0.32	1	0.21
SAIL	AER	500 ± 0	-99.35 ± 1.84	-78.84 ± 0.41	183.62 ± 5.63
	\mathcal{P}	1	0.99	0.99	0.83

draw the comparison to SAIL’s results. For the LunarLander environment, IUPE has the worst and only negative result. We believe that such a result originates from the fact that LunarLander optimal behaviour highly correlates to the agent and goal initialisation, which IUPE lacks mechanisms to understand from its proficient source.

Finally, we observe that for environments with more relation between states (*i.e.*, carrying momentum), such as MountainCar, SAIL performs best. While most baselines yielded policies closer or equal to a random one, SAIL achieved a performance ≈ 1 . During experimentation, we observe that most other methods require the agent to be in a specific state, *i.e.*, stopped or with almost no movement force. However, we did not notice a similar behaviour since SAIL receives information from its discriminator model.

VI. DISCUSSION

In this section, we consider the following: (i) how SAIL learns with different amounts of samples; and (ii) how π_θ behaves compared to its performance and discriminator accuracy. The first case allows us to understand the trade-off between having more or fewer trajectories than those presented in Section V. Understanding this balance is essential, so SAIL learns with as fewer samples as possible and, therefore, faster. We investigate the second case to understand how much π_θ approximates from π_e trajectories. IL works use performance and AER as metrics but do not consider how similar the policy is to its teacher.

A. Sample Efficiency

We observe that SAIL inherits two different behaviours from behavioural cloning methods. Firstly, it requires a sample size bigger than a single trajectory to learn how to achieve a performance similar to its teacher. This is no surprise since 1 trajectory barely provides any information for SAIL to learn how to encode different states and generalise significantly in each environment [24]. The second behaviour SAIL inherits is it fails to scale according to the number of samples due

to compounding error [25]. As the samples grow, the policy diverges less from its observed trajectories, decreasing the agent’s performance. Therefore, behavioural cloning methods need to find the correct number of trajectories they should use. Conversely, as Table II shows, SAIL achieves a performance close to 1 with 25 episodes, only decreasing its standard deviation for each row consecutively and increasing again with 100 trajectories. We believe that SAIL achieves this result due to the updates from \mathcal{G} and, therefore, \mathcal{D} . Using a different objective, *e.g.*, recreating trajectories closer to the teachers’, SAIL results in fewer episodes needed than previous methods would because π_θ is less dependent on its LfO objective.

Nevertheless, SAIL achieves these results also due to using a smaller discriminator model, which comes with the price of being sequential. We hypothesise that to keep the sample size small, SAIL cannot use a larger sequence model, such as Transformers [26], since the high number of parameters means that the discriminator model would easily overfit on its small dataset. We believe that increasing SAIL efficiency would require further experimentation regarding the augmentation of observations since performing modification of teacher samples requires prior domain knowledge so as not to augment tuples into impossible or undesired transitions.

B. Imitation Behaviour

IL metrics usually rely only on the accumulated reward of the agent to judge how well a policy learned to mimic its teacher’s behaviour. Most works, such as this, use \mathcal{P} as a metric, which, we believe, fails to show IL agents’ intricacies (such as trajectory). By only using the reward signal to evaluate these agents, we can not conclude that the agent behaves like its teacher. There might be a proficient behaviour desired in an environment that is not encoded in the reward function or a stochastic behaviour not apparent in its observation. Therefore, \mathcal{P} may fail to measure a divergence in trajectory since the accumulated reward might be equal. Let us use a maze environment as an example, where there are always two trajectories with equal lengths, but in one path, there is a

TABLE II: Results for SAIL with different sample sizes.

Environment	Trajectories	\mathcal{P}	AER (avg)	AER (min)	AER (max)	SD
CartPole	1	0.55	1.55	86.23	457.30	± 133.98
	25	1	500	500	500	± 0
	50	1	500	500	500	± 0
	75	1	500	500	500	± 0
	100	1	500	500	500	± 0
MountainCar	1	0	-200	-200	-200	± 0
	25	0.87	-109.96	-114.40	-103.60	± 4.21
	50	0.96	-101.78	-103.70	-99.11	± 2.02
	75	0.99	-99.35	-102.10	-97.38	± 1.84
	100	0.98	-101.31	-109.90	-97.87	± 4.93
Acrobot	1	0.98	-87.47	-112.20	-77.75	± 14.03
	25	0.99	-77.95	-79.75	-76.78	± 1.16
	50	0.99	-79.33	-80.33	-78.23	± 0.96
	75	0.99	-78.84	-79.46	-78.46	± 0.41
	100	0.99	-79.36	-80.72	-76.43	± 1.73
LunarLander	1	0.31	-30.13	-76.95	55.28	± 52.28
	25	0.86	196.50	148	242.20	± 36.86
	50	0.83	133.19	-24.84	207.30	± 98.04
	75	0.83	183.62	175.20	188.90	± 5.63
	100	0.81	151.26	10.72	204.50	± 79.45

slight chance of the floor breaking. In this case, a conservative agent might never take the trajectory with the faulty floor. On the other hand, a more aggressive agent will not consider this when making its way through the maze. An imitation learning agent that learns from the conservative policy might inherit the bias of never stepping into the floor that might break. However, the performance will not measure this behaviour, even if the student becomes closer to the aggressive policy.

To avoid this issue, we analyse not only π_θ 's performance but also \mathcal{D} accuracy and \mathcal{G} error. If π_θ achieves higher performance, while \mathcal{D} has high accuracy and \mathcal{G} has a lower error, it means that although π_θ correctly encoded proficient behaviour, it has a trajectory that is not close from its teacher. Moreover, if π_θ achieves a performance close to 1, and \mathcal{D} has lower accuracy and \mathcal{G} has a higher error, it means that π_θ yields the same reward as its teacher, but its generator learned how to encode next states to 'fool' \mathcal{D} , but does not follow the observations correctly. Therefore, we are looking for a scenario where π_θ has $\mathcal{P} \approx 1$, \mathcal{D} an accuracy $\approx 50\%$ and a lower error for its \mathcal{G} during BEHAVIOURALCLONNING (Line 9 in Algorithm 1. Table III shows all three metrics results for all four environments used in this work.

We observe that for all environments, when π_θ yields its best result, \mathcal{D} has results equal to a random model, *e.g.*, it is not

TABLE III: Discriminator's accuracy (\mathcal{D}), Generator's loss (\mathcal{G}) and Policy's performance (π_θ) for each environment.

Environment	\mathcal{D} 's Accuracy (%)	\mathcal{G} 's Loss	π_θ 's Performance
CartPole	49.44 ± 1.12	0.0015	1
MountainCar	49.76 ± 0.83	0.0029	0.98
Acrobot	50.13 ± 3.21	0.8685	0.99
LunarLander	49.08 ± 1.28	0.0308	0.81

able to discriminate whether a trajectory comes from teacher or student. Additionally, when we compare \mathcal{G} error during its first learning phase, where it tries to recreate the next state from a teacher's trajectory, it has a lower error, meaning it correctly learned how to encode state transitions. Therefore, SAIL yields a policy consistent with proficient rewards while keeping a consistent trajectory with its learned source. We note that error is highly contextual to each environment state encodings. For example, Acrobot's states is a vector with 6 values, which consists of 4 values varying from $[-1, 1]$ and two values from $[-12.57, 12.57]$ and $[28.27, 28.27]$. Figure 2 displays different error margins for 5 executions of SAIL. It is possible to note that, although \mathcal{G} has a higher error rate for the Acrobot environment, its initial value (≈ 2.5) is higher due to the environment encoding characteristics. Thus, its higher error rate (when compared with all other environments),

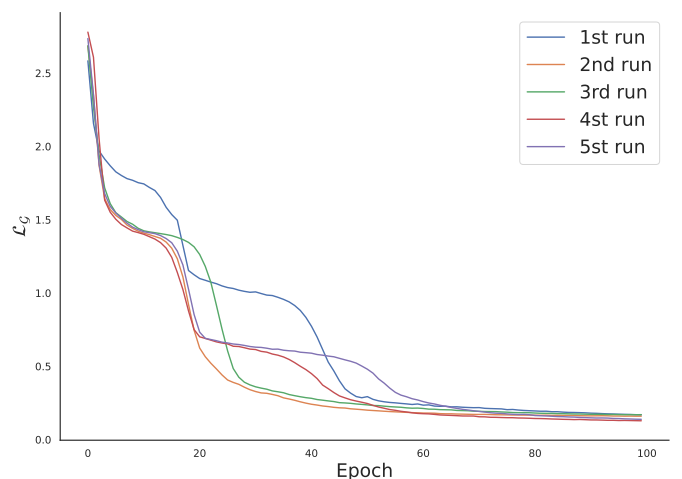


Fig. 2: Error rate for 5 different runs in Acrobot environment.

contextually, could be considered a lower margin rate.

VII. CONCLUSION

In this paper, we developed a novel LfO approach that uses an adversarial module to learn how to imitate the behaviour of teachers without accessing its actions and is capable of achieving state-of-the-art results in performance and efficiency. We evaluate our model under different amounts of sampled behaviour and compare it with various baselines from the literature. SAIL (Self-supervised Adversarial Imitation Learning) achieves significant results in both Performance and AER with fewer samples for two main reasons. First, we use an adversarial mechanism to better approximate our model's behaviour with the teacher's. Such a mechanism is connected to our end-to-end model and iteratively learns through a loss error, which makes the model achieve higher returns. Second, SAIL uses an exploration technique that helps the model collect the best data for each interaction, resulting in a model's convergence in fewer steps. Finally, we evaluate and discuss the metrics by which we can measure how much a policy actually imitates sampled behaviour. To that end, we carry out an ablation study in Section VI where we show that SAIL is capable of achieving proficient rewards while still 'fooling' its discriminator, including analysis through our obtained results.

In future work, we aim to test our method in a variety of different environments to understand better SAIL's ability to imitate other agent behaviour vis-à-vis other baselines from literature. We believe that by changing our model's topology, we can test with environments that represent their states with images, which could result in learning from teachers by collecting available videos from the internet (e.g., YouTube) since we do not need previous knowledge of actions used by the teachers. And further investigate how we can measure proficient behaviour in different environments for a better understanding of possible emergent behaviour by the agent, and its impact on real-world applications.

VIII. ACKNOWLEDGEMENTS

This work was supported by UK Research and Innovation [grant number EP/S023356/1], in the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence (www.safeandtrustedai.org) and made possible via King's Computational Research, Engineering and Technology Environment (CREATE) [27].

REFERENCES

- [1] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys*, vol. 50, no. 2, pp. 21:1–21:35, 2017.
- [2] Y. Liu, A. Gupta, P. Abbeel, and S. Levine, "Imitation from observation: Learning to imitate behaviors from raw video via context translation," in *Proceedings of ICRA 2018*, 2018, pp. 1118–1125.
- [3] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," in *Proceedings of IJCAI'18*, 2018, pp. 4950–4957.
- [4] J. Monteiro, N. Gavenski, R. Granada, F. Meneguzzi, and R. C. Barros, "Augmented behavioral cloning from observation," in *Proceedings of the 2020 International Conference on Neural Networks*, ser. IJCNN 2020, Proceedings of the 2020 International Conference on Neural Networks. IEEE, Jul 2020, pp. 1–8. [Online]. Available: <https://arxiv.org/abs/2004.13529>
- [5] N. Gavenski, J. Monteiro, R. Granada, F. Meneguzzi, and R. C. Barros, "Imitating unknown policies via exploration," *arXiv preprint arXiv:2008.05660*, 2020.
- [6] Z. Zhu, K. Lin, B. Dai, and J. Zhou, "Off-policy imitation learning from observations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 402–12 413, 2020.
- [7] L. Le Mero, D. Yi, M. Dianati, and A. Mouzakitis, "A survey on imitation learning techniques for end-to-end autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2022.
- [8] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Proceedings of the 1st Conference on Neural Information Processing Systems*, ser. NIPS 1988. Proceedings of the 1st Conference on Neural Information Processing Systems, 1988, pp. 305–313.
- [9] B. Zheng, S. Verma, J. Zhou, I. W. Tsang, and F. Chen, "Imitation learning: Progress, taxonomies and challenges," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–16, 2022.
- [10] F. Torabi, G. Warnell, and P. Stone, "Generative adversarial imitation from observation," in *I3 Workshop at ICML 2019*, 2019.
- [11] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Proceedings of NIPS 2016*, 2016, pp. 4565–4573.
- [12] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of NIPS'14*, 2014, pp. 2672–2680.
- [13] R. S. Sutton and A. G. Barto, *Reinforcement Learning*, 2nd ed. The MIT Press, 2018.
- [14] N. S. Gavenski, "Self-supervised imitation learning from observation," Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul, 2021.
- [15] R. Kidambi, J. Chang, and W. Sun, "Mobile: Model-based imitation learning from observation alone," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 28 598–28 611.
- [16] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & sons, 2009.
- [17] A. W. Moore, "Efficient memory-based learning for robot control," University of Cambridge, Tech. Rep., 1990.
- [18] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.
- [19] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, vol. 1, no. 5, pp. 834–846, Sep 1983.
- [20] N. Gavenski, J. Monteiro, A. Medronha, and R. C. Barros, "How resilient are imitation learning methods to sub-optimal experts?" in *Intelligent Systems*, J. C. Xavier-Junior and R. A. Rios, Eds. Cham: Springer International Publishing, 2022, pp. 449–463.
- [21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.
- [22] A. Graves and A. Graves, "Long short-term memory," *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.
- [23] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems*, D. Touretzky, M. Mozer, and M. Hasselmo, Eds., vol. 8. MIT Press, 1995.
- [24] H. M. Le and Y. Yue, "Imitation learning tutorial," 2018, iCML Presentation. [Online]. Available: <https://sites.google.com/view/icml2018-imitation-learning>
- [25] G. Swamy, S. Choudhury, J. A. Bagnell, and S. Wu, "Of moments and matching: A game-theoretic framework for closing the imitation gap," in *International Conference on Machine Learning*. PMLR, 2021, pp. 10 022–10 032.
- [26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [27] K. C. L. e Research team, "King's computational research, engineering and technology environment (create)," 2023. [Online]. Available: <https://doi.org/10.18742/rnvf-m076>