PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

JUAREZ MONTEIRO DOS SANTOS JÚNIOR

ADVANCES IN IMITATION LEARNING FROM OBSERVATION

Porto Alegre
2023

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

ADVANCES IN IMITATION LEARNING FROM OBSERVATION

JUAREZ MONTEIRO DOS SANTOS JÚNIOR

Doctoral Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science.

Advisor: Prof. Dr. Rodrigo Coelho Barros

Porto Alegre
2023

Ficha Catalográfica

S237a Santos Júnior, Juarez Monteiro dos

Advances in imitation learning from observation / Juarez Monteiro dos Santos Júnior. – 2023.

120 f.

Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Rodrigo Coelho Barros.

1. Imitation Learning. 2. Behavior Cloning. 3. Reinforcement Learning. 4. Machine Learning. I. Barros, Rodrigo Coelho. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Clarissa Jesinska Selbach CRB-10/2051

JUAREZ MONTEIRO DOS SANTOS JÚNIOR

ADVANCES IN IMITATION LEARNING FROM OBSERVATION

This Doctoral Thesis has been submitted in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science of the Computer Science Graduate Program, School of Technology of the Pontifical Catholic University of Rio Grande do Sul

Sanctioned on August 30th, 2023.

COMMITTEE MEMBERS:

Profª. Drª. Isabel Harb Manssour (PPGCC/PUCRS)

Prof. Dr. Gabriel de Oliveira Ramos (PPGCA/UNISINOS)

Prof. Dr. Adré Pereira Grahal (PPGC/UFRGS)

Prof. Dr. Rodrigo Coelho Barros (PPGCC/PUCRS - Advisor)

Com toda a minha gratidão, dedico este trabalho aos meus pais e à minha noiva. O amor incondicional, o apoio constante e a infinita paciência de vocês têm sido a minha fonte de energia. Vocês nunca deixaram de acreditar em mim.

“The ending isn’t any more important than any of the moments leading to it.”
(To the Moon)

ACKNOWLEDGMENTS

Gostaria de expressar a minha sincera gratidão à Pontifícia Universidade Católica do Rio Grande do Sul e a todos os envolvidos no Programa de Pós-Graduação em Ciência da Computação. Agradeço por confiarem em mim e por me proporcionarem a oportunidade de conduzir a minha pesquisa, oferecendo apoio inestimável ao longo de todo o processo acadêmico.

Quero estender meus agradecimentos ao meu orientador, Rodrigo Coelho Barros, pela sua paciência e orientação. Obrigado por compartilhar seu conhecimento e confiança ao longo destes anos. Juntos, enfrentamos uma série de desafios e momentos de aprendizado, e ao refletir sobre essa jornada, posso claramente ver a significativa evolução que essa colaboração proporcionou em minha vida acadêmica e profissional. Suas contribuições foram inestimáveis. Obrigado por tudo.

Também é de grande importância para mim expressar minha sincera gratidão a todos os professores que desempenharam um papel fundamental em minha formação como estudante de doutorado. Em particular, quero destacar o professor Felipe Rech Meneguzzi, cujo apoio e profundo conhecimento acadêmico sempre estiveram disponíveis para mim. Sua constante disposição em oferecer suporte e apoio contribuiu significativamente para a minha jornada acadêmica.

Não posso deixar de agradecer aos colegas do grupo de pesquisa *Machine Learning Theory and Applications Research Group* (MALTA), bem como meu amigo e parceiro de pesquisa, Nathan Schneider Gavenski. Seu apoio constante e amizade foram essenciais para que eu enxergasse a capacidade de concluir este capítulo da minha vida. Espero que nossa colaboração na pesquisa continue a trazer valiosas contribuições para a ciência.

Quero expressar minha profunda gratidão aos meus pais pelo amor incondicional e pela confiança constante que sempre depositaram em mim. Sem dúvida, esses fatores foram fundamentais para que eu alcançasse este marco.

Por último, mas definitivamente não menos importante, desejo agradecer à minha noiva, Raquel Alves Marini. Juntos, enfrentamos desafios que incluíram a busca por uma bolsa acadêmica no exterior e a jornada dessa durante uma pandemia global. Agradeço por ser a pessoa que esteve ao meu lado em cada momento. Seu poio foi inestimável, e me considero imensamente grato e afortunado por tê-la ao meu lado. Obrigado por tudo.

AVANÇOS EM APRENDIZADO DE IMITAÇÃO POR OBSERVAÇÃO

RESUMO

A Imitação por Observação, técnica computacional destinada ao ensino de agentes por meio da observação de demonstrações de especialistas, enfrenta desafios significativos como baixo desempenho, problemas com mínimos locais e exploração ineficaz do espaço de estados. Apesar das recentes abordagens empregarem dados não rotulados para decodificar informações de maneira auto-supervisionada, persistem os desafios a serem superados. Em resposta a tais desafios, a presente tese introduz quatro novos métodos destinados à imitação por observação. Ainda, apresenta um estudo aprofundado sobre a resiliência dos métodos de aprendizado por imitação, proporcionando uma melhor compreensão de seu desempenho e robustez em diversos contextos. As contribuições dos métodos propostos são evidenciadas pelos resultados positivos alcançados. Foi verificado que o uso de um mecanismo de amostragem pode aperfeiçoar os ciclos iterativos de aprendizado, tornando-os mais equilibrados. A inclusão de um mecanismo de exploração revelou potencial para exceder o desempenho de especialistas e estabelecer novos patamares na área. Além disso, o emprego de mecanismos de aprendizado por reforço e de aprendizado adversário mostrou-se capaz de gerar políticas mais eficientes, obtendo resultados significativos com menos amostras. As estratégias propostas melhoraram o desempenho e a eficiência dos atuais métodos, ao mesmo tempo que minimizam a complexidade da aquisição de dados de especialistas.

Palavras-Chave: Aprendizado por imitação, Aprendizado por observação, Aprendizado profundo.

ADVANCES IN IMITATION LEARNING FROM OBSERVATION

ABSTRACT

Imitation from Observation, a computational technique that instructs agents by observing expert demonstrations, suffers from considerable hurdles such as sub-optimal performance, local minima issues, and ineffective state-space exploration. Although recent strategies leverage unlabeled data to decode information self-supervisedly, persistent challenges remain. This thesis presents four novel methods for imitation learning from observation in response to those challenges. Furthermore, a comprehensive study on the resilience of imitation learning methods is provided to enable a nuanced comprehension of their robustness and performance across various scenarios. The achieved positive outcomes substantiate the merits of the proposed methods. A sampling mechanism is shown to enhance iterative learning cycles, rendering them more balanced. Integrating an exploration mechanism shows potential to surpass expert performance, establishing state-of-the-art results in the field. Moreover, the employment of reinforcement and adversarial learning mechanisms demonstrate their ability to forge more efficient policies, accomplishing good results with fewer samples. The proposed strategies boost performance and efficiency while minimizing the complexity of acquiring expert data.

Keywords: Imitation Learning, Behavioral Cloning, Self-supervised learning.

LIST OF FIGURES

Figure 3.1 – Representation of the Imitation Model (Policy) and the Discriminator. Image adapted from the work proposed by Torabi *et al.* [48]. 42

Figure 4.1 – Visual representation of the Acrobot environment. 48

Figure 4.2 – Visual representation of the CartPole environment. 48

Figure 4.3 – Visual representation of the Gym-Maze environment. 49

Figure 4.4 – Visual representation of the LunarLander environment. 49

Figure 4.5 – Visual representation of the MountainCar environment. 50

Figure 5.1 – Expert demonstrations of a 5×5 Gym-Maze configuration. Below each state image, we indicate the number of expert visits. The blue line depicts the path chosen by our ABCO agent. 59

Figure 5.2 – IDM predictions of the expert examples through time. 60

Figure 5.3 – *L2* distance for the average of each action for each iteration normalized by the expert and random samples in the 5×5 mazes. 60

Figure 6.1 – Heatmap visualization of the gradient filters activating for the maze environment. The first row shows the input image, while the second row shows the gradient activation. 68

Figure 6.2 – (a) IDM predictions for the expert examples through time. (b) Percentage of choices in which the MAP estimation is not selected by the self-decaying exploration rate. 69

Figure 7.1 – Results from multiple environments, using five different experts per environment and four algorithms. The x-axis represents the expert reward, while the y-axis represents the cumulative reward over a hundred runs for each agent. The lines on each graph show the patterns of the expert and one or more algorithms, highlighting their distinct behaviors. 79

Figure 8.1 – Combined Reinforcement and Imitation Learning (CRIL) framework. . 83

Figure 8.2 – Visualization of the MountainCar-v0 environment. Each figure illustrates the *maximum a posteriori* probabilities in a 3D mesh. 89

Figure 8.3 – Comparison between policies trained in the MountainCar-v0 environment. We only color the tiles that have the same action as π^* for easier visualization. 90

Figure 8.4 – Boxplot of the *Average Episodic Reward* for all methods and environments. 93

Figure 9.1 – Self-Supervised Adversarial Imitation Learning (SAIL) training pipeline. 98

Figure 9.2 – Error rate obtained from SAIL for 5 different runs in the Acrobot environment. 107
Figure 10.1 – Contributions during my PhD journey. 113

LIST OF TABLES

Table 3.1 – List of related works, as well as the models, metrics and environments used. 44

Table 4.1 – Information regarding the five selected environments for imitation learning. 47

Table 5.1 – Comparison of Performance and Average Episodic Reward between ABCO (our approach) and related work. 57

Table 5.2 – Ablation study evaluating the impact of the *attention* and *sampling* modules in the 5×5 Maze environment. 61

Table 6.1 – *Performance* and *Average Episode Reward* for IUPE and related work. 66

Table 6.2 – Ablation study considering IUPE's 3 main components in the maze environment. 67

Table 7.1 – Average Episodic Reward (AER) for experts with decreasing quality. Expert^1 is the optimal expert, while Expert^5 is the worst-performing expert in our experimental analyses. 74

Table 7.2 – Performance ($\mathcal{P}$) and Average Episodic Reward (AER) for all algorithms on all environments using the optimal expert. 75

Table 7.3 – Performance ($\mathcal{P}$) and Average Episodic Reward (AER) for all algorithms on all environments using Expert^2. 76

Table 7.4 – Performance ($\mathcal{P}$) and Average Episodic Reward (AER) for all algorithms on all environments using Expert^3. 76

Table 7.5 – Performance ($\mathcal{P}$) and Average Episodic Reward (AER) for all algorithms on all environments using Expert^4. 77

Table 7.6 – Performance ($\mathcal{P}$) and Average Episodic Reward (AER) for all algorithms on all environments using Expert^5. 78

Table 8.1 – KL Divergence from all three models, when compared to an optimal policy (π^*). 89

Table 8.2 – *Performance* ($\mathcal{P}$) and *Average Episode Reward* (AER) for each IL methods with only one expert's trajectory as data. 91

Table 8.3 – The average number of timesteps required by each algorithm to reach the maximum reward in each environment. 92

Table 8.4 – Quantitative results for all RL and IL algorithms used in this study as baselines. We also display the average Performance of all environments. DQN^1 is the unmodified DQN architecture [29], while DQN^2 is the version from Schaul *et al.* [39]. 93

Table 9.1 – SAIL and baselines results for all environments. 102
Table 9.2 – Results for SAIL with different sample sizes. 104
Table 9.3 – Results obtained from SAIL's modules, presenting the accuracy of the Discriminator ($\mathcal{D}$), the loss of the Generator ($\mathcal{G}$), and the performance of the Policy (π_θ) across different environments. 105

LIST OF ACRONYMS

ABCO – Augmented Behavioral Cloning from Observation

AER – Average Episodic Reward

BC – Behavioral Cloning

BCO – Behavioral Cloning from Observation

CNN – Convolutional Neural Network

CRIL – Combined Reinforcement and Imitation Learning

DQN – Deep Q-Network

GAIFO – Generative Adversarial Imitation from Observation

GAIL – Generative Adversarial Imitation Learning

IDM – Inverse Dynamics Model

IL – Imitation Learning

ILPO – Imitating Latent Policies from Observation

IFO – Imitation from Observation

IUPE – Imitating Unknown Policies via Exploration

LFD – Learning from Demonstration

LSTM – Long Short-Term Memory

MAP – Maximum a Posteriori

MDP – Markov Decision Process

ML – Machine Learning

MLP – Multilayer Perceptron

P – Performance

PM – Policy Model

SA – Self-Attention

TRPO – Trust Region Policy Optimization

CONTENTS

1 INTRODUCTION 27
1.1 DOCUMENT ORGANIZATION 28

2 BACKGROUND 29
2.1 REINFORCEMENT LEARNING 29
2.1.1 POLICY LEARNING 30
2.1.2 EXPLORATION VS. EXPLOITATION 31
2.2 IMITATION LEARNING 31
2.2.1 BEHAVIOR CLONING 32
2.2.2 IMITATION LEARNING FROM OBSERVATION 33
2.3 ADVERSARIAL LEARNING 35
2.4 TYPES OF ENVIRONMENT 35

3 RELATED WORK 37
3.1 BEHAVIOR CLONING FROM OBSERVATION 37
3.1.1 INVERSE DYNAMICS MODEL 38
3.1.2 POLICY MODEL 38
3.1.3 ITERATIVE MODEL 39
3.2 IMITATING LATENT POLICIES FROM OBSERVATION 39
3.2.1 LATENT FORWARD DYNAMICS 40
3.2.2 ACTION REMAPPING 40
3.3 GENERATIVE ADVERSARIAL IMITATION LEARNING 41
3.4 GENERATIVE ADVERSARIAL IMITATION FROM OBSERVATION 42
3.5 FINAL REMARKLS 43

4 METHODOLOGY 45
4.1 HYPOTHESIS 45
4.2 GOALS 45
4.3 METRICS 46
4.4 ENVIRONMENTS 47

5 AUGMENTED BEHAVIOR CLONING FROM OBSERVATION 51
5.1 INVERSE DYNAMICS MODEL AND POLICY MODEL 51

5.1.1 POLICY MODEL 52
5.1.2 ITERATIVE BEHAVIORAL CLONING FROM OBSERVATION 53
5.2 SAMPLING METHOD 54
5.3 SELF-ATTENTION 55
5.4 IMPLEMENTATION AND RESULTS 56
5.4.1 IMPLEMENTATION 56
5.4.2 RESULTS 57
5.5 DISCUSSION 58
5.5.1 ABCO AND SELF-ATTENTION 58
5.5.2 ABCO AND SAMPLING 59
5.6 FINAL REMARKS 61

6 IMITATING UNKNOWN POLICIES VIA EXPLORATION 63
6.1 SAMPLING METHOD 63
6.2 EXPLORATION 64
6.3 EXPERIMENTAL RESULTS 65
6.3.1 RESULTS 65
6.4 DISCUSSION 67
6.4.1 SELF-ATTENTION 67
6.4.2 SAMPLING 68
6.4.3 EXPLORATION OVER MAXIMIZATION 69
6.5 FINAL REMARKS 70

7 RESILIENCE OVER SUB-OPTIMAL SAMPLES 73
7.1 EXPERIMENTAL DESIGN 73
7.2 RESULTS 74
7.3 DISCUSSION 79
7.4 FINAL REMARKS 81

8 COMBINED REINFORCEMENT AND IMITATION LEARNING 83
8.1 REINFORCEMENT AND IMITATION LEARNING 83
8.1.1 SELF-SUPERVISED IMITATION LEARNING 84
8.1.2 EXPLORATION WITH NEURAL NETWORKS AND q-VALUES 84
8.2 COMBINING IMITATION AND REINFORCEMENT LEARNING 85
8.3 EXPERIMENTAL RESULTS 87
8.3.1 POLICY OPTIMIZATION BEHAVIOR 88

8.3.2 REINFORCEMENT LEARNING 91
8.3.3 QUANTITATIVE RESULTS 92
8.3.4 FINAL REMARKS 95

9 SELF-SUPERVISED ADVERSARIAL IMITATION LEARNING 97
9.1 ADVERSARIAL APPROACH 97
9.1.1 GOAL-AWARE FUNCTION 99
9.1.2 GENERATIVE MODEL 100
9.2 EXPERIMENTAL RESULTS 102
9.2.1 RESULTS 103
9.3 DISCUSSION 104
9.3.1 SAMPLE EFFICIENCY 105
9.3.2 IMITATION BEHAVIOR 106
9.4 FINAL REMARKS 107

10 CONCLUSION 109
10.1 LIMITATIONS 110
10.2 FUTURE WORK 111
10.3 PUBLISHED WORK 112
10.4 ON GOING WORK 113

REFERENCES 115

1. INTRODUCTION

Human nature has evolved various forms of comprehension and learning, including the capacity to learn through observation. Individuals have leveraged this intrinsic ability since their childhood. In the early stages, we can learn simple gestures by visually observing and mimicking others, even without direct access to the actions and intentions that led to that movements [36]. As individuals grow and mature, they can learn a wide range of skills from various sources, such as dancing, cooking, and painting, by simply observing and emulating a teacher in the field. However, there are instances where the process of acquiring skills by observation can pose challenges. For example, watching soccer players execute complex movements can be insufficient in helping an individual break down the actions into understandable steps for learning. Clegg and Legare [8] address this problem in "Instrumental and Conventional Interpretations of Behavior are Associated with Distinct Outcomes in Early Childhood", where they experiment with varying complexities and methods of observation.

Imitation Learning (IL), also known as Learning from Demonstration (LfD), is a Machine Learning (ML) technique addressed both by supervised and reinforcement learning perspectives. Its objective is to model how other entities or agents interact with their environment or solve a specific task, thereby emulating their actions. These teachers, as they are often referred to, provide practical examples for the learning algorithm [35]. This method involves representing the teacher's behavior as a series of demonstrations comprising state-action pairs that showcase how the teacher performs a given task across various situations. The learning entity, often called an agent, is then trained to mimic the teacher's strategies or approaches. This is done by establishing a connection between various scenarios and their corresponding actions, generating behavior that closely resembles that of the teacher in similar circumstances [18] Recent research has focused on approximating the scope of human learning through Imitation from Observation (IfO) [47, 48]. This learning strategy enables agents to learn by observing the teacher without direct access to its actions, the so-called labels. Such a strategy has gained popularity in the literature and has been the subject of several studies [27, 47].

While both LfD and IfO involve learning from an expert's behavior, IfO is particularly useful in scenarios where it is difficult or expensive to demonstrate the expert's actions. For example, having a human operator directly demonstrate a task to the robot in human-robot interaction may be impractical. Instead, the robot can observe and learn from human behavior through IfO. However, IfO brings specific challenges that can be detrimental to the imitation process. These include uncertainty in observed situations, the complexity of the examples used for learning, and situations where only partial information is available, among others [47].

This thesis presents five distinct contributions to the field of Imitation from Observation that address these challenges, including the introduction of four novel methods. First, we improve the Behavioral Cloning from Observation (BCO) framework [47] by introducing a unique sample selection strategy and a self-attention mechanism. Subsequently, drawing upon the improvements from the first method, we propose a second approach that incorporates an exploration strategy to boost performance even further. Third, we investigate the resilience of IfO with limited samples. Fourth, we combine reinforcement and imitation learning to improve sample efficiency. Finally, we introduce adversarial learning to guide the function approximation process based on the teacher's trajectories. These contributions improve sample quality, exploration, and diversity of trajectories while enhancing the agent's ability to imitate expert behavior.

1.1 Document Organization

This thesis is organized and structured into ten chapters: Chapters 2 and 3 provide an introduction to the learning paradigms and current methods in imitation learning. Chapter 4 outlines the methodology used in this work. Chapters 5 to 9 present and analyze the proposed methods and contributions for the Imitation Learning area. Finally, Chapter 10 concludes this thesis by summarizing the proposed methods, discussing their limitations, and suggesting directions for future research.

2. BACKGROUND

This chapter briefly introduces the concepts and techniques that form the foundation of this thesis. The chapter is organized into three sections, each of which covers a key area of this work: Reinforcement Learning (RL), Imitation Learning (IL), and types of environments.

The first section introduces RL, a paradigm of machine learning in which an agent learns to make decisions in an environment by interacting with it and receiving feedback via a reward function that can be positive or negative. We discuss how RL works, its formulation as a Markov Decision Process, and the key phases of the RL process: exploration and exploitation.

In Section 2.2, we introduce IL, a technique that enables an agent to learn from expert demonstrations. We discuss two common forms of IL: behavior cloning and imitation learning from observation. We also explore how IL can be used in conjunction with RL and Adversarial Learning to improve the performance of its algorithms.

Finally, in the third section, we discuss the different types of environments with which an agent can interact in RL and IL. We cover the following types of environments: deterministic, stochastic, episodic, and continuous. We also discuss how these types of environments affect the performance of RL and IL algorithms.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a paradigm of machine learning in which an agent learns to make decisions in an environment by interacting with it and receiving feedback in the form of rewards or punishments [45]. RL aims to take actions that maximize a reward signal over time. The agent achieves this by selecting actions based on a policy that maps states to actions and maximizes the expected cumulative reward, also known as the return.

The RL problem is commonly formulated as a Markov Decision Process (MDP) [45], a mathematical framework used to model sequential decision-making problems. An MDP specifies a set of states, actions, transition probabilities, rewards, and discount factors. The agent interacts with the environment by transitioning it from one state to another according to the transition probabilities. The agent receives a reward from the environment for each transition, which is used to update its policy and value estimates.

There are two main phases during the RL process: exploration and exploitation. In the exploration phase, the agent interacts with the environment to acquire information about the environment's dynamics and to determine the optimal actions to take in each state. Once

the exploration phase is complete, the agent transitions to the exploitation phase, utilizing its learned policy to take actions that maximize its expected cumulative reward.

RL practitioners have successfully applied the approach to various domains, including game playing [30], robotics [26], and finance [33]. One of the main strengths of RL is its capability to learn from feedback without the requirement of explicit supervision, making it a powerful tool for autonomous decision-making in complex and dynamic environments.

In this thesis, we will use RL as a framework for studying Imitation Learning from Observation (IfO). Specifically, we will investigate how RL can be used to learn from expert demonstrations without access to their actions, using only the states of the environment. We will explore how different types of environments (deterministic, stochastic, episodic, and continuous) impact the performance of RL algorithms in IfO settings. Furthermore, we will propose new algorithms that incorporate additional information, such as semantic labels or human feedback, to improve the performance of RL in IfO settings.

2.1.1 Policy Learning

Policy optimization constitutes a fundamental component within the reinforcement learning paradigm [45], where the overall objective is to enable an autonomous agent to learn how to select actions that augment the expected aggregate reward over a temporal continuum. Conceptually, the policy is a function that maps from states to actions, thus delineating the agent's interaction behavior within its operational environment. Consequently, the crux of policy optimization lies in the development of an optimal policy capable of augmenting the expected cumulative reward.

Policy optimization bifurcates into two primary categories: on-policy and off-policy learning [10]. On-policy learning is a methodology in which the agent procures the valuation of the policy currently employed for decision-making processes. Conversely, off-policy learning entails learning the valuation of an alternate policy differing from the policy actively employed in decision-making. Consequently, while on-policy strategies can deliver enhanced efficiency and stability, off-policy strategies offer expanded flexibility, facilitating learning from a broader spectrum of experiences.

Several techniques have been propagated for policy optimization, including value-based methods, policy gradient methods, and actor-critic methods [23]. Value-based methods target learning the optimal value function corresponding to a particular policy, subsequently deriving the optimal policy. Policy gradient methods focus on direct policy optimization, conventionally by computing the gradient of the expected cumulative reward in relation to the policy parameters. Lastly, actor-critic methods amalgamate value-based and policy gradient techniques to concurrently learn the value function and policy.

Policy optimization remains an important element of reinforcement learning, and its effective implementation is pivotal for achieving effective decision-making in an array of application domains.

2.1.2 Exploration vs. Exploitation

One of the main challenges in RL is the exploration-exploitation dilemma, where the agent must balance between exploring new actions to learn more about the environment and exploiting its current knowledge to maximize rewards [19]. Exploration is necessary to discover the optimal policy, while exploitation is necessary to reap the rewards of the already learned policy.

Researchers have proposed several methods to address this challenge. One popular approach is the ϵ-greedy method, which selects a random action with probability ϵ and the action with the highest estimated reward with probability $1 - \epsilon$ [45]. Another approach is the Upper Confidence Bound (UCB) method, which selects actions based on an uncertainty measure of their estimated reward [24].

The choice of exploration-exploitation method can significantly impact the agent's performance [4]. For example, excessive exploration can result in a slow convergence to the optimal policy, while insufficient exploration can cause the agent to get stuck in sub-optimal policies. Therefore, finding the right balance between exploration and exploitation is crucial for the success of RL algorithms.

An incorrect balance between exploration and exploitation can significantly impact the performance of reinforcement learning algorithms. If the agent explores too much, it may spend too much time in states that are not relevant to the task at hand and need more time exploiting the learned policy. On the other hand, if the agent exploits too much, it may be unable to discover better policies thus getting stuck in local optima. Thus, finding the right balance between exploration and exploitation is crucial for achieving good performance in reinforcement learning tasks [29].

2.2 Imitation Learning

Human beings possess the remarkable capacity of acquiring new skills and knowledge throughout their lives in a variety of ways. As a result, the process of learning through observation and imitation of actions performed by others has been extensively studied [36]. For instance, infants learn basic motor skills such as walking by observing and mimicking adult behavior. This ability to learn through imitation has been extensively examined in the

field of psychology [28], and more recently, it has emerged as a prominent area of investigation in the field of artificial intelligence.

Imitation learning, also known as learning from demonstration, is a sub-field of machine learning that aims to teach agents to perform tasks by imitating expert behavior [18]. One of the most popular methods of imitation learning in AI is behavior cloning [1], which allows AI models to learn complex behaviors from expert demonstrations without requiring explicit programming, making it a powerful tool for knowledge transfer and automation. Behavior cloning has been successfully applied in various domains, including robotics and video games [37, 47, 50]. However, this approach has important limitations, the main one being the difficulty of acquiring data of expert demonstrations that capture all possible variations of a given task.

To address these limitations, researchers have proposed various extensions to imitation learning, such as inverse reinforcement learning and apprenticeship learning, which aim to create more robust and adaptive models. Inverse reinforcement learning involves inferring the underlying reward function that motivated the expert's behavior, rather than simply replicating their actions [1]. This can lead to more flexible and generalist models that can adapt to new situations and tasks. Apprenticeship learning, on the other hand, involves directly interacting with the expert to learn the underlying decision-making process that leads to their behavior [1, 2]. This can be especially useful in domains where the behavior of the expert is difficult to capture in a dataset, such as in social or economic settings.

Behavior cloning, as one of the most popular methods of imitation learning, has received significant attention in the AI community due to its effectiveness and ease of use. However, there are ongoing efforts to improve the quality and variety of expert demonstrations used in this approach, as well as to explore alternative methods such as inverse reinforcement learning and apprenticeship learning. These extensions and variations in imitation learning continue to expand the possibilities for creating more robust and effective models, including the popular method of behavior cloning.

2.2.1 Behavior Cloning

Behavior Cloning (BC) is a popular method of imitation learning that involves training an AI model to imitate the behavior of a human expert by using a dataset of expert demonstrations [1]. BC receives information from the behavior of an expert acting in an environment and learns to map the behavior of the expert demonstration to an intelligent agent [18], where an intelligent agent is defined as an entity that autonomously interacts within an environment, attempting to achieve an objective [38]. In classic BC algorithms [1, 9], training examples are represented by pairs of *state* and *action*, similar to supervised learning algorithms where training examples are represented by pairs containing *features* and *class*.

A *state* describes the current behavior of the agent in the environment, which may include data on the position, velocity, and angle of the agent's joints, or even an image representing the agent's position in relation to the environment in which it is placed.

BC is commonly modeled using a Markov Decision Process (MDP) [45]. In an MDP, the problem is formulated as a tuple containing five elements $M = S, A, T, r, \gamma$, where S represents the set of states existing in the environment, A the set of possible actions, T the transition model, r a function that determines the immediate reward when the agent takes specific action in a given state, and γ is a discount factor. The solution to the problem of imitation learning using an MDP consists of finding a policy $\pi(a|s)$ that determines the probability distribution over the agent's actions so that the agent performs an action a when it is in a state s_t.

The normal sequence of an imitation learning process begins with the acquisition of expert demonstrations. These demonstrations are converted into a sequence of pairs of *state* and *action* that are used to train a policy that can imitate the behavior of the experts. While BC has shown promising results in many domains, it also faces several challenges and limitations. For instance, the quality of the expert demonstrations greatly impacts the performance of the BC model, and the model is often prone to overfitting and failing to generalize to new situations.

Despite these limitations, BC has a wide range of potential applications, such as autonomous driving, robotics, and video game AI, among others. Moreover, BC is a prominent example of the more general field of Imitation Learning, which encompasses a wide range of algorithms and methods used for training agents to imitate the behavior of experts. Imitation learning has been extensively studied in recent years, and it has shown great potential for solving complex tasks in various domains [18]. This field has seen significant advances in recent years, with new techniques and models being developed that aim to address some of the limitations of BC and other methods of imitation learning. As such, imitation learning from observation is a vibrant and active area of research, with exciting new possibilities for the development of intelligent systems in various domains.

2.2.2 Imitation Learning from Observation

Unlike humans who can learn to imitate without having direct access to the actions performed in a demonstration [36], classical behavioral cloning algorithms [34, 9] use action labels to imitate the expert's behavior. This assumption is restrictive and unrealistic for most cases, as direct access to the actions being executed by an expert is usually not available. For example, suppose a person is watching online videos to learn how to play a particular game. In that case, the person does not have direct access to the buttons being pressed by the demonstration player, but they can infer which buttons are being pressed through the

images. On the other hand, if we want to make an intelligent agent learn to play the same game by only observing online videos, we have to make it infer which actions are being executed by the demonstrator, as we do not have direct access to them.

Recent approaches take this aspect into consideration and perform imitation from observation (IfO) [27, 47]. In this type of approach, only the agent's states from the expert's demonstrations are considered, and the learning algorithm must infer the action that describes the change between two subsequent states. Thus, such approaches must learn two models: the Inverse Dynamics Model (IDM) and a Policy Model (PM). The IDM must 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 for each action a given that an agent has passed from state s_t to state s_{t+1}. In this problem, the reward function described by the MDP is not explicitly defined, and the actions performed by the expert are unknown. Thus, the problem consists of finding a policy that imitates the behavior of the experts from a set of demonstrations containing only the agent's states $D = \{\zeta_1, \zeta_2, ..., \zeta_N\}$, where ζ is a trajectory containing only states $\{s_0, s_1, ..., s_N\}$.

As the IDM needs to perform the mapping between state transitions (s_t and s_{t+1}) and actions (a), current approaches [47, 48] use self-supervised learning to perform such mapping. Thus, the agent in state s_t interacts with the environment by performing an action a using a random policy π, which consequently generates an agent state s_{t+1} as a result of the action's effect. These tuples containing pairs of state and action are saved as pre-demonstrations $\mathcal{I}^{pre} = (s_t, a_t, s_{t+1})$. The pre-demonstrations are used as ground truth for learning the inverse dynamics of the agent $\mathcal{M}_\theta$, which seeks to find the parameters θ^* that best describe the state transition.

On the other hand, the model containing the policy (PM) is responsible for cloning the expert's behavior. For this purpose, each pair of states from the expert's demonstrations $\mathcal{T}^e = \{(s_t, s_{t+1})\}$ is used by the IDM to identify the action distribution $\mathcal{M}_\theta(s_t, s_{t+1})$ and predict the action â that corresponds to the movement executed by the expert when the state changes from s_t to s_{t+1}. Once the most likely action executed for each state s_t has been identified using self-supervised learning, the method learns a policy to imitate the expert, *i.e.*, it finds the parameters ϕ^* that approximate the sequence of actions taken by the expert.

Recent studies [27, 47] have shown that IfO is a promising alternative to traditional BC methods, particularly in situations where direct access to the actions taken by the expert is limited or unavailable. By leveraging the self-supervised learning techniques to learn the inverse dynamics model, IfO can infer the expert's actions from state observations and generate action sequences that are closer to those demonstrated by the expert.

2.3 Adversarial Learning

Adversarial learning is a machine learning strategy that leverages the competitive dynamics between models to enhance learning effectiveness [14]. A prominent example of adversarial learning is Generative Adversarial Networks (GANs), wherein two neural networks, a generator and a discriminator, compete against each other within a zero-sum game framework. The generator network strives to create synthetic data that closely resemble the real data, while the discriminator network attempts to distinguish between the real and synthetic data [14]. The ultimate goal is for the generator network to produce data so convincing that the discriminator network cannot differentiate it from the real data.

Adversarial learning offers a broad spectrum of applications, ranging from generating synthetic data to enhancing the robustness of machine learning models [14]. Furthermore, adversarial learning has been employed to augment the effectiveness of model training, enabling them to learn from a more diverse array of examples. Although adversarial learning presents unique challenges, such as the difficulty of achieving equilibrium between the generator and discriminator networks, it also offers opportunities for enhancing the robustness and generalization of machine learning models [3]. As machine learning systems become increasingly ubiquitous, the importance of understanding and applying adversarial learning techniques will continue to escalate.

2.4 Types of Environment

In the field of AI, different types of environments are used to model the problem space that an agent must navigate to achieve its objectives. The type of environment that is used can have a significant impact on the performance of the agent and the effectiveness of the learning algorithms used to train it [45].

One important distinction is between deterministic and stochastic environments. In a deterministic environment, the outcome of each action taken by the agent is known with certainty, while in a stochastic environment, there is some degree of uncertainty associated with each action. This uncertainty can arise from factors such as sensor noise or unpredictable changes in the environment [45].

Another important distinction is between episodic and continuous environments. In an episodic environment, the agent's actions occur in discrete episodes, where each episode begins with the agent in a particular state and ends when the agent reaches a terminal state or a predetermined number of time steps have elapsed. In contrast, in a continuous environment, the agent's actions occur continuously over time, without any clear episodic structure [45].

Understanding the type of environment being used is crucial when designing and training an intelligent agent. In this thesis, we will be using these environment types to test our hypotheses on the effectiveness of imitation learning from observation. By studying how our algorithms perform in different environments, we can gain insights into their strengths and limitations, and refine our methods to achieve better performance in real-world applications.

3. RELATED WORK

The field of imitation from observation is becoming increasingly important as there is a need to create intelligent systems capable of imitating human behavior. Advancements in this area and the effectiveness of imitation have motivated various applications, ranging from enhancing realism in gaming characters and improving physics to using human models for animation in the film industry. Imitating via observation requires access to the real or simulated environment where expert data is generated. By accessing the environment, we can learn from unlabeled expert data, resembling human learning from video observations. Researchers have recently proposed several approaches and advances to imitation from observation field. This chapter summarizes the main approaches most relevant to this thesis and its contributions. We discuss the works of Torabi *et al.* [47] on "Behavior Cloning from Observation", Edwards *et al.* [11] on "Imitating Latent Policies from Observation", Ho and Ermon [17] on "Generative Adversarial Imitation Learning", and Torabi *et al.* [48] on "Generative Adversarial Imitation from Observation". These methods represent significant contributions to the imitation learning from observation field, and understanding them provides valuable insights and context for the contributions presented in this thesis. In addition, by reviewing these approaches, we can emphasize their key concepts, methodologies, and how they relate to and complement the proposed contributions of this thesis.

3.1 Behavior Cloning from Observation

Torabi *et al.* [47] present the work *Behavioral Cloning from Observation* (BCO) to imitate expert behaviors through a self-supervised approach based on state observation. They employ an Inverse Dynamics Model (IDM) to infer actions in a self-supervised manner, making it possible to learn a Policy Model (PM) responsible for informing the agent what to do in each state of the environment. BCO is one of the most important works for this thesis, as they share the same motivations. It does not, however, perform tests considering temporal scenarios, and it uses only numerical values directly collected from the experimented environments (no images nor videos). The authors present the results of their approach in *Cartpole*, *Mountain Car*, *Reacher*, and *Ant* environments, all of which are available in OpenAI Gym[1], which is a toolkit for developing and comparing reinforcement learning algorithms and similar techniques. BCO presented results comparable to supervised approaches in the employed environments.

[1] OpenAI Gym - https://gym.openai.com/.

3.1.1 Inverse Dynamics Model

The IDM, implemented in [47], utilizes a neural network to learn the actions that enable the agent to transition from state s_t to state s_{t+1}. In order to learn these actions without labeled state-action pairs, the agent interacts with the environment using a random policy π, generating state pairs $\mathcal{T}_{\pi_\phi} = \{(s_t, s_{t+1}), ...\}$ and their corresponding actions $A_{\pi_\phi} = \{a_t, a_{t+1}, ...\}$.

The state pairs, along with their corresponding actions (s_t, a_t, s_{t+1}), are stored as a pre-demonstration ($\mathcal{I}^{pre}$) generated by the IDM. By randomly transitioning between states in $\mathcal{I}^{pre}$, the model learns the inverse dynamics $\mathcal{M}_\theta$ of the agent, finding the optimal parameters θ^* that best describe the actions leading to the transitions in $\mathcal{T}_{\pi_\phi}$.

BCO employs maximum likelihood estimation (Equation 3.1) to find the optimal parameters, where p_θ represents the probability distribution of actions given a pair of states representing a transition. During the testing phase, the IDM uses the learned parameters to predict an action $\hat{a}$ given a state transition (s_t, s_{t+1}). After obtaining the parameters for the IDM, the next step involves pseudo-labeling expert state pairs and training the Policy Model.

$$\theta^* = \arg\max_\theta \prod_{(s_t, a_t, s_{t+1}) \in \mathcal{I}^{pre}} p_\theta(a_t | s_t, s_{t+1}) \tag{3.1}$$

3.1.2 Policy Model

The PM is responsible for cloning the expert's behavior in a given task. For example, based on expert demonstrations in a given task $D = \{\zeta_1, \zeta_2, ..., \zeta_N\}$, where each demonstration consists of state pairs $(s_t^e, s_{t+1}^e) \in \mathcal{T}^e$, BCO uses the IDM to calculate the probability distribution over actions $\mathcal{M}_\theta(s_t^e, s_{t+1}^e)$ and predict actions $\hat{a}$ that correspond to the expert's movement from state s_t to s_{t+1}.

With the predicted action, through self-supervised training, the method creates a set of state-action pairs $\{(s_t^e, \hat{a})\}$ corresponding to the action $\hat{a}$ taken in state s_t. By having s_t and the predicted action $\hat{a}$, it is possible to train the PM to learn the imitation policy π_ϕ, which is responsible for imitating the expert's behavior in a supervised manner.

Following the field of Imitation Learning, learning an imitation policy π_ϕ using state-action tuples $\{(s_t^e, \hat{a})\}$ involves finding the parameters ϕ^* that make π_ϕ best match the given tuples. Furthermore, the authors also implement maximum likelihood estimation in the PM, following Equation 3.2, to find the optimal parameter set ϕ^*. After training the PM, represented by a neural network [47], the model is capable of receiving any state (s_t) and pre-

dicting the action (a) responsible for transitioning from the current state (s_t) to the next state (s_{t+1}) that closely resembles the behavior of the experts observed by the model.

$$\phi^* = \arg\max_{\phi} \prod_{t=0}^{N} \pi_\phi(\hat{a}_t \mid s_t) \tag{3.2}$$

3.1.3 Iterative Model

Torabi *et al.* [47] extend their proposed method to incorporate an iterative process. After training the neural network that represents the PM, it learns observations and stores the sequences of states and predicted actions as post-demonstrations ($\mathcal{I}^{pos}$). The purpose of $\mathcal{I}^{pos}$ is to be used to adjust the parameters of the IDM in the next iteration, enabling the model to classify state transitions that resemble those of experts.

The improvement, called BCO (α), where α represents a parameter to control the number of post-demonstration iterations, works as follows. After learning the imitation policy, the agent executes the environment to acquire new state-action sequences as post-demonstrations ($\mathcal{I}^{pos}$). These post-demonstrations are used to update the IDM and, subsequently, the PM itself.

The challenge with iterating in BCO is that it solely utilizes the set of post-demonstrations to retrain the IDM. Thus, in cases where the policy still needs to have sufficiently good predictive performance, the generated set of post-demonstrations will contain mislabeled actions for the presented state pairs. These erroneous actions tend to degrade the predictive performance of the IDM, leading to a decrease in the quality of policy predictions in a negative feedback cycle.

3.2 Imitating Latent Policies from Observation

Edwards *et al.* [11] present a Forward Dynamics Model responsible for mapping state-action pairs $\{(s_t, a_t)\}$ to the next state s_{t+1}, naming this approach "Imitating Latent Policies from Observation" (ILPO). First, it is important to define that a latent action can be one or more actions responsible for causing a transition in a given state. This definition is necessary because ILPO does not define all generated actions as latent ones. The authors also consider that for all environments, there is a set of known actions A and a set of latent actions $\mathcal{Z}$, which they define as $\{z_1 ... z_A\} \in \mathcal{Z}$.

The authors also note that not all static state transitions have accurate associations with their original actions. Therefore, ILPO empirically assumes that $|\mathcal{Z}| \neq |A|$. The approach consists of two steps: first, the agent learns a latent policy offline (Section 3.2.1), which

estimates the probability of a latent action given the current state. Then, in a limited number of steps in the environment, the model remaps actions (Section 3.2.2), associating latent actions with the corresponding correct actions.

Finally, they train the models and test them using environments available in the OpenAI Gym, such as Cartpole, Acrobot, Mountain Car, and CoinRun. The evaluation metrics include the Average Episodic Reward (AER) and Performance.

3.2.1 Latent Forward Dynamics

The first step of ILPO involves executing Latent Forward Dynamics, which is a generative model represented by $G_\theta\left(E_p\left(s_t\right), z\right)$. This model aims to generate a state s_{t+1} given the current state s_t and a latent action z. The Latent Forward Dynamics model predicts the state difference $\Delta_t = s_{t+1} - s_t$ instead of the absolute next state and computes $s_{t+1} = s_t + \Delta_t$. However, after learning, the generator may start predicting average transitions instead of learning the desired transition. Since ILPO lacks action information to condition state generation, the authors proposed to train the Latent Forward Dynamics model to make predictions based on each latent action $z \in Z$, $f\left(s_{t+1}|s_t, z\right)$ instead of $f\left(s_{t+1}|s_t, a\right)$. To train the generator, ILPO computes the error using Equation 3.3:

$$\mathcal{L}_{\text{min}} = \min_z \left\|\Delta_t - G_\theta\left(E_p\left(s_t\right), z\right)\right\|^2 \tag{3.3}$$

Thus, the goal of the policy generated by ILPO is to minimize the error presented in Equation 3.4. For this purpose, they fix the predictions to produce the most likely next state without impacting the generator's output.

$$\mathcal{L}_{exp} = \left\|s_{t+1} - \widehat{s}_{t+1}\right\|^2 \tag{3.4}$$

The process is to train the network by considering the errors presented in Equations 3.3 and 3.4.

3.2.2 Action Remapping

The second model presented in ILPO is responsible for learning how to map latent actions discovered by the previous model to real actions. To achieve this, ILPO collects tuples of the current state, the action taken, and the next state $\{s_t, a_t, s_{t+1}\}$ through a random policy or an iterative policy remapping process π_ξ.

By collecting data while running an agent in a given environment, ILPO first identifies the latent action corresponding to the state transition. It then uses the action taken in the environment to label the data and train its policy $\pi_\xi(a_t|z_t, E_a(s_t))$. The authors use the L_2 distance between the generator and the actual state s_{t+1} for environments with lower-dimensional state spaces, as shown in Equation 3.5. For environments with higher dimensional state spaces, such as chess, the L_2 distance using E_p is applied, as shown in Equation 3.6.

$$z_t = \arg\min \left\| s_{t+1} - G_\theta \left(E_p(s_t), z \right) \right\|_2 \tag{3.5}$$

$$z_t = \arg\min_z \left\| E_p(s_{t+1}) - E_p \left(G_\theta \left(E_p(s_t), z \right) \right) \right\|_2 \tag{3.6}$$

Once the latent action z_t that best approximates the action taken in the environment a_t is obtained, the network learns using the cross-entropy loss in a supervised manner.

3.3 Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning (GAIL) [17] leans heavily on the generative adversarial framework. In GAIL, the goal is to learn how to mimic expert policies, which are represented as state-action pairs, using adversarial training techniques [15]. This approach integrates the structure of reinforcement learning with adversarial networks, providing an intricate balance between exploration and exploitation of the policy space.

As a generative adversarial training-based model, GAIL comprises two main components: the discriminator and the generator. The discriminator is responsible for the distribution of states and actions which define expert behavior, while the generator represents the imitation policy. The proposed model seeks to find a policy π_θ for which the discriminator $\mathcal{D}_\mathcal{R}$ cannot distinguish between states explored by expert policies π_E and states explored by the imitation policy generated by GAIL π_θ. The equilibrium that the discriminator is expected to attain is formulated by Equation 3.7:

$$\max_{\pi_\theta} \min_{\mathcal{D}_\mathcal{R}} -E_{\pi_\theta}[\log \mathcal{D}_\mathcal{R}(s)] - E_{\pi_E}[\log(1 - \mathcal{D}_\mathcal{R}(s))]. \tag{3.7}$$

The authors implemented the policy and the discriminator using deep neural networks, and the training was performed iteratively by updating their gradients. The discriminator ($\mathcal{D}_\mathcal{R}$) is trained in a supervised manner using data generated by the evolving policy π_θ as well as data from expert demonstrations. Once $\mathcal{D}_\mathcal{R}$ converges, the pursuit of the optimal policy for a given problem begins. The search for the best policy π_θ is conducted using $-\log \mathcal{D}_\mathcal{R}(s)$ as the reward value and the Trust Region Policy Optimization method [40].

To test their new method, the authors used the same steps as in the BCO study [47]. They chose OpenAI Gym as the place to run their tests: Cartpole, Acrobot, Mountain Car, HalfCheetah, Hopper, Walker, Ant, Humanoid, and Reacher. Each environment has different challenges, which made sure the model was tested thoroughly. After they trained their model in those environments, they measured its success using the Performance metric, which we further explain in the following chapter. Hence, they could understand how well the model learned and copied expert behavior across all different tasks.

3.4 Generative Adversarial Imitation from Observation

Inspired by GAIL, Torabi *et al.* [48] proposed an approach that uses a Generative Adversarial Network (GAN) called *Generative Adversarial Imitation from Observation* (GAIfO). The method uses demonstrations from experts and demonstrations generated by a model trained to imitate, while the discriminator classifies the source of the data.

The imitation model's policy aims to generate state transitions that appear to be generated by an expert, *i.e.*, it tries to align the distribution of state transitions from the imitation model to that of the expert. The authors use deep neural networks to implement the imitation model (policy) and the discriminator.

The overall architecture of the method is depicted in Figure 3.1. The policy model (top part of Figure 3.1) takes four gray-scale images. These images are arranged from $t-3$

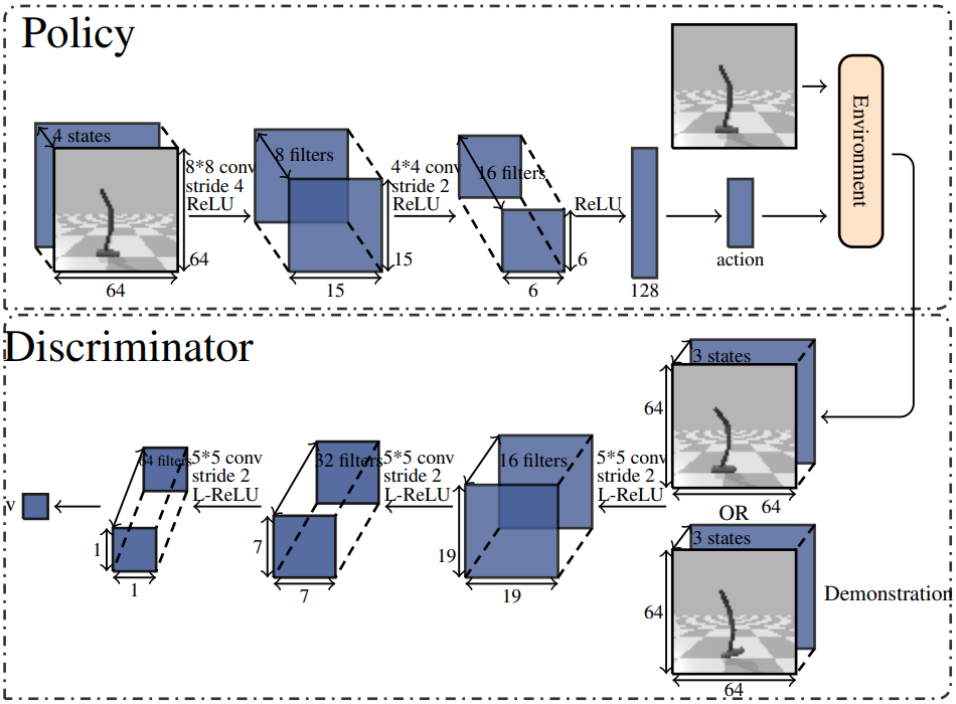


Figure 3.1 – Representation of the Imitation Model (Policy) and the Discriminator. Image adapted from the work proposed by Torabi *et al.* [48].

to t, where t represents the current state. The network applies convolutions and activation functions to these states, producing the action that transitions the state from t to $t+1$ through the environment.

In the discriminator (bottom part of Figure 3.1), the model receives three gray-scale images representing states from $t-1$ to $t+1$. These states can come from expert demonstrations or the imitation model. The discriminator outputs a value of 0 if the states are from the imitation model or 1 from the experts. The objective is for the imitation model to generate states so similar to those generated by the experts that the discriminator classifies them as such.

To evaluate the proposed method, the authors conduct experiments using eight environments from OpenAI Gym: *Inverted Pendulum*, *Inverted Double Pendulum*, *Inverted Pendulum Swing Up*, *Hopper*, *Walker 2D*, and *Reacher*.

3.5 Final Remarkls

In this chapter, we introduced the key studies related to this thesis, along with their features and the environments they utilized. However, the area of Imitation from Observation (IfO), being relatively new, still lacks a substantial amount of published research. We highlight the most relevant studies in the field that closely align with our thesis and those that have been published in prominent conferences.

Most of the selected works in this chapter concentrate primarily on imitation via observation, without any pre-existing labels identifying the actions of the experts. We emphasize the study that pioneered the field of IfO [47], along with the ones that started incorporating images and Deep Learning models [17, 11, 48].

Every study discussed in this chapter still has room for improvement, such as issues during the training phase (due to vanishing gradients), the use of more complex environments, more effective exploration of problem states, using images as input for the models, testing in discrete environments, among other aspects.

The works presented in this chapter share several characteristics, as seen in Table 3.1. Besides employing some form of Neural Network and Deep Learning, they also use the *Performance* metric to measure the quality of their proposed models.

One of the works presented, ILPO [11], introduces the AER metric. This metric is essential for evaluating the quality of the policy in achieving the highest possible reward in a given environment. Both AER and Performance are discussed in greater detail in Chapter 4.

Another similar feature among the studies is their use of environments derived from the OpenAI Gym tool due to its convenience in implementing these environments in their proposed models. These shared characteristics inspire us to develop this thesis with similar

Table 3.1 – List of related works, as well as the models, metrics and environments used.

Method	Year	Model	Metrics	Environments
BCO [47]	2018	Artificial Neural Networks	*Performance*	*Cartpole*, *Mountain Car*, *Reacher*, *Ant*.
ILPO [11]	2019	Convolutional Neural Networks	AER, *Performance*	*Cartpole*, *Acrobot*, *Mountain Car*, *CoinRun*.
GAIL [17]	2016	Generative Adversarial Networks	*Performance*	*Cartpole*, *Acrobot*, *Mountain Car*, *HalfCheetah*, *Hopper*, *Walker*, *Ant*, *Humanoid*, *Reacher*.
GAIfO [48]	2019	Generative Adversarial Networks	*Performance*	*Inverted Pendulum*, *Inverted Double Pendulum*, *Inverted Pendulum Swing Up*, *Hopper*, *Walker-2D*, *Reacher*.

features (environments and metrics) to use as benchmarks against our methods. In the next chapter, we will present the methodological elements that guide this thesis: hypothesis, metrics, application scenarios, and the neural network topology that is used to throughout this work.

4. METHODOLOGY

This chapter presents the backbone of this thesis. It details the systematic approach to achieve the research goals and test the thesis' primary hypothesis.

4.1 Hypothesis

The central hypothesis of this research is a practical and targeted proposition aimed at advancing the field of Imitation Learning from Observation. We hypothesize that the classic Imitation from Observation framework (BCO) can be significantly improved by incorporating different techniques, such as state exploration, reinforcement learning, and adversarial training.

This hypothesis is based on the premise that these techniques can enhance the robustness and efficiency of the learning model, enabling it to outperform the current state-of-the-art. By integrating these techniques, we assume the resulting model will be more effective in imitating expert behavior across a variety of environments.

Throughout this research, we rigorously test this hypothesis using the metrics and environments detailed in the subsequent sections. The outcomes of these tests provide valuable insights into the validity of this hypothesis and the effectiveness of the proposed improvements.

4.2 Goals

The goals of this thesis were carefully crafted and strategically aligned aiming at addressing the existing challenges, enhancing the current methodologies, and pushing the boundaries of the state-of-the-art.

- General Goal
  - Significantly contributing to Imitation Learning from Observation by **improving** the Behavior Cloning from Observation framework.
- Specific Goals
  - **Environment Selection and Preparation**: selecting and preparing environments with varying characteristics suitable for Imitation Learning. The diversity in these environments will allow us to test the robustness and adaptability of the proposed improvements.

- **Expert Behavior Selection and Preparation**: selecting and preparing expert behaviors for the chosen environments alongside the environment preparation. These behaviors will serve as the benchmark for the imitation learning models, providing a clear target for the learning algorithms.
- **Framework Improvement**: enhancing the existing Imitation from Observation framework. We aim to achieve superior results compared to the current state-of-the-art by refining the framework's performance, efficiency, and reliability.
- **Problem Solving**: proposing solutions to current problems faced in the field, such as the issue of gradient vanishing. By addressing these challenges, we focus on improving the stability and effectiveness of imitation learning models.
- **Results Comparison**: comparing the achieved results with the current state-of-the-art approach for each selected environment. This comparison will provide a clear measure of our progress and the effectiveness of the proposed improvements.

4.3 Metrics

To evaluate the quality of the methods we have developed, we utilize two primary metrics: Average Episodic Reward (AER) and Performance ($\mathcal{P}$) [17]. We chose these metrics because they have been extensively used in imitation learning tasks and studies, as demonstrated in Table 3.1.

AER is a standard metric to verify how well the generated Policy performs in a given environment. The metric consists of the average value of 100 runs for each episode in a given environment. For example, this could involve averaging the rewards of 100 runs in different mazes in the Gym-Maze environment or averaging the rewards generated through 100 consecutive episodes for the CartPole problem. Expert policies with excellent performance in a given task are usually difficult to imitate. In contrast, experts with low performance may be easier to imitate. The AER value is a good metric to verify these characteristics, as we can understand how the expert performed its task and how difficult it was to imitate its behavior. The calculation of the AER metric involves the following formula, where E represents the number of performed episodes (100), F represents the number of steps, and $\pi_\phi(e_{ij})$ represents the reward obtained by Policy π_ϕ during execution i at step j:

$$AER = \frac{\sum_{i=1}^{E} \sum_{j=1}^{F_i} \pi_{\phi_{ij}}}{E} \tag{4.1}$$

No less important than AER, the Performance metric is calculated as the average reward for each run, normalized between 0 and 1. Note that each environment often provides its minimum and maximum values of rewards. We represent the value 0 of $\mathcal{P}$ as the reward obtained by a random policy running in a given environment and the value 1 as the reward obtained by the expert's policy. Similarly to the available imitation learning works in the literature, we do not use metrics such as accuracy to measure the quality of the generated policies, as this metric cannot guarantee high-quality results for this problem. IfO techniques tend to use two models to generate a policy in a self-supervised manner. However, the error of the first model can easily be propagated to the second, hiding the real quality of the method. Therefore, we cannot use accuracy as a metric to verify the quality of the generated policies, as achieving 100% accuracy with a policy generated using a poor Inverse Dynamics Model will consequently reduce the AER and $\mathcal{P}$. The following equation illustrates the Performance metric, where E represents the number of performed episodes, π_{ϕ_e} represents the reward of the model π_ϕ in episode e, π_{ϵ_e} represents the reward of the random model π_ϵ in episode e, and π_e represents the reward of the expert's policy π for episode e:

$$\mathcal{P} = \frac{\sum_{e=1}^{E} \frac{\pi_{\phi e} - \pi_{\epsilon e}}{\pi e - \pi_{\epsilon e}}}{E} \tag{4.2}$$

4.4 Environments

This section presents the selected environments for training and testing the proposed methods. All chosen environments are applicable in the Imitation from Observation field. The primary basis for selecting the environments was the distinct characteristics they have. We aimed at selecting environments regarding the number of available actions, dimensions, environment observation, type (discrete or continuous), and whether they are static or dynamic. The goal is to measure the real performance of the proposed methods across different scenarios. All of these environments are available through the OpenAI Gym toolkit. The characteristics of each environment are described below and summarized in Table 4.1.

Table 4.1 – Information regarding the five selected environments for imitation learning.

Environment	#Actions	Dimension	Observation	Type	Discrete/Continuous	Static/Dynamic
Acrobot	3	6	full	stochastic	continuous	dynamic
CartPole	2	4	full	stochastic	continuous	dynamic
Gym-Maze	4	128×128	full	deterministic	discrete	static
LunarLander	4	128×128	full	deterministic	continuous	dynamic
MountainCar	3	2	full	stochastic	continuous	dynamic

Acrobot

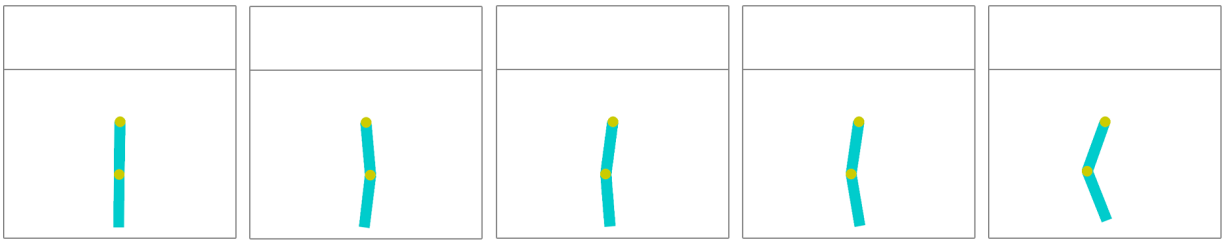


Figure 4.1 – Visual representation of the Acrobot environment.

Acrobot, originally proposed by Sutton [44], is an environment featuring two joints and two mechanical links, with the joint between the two links being actuated. Initially, the links hang downward, and the objective is to swing the end of the lower link up to a specific height. The state space consists of 6 dimensions, representing the cosine, sine, and position of both joints: $\{\cos\theta_1, \sin\theta_1, \cos\theta_2, \sin\theta_2, \theta_1, \theta_2\}$. The action space consists of three possible forces: applying force to the right, applying force to the left, or not applying any force. It is worth mentioning that Acrobot is an unsolved environment, meaning it does not have a specified reward threshold for being considered solved. Acrobot (Figure 4.1) was chosen for this thesis due to its continuous and dynamic nature, as well as its significant difficulty, which poses challenges for imitation learning models. Acrobot serves as a challenging task in the field of reinforcement learning and provides a platform for testing the capabilities of various learning algorithms. Its complex dynamics and the absence of a predefined success threshold make it a suitable environment for evaluating imitation learning models' performance and generalization abilities.

CartPole

The CartPole environment, first introduced by Barto *et al.* [5], is a task where an agent applies forces to the side of a cart to balance a pole vertically for as long as possible. This environment has two discrete actions: applying force to the left or right side of the cart. The state space consists of four dimensions: cart position, cart velocity, pole angle, and velocity at the tip of the pole. The objective of CartPole is to maintain the pole upright for as many time steps as possible. The CartPole environment defines a successful solu-

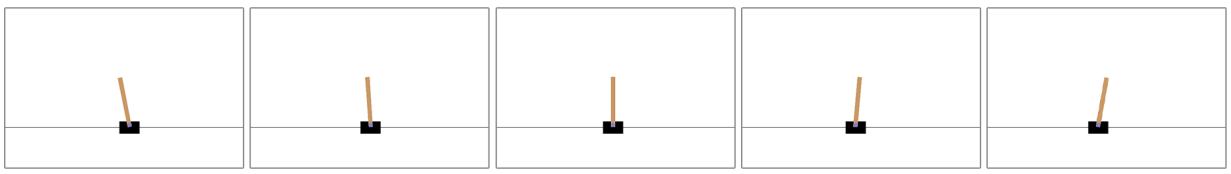


Figure 4.2 – Visual representation of the CartPole environment.

tion as achieving an average reward of 195 over 100 consecutive attempts. An example of the CartPole environment can be seen in Figure 4.2. CartPole is a classic reinforcement learning benchmark and is a suitable environment for testing and evaluating different algorithms and approaches. CartPole's simplicity and well-defined objective make it popular for experimenting with reinforcement learning agents.

Gym-Maze

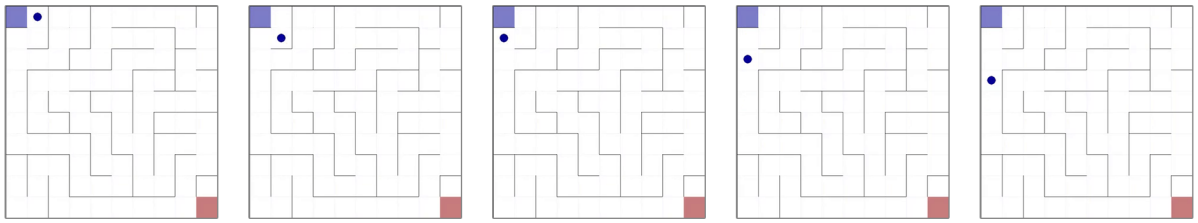


Figure 4.3 – Visual representation of the Gym-Maze environment.

Gym-Maze is a 2D maze environment where an agent, represented by a blue circle in Figure 4.3, aims to find the shortest path from the starting point (a blue square in the top-left corner) to the goal (a red square in the bottom-right corner). Each maze within Gym-Maze can have various configurations of walls and different sizes, such as 3×3, 5×5, or 10×10. The agent can move in four directions: west, north, east, and south. The state space of the Gym-Maze environment consists of rendered images of the maze, with a resolution of 128×128 pixels, as shown in Table 4.1. The action space is discrete and allows the agent to select from four possible actions: N (north), S (south), W (west), and E (east). Additionally, the Gym-Maze environment is deterministic and discrete, representing a novel type that has not been extensively explored in the existing literature. Gym-Maze is a platform for evaluating navigation and pathfinding algorithms in complex 2D mazes. Its varying maze configurations and sizes provide a challenging setting for testing and benchmarking different approaches to maze-solving tasks.

LunarLander

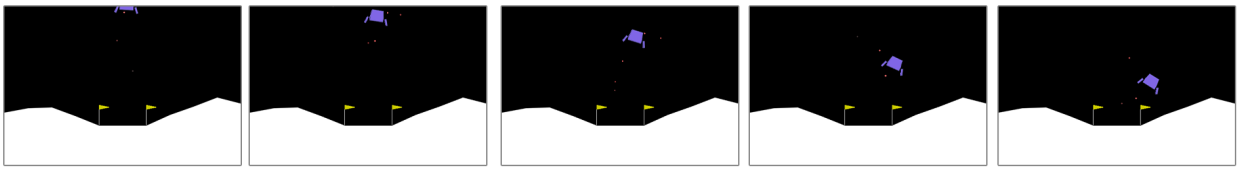


Figure 4.4 – Visual representation of the LunarLander environment.

LunarLander, developed by Klimov [6], is an environment where an agent must successfully land a spacecraft on the moon's surface under low gravity conditions. An illustration

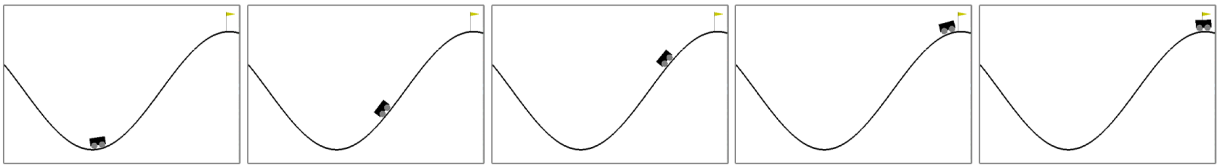


Figure 4.5 – Visual representation of the MountainCar environment.

of LunarLander is presented in Figure 4.4. The state space in LunarLander is continuous, while the action space is discrete. There are four possible actions available to the agent: *do nothing*, *move left*, *move right*, and *reduce the falling velocity*. In LunarLander, each action except for the *do nothing* state incurs a reward of -1, whereas the *do nothing* state has a reward of -0.3. The agent receives a positive reward when it moves in the correct direction, always at the coordinates $(0, 0)$. To consider LunarLander as solved, the agent must accumulate a total reward of 200 over 100 consecutive trial runs. LunarLander is a challenging task in the Reinforcement Learning field, requiring precise control and navigation in a simulated lunar landing scenario.

MountainCar

MountainCar [32] is an environment where a car is positioned on a one-dimensional track between two "mountains". The goal for the agent is to learn how to leverage potential energy by driving up the opposite hill to reach the designated goal position. The state space in MountainCar consists of two continuous attributes: *velocity* and *position*. The action space is discrete and offers three possible actions: *left*, *neutral*, and *right*. In MountainCar, a reward of -1 is provided for each time step until the car reaches the goal position of 0.5. The initial state starts with the car in a random position and with no velocity. Its goal is reached by achieving an average reward of -110 over 100 consecutive trial runs, as defined by Moore [32].

MountainCar challenges the agent to overcome the forces of gravity and limited car dynamics to climb the hill and reach the goal position successfully. It provides a testbed for evaluating and comparing different reinforcement learning algorithms' capabilities in solving complex continuous control problems. Figure 4.5 provides a visual representation of the MountainCar environment.

5. AUGMENTED BEHAVIOR CLONING FROM OBSERVATION

Recent approaches perform *imitation from observation* (IfO) [27, 47], which uses only the sequence of state observations from the expert. Such approaches learn two models: the inverse dynamics of the environment (Inverse Dynamics Model, IDM) and an imitation policy model (PM). Current approaches learn both models iteratively from samples based on each other, *i.e.*, the IDM uses demonstrations generated with a specific policy from PM to update its model, and then the PM is updated using the new outcomes from the updated IDM. Using iterations during the learning process allows the policy model to approximate the distribution of actions used by the expert, which improves the imitation process. However, performing IfO using this type of iteration has the drawback of overfitting the policy demonstrations, primarily in the first iterations, and sometimes, causing some of the actions to be ignored altogether during learning of the PM due to errors in the IDM.

To address this problem in IfO, we design an architecture, named *Augmented Behavior Cloning from Observations* (ABCO), that uses attention models and a sampling mechanism to regulate the observations that feed the inverse dynamics model, preventing the models from reaching undesirable local minima. The idea is to train a model with the inverse dynamics of the environment to infer actions from state changes and a policy model to mimic the expert via behavior cloning. ABCO considerably boosts both the sample efficiency and the quality of the imitation policy model, surpassing traditional behavior cloning. This is achieved first by incorporating an inverse dynamic model (Section 5.1) and a carefully devised sampling strategy (Section 5.2), which together determine the observations inputted into the inverse dynamic model. The system is further improved by integrating attention mechanisms (Section 5.3) within both the inverse dynamic model and the sampling strategy. Results (Section 5.4) show that by using either low-dimensional state spaces or raw images as input, ABCO outperforms the main IfO algorithms regarding both *Performance* and *Average Episodic Reward*.

5.1 Inverse Dynamics Model and Policy Model

Our approach represents the inverse dynamics model (IDM) as a neural network that learns the actions that enable an agent's transition from state s_t to s_{t+1}. The agent interacts unsupervised with the environment, using a random policy π, thereby generating pairs of states $\mathcal{T}^{ag}_{\pi_\phi} = (s^{ag}_t, s^{ag}_{t+1}), ...$ for the agent *ag* along with their corresponding actions $A\pi\phi = a_t, ...$ These state pairs and their associated actions (s_t, a_t, s_{t+1}) are stored as a pre-demonstration ($\mathcal{I}^{pre}$).

While transitioning randomly between states in $\mathcal{I}^{pre}$, the model learns the inverse dynamics $\mathcal{M}\theta$ of the agent, finding the parameters θ^* that optimally describe the actions leading to transitions from $\mathcal{T}\pi_\phi{}^{ag}$. The BCO employs maximum-likelihood estimation (Equation 5.1) to identify the best parameters, where p_θ represents the action probability distribution given a pair of states. At test time, the IDM utilizes the learned parameters to predict an action $\hat{a}$ for a state transition (s_t^{ag}, s_{t+1}^{ag}).

$$\theta^* = \arg, \max_\theta, \prod_{t=0}^{|\mathcal{I}^{pre}|} p_\theta(a_t \mid s_t^{\pi_\phi}, s_{t+1}^{\pi_\phi}) \tag{5.1}$$

ABCO enhances the conventional IDM by incorporating a Self-Attention (SA) module [49, 54] (Section 5.3). This helps accommodating the substantial sample variations from $\mathcal{I}^{pre}$ to $\mathcal{I}^{pos}$ during iterative processes. The SA component compels the IDM to discern what is crucial to learn from each state. When applied to images, the SA can identify which portion of the image representing the state is essential for predicting the correct action.

5.1.1 Policy Model

The Policy Model (PM) seeks to emulate the behavior of an expert. It bases its process on expert demonstrations $D = \zeta_1, \zeta_2, \ldots, \zeta_N$, where each demonstration contains pairs of subsequent states $(s_t^e, s_{t+1}^e) \in \mathcal{T}^e$. ABCO uses the IDM to calculate the action distribution $\mathcal{M}\theta(s_t^e, st + 1^e)$ and predict the action $\hat{a}$ that corresponds to the movement executed by the expert to transition from state s_t to s_{t+1}. With the predicted action, the method constructs a set of state-action pairs $(s_t^e, \hat{a})$ that correspond to the action $\hat{a}$ taken in state s_t. This information is subsequently used to learn the imitation policy π_ϕ that replicates the expert behavior in a supervised manner.

For behavior cloning, the task of learning an imitation policy π_ϕ from state-action tuples $\{(s_t^e, \hat{a})\}$ involves finding parameters ϕ^* that make π_ϕ align as closely as possible with the given tuples. Originally, BCO employed maximum-likelihood estimation (Equation 5.2) to determine the optimal parameters ϕ^*. Following policy network training, it performs imitation learning and stores the state sequences and predicted actions as post-demonstrations $(\mathcal{I}^{pos})$.

$$\phi^* = \arg, \max_\phi, \prod_{t=0}^{N} \pi_\phi(\hat{a}_t \mid s_t) \tag{5.2}$$

Contrasting with the original BCO, we improve the PM by introducing a self-attention module [49, 54], elaborated further in Section 5.3. As opposed to its usage in the IDM, we deploy the SA in the PM to limit state changes in each iteration, given that the SA module

hones in on minor details and differentiates classes provided by the IDM more effectively. The SA also enables the policy to examine states non-locally, thereby promoting faster learning for high-dimensional states (Maze and Acrobot) with a more incremental success rate across iterations.

5.1.2 Iterative Behavioral Cloning from Observation

The BCO algorithm was extended by Torabi *et al.* [47] through incorporating interactions with the post-demonstration environment to enhance both the IDM and the imitation policy. This improvement, named BCO(α), with α symbolizing a user-defined hyperparameter controlling the quantity of post-demonstration interactions, operates as follows: Following the learning of the imitation policy, the agent engages with the environment to accumulate new state-action sequences as post-demonstrations ($\mathcal{I}^{pos}$). These post-demonstrations are subsequently used to update the IDM and, by extension, the imitation policy.

A limitation of the iterative BCO is its exclusive reliance on the set of $\mathcal{I}^{pos}$ for retraining the IDM. As such, in situations where the policy's predictive performance remains sub-optimal, the generated set of post-demonstrations might include incorrect actions for specific state pairs. These wrong actions could degrade the IDM's predictive performance, resulting in a negative feedback loop on the policy's predictions.

To address these challenges, ABCO(α), a model that iteratively refines ABCO through a sampling method that balances the extent to which the IDM learns from pre-demonstrations ($\mathcal{I}^{pre}$) and post-demonstrations ($\mathcal{I}^{pos}$). Algorithm 5.1 outlines the ABCO(α) training process, wherein TRAINIDM($\mathcal{I}^s$) represents the use of $\mathcal{I}^s$ to identify a θ^* that best accounts for the transitions in the demonstration $\mathcal{I}^s$ as presented in Equation 3.1.

Algorithm 5.1 Augmented Behavioral Cloning from Observation (ABCO)

Initialize the model $\mathcal{M}_\theta$ as a random approximator
Initialize the policy π_ϕ with random weights
Generate $\mathcal{I}^{pre}$ using policy π_ϕ
Generate state transitions $\mathcal{T}^e$ from demonstrations D
Set $\mathcal{I}^s = \mathcal{I}^{pre}$
for $i \leftarrow 0$ to α **do**
    Improve $\mathcal{M}_\theta$ by TRAINIDM($\mathcal{I}^s$)
    Use $\mathcal{M}_\theta$ with $\mathcal{T}^e$ to predict actions $\hat{A}$
    Improve π_ϕ by behavioralCloning($\mathcal{T}^e$, $\hat{A}$)
    for $e \leftarrow 1$ to $|E|$ **do**
        Use π_ϕ to solve environment e
        Append samples $\mathcal{I}^{pos} \leftarrow (s_t, \hat{a}_t, s_{t+1})$
        if π_ϕ at goal g **then**
            Append $v_e \leftarrow 1$
        else
            Append $v_e \leftarrow 0$
        end if
    end for
    Set $\mathcal{I}^s$ = SAMPLING($\mathcal{I}^{pre}$, $\mathcal{I}^{pos}$, $P(g \mid E)$, v_e)
end for

5.2 Sampling Method

In the process of each iteration, our approach focuses on a sampling technique that generates an improved dataset, $\mathcal{I}^s$, which contains a subset of post-demonstrations $\mathcal{I}^{pos}_{spl}$ along with a subset of pre-demonstrations $\mathcal{I}^{pre}_{spl}$. To derive the subset from $\mathcal{I}^{pos}_{spl}$, we initially determine the distribution of actions, given a specific run E within the environment, and the current policy $P(A \mid E; \mathcal{I}^{pos})$.

We restrict our focus to only successful runs from $\mathcal{I}^{pos}$, which implies that we consider only those state-action sequences where the agent accomplished the goal set by the environment. This goal could be to reach a specific state or to avoid an undesirable state for a predetermined number of transitions. The type of expert demonstration we receive guides us in inferring these goal states. We denote it as v_e in Equation 5.3, where v_e is set to 1 if the agent accomplishes the environmental objective and zero otherwise, and E is the total number of runs within an environment.

$$P(A \mid E; \mathcal{I}^{pos}) = \frac{\sum_{e \in E} v_e \cdot P(A \mid e)}{|E|} \tag{5.3}$$

The logic behind restricting the use of post-demonstration to successful runs is that if a policy fails to achieve the environmental goal, then exclusively relying on post-demonstration would not sufficiently bridge the learning gap between what the model previously learned with $\mathcal{I}^{pre}$ and what the expert performs within the environment. By focusing only on successful runs, we also ensure a more accurate expert distribution. This approach also solves the issue where BCO(α) decreases the performance in both models.

With the action distribution derived from successful runs, we select the subset $\mathcal{I}^{pos}_{spl}$ from these runs, following the win probability $P(g \mid E)$, meaning the probability of achieving a goal in a given environment, as illustrated in Equation 5.4.

$$\mathcal{I}^{pos}_{spl} = (P(g \mid E) \times P(A \mid E, \mathcal{I}^{pos})) \sim \mathcal{I}^{pos} \tag{5.4}$$

The subset from pre-demonstrations complements the size of the $\mathcal{I}^{pos}$. To create the subset from $\mathcal{I}^{pre}_{spl}$, we apply the loss probability combined with the distribution of actions in $\mathcal{I}^{pre}$, denoted as $P(A \mid \mathcal{I}^{pre})$, as shown in Equation 5.5.

$$\mathcal{I}^{pre}_{spl} = ((1 - P(g \mid E)) \times P(A \mid \mathcal{I}^{pre})) \sim \mathcal{I}^{pre} \tag{5.5}$$

Integrating the training dataset with random demonstrations offers two main benefits. Firstly, it prevents the model from overfitting on the policy demonstrations. Secondly, during the

early iterations, when the policy generates only a limited number of successful runs, the training data ensures exploration by the IDM.

Using a win-loss probability, we guide the training data to align with the expert demonstration, enhancing the model's ability to imitate the expert. In this configuration, the more frequently an agent accomplishes its goal, the less we need $\mathcal{I}^s$ consisting of $\mathcal{I}^{pre}$ and more of $\mathcal{I}^{pos}$. It's crucial to note that our method is solely goal-oriented as it considers tuples from successful runs in the subset from post-demonstration and doesn't use the reward information for learning or optimization.

We do not use rewards because not all environments have a dedicated function to provide them. However, most agents do have a goal that is relatively easy to visually identify by examining the last transition, such as the mountain car reaching the flag pole, arriving at the final square in a maze, the acrobot reaching the horizontal line, and the Cart-Pole surviving up to 195 steps, as presented in Section 4.4.

5.3 Self-Attention

The Self-Attention (SA) module [49] is a neural network component capable of contrasting global relationships within the network's internal representation. This contrast is achieved by calculating non-local responses as a weighted sum of features across all positions. The SA mechanism enables the network to concentrate on task-relevant features at each stage and establishes correlations between global features [12].

Our approach, ABCO, employs the SA module drawing from the Self-Attention Generative Adversarial Network (SAGAN) [54], which demonstrated superior performance in image synthesis. Within the SAGAN framework, the self-attention module formulates the key $f(x)$, the query $g(x)$, and the value $h(x)$, given a feature map x, through convolutional filters based on the formulas $f(x) = W_f x$, $g(x) = W_g x$, and $h(x) = W_h x$.

The construction of the attention map entails two primary steps. Initially, we employ Equation 5.6 to the present key f and query g.

$$s_{ij} = f(x_i)^T g(x_j) \tag{5.6}$$

Subsequently, we compute the softmax function $\beta_{j,i}$, which is applied to the attention module at the i^{th} position when generating the j^{th} region. Using the attention map β and the values $h(x)$, we determine the self-attention feature maps $a = (a_1, a_2, ..., a_N) \in \mathbb{R}^{C \times N}$, where N is the count of feature locations and C is the channel count, as depicted in Equation 5.7.

$$a_j = \upsilon \left(\sum_{i=1}^{N} \beta_{j,i} h(x_i) \right), \upsilon(x_i) = W_\upsilon x_i \tag{5.7}$$

In this equation, W_f, W_g, and $W_h \in \mathbb{R}^{\hat{C} \times C}$ and $W_v \in \mathbb{R}^{C \times \hat{C}}$, where $\hat{C}$ is C/k, a division applied to reduce the numbers of feature map. Furthermore, we have the SA feature map a, weighted by μ, an adaptable parameter initialized as zero.

The SA module in our approach is specifically designed to mitigate the effects of constant instabilities brought by the iterative process through a comprehensive feature weighting mechanism. This module allows the model to overlook any potential local noise an agent may introduce, enabling it to concentrate on the most pertinent features for action prediction. It also ensures more uniform weight updates due to the complete feature weighting. We hypothesize that, during the initial iterations, SA modules will learn from the random policy dataset how to weigh each state properly. This understanding will subsequently lead to more precise labeling when $\mathcal{I}^s$ predominantly consists of $\mathcal{I}^{pos}$ over $\mathcal{I}^{pre}$.

5.4 Implementation and Results

In order to test ABCO, we perform experiments using five environments available at OpenAI Gym [6] toolkit. The selected environments are separated in vector-based environments (*Acrobot-v1*, *Cart-Pole-v1*, *MountainCar-v0*), and image-base environments (*Gym-Maze* 3×3, 5×5, and 10×10). Each environment is described in Section 4.4 and illustrated in the respective Figures (4.1, 4.2, 4.5, 4.3).

We developed two distinct networks to accommodate both vector-based and image-based environments (Section 5.4.1). Then, we evaluated the results in terms of *Average Episodic Reward* (AER) and *Performance* ($\mathcal{P}$), and we present a comparative analysis with existing state-of-the-art methods (Section 5.4.2).

5.4.1 Implementation

Our experiments focus on two types of environments: the first, low-dimensional, vector-based environments and the second, high-dimensional, image-based environments. To address these distinct scenarios, we have designed two corresponding neural networks. All models have been constructed with the PyTorch framework, utilizing the Cross-Entropy loss function and the Adam optimizer. In both the IDM and PM, we have incorporated self-attention modules.

Here is the detailed composition of each network applied in our experimentation. In this context, FC_d symbolizes a fully connected layer of d dimensions, SA_d represents a self-attention layer and ... indicates the sequence of layers from the initial architecture until the next described layer:

• **Vector-based Environments**: The network starts with an input of dimensions $Input_{\text{dims}}$, followed by a series of layers: $FC_{12} \rightarrow SA_{12} \rightarrow FC_{12} \rightarrow SA_{12} \rightarrow FC_{12} \rightarrow FC_{12} \rightarrow Output_6$. Here, *dims* corresponds to a vector housing twelve and six states for the IDM and PM, respectively.

• **Image-based Environments**: We modified the ResNet architecture to accommodate these environments, integrating two self-attention modules. The layout unfolds as follows: $Input_{224\times224} \rightarrow \cdots \rightarrow ResBlock_2 \rightarrow SA_{64} \rightarrow \cdots \rightarrow ResBlock_4 \rightarrow SA_{128} \rightarrow \cdots \rightarrow FC_{\text{dims}} \rightarrow LeakyRelu \rightarrow Dropout_{0.5} \rightarrow FC_{512} \rightarrow LeakyRelu \rightarrow Dropout_{0.5} \rightarrow Output_4$. In this case, *dims* indicates a vector of 1024 and 512 features for the IDM and PM, respectively.

5.4.2 Results

To evaluate our approach, we compare it against the state-of-the-art methods. Each model is trained using the identical initial set of random pre-demonstrations, $\mathcal{I}^{pre}$. Table 5.1 presents the results in terms of Average Episodic Reward (*AER*) and Performance ($\mathcal{P}$) for our models, as well as for related work, BCO [47] and ILPO [11]. As a benchmark, we also present the results of Behavioral Cloning (BC), a supervised approach.

Our method is found to be either equivalent to or superior to the state-of-the-art in every environment except for the Maze 3×3, where we performed comparably to BCO. The general results confirm that the attention module and our sampling strategy enhance the imitation process. All models successfully achieved the maximum score for the CartPole in both *AER* and Performance, indicating that this problem is relatively straightforward to learn. Despite our model boasting the best $\mathcal{P}$and *AER* scores in the Acrobot environment, related work also presented similar results with $\mathcal{P} \approx 1.00$ and $AER = -85.300$. The superior *AER* score our model obtained implies that ABCO can solve the problem with fewer frames. However, the similar imitation capabilities of both models are apparent as all models

Table 5.1 – Comparison of Performance and Average Episodic Reward between ABCO (our approach) and related work.

Model	Metric	CartPole	Acrobot	MountainCar	Maze 3×3	Maze 5×5	Maze 10×10
BC	$\mathcal{P}$	**1.000**	1.071	1.560	-1.207	-0.921	-0.470
	AER	**500.000**	-83.590	-117.720	0.180	-0.507	-1.000
BCO	$\mathcal{P}$	**1.000**	0.980	0.948	0.883	-0.112	-0.416
	AER	**500.000**	-117.600	-150.00	**0.927**	0.104	-0.941
ILPO	$\mathcal{P}$	**1.000**	1.067	0.626	-1.711	-0.398	0.257
	AER	**500.000**	-85.300	-167.00	-0.026	-0.059	-0.020
ABCO	$\mathcal{P}$	**1.000**	**1.086**	**1.289**	**1.159**	**0.960**	**0.860**
	AER	**500.000**	**-77.900**	**-132.30**	0.908	**0.932**	**0.784**

attained $\mathcal{P} \approx 1.00$. It is worth noting that even though ABCO does not use labeled data, it still outperforms the BC approach, which relies on action labels. A noticeable difference in Performance is seen in the MountainCar, with our model reaching $\mathcal{P}$ = 1.289, about 0.34 more than the second-best result achieved by BCO.

Although we recorded the highest Performance scores in all Maze environments, we noticed that as the complexity of the environment increased, our Performance declined. However, ABCO was less affected than BCO when faced with increasing complexity. This is evident from the Performance results we achieved in Mazes 3×3, 5×5 and 10×10 which were 1.159, 0.960 and 0.860 respectively, compared to BCO's 0.883, -0.112 and -0.416. In terms of *AER*, ABCO was only surpassed by BCO in Maze 3×3 by about 0.02, where Torabi *et al.* [47] scored an *AER* = 0.927. Comparing the results of ABCO with ILPO, it can be seen that ILPO's Performance improves as the maze size increases, but it is still significantly lower than ABCO for the 10×10 Maze. We attribute this discrepancy to two key factors. First, our method incorporates an attention module (5.3), enhancing ABCO's ability to concentrate on critical features in non-visited state spaces. Second, ILPO does not account for a comprehensive view of the scenario, as it employs crop mechanisms and performs internal manipulations with the state images. Partial environmental observation means the approach could miss essential image features (e.g., the initial state, the goal state, the agent, *etc*). Conversely, as the maze size increases, ILPO gains more local information through the crops, thus increasing its Performance.

5.5 Discussion

An ablation study was performed to examine the impact of each component of our proposed method. We measured Performance and AER in scenarios using only the self-attention mechanism without sampling, the sampling strategy without self-attention, and the combination of attention with different samplings. Table 5.2 presents all results that were generated for this discussion using the Maze 5×5 environment.

5.5.1 ABCO and Self-attention

ABCO was trained using only the self-attention module to measure the impact on the learning process. It was observed that using self-attention alone led to higher model accuracy than the original method. However, high accuracy does not always imply excellent performance. Without the sampling method, some actions may not occur in later iterations, resulting in the IDM failing to predict less common actions and forming a subset of all pos-

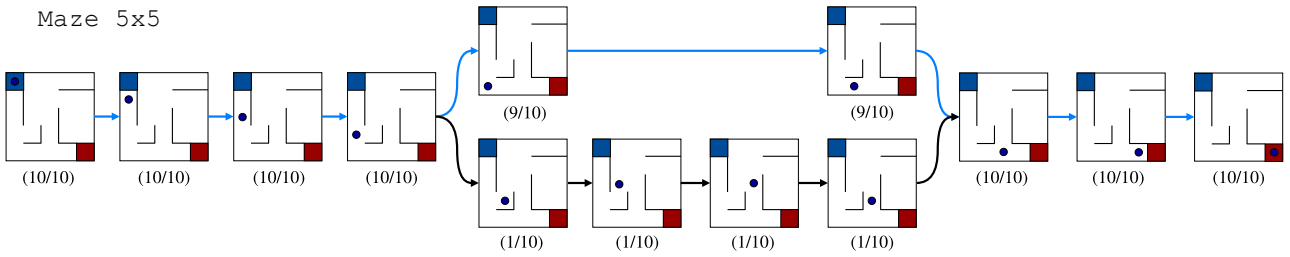


Figure 5.1 – Expert demonstrations of a 5 $\times$ 5 Gym-Maze configuration. Below each state image, we indicate the number of expert visits. The blue line depicts the path chosen by our ABCO agent.

sible actions. Consequently, the policy learns this new subset of actions rather than the real ones.

The IDM can predict the most common path even when features are weighed. The agent imitates the most common path when provided with ten different solutions for each maze, as shown in Figure 5.1. Although the self-attention mechanism yields results similar to BCO(α), its combination with the sampling method significantly impacts the results.

5.5.2 ABCO and Sampling

In this experiment, ABCO(α) was trained using only the sampling module, with the self-attention module disabled. It was hypothesized that sampling from the original random policy dataset would help solve the vanishing actions issue and close the difference between the initial iteration $\mathcal{I}$ and the expert. Vanishing of actions from the IDM prediction occurs due to the weak policy inference resulting in a $\mathcal{I}^{pos}$ that does not contain all actions or sparse representations that cause the inverse dynamic model to underfit.

During early iterations under these conditions, IDM ceases predicting classes that are the minority in the expert dataset. This misclassification leads to the policy looping between actions that hinder the model from reaching its goal. The distribution of all predictions from the IDM from BCO(α) and ABCO is compared in Figure 5.2. This comparison shows that our sampling method can predict all classes better due to the artificial growth of our dataset caused by sampling from the $\mathcal{I}^{pre}$.

Moreover, we calculated the *L2* distances from the average of all images from each action during each iteration to determine if the policy can generate samples closer to the expert than the random dataset. We normalized them between zero for the expert, and one, for the $\mathcal{I}^{pre}$ samples. The results (Figure 5.3) confirm that our model learns a policy that generates a better $\mathcal{I}$ for majority classes (e.g., S and E) and even for minority classes (e.g., N and W). The difference in the approximation of the expert dataset is assumed to be due to the minority classes consisting mostly of the $\mathcal{I}^{pre}$ since most mazes do not require those

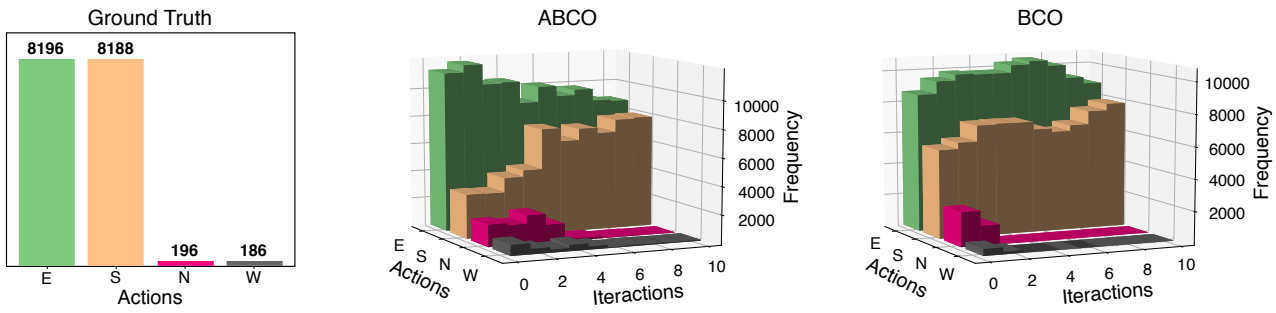


Figure 5.2 – IDM predictions of the expert examples through time.

actions. Sampling from the random dataset forces our IDM to balance its labeling and create further distant iterations. However, as the policy progresses and solves more runs, it gets closer to the expert. The new samples allow the IDM to fine-tune itself and predict expert labels more precisely.

We hypothesize that not all interactions following a sub-optimal policy are relevant for IDM's learning. To test this, we used a Resnet without attention modules and created $\mathcal{I}^s$ with all $\mathcal{I}^{pos}$ and the same ratio used in the original sampling method for all $\mathcal{I}^{pre}$. This approach resulted in lower AER and $\mathcal{P}$as expected, supporting our hypothesis.

In conclusion, the new sampling method alone can enhance the learning experience by providing a more balanced dataset to the IDM. However, when paired with the self-attention modules, it improves the model's generalization by learning to weigh each sample accordingly, further boosting the performance of our method.

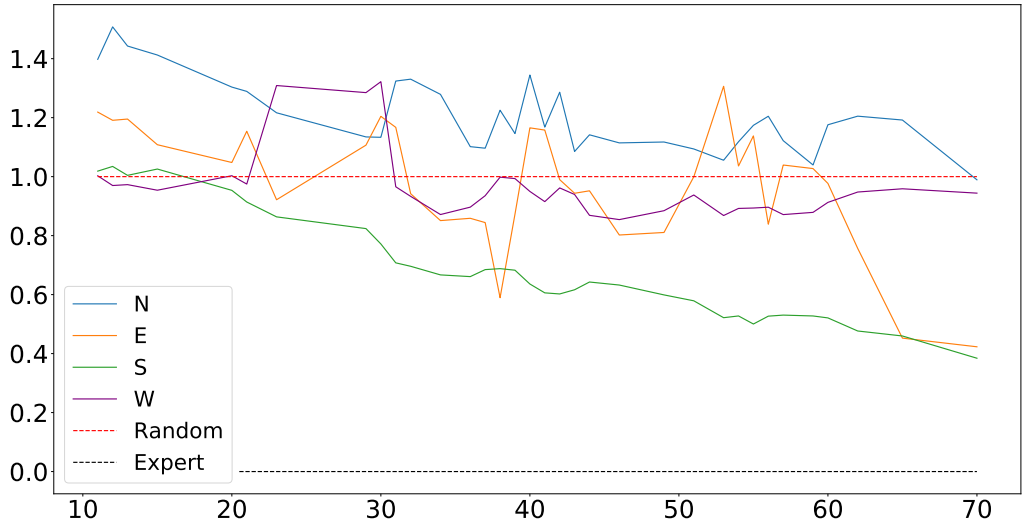


Figure 5.3 – *L2* distance for the average of each action for each iteration normalized by the expert and random samples in the 5×5 mazes.

Table 5.2 – Ablation study evaluating the impact of the *attention* and *sampling* modules in the 5×5 Maze environment.

Model	Performance	Average Episodic Reward
BCO [47]	-0.112	-0.941
Attention	-0.415	-0.940
Partial Sampling	0.717	0.716
Whole Sampling	0.628	0.676
ABCO (Attention + Partial Sampling)	0.960	0.932
ABCO (Attention + Whole Sampling)	0.759	0.755

5.6 Final Remarks

In this chapter, we proposed a novel approach, Augmented Behavior Cloning from Observation (ABCO), to address the challenging task of imitation learning. The model combines a self-attention mechanism with a unique sampling strategy, substantially overcoming the limitations associated with standard imitation learning methods. The main contributions of this work, which was later accepted for publication in the *International Joint Conference on Neural Networks* (IJCNN), are:

- **Self-attention Mechanism:** the architecture incorporates a self-attention mechanism designed to capture global features effectively. This feature allows the model to concentrate on crucial aspects of the input space, thereby facilitating a more refined understanding of complex environments.
- **Sampling Strategy:** we have developed and implemented a unique sampling strategy to regulate the observations used for learning. By actively managing the balance of the samples, our strategy avoids overfitting to more common actions and enhances the model's ability to predict less frequent but essential actions.
- **Empirical Validation:** we conducted extensive empirical evaluations on four environments where ABCO significantly outperformed state-of-the-art methods. These experiments highlight the model's robustness and efficacy in learning from observation and underscore its capacity to work with low-dimensional and raw image data.

6. IMITATING UNKNOWN POLICIES VIA EXPLORATION

This chapter presents Imitating Unknown Policies via Exploration (IUPE). IUPE advances significantly over Augmented Behavior Cloning from Observation (ABCO), which was presented in the previous chapter. ABCO provided a foundational framework by improving the gaps found in the original BCO framework, but encountered challenges in the initial stages of training due to the application of maximum-likelihood estimation in situations of uncertainty.

IUPE mitigates these issues by incorporating exploration and sampling strategies, enhancing model performance by effectively avoiding local minima. Further, IUPE integrates self-attention modules, augmenting ABCO's approach and facilitating the model's focus on the most relevant aspects of the input space. This strategy improves generalization and allows the model to understand complex environments more efficiently.

IUPE has demonstrated its effectiveness by outperforming ABCO and other state-of-the-art behavior cloning methods, reinforcing its potential in imitation learning. Additionally, IUPE contributes to a deeper understanding of the influence of exploration mechanisms in the imitation learning paradigm and the evolution of distances from $\mathcal{I}^{pre}$ and $\mathcal{I}^{pos}$ during iterative processes.

Recognizing these advancements, IUPE was published in the proceedings of the British Machine Vision Conference (BMVC), affirming its position as a significant contribution to imitation learning.

6.1 Sampling Method

IUPE employs a strategy similar to ABCO's sampling approach. It selects from the post-demonstration using only the runs that have met the goal of the environment, as depicted in Equation 5.3. However, following this selection of successful runs, IUPE incorporates all runs from $\mathcal{I}^{pos}$ rather than re-sampling according to the win probability.

In the subsequent steps, IUPE selects from the pre-demonstrations, $\mathcal{I}^{pre}_{spl}$, in alignment with the inverse probability of the post-demonstrations, essentially, the loss probability distribution as given by $1 - P(A \mid E; \mathcal{I}^{pos})$. To simplify, the samples (observations) that constitute the new post-demonstration dataset are selected proportionately to $P(A \mid E; \mathcal{I}^{pos})$ for winning executions. The dataset is then populated with the pre-demonstrations in proportion to the number of loss runs. IUPE and its sampling strategy are outlined in Algorithm 6.1.

It is important to mention that IUPE and ABCO are both goal-aware methods, and they do not rely on any reward information for learning or model optimization. The idea

Algorithm 6.1 Imitating Unknown Policies via Exploration (IUPE)

Initialize model $\mathcal{M}_\theta$ as a random approximator
Initialize policy π_ϕ with random weights
Generate $\mathcal{I}^{pre}$ using policy π_ϕ
Generate state transitions $\mathcal{T}^e$ from demonstrations D
Set $\mathcal{I}^s = \mathcal{I}^{pre}$
Let α be the number of improvement cycles
for $i \leftarrow 0$ to α **do**
    Improve $\mathcal{M}_\theta$ by trainIDM($\mathcal{I}^s$)
    Use $\mathcal{M}_\theta$ with $\mathcal{T}^e$ to predict actions $\hat{A}$
    Improve π_ϕ by behavioralCloning($\mathcal{T}^e$, $\hat{A}$)
    for $e \leftarrow 1$ to $|E|$ **do**
        Use π_ϕ to solve environment e
        Append samples $\mathcal{I}^{pos} \leftarrow (s_t, \hat{a}_t, s_{t+1})$
        if π_ϕ at goal g **then**
            Append $v_e \leftarrow 1$
        else
            Append $v_e \leftarrow 0$
        end if
    end for
    Set $\mathcal{I}^s$ = sampling($\mathcal{I}^{pre}$, $\mathcal{I}^{pos}$, $P(g|E)$, v_e)
end for

behind this choice is that not all environments offer intuitive reward functions conducive to problem-solving.

6.2 Exploration

The original behavioral cloning framework employs a *maximum a posteriori* (MAP) estimation technique, predicting the most probable action given a pair of states according to the model. This applies to both its original form and the α-iterations variant. However, we noted that in the initial iterations, the model often has a level of uncertainty about the correct action, leading to local minima that are not desirable.

To tackle this, we adopt a straightforward solution from the domain of language modeling: we sample actions from the softmax distribution of both models (the IDM during expert labeling and the PM during environment execution) instead of using MAP estimations. This strategy generates a stochastic policy that enables more thorough exploration in the early iterations, taking into account the uncertainty of the model. We demonstrate that this sampling strategy not only facilitates the convergence of IDM in fewer iterations but also ensures a more diverse dataset composed of $\mathcal{I}^{pre}$ and $\mathcal{I}^{pos}$. Further, the stochastic policy enhances the exploration of the search space for effectively achieving the environment goal.

Additionally, a stochastic policy in dynamic environments is crucial in reaching the goal where deterministic behavior falls short. This distinction is essential for avoiding local minima during iterations. In scenarios where the model cannot sample a sub-optimal action during the training phase, the agent's actions may default to the most common action in the expert samples. With this preference to sample the most frequent action, the policy risks getting caught in a loop between states, *e.g.*, alternating between left and right in a Maze

environment. Thus, our method mitigates such instances, contributing to a more effective learning process.

6.3 Experimental Results

This section presents the experiment details and the final results obtained during the IUPE implementation. Our experimental framework is grounded in four diverse environments derived from OpenAI Gym [6] (see in Section 4.4). These environments encompass vector-based scenarios such as *Acrobot-v1*, *Cart-Pole-v1*, *MountainCar-v0*, and image-based scenarios such as *Gym-Maze* with variations in dimensions (3×3, 5×5, and 10×10). All experimental runs, including the benchmark models, extend across 100 epochs utilizing the same set of expert data.

For each environment type, we created two unique networks: one catering to low-dimensions, *vector-based environments*, and the other tailored to high-dimensions, *image-based environments*. All the networks were built on the *PyTorch* platform. The networks utilized the Adam optimizer [22] to minimize the cross-entropy loss function. Consistent with the network topology implemented in the preceding ABCO (Chapter 5) method, as described in the previous chapter, we integrated self-attention modules [49, 54] into both the IDM and PM.

To evaluate IUPE, and following the metrics presented in the methodology chapter of this thesis (Chapter 4), we applied the *Average Episodic Reward* (AER) and *Performance* ($\mathcal{P}$). The AER represents the mean reward across 100 runs for each tested environment. Given that AER is contingent upon an environment's reward function, its numeric value varies across tasks. Essentially, AER quantifies the proficiency of the expert in executing the task, thereby indicating the degree of difficulty for the agent to mirror the expert's behavior. The AER is ascertained by averaging 100 distinct mazes for the Gym-maze environment and 100 sequential runs for the remaining environments.

6.3.1 Results

We evaluate IUPE by comparing it with the leading methods in IfO: BCO [47] and ILPO [11]). The performance outcomes of each technique are represented in Table 6.1. For the purpose of making an effective comparison, we incorporate the results (in terms of both AER and $\mathcal{P}$) of an expert, a randomly acting policy, and behavioral cloning (supervised fashion). All models are trained on the same starting set of random pre-demonstrations, denoted as $\mathcal{I}^{pre}$.

Models	Metrics	CartPole	Acrobot	MountainCar	Maze 3×3	Maze 5×5	Maze 10×10
Expert	$\mathcal{P}$	1.000	1.000	1.000	1.000	1.000	1.000
	AER	442.628	−110.109	−147.265	0.963	0.970	0.981
Random	$\mathcal{P}$	0.000	0.000	0.000	0.000	0.000	0.000
	AER	18.700	−482.600	−200.000	0.557	0.166	−0.415
BC	$\mathcal{P}$	1.135	1.071	1.560	−1.207	−0.921	−0.470
	AER	500.000	−83.590	−117.720	0.180	−0.507	−1.000
BCO	$\mathcal{P}$	**1.135**	0.980	0.948	0.883	−0.112	−0.416
	AER	**500.000**	−117.600	−150.000	**0.927**	0.104	−0.941
ILPO	$\mathcal{P}$	**1.135**	1.067	0.626	−1.711	−0.398	0.257
	AER	**500.000**	−85.300	−167.000	−0.026	−0.059	−0.020
IUPE	$\mathcal{P}$	**1.135**	**1.086**	**1.314**	**1.361**	**1.000**	**1.000**
	AER	**500.000**	**−78.100**	**−130.700**	**0.927**	**0.971**	**0.981**

Table 6.1 – *Performance* and *Average Episode Reward* for IUPE and related work.

Overall Results: the results, shown in Table 6.1, illustrate IUPE's superior performance over leading approaches in all environments, but for CartPole, where it is similar to the baseline results. Predictable results are observed in the CartPole environment, given its relatively simple state representation of four dimensions and easily distinguishable actions. In the Acrobot environment, IUPE and ILPO demonstrate near equivalent Performance ($\mathcal{P} \approx$ 1.00), reflecting their comparable imitation capabilities. However, IUPE delivers an *AER* = −78.10, surpassing ILPO's score of −85.30, indicating IUPE's ability to resolve the environment approximately 10 frames faster than ILPO. In the MountainCar environment, IUPE achieved a $\mathcal{P}$= 1.314, setting the highest benchmark compared to the baselines in this environment, surpassing the next best result (BCO) by approximately 0.37. IUPE also surpassed others with a considerably better AER value in the MountainCar environment: −130.70, which is 19.3 more effective than BCO. In the Maze environments, IUPE confirms the highest $\mathcal{P}$values across all tested Maze types. As for AER, IUPE was only matched by BCO in the 3×3 Maze, where both models scored *AER* = 0.927. Unlike the Performance of other approaches that decline as Maze complexity increases, IUPE remains relatively stable. IUPE's robustness can be attributed to the stochastic nature of the model, which facilitates a broader exploration of the environment, avoids local minima, and ensures goal achievement. When compared with the *Expert*, IUPE either matches or outperforms it in most of the tested environments.

IUPE *vs.* BCO: IUPE, as an enhanced version of BCO, reflects a significant positive impact on the imitation process in the results. Across all environments, IUPE consistently matches or exceeds the performance of BCO. In the 3×3 Maze, both IUPE and BCO earned the same AER score, attributable to the methods solving the same number of environments. However, IUPE surpassed BCO in the same environment with higher reward scores. Importantly, as the number of states in the environment escalates, IUPE continues to learn, differentiating from the BCO method.

Model	$\mathcal{P}$	AER
BCO	-0.416	-0.941
Attention	-0.415	-0.940
Sampling	0.534	0.348
Exploration	0.734	0.605
Attention + Sampling	0.367	0.088
Attention + Exploration	-0.407	-0.921
Sampling + Exploration	0.943	0.901
Attention + Sampling + Exploration (IUPE)	**1.000**	**0.981**

Table 6.2 – Ablation study considering IUPE's 3 main components in the maze environment.

IUPE *vs.* ILPO: in the CartPole and Acrobot environments, IUPE and ILPO display similar results. However, IUPE outperforms with superior $\mathcal{P}$ and AER scores in the MountainCar and for all Maze environments. In the MountainCar scenario, IUPE achieved a $\mathcal{P}$= 1.314, demonstrating its ability to imitate effectively and even outcoming the expert results. With respect to AER, IUPE achieved a score of -130.7 compared to ILPO's -167, representing that our method achieved superior rewards in MountainCar across 100 episodes. When tested in Maze environments, ILPO's AER scores fell below zero, suggesting that it was unable to solve the majority of the mazes. We believe that ILPO's underperformance may occur from its failure to incorporate the complete scenario image as it employs crop mechanisms and manipulates state images. Such practices could inadvertently exclude crucial elements from the images (*e.g.*, the agent, the goal state, etc.) during training.

6.4 Discussion

We discuss three key strategies that we explore in the context of behavioral cloning from observations: *self-attention*, *sampling*, and *exploration* over *maximization*. We run several experiments in order to analyze the impact of each feature separately, as well as combined with the other features. Table 6.2 shows the results of this analysis for the 10×10 Maze environment, where BCO represents the approach without considering any improvement.

6.4.1 Self-Attention

The usage of self-attention modules by IUPE may result from having similar results when compared with BCO's performance in terms of $\mathcal{P}$ and AER while also converging faster during the training phase. The significance of the SA modules becomes more evident when transitioning from $\mathcal{I}^s$ to $\mathcal{T}^e$ due to the comprehensive feature weighting provided by the attention mechanism, which preserves high accuracy across epochs. This phenomenon is clarified in Figure 6.1, where we use the Grad-CAM [43] technique to illustrate the self-

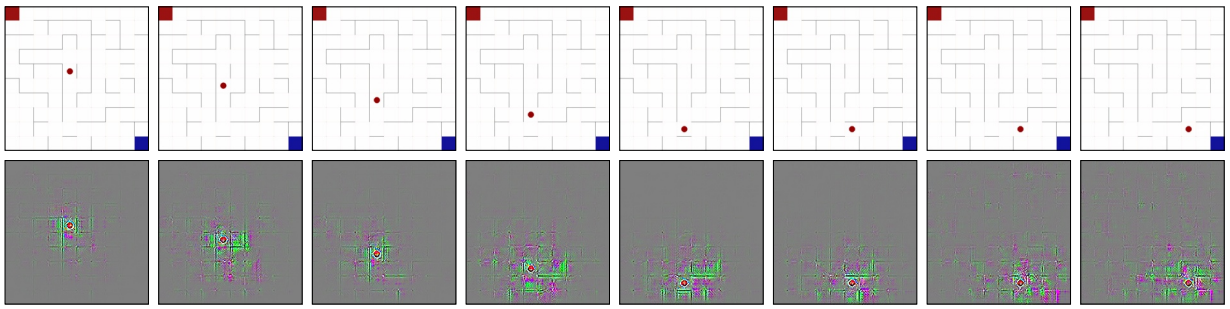


Figure 6.1 – Heatmap visualization of the gradient filters activating for the maze environment. The first row shows the input image, while the second row shows the gradient activation.

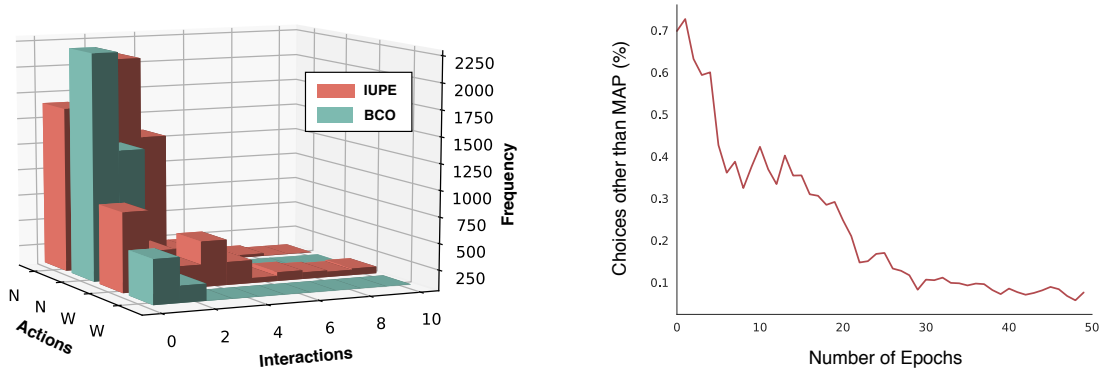
attention gradient activations from an image incorporated in a trained policy. After examining the gradient activations, it is inferred that the self-attention modules direct the model's focus towards the agent while simultaneously accounting for proximal walls. Moreover, it is observed that activation regions are more expansive when the agent navigates through open passages compared to corridors. This pattern mirrors human visual perception, as evidenced in the first and third frames, where the agent is between two walls, contrasted with other frames where the agent possesses a wider viewpoint. Nevertheless, the absence of the sampling method and the use of BCO's original reconstruction for the IDM dataset results in action predictions disappearing from the model prediction distribution, as depicted in Figure 6.2(a). Such an issue might occur when the PM fails to execute all actions or behaves differently from the expert, thereby distancing the feature space from the expert actions.

IUPE demonstrates less score variability during the validation phase comparable to BCO, suggesting that our model has a superior ability to discern the appropriate action from the state tuples. After assessing the influence of self-attention (Table 6.2) and the issue of vanishing actions, we conducted an investigation of the sampling mechanism that guides the observations input into the IDM, hence preventing it from settling into sub-optimal local minima.

6.4.2 Sampling

To verify the impact of sampling, we can see that Table 6.2 reveals that the mere application of the sampling method can ameliorate the vanishing action issue, thus marking a significant advancement over both BCO and self-attention in isolation. Given that the sampling procedure either retains $\mathcal{I}^{pre}$ in entirety or in fragments during the iterations, it ensures the IDM continues to receive a selection of random actions as inputs, thereby leading to more evenly distributed predictions.

As the training progresses and $\mathcal{I}^{s}$ increasingly becomes more representative of $\mathcal{I}^{pos}$ than $\mathcal{I}^{pre}$, we observe a shift in the action distribution by the IDM that guides the PM to



(a) (b)

Figure 6.2 – (a) IDM predictions for the expert examples through time. (b) Percentage of choices in which the MAP estimation is not selected by the self-decaying exploration rate.

concentrate more on actions that frequently appear in the expert's actions. We hypothesize that this skewing of $\mathcal{I}^s$ facilitates the IDM's understanding that the probability of all actions is not equal, and thus it should prioritize certain actions over others.

In conjunction with the SA module, the sampling mechanism causes the PM to yield less favorable results when compared to the sole use of the sampling method, although it still surpasses BCO in performance. Upon investigating the cause of this performance degradation, we discovered that the *Attention+Sampling* model provided higher per-action accuracy during validation. Even though it demonstrated greater confidence in action selection, the agent was also considerably more prone to becoming trapped between states, predicting an action that would revert the agent to a prior state and leading to a cyclical pattern of movement back and forth.

We hence observed that the new sampling method alone could enhance the learning experience across iterations, but when paired with SA, it inadvertently compromised the model's generalization capacity. Given that the issue lay in the excessive certainty associated with a few actions, we decided to experiment further with the stochastic mechanism we labeled as *exploration*.

6.4.3 Exploration over Maximization

Stochastic learning algorithms typically decrease their exploration rate over time, under the assumption that the agent incrementally discovers the optimal solution as time passes [30]. If the rate of decay is too slow, the agent may settle on the first solution it encounters, while an excessively rapid decay might cause the agent to expend too much time investigating sub-optimal states. By employing the softmax distribution of actions, we develop an exploration mechanism that gradually diminishes naturally as the neural network

improves at segregating the feature space (thereby eliminating the need for tuning a decay hyperparameter). The self-decaying exploration rate can be observed in Figure 6.2(b).

To assess whether IUPE derives any benefit from this exploration rate, we can refer to Table 6.2 and observe the impact of exploration on the baseline (BCO), as well as in combination with the other features (attention and sampling). By solely deploying this mechanism, we manage to address the issue of repetitive cycling between actions, and while the PM may achieve the goal via non-optimal paths, it facilitates the generation of samples that more closely mimic the expert than before.

When the exploration mechanism is paired with the sampling method, it yields results comparable to IUPE (complete method incorporating all 3 strategies). We understand that this is due to the increased stochasticity that helps in disrupting loops of actions. While using exploration with SA results in a model with superior per-action accuracy, we observe a decrease in both $\mathcal{P}$ and AER. We hypothesize that this occurs due to the vanishing action problem triggered by the absence of the sampling method. When all mechanisms are integrated, the SA modules no longer negatively impact the model, instead, they enhance both $\mathcal{P}$ and AER. With the exploration mechanism supporting the model and preventing it from getting trapped between states, IUPE can refine its predictions earlier because the non-optimal actions are more similar to the expert. With the inclusion of the sampling mechanism, which is responsible for balancing $\mathcal{I}^s$, IUPE yields the best results in this analysis.

6.5 Final Remarks

In this chapter, we introduced a new approach, Imitating Unknown Policies via Exploration (IUPE), that aims to enhance imitation learning performance. IUPE uniquely integrates a self-attention mechanism, a strategic sampling methodology, and an exploration-driven learning mechanism in order to significantly advance conventional imitation learning methods. Our proposed approach was published in the proceedings of the British Machine Vision Conference (BMVC).

The key contributions of IUPE are:

- **Self-attention Mechanism:** IUPE's architecture employs a self-attention mechanism that effectively captures global features from the input data. This feature enables the model to focus on the most vital aspects of the input space, thus offering a more nuanced understanding of complex environments and actions.

- **Sampling Strategy:** we have developed a novel sampling strategy that regulates the input observations used for the imitation learning process. This strategy not only ensures the balance of the samples across different actions but also prevents overfitting

to prevalent actions, thereby improving the model's ability to predict less frequent yet crucial actions.

- **Exploration-driven Learning:** IUPE introduces a mechanism of exploration over maximization that offers a dynamic, self-decaying exploration rate for action selection, enhancing the model's performance and ability to imitate unknown policies effectively.
- **Empirical Validation:** we conducted an exhaustive series of empirical evaluations on several environments, demonstrating that IUPE significantly outperforms state-of-the-art imitation learning methods. The results highlight the model's robustness and effectiveness in learning from observations and its capacity to work with both low-dimensional data and raw image data.

7. RESILIENCE OVER SUB-OPTIMAL SAMPLES

Imitation Learning (IL) algorithms aim to emulate the behaviors of experts to execute specific tasks. Nevertheless, the potential outcomes of these strategies when they learn from sub-optimal demonstrations are still uncertain. By exploring how IL methods adapt when faced with varying levels of observational quality, we can enhance areas such as data collection optimization, model interpretability, reduction of bias in sub-optimal experts, and beyond. This chapter presents an extensive set of experiments to analyze how diverse IL methods learn under an array of expert optimality degrees. In this study, we selected four IL algorithms, three that learn through a self-supervised approach and one that uses ground-truth labels to learn in supervised mode (Behavioral Cloning) in four different environments. We subsequently compare their performances when using optimal and sub-optimal expert demonstrations. Our research reveals that Self-Supervised IL methods demonstrate considerable resilience to sub-optimal experts, a characteristic not presented by the supervised approach. Interestingly, sub-optimal experts can occasionally provide benefits, as they serve as a form of regularization technique, protecting models against overfitting.

7.1 Experimental Design

In this study, we aim to investigate the resilience of IL algorithms to varying qualities of expert samples. Theoretically, if an agent learns behavior from sub-optimal trajectories, it should perform sub-optimally. That is, it should display lower performance compared to learning from an optimal expert. Still, we hypothesize that some IL algorithms will be able to bridge this disparity. Therefore, we search the available literature for various IL methods that do not depend on direct environment supervision through self-experience, such as a reward signal.

While many recent IL methods propose hybrid strategies that use the advantages of reinforcement learning [21, 7], we believe that utilizing a direct signal from the environment about action optimality would undermine the objective of these experiments. Thus, we have intentionally decided against including approaches like MobILE [21] in our experimental framework. Additionally, we exclude recent IL methods that necessitate any action information from the expert, such as OPOLO [55].

For this study, we selected four different methods: Behavioral Cloning (BC) [34], Generative Adversarial Imitation Learning (GAIL) [17], Imitating Latent Policies from Observation (ILPO) [11] and Imitating Unknown Policies via Exploration (IUPE) [13]. All the other methods, but IUPE, presented in the previous chapter, can be seen in greater depth in Chapter 3.

Table 7.1 – Average Episodic Reward (AER) for experts with decreasing quality. Expert[1] is the optimal expert, while Expert[5] is the worst-performing expert in our experimental analyses.

Experts	CartPole	Acrobot	MountainCar	LunarLander
Expert[1]	500 ± 0.00	-82 ± 22.46	-99 ± 8.60	228 ± 62.33
Expert[2]	435 ± 70.51	-97 ± 24.41	-129 ± 25.50	151 ± 106.18
Expert[3]	354 ± 137.07	-138 ± 58.15	-137 ± 25.89	98 ± 103.76
Expert[4]	220 ± 47.42	-199 ± 79.85	-149 ± 32.02	60 ± 119.35
Expert[5]	112 ± 7.50	-243 ± 136.82	-156 ± 47.76	4 ± 110.92

To measure the quality of each approach, we use the same two metrics presented in the methodology of this thesis (Chapter 4): *Performance* ($\mathcal{P}$) and *Average Episodic Reward* (AER). As environments, we selected CartPole, Acrobot, MountainCar and LunarLander, all of them presented and explained in depth in Section 4.4.

In order to generate the expert samples, we train different policies, primarily with DQN [29] and PPO [42] algorithms, with different levels of optimality. We define optimality in these contexts as a threshold in accumulated reward over 100 episodes. When training an agent to act as the expert, if it meets the predetermined threshold (specific values can be seen in Table 7.1), we utilize its behavior for the creation of the (expert) dataset. It is important to note that the AER values in Table 7.1 decrease, implying that Expert[1] outperforms Expert[2], and so forth. The provided values for each expert in the corresponding environment are averages over $1,000$ episodes. We use the same number of episodes ($1,000$) per environment to train each IL method in our experimental analysis.

7.2 Results

In this study, we focus on the performance of different algorithms when provided with non-optimal samples for each domain. We execute each algorithm detailed in the previous section 10 times, using the data from each expert for 100 epochs. The network topology is the same for each approach, which consists of a multi-layer perceptron containing two layers and 32 neurons in each layer.

Table 7.2 displays the average and standard deviation of both metrics for each algorithm when using Expert[1] for each environment. The corresponding AER values for the experts are also included.

CartPole was the simplest environment for all methods to learn from, with all approaches achieving a performance score of 1. The second most accessible environment was Acrobot, with all algorithms reaching an average Performance score of 0.865. In this

Table 7.2 – Performance ($\mathcal{P}$) and Average Episodic Reward (AER) for all algorithms on all environments using the optimal expert.

Experts	Metrics	CartPole	Acrobot	MountainCar	LunarLander
Random	AER	18.7 ± 0	-482.6 ± 0	-200 ± 0	-182.72 ± 0
	$\mathcal{P}$	0 ± 0	0 ± 0	0 ± 0	0 ± 0
Expert[1]	AER	500 ± 0	-82 ± 22.46	-99 ± 8.60	228 ± 62.33
	$\mathcal{P}$	$\mathbf{1 \pm 0}$	1 ± 0	1 ± 0	1 ± 0
BC	AER	$\mathbf{500.0 \pm 0.0}$	-93.6 ± 18.5	-200.0 ± 0.0	$\mathbf{-80.9 \pm 188.15}$
	$\mathcal{P}$	$\mathbf{1 \pm 0}$	0.97 ± 0.0	0.0 ± 0.0	$\mathbf{0.25 \pm 0.03}$
GAIL	AER	$\mathbf{500.0 \pm 0.0}$	-286.23 ± 90.86	-200.0 ± 0.0	-92.26 ± 114.67
	$\mathcal{P}$	$\mathbf{1 \pm 0}$	0.48 ± 0.15	0.0 ± 0.0	0.22 ± 0.13
ILPO	AER	$\mathbf{500.0 \pm 0.0}$	$\mathbf{-75.65 \pm 12.85}$	-184.54 ± 10.06	-98.46 ± 56.54
	$\mathcal{P}$	$\mathbf{1 \pm 0}$	$\mathbf{1.02 \pm 0.0}$	0.15 ± 0.31	0.2 ± 0.07
IUPE	AER	$\mathbf{500.0 \pm 0.0}$	-87.26 ± 22.19	$\mathbf{-166.97 \pm 18.34}$	-81.34 ± 74.5
	$\mathcal{P}$	$\mathbf{1 \pm 0}$	0.99 ± 0.05	$\mathbf{0.38 \pm 0.15}$	0.25 ± 0.25

context, ILPO achieved the highest reward of -75.42 (surpassing the expert), while GAIL recorded the lowest reward of -290.57.

One significant observation is the results exhibited by IUPE in the LunarLander environment. Despite not reaching the expert's proficiency, IUPE performed on par with BC. It is important to remember that BC is a robust baseline with access to the expert's actions at every state, also known as ground-truth labels.

Comparing the results from CartPole and Acrobot with those from MountainCar and LunarLander, where the average performance for all algorithms was approximately 0.095 and 0.2425, respectively, we can infer that: (i) High rewards are closely correlated to accurate overall trajectories in both MountainCar and LunarLander. (ii) In environments where trajectories can be noisy yet contain a few correctly executed actions, non-exploratory agents that operate locally tend to outperform their exploratory counterparts. This tendency is evident in the results observed in the Acrobot and CartPole environments.

Table 7.3 provides the results for all methods when using Expert[2]. CartPole maintains its status as the easiest environment for all methods, except for BC, which cannot achieve a Performance score of 1. Given that a reward of 195 is considered as solving the CartPole environment, however, it is expected that other algorithms will retain similar outcomes even as we introduce more sub-optimal experts.

Regarding the Acrobot environment, with an average performance score of 0.905, all IL approaches demonstrate a minor decline in Performance compared to the optimal expert. This pattern emerges from Acrobot's nature towards random actions and further supports our secondary conclusion drawn from Table 7.2.

For the MountainCar environment, a reduction in Performance is observed across all approaches. This is expected as maintaining momentum is essential for solving the task. A single non-optimal action can result in a significant divergence from the optimal trajectory, making it challenging for IL algorithms to recover from unseen states [18].

Table 7.3 – Performance ($\mathcal{P}$) and Average Episodic Reward (AER) for all algorithms on all environments using Expert^2.

Algorithms	Metrics	CartPole	Acrobot	MountainCar	LunarLander
Random	AER	18.7 ± 0	-482.6 ± 0	-200 ± 0	-182.72 ± 0
	$\mathcal{P}$	0 ± 0	0 ± 0	0 ± 0	0 ± 0
Expert^2	AER	435 ± 70.51	-97 ± 24.41	-129 ± 25.50	151 ± 106.18
	$\mathcal{P}$	1 ± 0	1 ± 0	1 ± 0	1 ± 0
BC	AER	415.84 ± 81.46	-103.36 ± 18.0	-200.0 ± 0.0	$\mathbf{32.45 \pm 63.39}$
	$\mathcal{P}$	0.95 ± 0.04	0.98 ± 0.0	0.0 ± 0.0	$\mathbf{0.64 \pm 0.01}$
GAIL	AER	$\mathbf{500.0 \pm 0.0}$	-253.55 ± 84.78	-199.26 ± 2.73	-100.21 ± 99.02
	$\mathcal{P}$	$\mathbf{1.16 \pm 0.0}$	0.61 ± 0.17	0.0 ± 0.1	0.23 ± 0.14
ILPO	AER	$\mathbf{500.0 \pm 0.0}$	$\mathbf{-76.07 \pm 13.48}$	-192.7 ± 2.63	-89.06 ± 76.45
	$\mathcal{P}$	$\mathbf{1.16 \pm 0.0}$	$\mathbf{1.05 \pm 0.0}$	0.0 ± 0.4	0.26 ± 0.11
IUPE	AER	489.61 ± 9.06	-120.36 ± 36.35	$\mathbf{-190.26 \pm 11.03}$	-57.93 ± 91.75
	$\mathcal{P}$	1.16 ± 0.24	0.98 ± 0.16	$\mathbf{0.12 \pm 0.22}$	0.35 ± 0.22

Table 7.4 – Performance ($\mathcal{P}$) and Average Episodic Reward (AER) for all algorithms on all environments using Expert^3.

Experts	Metrics	CartPole	Acrobot	MountainCar	LunarLander
Random	AER	18.7 ± 0	-482.6 ± 0	-200 ± 0	-182.72 ± 0
	$\mathcal{P}$	0 ± 0	0 ± 0	0 ± 0	0 ± 0
Expert^3	AER	354 ± 137.07	-138 ± 58.15	-137 ± 25.89	98 ± 103.76
	$\mathcal{P}$	1 ± 0	1 ± 0	1 ± 0	1 ± 0
BC	AER	181.99 ± 78.5	-130.94 ± 25.4	-200.0 ± 0.0	$\mathbf{107.48 \pm 101.49}$
	$\mathcal{P}$	0.49 ± 0.02	1.02 ± 0.0	0.0 ± 0.0	$\mathbf{1.03 \pm 0.03}$
GAIL	AER	$\mathbf{500.0 \pm 0.0}$	-248.78 ± 82.66	-199.96 ± 1.6	-96.88 ± 106.57
	$\mathcal{P}$	$\mathbf{1.44 \pm 0.0}$	0.66 ± 0.14	0.0 ± 0.03	0.29 ± 0.19
ILPO	AER	500.0 ± 0.0	$\mathbf{-75.63 \pm 14.05}$	-199.82 ± 4.36	-90.88 ± 68.94
	$\mathcal{P}$	1.44 ± 0.0	$\mathbf{1.18 \pm 0.0}$	0.0 ± 0.32	0.33 ± 0.06
IUPE	AER	177.73 ± 85.75	-85.59 ± 22.59	$\mathbf{-173.5 \pm 14.55}$	-190.7 ± 89.16
	$\mathcal{P}$	0.63 ± 0.38	1.15 ± 0.07	$\mathbf{0.43 \pm 0.3}$	0.12 ± 0.46

In contrast, the results for the LunarLander environment are quite unexpected. Despite a 50-point reduction in the expert result, they are transitioning from trajectories that solve the environment to those that do not. The Performance increases for BC and remains nearly unchanged for the remaining approaches. This intriguing behavior suggests that a sub-optimal expert might assist in generalization despite the high correlation between LunarLander's trajectories and the final reward. When faced with slightly less optimal trajectories that do not solve the environment, the IL agent avoids overfitting to specific behaviors and potentially learns to diverge from landing positions observed during training. This unexpected adjustment leads to a surprising enhancement in terms of generalization.

Table 7.4 presents the results for Expert^3. In this setup, we observe a drop in AER and $\mathcal{P}$for IUPE, alongside BC, in the CartPole environment. IUPE's success is more strongly

Table 7.5 – Performance ($\mathcal{P}$) and Average Episodic Reward (AER) for all algorithms on all environments using Expert[4].

Experts	Metrics	CartPole	Acrobot	MountainCar	LunarLander
Random	AER	18.7 ± 0	-482.6 ± 0	-200 ± 0	-182.72 ± 0
	$\mathcal{P}$	0 ± 0	0 ± 0	0 ± 0	0 ± 0
Expert[4]	AER	220 ± 47.42	-199 ± 79.85	-149 ± 32.02	60 ± 119.35
	$\mathcal{P}$	1 ± 0	1 ± 0	1 ± 0	1 ± 0
BC	AER	372.76 ± 120.85	-179.18 ± 26.72	-200.0 ± 0.0	$\mathbf{71.48 \pm 136.87}$
	$\mathcal{P}$	1.76 ± 0.05	1.07 ± 0.0	0.0 ± 0.0	$\mathbf{1.04 \pm 0.05}$
GAIL	AER	498.28 ± 16.08	-316.42 ± 79.66	-200.0 ± 0.37	-88.32 ± 91.24
	$\mathcal{P}$	2.39 ± 0.04	0.53 ± 0.2	0.0 ± 0.0	0.39 ± 0.16
ILPO	AER	$\mathbf{500.0 \pm 0.0}$	$\mathbf{-75.78 \pm 13.8}$	-190.47 ± 5.25	-103.39 ± 80.0
	$\mathcal{P}$	$\mathbf{2.39 \pm 0.0}$	$\mathbf{1.43 \pm 0.0}$	0.0 ± 0.67	0.32 ± 0.09
IUPE	AER	357.65 ± 61.64	-338.1 ± 40.02	$\mathbf{-161.05 \pm 20.11}$	-79.54 ± 110.0
	$\mathcal{P}$	1.84 ± 0.69	0.24 ± 0.64	$\mathbf{0.8 \pm 0.51}$	0.49 ± 0.2

connected to the expert's optimality than GAIL and ILPO, which can maintain their maximum results (500 reward points) in this environment.

In the Acrobot environment, all algorithms sustain their performance with minimal degradation in reward, with their standard deviations remaining largely consistent. IUPE continues to perform best in the MountainCar environment, reaching -172.92 of AER, close to the optimal expert. Although IUPE tracks behind ILPO in terms of overall reward, its stability, indicated by standard deviation values, surpasses ILPO in most environments, except CartPole.

The results from LunarLander in Table 7.4 further underscore the point that when dealing with an environment that necessitates specific trajectories (*e.g.*, landing within the flags), fewer optimal experts can aid IL algorithms in achieving better generalization. However, it is important to note that good performance here (behavior similar to the expert) may not indicate the optimal behavior, given that the expert is flawed. Thus, achieving $\mathcal{P}$ of 1 no longer correlates with optimal behavior.

Table 7.5 presents the results for Expert[4]. These results indicate a mid-way point in solving each environment, except for the CartPole environment, which maintains a reward score of 195. We expected continual performance degradation for IUPE and BC during these trials in the Acrobot and CartPole environments. However, this occurred only in the Acrobot experiments.

In the CartPole environment, while ILPO and GAIL retained the maximum reward, as expected, an intriguing reward increase was noticed for both BC and IUPE. Furthermore, an increased standard deviation was evident in the 10 runs of BC. IUPE consistently performed well in the MountainCar environment, despite reducing expert optimality. These observations, associated with the ILPO performance on Acrobot, lead us to conclude that the intrinsic explorative nature of these methods primarily drives this behavior. While ILPO only relies on expert and random samples, IUPE constantly receives varied samples for its

Table 7.6 – Performance ($\mathcal{P}$) and Average Episodic Reward (AER) for all algorithms on all environments using Expert^5.

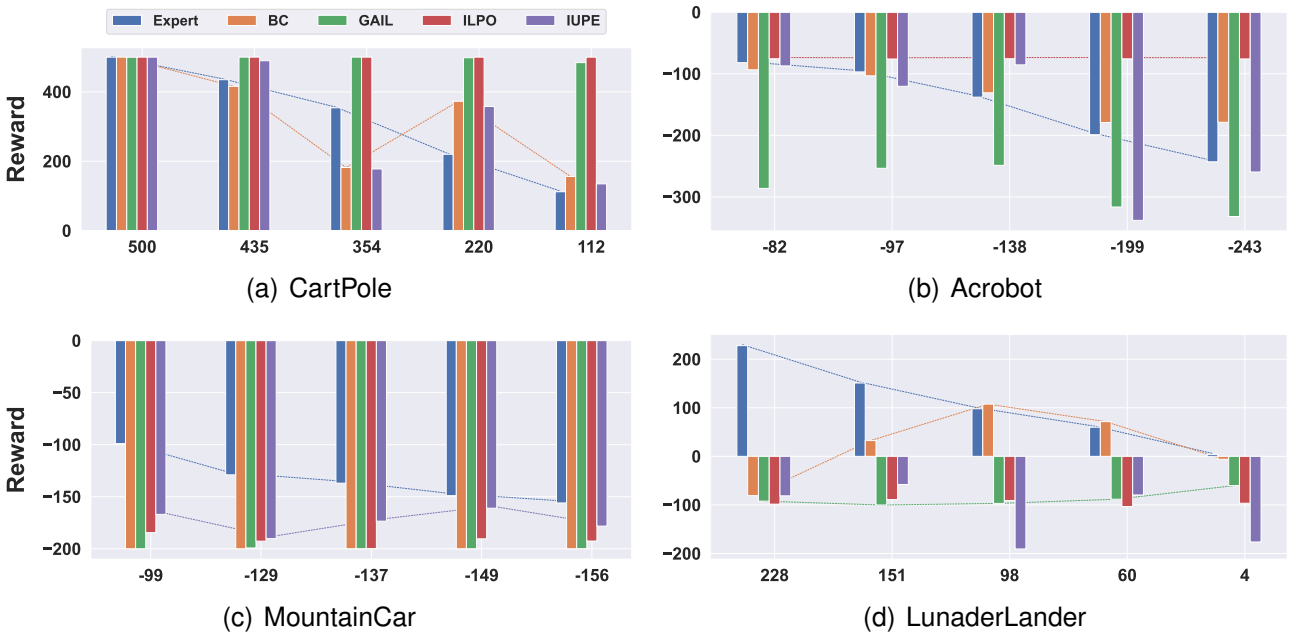
Algos	Metrics	CartPole	Acrobot	MountainCar	LunarLander
Random	AER	18.7 ± 0	-482.6 ± 0	-200 ± 0	-182.72 ± 0
	$\mathcal{P}$	0 ± 0	0 ± 0	0 ± 0	0 ± 0
Expert^5	AER	112 ± 7.50	-243 ± 136.82	-156 ± 47.76	4 ± 110.92
	$\mathcal{P}$	1 ± 0	1 ± 0	1 ± 0	1 ± 0
BC	AER	155.96 ± 71.82	-178.82 ± 55.38	-200.0 ± 0.1	$\mathbf{-6.03 \pm 93.79}$
	$\mathcal{P}$	1.46 ± 0.05	1.27 ± 0.02	0.0 ± 0.0	$\mathbf{0.96 \pm 0.02}$
GAIL	AER	484.22 ± 40.29	-331.85 ± 84.55	-199.94 ± 2.14	-60.28 ± 101.21
	$\mathcal{P}$	5.11 ± 0.31	0.61 ± 0.26	0.0 ± 0.03	0.77 ± 0.4
ILPO	AER	$\mathbf{500.0 \pm 0.0}$	$\mathbf{-75.93 \pm 15.15}$	-192.58 ± 7.65	-96.57 ± 76.11
	$\mathcal{P}$	$\mathbf{5.16 \pm 0.0}$	$\mathbf{1.7 \pm 0.0}$	0.0 ± 0.64	0.41 ± 0.14
IUPE	AER	134.72 ± 22.51	-259.57 ± 23.58	$\mathbf{-178.24 \pm 23.54}$	-176.33 ± 85.52
	$\mathcal{P}$	1.33 ± 0.55	1.33 ± 0.82	$\mathbf{0.6 \pm 0.36}$	0.14 ± 0.57

IDM. A plausible hypothesis is that self-supervised methods with exploration mechanisms can enable more dynamic learning models (as evidenced by the MountainCar outcomes). In contrast, those lacking exploration excel in more stochastic environments.

Table 7.6 reveals results for the least effective expert, Expert^5. In this scenario, none of the experts resolve the tasks, yet they significantly outperform the random agent. A considerable performance drop is observed for all algorithms across the CartPole, Acrobot, and MountainCar environments, except for ILPO, which remains consistent in the first two experiments. This consistency observed with ILPO supports our inference that self-supervised IL methods without exploration perform better in environments where trajectory formation is less crucial (*e.g.*, in Acrobot, where the impact of any action on the overall trajectory is minimal compared to MountainCar).

IUPE sustains its results with minimal alterations, even with degraded expert samples. This behavior strengthens our assertion that self-supervised IL methods with exploratory mechanisms can assist IL agents in executing smoother actions in environments more predisposed to trajectory deviations (*e.g.*, momentum in MountainCar). Lastly, as anticipated, in the LunarLander environment, all methods experienced diminishing results in this final scenario.

Figure 7.1 shows the performance of all the approaches across four environments and using five experts. The lines indicate the performance tendency of each algorithm as the quality of the samples decreases, with the blue line representing the expert. In the CartPole environment (Figure 7.1(a)), BC's performance decreases linearly even though it had access to all labels from the expert. IUPE's performance pattern follows BC, while GAIL and ILPO maintain relatively stable performance throughout the sample-quality degradation process. Next, we analyze the Acrobot (Figure 7.1(b)) and MountainCar (Figure 7.1(c)) environments. In Acrobot, ILPO is able to maintain its performance throughout the sample-quality degradation, suggesting that ILPO has learned the dynamics of the environment, and



(a) CartPole (b) Acrobot

(c) MountainCar (d) LunaderLander

Figure 7.1 – Results from multiple environments, using five different experts per environment and four algorithms. The x-axis represents the expert reward, while the y-axis represents the cumulative reward over a hundred runs for each agent. The lines on each graph show the patterns of the expert and one or more algorithms, highlighting their distinct behaviors.

the expert's behavior only works as a guide to its goal. In the MountainCar environment, IUPE exhibits a similar pattern of consistent performance, showing some correlation with the expert. Finally, in the LunarLander environment (Figure 7.1(d)), BC shows the best overall results, GAIL displays the most consistent and robust performance, and IUPE has the best performance in 3 out of 5 experts among those that do not have access to the ground-truth labels. Overall, receiving sub-optimal trajectories allows the approaches to learn the dynamics of the environment more effectively and achieve better results. However, this does not occur in all environments. We hypothesize that the LunarLander environment might provide a better scenario for testing the generalization capabilities of IL approaches since merely learning to mimic the trajectory of the expert without understanding how and where to land does not lead to high performance, especially given that the final goal changes depending on the chosen seeds.

7.3 Discussion

In this study, similar to previous research in this field and following the metrics presented in our methodology, we employed the *Performance* metric to evaluate the results of our experiments. As elaborated in Section 4.3, this metric normalizes the agent's performance by computing the average reward across 100 executions and comparing it to the expert's performance and the expected performance of a random policy. Consequently, an

agent surpassing the expert's performance can achieve results greater than 1, while poor performance may yield negative results.

However, it is important to note that this metric may not always obtain accurate information or align with common expectations. When applied to skewed data, it inherits the same challenges as linear normalization methods. Although we do not use it for classification or regression optimization, it can misrepresent the agent's behavior in a given environment. To illustrate this point, the experiments conducted in the MountainCar domain can be examined under two scenarios: (i) when the expert achieves a reward of -160 and the random policy achieves -200 (the lowest possible reward); and (ii) when the expert's reward is -110 (considered the threshold for solving the task) and the random policy still receives -200.

In the first scenario, if an agent replicates the expert's behavior, it would be given a Performance score of 1, whereas in the second scenario, achieving the same reward would result in a Performance score of approximately 0.44. This example highlights the issue of assuming the expert policy is optimal, as also observed in Tables 7.5 and 7.6. If the selected expert samples are not optimal, the performance metric will fail to represent the agent's competence accurately. In this case, the agent falls short of solving the task by 50 reward points, yet the metric suggests it is optimal.

Therefore, it becomes crucial to have prior knowledge regarding the optimality level of the expert, which can present a complex challenge across various application domains. Furthermore, if an agent achieves a reward of -100 in the same example, the performance scores would be 2.5 and 1.11 for the first and second scenarios, respectively. This example demonstrates that (i) the samples may not be ideal, as was the case with most of the samples in this study, and (ii) the policy may not have fully learned the precise behavior exhibited by the expert. The results support both hypotheses, indicating that the proposed method successfully learned from and even surpassed the expert's performance.

We can conclude that the reward function alone does not accurately represent the expert's trajectory. To illustrate this point, consider an environment where an agent's task is to drive a car from point A to point B and then pick up a passenger. Suppose the reward function solely tracks the distance driven by the agent and fails to account for the act of picking up the passenger. In that case, the agent's and expert's rewards will be the same. Consequently, the agent's performance would be assigned a value of 1. Similarly, if the reward function only considers the act of picking up the passenger and ignores the distance driven, an agent that takes a longer path than the expert would still achieve a performance score of 1, despite its sub-optimal behavior. This example can be applied to the LunarLander environment, where the agent is rewarded for landing between flags. Even if the agent takes longer to reach its destination, it can still attain a performance score close to 1.

Furthermore, it is important to note that the performance metric fails to provide insights into the agent's generalization abilities. When utilizing classic control environments, IL can easily force the agent to closely reproduce the expert's trajectory without requiring sig-

nificant effort to map unseen states correctly. This issue becomes apparent when examining how ILPO maintains its performance across multiple sub-optimal experts, as depicted in Figure 4.1. Despite the consistent degradation of the expert's performance, ILPO manages to map the trajectory accurately. While this behavior is useful in cases where the environment task does not necessitate generalization over time, such as in MountainCar, a closer examination of Figures 4.5 and 4.4 reveals that these algorithms perform similarly to the random policy in environments where a moderate level of generalization is required.

We highlight that the performance metric can still produce positive results even when evaluating sub-optimal agents. This observation emphasizes the necessity for caution when using the metric, as it underscores the importance of fully considering its inherent flaws and limitations.

7.4 Final Remarks

This study explored the efficacy of four Imitation Learning algorithms under varying degrees of expert optimality in diverse scenarios. Specifically, we assess the impact of reducing expert quality on these approaches while solving different tasks. Notably, two IL algorithms, ILPO and IUPE, exhibit superior performance and demonstrate less dependence on expert correlation compared to the other methods. Employing approaches that involve the initial classification of state transitions by the agent, followed by learning the expert policy, enhances their resilience to different levels of sample quality. Although IUPE follows a distinct model for classifying transitions, its iterative nature aids in generalization by providing diverse labels for the same state samples during training.

This study was published in the proceedings of the Brazilian Conference on Intelligent Systems (BRACIS).

The key contributions and findings of this study are:

- **Analysis of IL Algorithms**: we experiment with four different IL algorithms, evaluating their performance in scenarios with varying levels of expert quality.
- **Impact of Expert Quality**: we investigate the impact of decreasing expert quality on IL algorithms' performance when solving different tasks.
- **Superior Performance of ILPO and IUPE**: ILPO and IUPE demonstrate better performance and exhibit less correlation with the expert than BC and GAIL.
- **Robustness to Sample Quality**: approaches involving initial classification of state transitions enhance resilience to different degrees of sample quality.
- **Iterative Nature of IUPE**: IUPE's iterative nature aids in generalization by providing diverse labels for the same state samples throughout training.

- **Trade-off between Optimality and Generalization**: certain environments showcase a clear trade-off between IL agents' optimality and generalization capabilities.
- **Proposal for New Evaluation Metrics**: given the possibility of experts being sub-optimal, we emphasize the importance of proposing new evaluation metrics for IL.

8. COMBINED REINFORCEMENT AND IMITATION LEARNING

While successful in reproducing observed behaviors, imitation learning approaches are constrained by the quality and availability of expert-generated trajectories. In the previous chapter (Chapter 7), we delved deep into this subject and conducted a series of comprehensive experiments to gain a thorough understanding of the impact of sub-optimality on IL algorithms. In contrast, Reinforcement Learning (RL), although not reliant on a supervised signal for task learning, often demands extensive computational resources, potentially leading to sub-optimal policies under resource constraints.

To address these limitations, we present a novel method called Combined Reinforcement and Imitation Learning (CRIL), which leverages a small sample of expert behavior to accelerate the reinforcement learning process and achieve optimal policies. CRIL (Figure 8) combines the strengths of both IL and RL by using expert trajectories to learn how to behave while concurrently learning from its own experiences in a traditional RL manner.

8.1 Reinforcement and Imitation Learning

CRIL adopts an interleaved process of Imitation and Reinforcement Learning to converge toward an optimal policy efficiently. Our method incorporates the concept of self-supervised learning of policies using an IDM, as explored and presented in previous approaches (Chapters 5 and 6). We then enhance and refine this policy through reward-based exploration inspired by *q*-learning techniques described in Watkins and Dayan [51], Mnih *et*

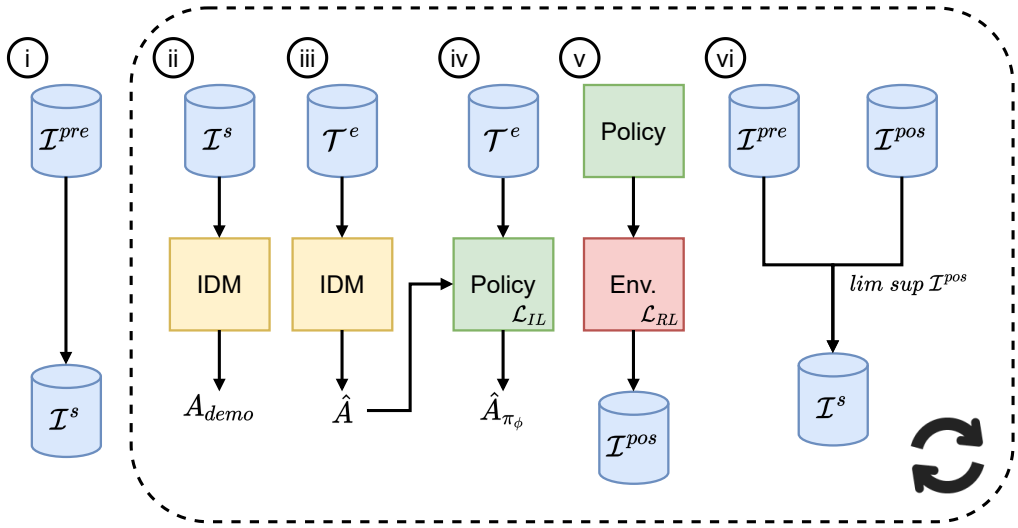


Figure 8.1 – Combined Reinforcement and Imitation Learning (CRIL) framework.

al.[29], and Kaiser *et al.* [20]. The final goal is to develop a novel algorithm that uses expert trajectory samples to guide policy design while leveraging its own experiences to explore states beyond the expert's trajectories and refine its trajectory.

For the RL component of our method, we base our implementation on the original, unmodified Deep Q-Network (DQN) [29] architecture. We chose DQN as it exhibits interesting parallels with the self-supervised IL component, particularly concerning being off-policy. Additionally, we acknowledge the pivotal role of the exploration versus exploitation trade-off in achieving higher rewards. This consideration aligns with our approach's Imitation and Reinforcement Learning components.

8.1.1 Self-Supervised Imitation Learning

Self-supervised Imitation Learning usually encompasses two main modules: the Inverse Dynamics Model (IDM) and the Policy Model (PM). The IDM learns to predict actions based on state transitions, denoted as $\mathcal{M}_\theta(a \mid s_t, s_{t+1})$, while the policy π_ϕ acts as a stationary model, predicting the most likely action a given the state s_t. To learn these transitions, a pre-demonstration dataset ($\mathcal{I}^{pre}$) is generated using π_ϕ with random weights, consisting of samples in the form of (s_t, a, s_{t+1}). The IDM then leverages $\mathcal{I}^{pre}$ to learn the inverse dynamics of the agent, aiming to find optimal parameters θ^* that accurately describe state transitions. As expert labels are unavailable, pairs of states from expert demonstrations (s_t^e, s_{t+1}^e) are used with the IDM to predict the actions associated with expert transitions. Subsequently, using these self-supervised labels, the PM learns to predict actions $\pi(\hat{a} \mid s_t)$. However, since $\mathcal{I}^{pre}$ includes random actions, the pseudo-labels generated by the IDM may deviate significantly from those of the expert. Chapter 2.2 further explores this topic in greater detail.

To address this issue, an iterative process is employed. The updated policy generates new samples $\mathcal{I}^{pos}$, and the dataset $\mathcal{I}^s$ is balanced with all trajectories that successfully reach the environment goal. This iterative approach maintains a weighted distribution between random and updated policy samples, preventing the model from getting stuck in local minima. Importantly, the probability of actions vanishing in each iteration is minimal, ensuring effective learning and avoiding convergence issues.

8.1.2 Exploration with Neural Networks and *q*-values

Another essential component of our proposed approach involves an exploration mechanism through RL using *q*-values and neural networks. One popular method in this context is Deep Q-Network (DQN), introduced by Mnih *et al.* [29]. DQN uses a Deep Neural Network to approximate the optimal *Q*-function, as defined in Equation 8.1. In this equation,

r represents the reward received during the transition from state s_t to s_{t+1}, α denotes the learning rate, a corresponds to the action, and Q refers to the deep neural network.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{8.1}$$

DQN incorporates an experience replay mechanism, which stores a set of observations from the environment to update the Q-function using random samples. This approach addresses the correlation issue between observation sequences and facilitates smoother changes in the data distribution.

During each episode, the algorithm selects a random action with a probability of ϵ or uses the action-value function. Typically, the value of ϵ decreases as training progresses to shift the agent's focus from exploring early states to exploiting late states, particularly those close to the goal. The agent executes the ϵ-greedy action in the environment, receiving a reward and transitioning to the next state.

8.2 Combining Imitation and Reinforcement Learning

CRIL follows an iterative approach that combines IL and RL ideas, leveraging both paradigms' benefits. The algorithm generates new samples using the environment and employs RL techniques.

CRIL introduces a Reinforcement Learning approach to learning through experience by accessing states and rewards from the environment. The complete learning pipeline of CRIL is outlined in Algorithm 8.1, with the following main steps: (i) Creation of dataset $\mathcal{I}^{pre}$ by using the imitation policy π_ϕ as $\mathcal{I}^s$ (lines 4-5); (ii) Utilization of $\mathcal{I}^s$ to learn the inverse dy-

Algorithm 8.1 Combined Reinforcement Learning (CRIL)

```
 1: Initialize model M_θ as a random approximator
 2: Initialize policy π_φ with random weights
 3: Generate state transitions T^e from expert demonstration
 4: Generate I^pre using policy π_φ
 5: Set I^s = I^pre
 6: while π_φ improves from either method do
 7:     Update M_θ by TRAIN(M_θ, I^s)
 8:     Generate pseudo-labels Â by M_θ(T^e)
 9:     Update π_φ by BCLOSS(T^e, Â)
10:     for e ← 1 to |E| do
11:         Use π_φ to solve environment e
12:         Append samples I^pos ← (s_t, a_t, s_{t+1})
13:         Update π_φ by TDLOSS(I^pos, A)
14:     end for
15:     I^s ← GOALSAMPLER(I^pos)
16: end while
```

namics of the environment (line 7); (iii) Labeling the expert actions $\hat{A}$ responsible for the state transitions in the expert samples $\mathcal{T}^e$ using the IDM network (line 8); (iv) Training the policy π_ϕ in an imitation learning fashion using $\mathcal{T}^e$ and the pseudo-labels $\hat{A}$ (line 9). (v) Using the policy π_ϕ in the environment to create new state transitions $\mathcal{I}^{pos}$ and employing the temporal difference update to learn from its own experiences (lines 10-14); (vi) Employing a sampling mechanism to create a new dataset $\mathcal{I}^s$ for feeding the IDM network (line 15); (vii) Repeating steps ii-vi until no further improvement is observed. This occurs when there are no changes in actions between two consecutive epochs or when there is no significant reduction in loss over consecutive epochs.

By iteratively following these steps, CRIL integrates IL's and RL's strengths, allowing the policy to benefit from expert guidance while learning from its own experiences in the environment.

In the RL-based component of CRIL, ϵ-greedy exploration is applied to learn states outside the expert's scope. While traditional ϵ-greedy approaches often decrease the exploration chance over time, CRIL alternates between learning from demonstration and experience, which can result in acquiring sub-optimal information from both perspectives. To address this, CRIL adapts its exploration behavior based on the model's accuracy.

The policy of CRIL uses the softmax distribution from its output to predict actions. As the policy learns to differentiate between actions, it becomes less predisposed to choose actions based solely on the maximum *a posteriori* label. In each iteration of CRIL, the exploration ratio of the policy is computed during the self-supervised learning component (Line 9). The same value is used as the epsilon for exploration in the RL-based component (Lines 10-14). This approach allows the model to explore more initially, similar to ϵ-greedy policies. As the policy evolves, it allows for less exploration and more exploitation. This strategy eliminates the need for a time-dependent decaying function for exploration values commonly observed in RL algorithms and enables increased exploration when the policy encounters local minima.

It is important to note that RL approaches are typically not designed to learn an optimal policy in a limited number of episodes. Therefore, it is crucial to adapt the size of the experience replay. If the size is too large, there will be fewer updates during each iteration, potentially leading to less desirable actions. Contrarily, if the size is too small, fewer samples will result in sub-optimal weight updates. To tackle this issue, CRIL utilizes the average size of expert trajectories available to determine the size of the replay memory for each environment. Specifically, CRIL sets the experience replay size to be $10\times$ the average size of the expert samples available in each domain. This adaptive approach ensures a balance between updating the weights effectively and utilizing the available expert knowledge for efficient learning.

One potential drawback of CRIL is its reliance on RL and IL methods. This iterative process constantly shifts data and labels, with one part depending on a reward signal and the other being self-supervised.

To address this limitation, we propose three modifications to both components: (i) We introduce gradient clipping for both the Reinforcement and Imitation Learning components. By limiting the magnitude of the gradients, we reduce variance during the learning process. Since the two learning methods have different objectives, this clipping helps stabilize the training; (ii) We incorporate layer normalization into both the IDM and PM. This addition is motivated by the continuous shift in data and labels due to the approach's self-supervised iterative nature. While traditional IL methods benefit from large expert datasets that mitigate the effects of covariance shift, CRIL relies on fewer expert samples. By incorporating layer normalization into CRIL's topology, the model became capable of learning the correct normalization of all neurons in each layer based on the samples. This addition boosts the learning process of the policy; (iii) We modify the sampling mechanism employed by the self-supervised strategy to reduce the impact of covariate shift. In its original form, the sampling is performed from a softmax distribution, allowing the IDM to quickly learn a distribution beyond $\mathcal{I}^{pre}$, which ensures balanced actions. However, this approach proves less effective when expert samples are scarce. The constant changes in IDM's predictions, combined with the limited number of examples, negatively affect the policy.

To mitigate this issue, we introduce an upper limit on the number of samples $\mathcal{I}$ from $\mathcal{I}^{pos}$. We determine this limit using Equation 8.2, where n represents a hyperparameter defining the number of epochs required for the upper limit to reach 100%, e denotes the current epoch, and k indicates the slope of the curvature. This approach promotes smoother transitions between epochs, reducing the covariate shift experienced by CRIL during training.

$$\limsup \ \mathcal{I}^{pos} = 1 - \frac{1}{1 + (\frac{n}{e} - 1)^{-k}}, \tag{8.2}$$

8.3 Experimental Results

In our main experiments, we set the value of k to 2, and each algorithm was trained for 150 epochs. These values may vary in later ablation studies. For the RL model, gradient clipping was applied within the range of $[-0.5, 0.5]$, while the IL models used $[-1, 1]$ for clipping. The evaluation of CRIL and other IL-related approaches is based on Average Episodic Reward (AER) and Performance ($\mathcal{P}$).

To compare RL approaches' sample efficiency, we count the number of samples each algorithm receives to reach a specific reward rather than measuring timesteps. This

ensures a fair comparison considering CRIL's utilization of both expert and environment samples. Two DQN methods are tested: the original version (DQN_1) by Mnih *et al.* [29], which serves as a reference for several mechanisms in CRIL, and the prioritized experience replay variant (DQN_2) by Schaul *et al.* [39], which is state-of-the-art for most tested environments. Additionally, we include two other RL algorithms, PPO [42] and ACKTR [53], as baselines.

These algorithms were chosen due to their distinct approaches, both achieving optimal policies for the Acrobot and MountainCar environments. Unlike CRIL, the other IL methods in this study do not utilize any RL mechanisms.

8.3.1 Policy Optimization Behavior

IL and RL methods work on the same premise that an agent needs to learn an approximation to a theoretical optimal policy in the form of an MDP. Nevertheless, IL focuses on a more specific optimal policy: the expert's. At the same time, RL learns how to optimize its value function, thus achieving one of many possible optimal functions for each environment. We hypothesize that CRIL yields better policies than IL methods since it learns with its own experiences, while also achieving results much more efficiently than RL methods. To validate that hypothesis, we conduct an experiment where we compute the KL Divergence of a trajectory with four different policies: (i) the optimal policy (π^*); (ii) an RL policy [29]; (iii) an IL policy [13]; and (iv) the CRIL policy. Since π^* may be one of many theoretical optimal policies, we do not use these results as a form of quantifying any of the policies created. However, upon carefully analyzing them, we can draw intuitions regarding the combination of RL and IL into a single policy.

We compute two different KL Divergence values. First, we compare all models with π^* and compute the difference with the probability of all possible actions. This result shows how similar a policy is to the optimal one regarding its mapping of the likelihood of actions given a state. However, such difference is trivial when an agent during evaluation uses only a greedy approach for choosing an action. Thus, we also compute the KL Divergence using one-hot encodings for the specific action given a state (KL-Divergence*).

In Table 8.1, it is evident that the CRIL policy exhibits the highest divergence from π^* according to the first metric, with a value of 9.6476. This difference can be attributed to two primary factors: the expert's actions may not be optimal, and the reinforcement learning updates can significantly vary the softmax probabilities.

To explore the first hypothesis, we can refer to Figure 8.2, which visualizes the probability of the *maximum a posteriori* action across all discretized values. Comparing π^* with the other policies, we observe that in the middle of the valley, π^* demonstrates a certain level of uncertainty (approximately 33% for each action), which is not apparent in the other policies. Furthermore, when comparing RL and IL policies, we find that the RL policy

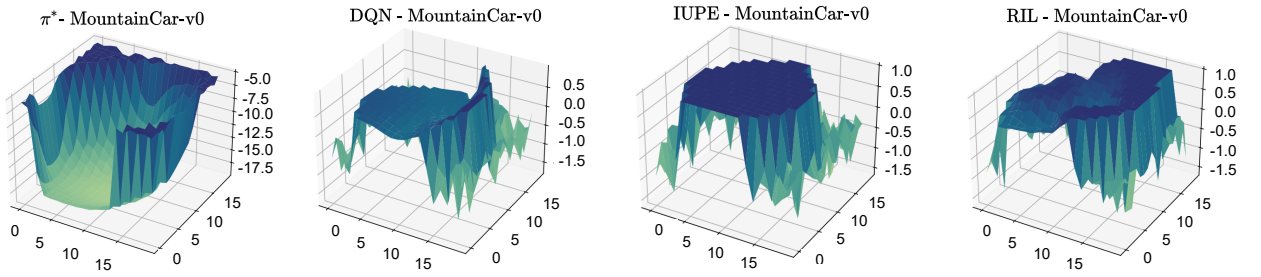


Figure 8.2 – Visualization of the MountainCar-v0 environment. Each figure illustrates the *maximum a posteriori* probabilities in a 3D mesh.

exhibits behavior more similar to π^* than the IL policy. This indicates that the RL approach better captures the optimal policy's characteristics than the IL approach.

The IL policy demonstrates a higher level of certainty compared to the other policies across all discretized values (approximately 58%). This characteristic can help explain the poor performance of the IL model. Since IL methods solely rely on expert samples in a supervised manner, the model's certainty only considers the classification problem without considering the sparse rewards that the MountainCar environment presents.

In contrast, when comparing CRIL with the other methods, we observe that CRIL manifests a combination of RL and IL characteristics within the "valley" region. Although CRIL does not yet generate results more closely aligned with π^*, it strikes a balance by creating a more moderate mapping of discretized values.

When comparing CRIL with the greedy optimal policy, our method achieves the highest similarity, with a value of 0.8094. This result indicates that despite their divergence in probabilities, both methods exhibit considerable similarity. In other words, both methods agree on the action to be taken for the same state.

Figure 8.3 visually illustrates the proximity of CRIL to π^* by discretizing the continuous state-space of MountainCar into a 20×20 Q-table. The plot depicts the maximum a posteriori action and colors the states equivalent to π^*. The figure shows that in a discrete space, CRIL is closer to the optimal policy than the other methods.

Table 8.1 – KL Divergence from all three models, when compared to an optimal policy (π^*).

Metric	π^*	RL (DQN)	IL (IUPE)	RIL
Reward	-86	-87	-162	-84
KL Divergence	-	2.0869	4.9863	9.6476
KL Divergence*	-	1.7345	1.7345	0.8094

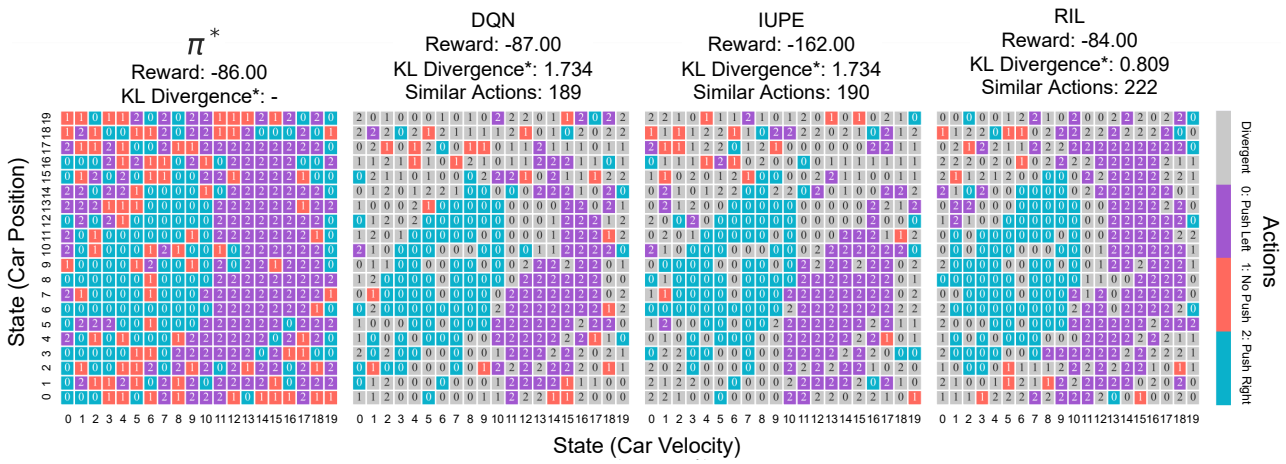


Figure 8.3 – Comparison between policies trained in the MountainCar-v0 environment. We only color the tiles that have the same action as π^* for easier visualization.

Regarding the regular KL Divergence, it is worth noting that the reward in this episode does not appear to be a decisive factor for the result in KL-Divergence*. This is evident as RL and IL policies exhibit similar distances from the optimal policy. Finally, if the reward plays a significant role in achieving good performance in the MountainCar environment, CRIL should be closer to zero in the KL Divergence compared to RL.

Imitation Learning

Since CRIL reaches $\mathcal{P} \geqslant 1$ with only one episode from the expert, we give the same single trajectory for all other IL methods during this experiment. The results in Table 8.2 show the average and standard deviation of 10 different runs for each learning algorithm.

Considering that for the CartPole and Acrobot environments there is almost no variation in their initial states, one trajectory should be enough to achieve their goal, even though not optimally. We hypothesize that all methods have comparable results in these cases. Nevertheless, just ILPO and CRIL results were good enough to achieve the goal for the CartPole environment, *i.e.* $r \geqslant 195$. In contrast, GAIL and IUPE achieved performance around 0.30, with GAIL being only 10 reward points from the goal, though far from the expert. The Acrobot environment does not have a defined goal, but we can define that a reward close to -80 can be considered ideal, as is the case for the expert. However, only CRIL was able to reach such a result. ILPO achieves a performance of 0.9, with IUPE being close with 0.8 performance points. Since a random policy achieves an AER of -482.6, both methods converge to policies closer to the expert than GAIL, which only achieves -279.02 reward points.

By contrast, MountainCar depends heavily on the agent starting position, while LunarLander alters its objective during each iteration. Having only one trajectory to learn how to mimic the expert is a significant disadvantage. This limitation is evident in the overall

Table 8.2 – *Performance* ($\mathcal{P}$) and *Average Episode Reward* (AER) for each IL methods with only one expert's trajectory as data.

Algorithms	Metric	CartPole	Acrobot	MountainCar	LunarLander	Average $\mathcal{P}$
Random	AER	18.7	-482.6	-200	-182.72	0 ± 0
	$\mathcal{P}$	0	0	0	0	
Expert	AER	500	-85	-106	235.96	1 ± 0
	$\mathcal{P}$	1	1	1	1	
BC	AER	490.96 ± 19.65	-122.75 ± 2.99	-129.92 ± 4.14	131.84 ± 53.25	0.84 ± 0.12
	$\mathcal{P}$	0.98 ± 0.04	0.91 ± 0.01	0.75 ± 0.04	0.75 ± 0.13	
GAIL	AER	185.07 ± 168.25	-279.02 ± 104.91	-196 ± 10.99	59.03 ± 87.76	0.36 ± 0.23
	$\mathcal{P}$	0.35 ± 0.34	0.51 ± 0.26	0.04 ± 0.12	0.58 ± 0.21	
ILPO	AER	456.87 ± 4.10	-125.92 ± 19.23	-200 ± 0	-451.81 ± 247.53	0.29 ± 0.75
	$\mathcal{P}$	0.91 ± 0.01	0.90 ± 0.04	0 ± 0	-0.64 ± 0.59	
IUPE	AER	144.64 ± 11.65	-232.38 ± 50.92	-198.00 ± 6.00	-203.05 ± 35.51	0.19 ± 0.27
	$\mathcal{P}$	0.26 ± 0.02	0.55 ± 0.16	0.02 ± 0.06	-0.05 ± 0.08	
CRIL	AER	$\mathbf{500 \pm 0}$	$\mathbf{-79.52 \pm 4.49}$	$\mathbf{-100.29 \pm 1.59}$	$\mathbf{261.73 \pm 9.91}$	$\mathbf{1.04 \pm 0.03}$
	$\mathcal{P}$	$\mathbf{1 \pm 0}$	$\mathbf{1.01 \pm 0.01}$	$\mathbf{1.06 \pm 0.02}$	$\mathbf{1.06 \pm 0.02}$	

results for all IL methods that reach a performance of ≈ 0.06 in MountainCar, and of ≈ -0.04 in LunarLander. These policies are further away from the expert and the goal of the environment (-110 and 200).

Since the BC method uses the ground-truth labels, we hypothesize that this approach yield similar results to CRIL, even though the number of trajectories is limited. In the CartPole and Acrobot environments, the BC method achieves results closer to the expert ($\mathcal{P} \simeq 0.9$); however, performance and rewards decrease significantly during the MountainCar and LunarLander environments. This experiment shows that CRIL's capability of learning with its own experience is a substantial advantage.

8.3.2 Reinforcement Learning

By comparing CRIL with IL methods, we demonstrate that CRIL achieves better results with fewer expert samples. However, CRIL leverages its own experiences through *q*-value mapping to create an optimal policy, which other IL methods cannot access. Thus, we also compare CRIL with RL methods to understand whether CRIL achieves comparable results with fewer samples than RL.

The results in Table 8.3 indicate the timesteps required for each method to reach its maximum reward. Since CRIL combines off-policy RL training with IL training, we consider both the samples used during RL training and the expert samples used during IL training in these results. For each iteration of CRIL, we count the number of timesteps from the RL component training plus 500 expert samples.

Table 8.3 – The average number of timesteps required by each algorithm to reach the maximum reward in each environment.

Environment	DQN_1	DQN_2	PPO	ACKTR	CRIL
CartPole	211,000	64,500	15,000	169,500	**14,800**
Acrobot	427,500	498,000	98,000	482,000	**16,365**
MountainCar	224,500	155,500	680,000	507,500	**29,745**
LunarLander	357,000	**239,000**	$154,000^{\dagger}$	646,000	281,204

As expected, DQN_1 and DQN_2 demonstrate similar results in terms of reaching their maximum rewards, although DQN_2 solves most environments while DQN_1 does not. DQN_2 achieves its maximum reward more efficiently than the other RL methods while achieving higher or comparable rewards. Specifically, DQN_2 exhibits a success rate ($\mathcal{P}$) of 0.92, whereas PPO and ACKTR achieve success rates of approximately 0.85. The difference between PPO and DQN_2 is minor for the CartPole environment but significant for MountainCar. PPO utilizes a smoother exploration mechanism, enabling it to attain superior outcomes with fewer samples ($524,500$) in the Acrobot environment. Similar behavior is observed with CRIL, as the ϵ-greedy strategy used during RL training becomes less frequent due to the initialization with the exploration rate from the previous epoch. Consequently, it enables a more efficient path to reach the maximum reward.

Although ACKTR achieves the best result among the RL methods for the MountainCar environment, it requires approximately 500,000 timesteps ($477,755$ more than CRIL). Notably, PPO requires fewer timesteps than all other algorithms to reach its maximum reward in the MountainCar environment. However, since PPO does not achieve the environmental goal, we do not consider it a significant result.

These experiments demonstrate that leveraging the expert trajectory as a guide allows CRIL to reach its maximum reward and accomplish the goal of all environments more efficiently than RL approaches. However, CRIL inherits the challenge from IL of relying on an expert's trajectory, making it challenging to adapt to environments where the goal constantly shifts, such as LunarLander.

8.3.3 Quantitative Results

Table 8.4 presents the results for both paradigms, showcasing the Performance of each method in terms of their ability to learn a policy capable of achieving the highest reward. The average $\mathcal{P}$ is also presented, indicating how well each method compares to the expert's Performance. We extended the evaluation to include 100 different expert trajectories in this

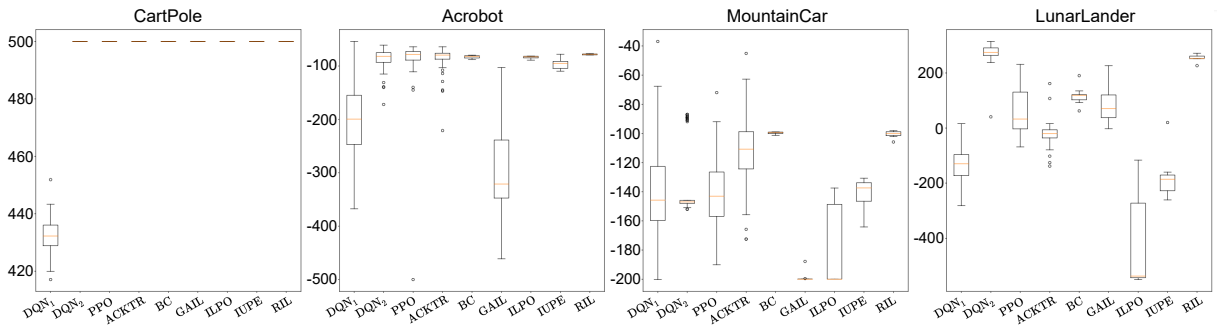


Figure 8.4 – Boxplot of the *Average Episodic Reward* for all methods and environments.

experiment, providing approximately 100 times more samples than the previous analysis. For the RL approaches, we trained each algorithm for $2,000,000$ timesteps.

As expected, DQN_1 achieved lower rewards than all other RL algorithms, supporting our premise that the RL component alone in CRIL may not solve the environment. Except for the CartPole environment, where all methods achieved $r \geqslant 195$, DQN_1 performed closer to the random policy than the expert's reward. On the other hand, DQN_2 achieved the environmental goal. It came close to the ideal reward in the Acrobot environment for most scenarios, surpassing the Performance of other RL approaches. However, unlike the other RL methods, DQN_2 struggled to solve the MountainCar environment.

Regarding the results of PPO, while it outperformed other RL methods in the Acrobot environment, it performed worse than ACKTR and DQN_2 in the remaining environments. ACKTR achieved the best result for the MountainCar environment but performed less favorably in the other environments. MountainCar is expected to be a challenging task due to its sparse reward function and a goal that incentives less exploration from the policy.

Table 8.4 – Quantitative results for all RL and IL algorithms used in this study as baselines. We also display the average Performance of all environments. DQN^1 is the unmodified DQN architecture [29], while DQN^2 is the version from Schaul *et al.* [39].

Environments	Reinforcement Learning				Imitation Learning				
	DQN_1	DQN_2	PPO	ACKTR	BC	GAIL	ILPO	IUPE	CRIL
CartPole	431.87 ±5.31	**500.00 ±0.00**	**500.00 ±0.00**	487.70 ±64.76	**500.00 ±0.00**	**500.00 ±0.00**	**500.00 ±0.00**	**500.00 ±0.00**	**500.00 ±0.00**
Acrobot	-191.51 ±64.29	-87.83 ±27.96	-83.43 ±23.29	-89.36 ±24.89	-82.92 ±2.63	-83.12 ±20.95	-83.84 ±2.50	-78.10 ±10.56	**-75.72 ±4.49**
MountainCar	-145.00 ±31.18	-135.28 ±24.10	-142.16 ±21.67	-112.55 ±21.19	**-99.69 ±0.69**	-186.74 ±0.65	-177.56 ±27.77	-130.70 ±15.23	**-100.37 ±2.50**
LunarLander	-127.64 ±70.58	**273.07 ±37.92**	105.24 ±51.90	85.85 ±64.72	214.93 ±5.56	83.56 ±65.26	-421.62 ±180.41	-211.2 ±40.77	**266.55 ±19.34**
Average $\mathcal{P}$	0.53	0.92	0.83	0.88	1.01	0.70	0.42	0.67	**1.04**

Moreover, for the LunarLander environment, most RL algorithms did not achieve the goal ($r \geqslant 200$), highlighting the difficulty of obtaining positive results. LunarLander has the lowest Performance among all tested environments, as its goal and reward system lack a strong correlation compared to environments like CartPole, and the goal itself is not fixed.

Analyzing the IL methods (excluding BC), we observe that they also face significant challenges in the LunarLander environment. While the RL methods achieved positive results, the IL strategies deteriorated over time. This behavior can be attributed to the policies learning to mimic the expert's landing position, which does not correlate with the actual goal. Without a reward signal to guide their learning, these methods tend to perform closer to the random policy rather than matching the expert's Performance.

The IL methods, on average, achieve higher results in the Acrobot environment (≈ 81.69) compared to the RL methods (≈ 86.87, excluding DQN_1). This can be attributed to the fact that IL methods learn an optimal trajectory without significant exploration. However, it is worth noting that IL approaches tend to replicate the average actions in a given state as the number of trajectories increases. This behavior benefits environments like CartPole and Acrobot, where the expert's states do not vary significantly. The policies can accurately predict the correct actions even when a particular state was not encountered in the learned trajectories or rapidly correct themselves.

Due to the self-supervised nature of IL methods, in the MountainCar environment, there is a decrease in reward. This occurs because these algorithms rely on pseudo-labels and only approximate the expert's actions. An incorrect action may cause the car to lose momentum in this environment, resulting in fewer reward points. One potential solution to address this issue is to use ground-truth labels from the expert.

Behavioral Cloning (BC) exhibits similar results to the best RL method (DQN_2) and is close to CRIL with a performance difference of 1.01. Although BC's performance is similar to the expert, it has access to ground-truth labels. However, acquiring these labels can be challenging or ineffective if the agent needs to learn the environment to generate a significant number of annotated trajectories. Therefore, CRIL's overall performance significantly improved over current IL methods. CRIL performs equally or better than the expert in all environments without relying on ground-truth labels. It also achieves higher rewards than the RL methods, except for DQN_2 in the LunarLander environment.

To further analyze the performance comparison between CRIL and DQN_2 in the LunarLander environment, we examine the standard deviation of rewards and the boxplot graph presented in Figure 8.4. CRIL achieves a reward of 266.55 with a standard deviation of 19.34, while DQN_2 achieves a higher reward of 273.07 with a larger deviation (37.92). CRIL falls within the reward interval of DQN_2 and achieves a significantly higher reward than the threshold required to solve the LunarLander environment (i.e., $r \geqslant 200$). We hypothesize that applying stricter gradient clipping during the RL training phase in CRIL contributes to

this result. One potential solution is to adjust the clipping values during the IL and RL training phases as the epochs progress, allowing for better convergence.

Notably, CRIL exhibits a smaller standard deviation than all other methods except for BC. The average deviation across all environments is approximately 6.58, while DQN_2, performing best among the RL methods, has an average standard deviation of approximately 23. BC, on the other hand, has an average standard deviation of 2.22, which is only 4.36 lower than CRIL. However, this difference may not be statistically significant, especially considering BC uses ground-truth labels for policy learning. For that reason, we plot the interval for all methods in all environments in Figure 8.4, allowing us to understand how CRIL compares to other methods in terms of variance (stability).

Except for BC, CRIL demonstrates the lowest variance among all methods. In the case of the MountainCar environment, CRIL may not achieve the highest possible value, which ACKTR achieves with a non-outlier maximum value of approximately -60. However, it is important to mention that CRIL's behavior surpasses the median behavior of ACKTR. A similar observation can be made for the LunarLander environment. Despite DQN_2 having the highest average reward of 273.07 and a maximum value of approximately 290, the behavior of CRIL falls within the interval of the RL method, with a difference of 6.52. It is important to recall that the LunarLander environment poses significant challenges for IL methods due to the strong correlation between the landing position and the final reward. The average reward is computed over 100 episodes, representing 100 different landing positions. None of the other environments considered in this work have the same characteristic of a moving goal.

The behavior of CRIL demonstrates that employing a hybrid RL/IL approach results in policies with lower variance. Furthermore, when comparing the stability of CRIL with RL approaches, it becomes evident that the adaptation capability of the RL methods is not as refined as that of CRIL. A similar tendency can be observed when comparing CRIL with IL approaches, although the average results of the IL methods are more aligned with CRIL's performance in the Acrobot environment.

8.3.4 Final Remarks

In this work, we introduced the Combined Reinforcement and Imitation Learning (CRIL) framework, which integrates IL and RL components in an iterative and interconnected process. CRIL leverages unlabeled expert samples and its own experiences to achieve state-of-the-art results across various benchmark environments.

The IL component of CRIL uses unlabeled expert trajectories to guide the policy toward a theoretically optimal policy. By learning from the expert's behavior, CRIL can benefit from the expertise encapsulated in the trajectories. On the other hand, the RL component

of CRIL explores the q-value functions derived from its own experiences, allowing the policy to adapt and refine its behavior more coherently.

CRIL offers two significant advantages. It demonstrates robust performance even with limited expert trajectories, thanks to its self-supervised learning strategy. CRIL effectively utilizes the available expert samples to guide its learning process, making it applicable when expert demonstrations are scarce. Additionally, CRIL achieves state-of-the-art results without requiring large amounts of data or extensive training iterations. By combining the strengths of both IL and RL paradigms, CRIL effectively leverages the advantages of each approach to achieve superior performance.

Compared to other IL methods, CRIL demonstrates superior results with fewer samples while maintaining comparable performance in scenarios with a higher volume of expert trajectories. This highlights CRIL's capability to extract valuable information from limited expert demonstrations, making it a highly efficient and effective approach. Furthermore, experimental evaluations reveal that CRIL achieves comparable or even better results than RL baselines while significantly reducing the number of required timesteps. This efficiency makes CRIL a promising framework for real-world applications, enabling the learning process to be more time and resource-efficient without compromising performance.

9. SELF-SUPERVISED ADVERSARIAL IMITATION LEARNING

As presented in this thesis, Imitation learning methods, such as Behavioral Cloning, instruct agents on expected conduct using expert demonstrations. State-of-the-art methods employ self-supervision techniques using fully-observable, unlabeled state snapshots to reverse-engineer state pairs into actions. Nevertheless, their iterative learning process is sensitive to getting stuck in sub-optimal local minima. Previous studies have attempted to mitigate this issue by employing goal-aware strategies, which necessitate manual intervention to ascertain if an agent has achieved its goal. This final study addresses this constraint by integrating a discriminator into the established framework. We named our approach Self-Supervised Adversarial Imitation Learning (SAIL). SAIL delivers three main benefits and directly tackles a learning challenge identified in earlier work: (i) it eliminates manual verification; (ii) it enables learning by straightforward function approximation based on the state transitions observed in expert trajectories; (iii) the discriminator solves a commonly encountered learning challenge in the Policy Model, where the agent occasionally refrains from performing any action in the environment until it ultimately comes to a deadlock.

As we saw previously in this thesis, the evaluation of LfO techniques is commonly based on metrics of performance and efficiency, as outlined in specific formalization [47, 31, 13] (Chapter 2). One alternative approach is to evaluate imitation by comparing the trajectories of the agent and the expert in executing a given task. However, both methodologies come with their own set of limitations.

In the case of traditional metrics, an agent can deviate significantly from the expert's path yet achieve the same reward, undermining the aim of the task. Sparse rewards in the environment can further complicate the identification of proficient behavior. By focusing only on rewards and disregarding trajectories or intent, an agent may achieve the same reward as a more proficient counterpart while failing to demonstrate part of the anticipated behavior.

An analogous issue can be seen when evaluating based on trajectory comparison. Consider an agent operating in a maze that mimics the teacher's trajectory. However, suppose the agent fails to reach the final state. In that case, it may receive a drastically different reward despite following a similar path.

9.1 Adversarial Approach

SAIL utilizes an exploration mechanism based on previous work [13, 21] 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 teacher's one.

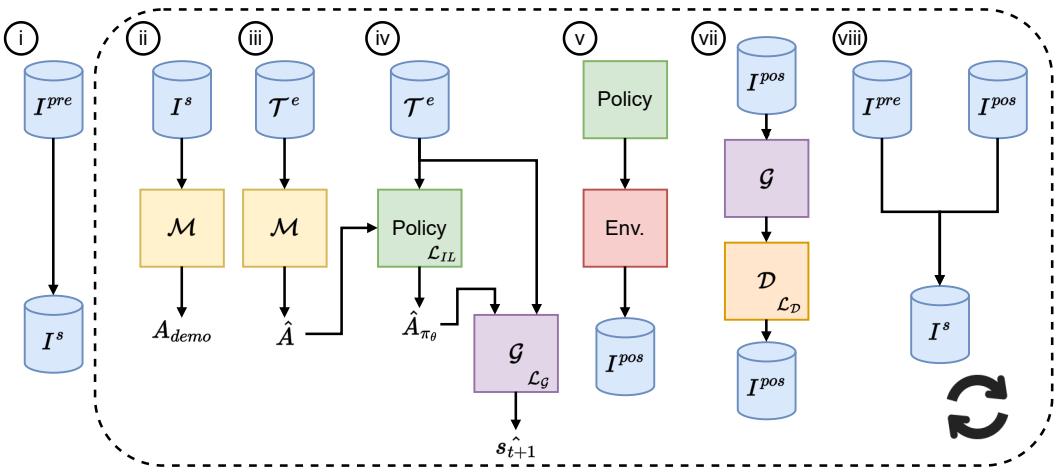


Figure 9.1 – Self-Supervised Adversarial Imitation Learning (SAIL) training pipeline.

SAIL comprises four different models: (i) model $\mathcal{M}$, which predicts an action given a state transition $P(\hat{a} \mid s_t, s_{t+1})$; (ii) a policy model π_θ that uses the self-supervised labels $\hat{a}$ to mimic a teacher, given a state $P(\hat{a} \mid s_t)$; (iii) a generative model $\mathcal{G}$ conditioned by predictions from π_θ and the current state $P(s_{t+1} \mid s_t, \tilde{a})$; 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.

The complete training process is presented in Algorithm 9.1 and the pipeline is illustrated in Figure 9.1.

Once all samples consisting of (s_t, a, s_{t+1}) tuples are added to the dataset I^s, SAIL proceeds to train its inverse dynamic model in a supervised manner (referred to as Function SUPERVISED in Line 7). Using the updated weights θ, the model $\mathcal{M}$ predicts pseudo-labels $\hat{A}$ for all teacher's transitions in $\mathcal{T}^e$ (Line 8). Since the model's weights may not be optimal at each iteration, SAIL incorporates an exploration mechanism that enables $\mathcal{M}_\theta$ to deviate from its *Maximum A Posteriori* (MAP) prediction by sampling from its softmax distributions, which

Algorithm 9.1 Self-Supervised Adversarial Imitation Learning (SAIL)

1: Initialize model $\mathcal{M}_\theta$ as a random approximator
2: Initialize policy π_θ with random weights
3: Initialize generative model $\mathcal{G}$ with random weights
4: Initialize discriminator $\mathcal{D}$ with random weights
5: Generate I^s using π_θ
6: **for** $i \leftarrow 1$ to epochs **do**
7: Improve $\mathcal{M}_\theta$ by SUPERVISED(I^s)
8: Use $\mathcal{M}_\theta$ 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 $I^s \leftarrow \forall i \in I^{pos} \mid \mathcal{D}(\mathcal{G}(I_i^{pos})) = 1$
13: **end for**

serve as weights for the predictions. This mechanism allows SAIL to dynamically explore alternative labels when uncertain, indicated by more uniformly distributed MAP values, and to exploit its MAP values when they are further apart.

After generating all self-supervised labels, we train the policy model π_θ using a behavioral cloning approach (Function BEHAVIOURALCLONING in Line 9). In contrast to other behavioral cloning approaches, SAIL also incorporates a generative model $\mathcal{G}$ that predicts the next state conditioned on the action predicted by π_θ. Thus, during the training of π_θ, $\mathcal{G}$ is also updated. Subsequently, SAIL employs π_θ to generate new samples that can assist $\mathcal{M}$ in approximating the unknown ground-truth actions from $\mathcal{T}^e$ (Line 10).

Finally, SAIL appends to I^s those samples for which the discriminator model $\mathcal{D}$ cannot differentiate between $\mathcal{T}^e$ and $\mathcal{G}(I^{pos})$ (Line 12). This step aims to discard trajectories that may cause $\mathcal{M}$ to converge to poor local minima and update π_θ to correct certain behaviors that $\mathcal{D}$ utilizes to distinguish between the teacher and the student.

9.1.1 Goal-aware function

Developing a goal-aware function can be challenging, particularly because different environments in the agent literature have varying definitions of what constitutes a goal. Typically, environments fall into two categories: maintenance tasks or achievement tasks [52].

Environments with achievement tasks define a specific end goal, such as the case of MountainCar [32], where the agent aims to reach a flag located at the top of a mountain while accumulating a reward greater than or equal to -110 (see Chapter 4). However, in the context of Imitation Learning, agents do not have direct access to the reward signal, so the number of steps taken by an agent before reaching its objective becomes a crucial consideration. Designing a goal-aware function to encode such objectives becomes increasingly difficult as environments become more complex.

On the other hand, maintenance-task environments typically involve defining a set of states that an agent should avoid reaching. For example, in the Ant environment [41], the agent attempts to walk as far as possible without encountering angles that would classify its state as "falling." In environments like CartPole [5], a stopping criterion is defined, such as when the pole of the cart reaches a specific angle, and an optimal threshold is set, such as maintaining the task for 195 consecutive steps. However, incorporating a goal-aware function adds complexity to the learning algorithm, which may not be desirable.

SAIL addresses these challenges by exclusively utilizing samples that reach a predefined goal, thereby acquiring examples that exhibit some degree of optimality and approximating the samples from the dataset I to align with the teacher's behavior [13]. However, accurately classifying whether samples are close to the proficient teacher's behavior can

be difficult. Defining the criteria for determining the proximity of a sample to the proficient teacher is challenging. If we consider a stationary agent, as in the case of SAIL, we risk discarding samples solely based on their distance from the states observed in the proficient teacher's behavior. Consequently, SAIL requires a goal-aware function that allows the model $\mathcal{M}$ to handle sub-optimal samples effectively to enhance the quality of learned policies.

To eliminate the need for hand-crafted goal-aware functions, SAIL incorporates a discriminator model $\mathcal{D}$ that distinguishes between the policy π_θ and an exploration policy π_e. This approach allows SAIL to operate without any human intervention and introduces a non-greedy sampling mechanism by employing a model to classify the origin of each trajectory. Additionally, since $\mathcal{D}$ is initialized with random weights (Line 4), it enables the inclusion of sub-optimal samples, *i.e.*, samples that did not reach a specific goal, into the dataset I^s.

However, as SAIL operates under LfO constraints, where the actions of the exploration policy π_e are not accessible, there is a need to develop a mechanism that discriminates between state-only trajectories generated by the teacher and the student. SAIL accomplishes this by utilizing state-only trajectories from π_θ and π_e and adopting an adversarial learning approach (described in Equation 9.1). This allows the policy π_θ and the generative model $\mathcal{G}$ to be updated based on the gradient flow derived from the discriminator $\mathcal{D}$.

$$\begin{aligned}\min_{\mathcal{M}\cup\pi}\max_{\mathcal{D}} & SAIL(\mathcal{M},\pi,\mathcal{G},\mathcal{D}) = \\ & \mathbb{E}_{r\sim R(\pi_\epsilon,Env)}[log(\mathcal{D}(r))] + \\ & \mathbb{E}_{r'\sim R(\pi_\theta,Env)}[log(1-\mathcal{D}(\mathcal{G}(r',\pi_\theta(r'))))]\end{aligned} \tag{9.1}$$

Considering that SAIL operates with a limited number of samples, represented by the sets $\mathcal{T}^e \cup \mathcal{T}^\pi$, it is crucial to prevent overfitting of $\mathcal{D}$. To address this, SAIL maintains a replay buffer *RB* where all trajectories generated by π_θ are recorded. At each iteration, a small subset of trajectories is sampled from both sets, denoted as $RB_{(s_t,\cdots,s_{t+n})} \sim \mathcal{T}^e \cup \mathcal{T}^\pi$. By using only a subset of trajectories from each sample pool, SAIL mitigates the risk of $\mathcal{D}$ overfitting, particularly during the initial iterations when the trajectories generated by π_θ significantly differ from the proficient teacher's behavior.

9.1.2 Generative model

SAIL effectively utilizes a generative model in two distinct phases: firstly, during the Function BEHAVIORALCLONING(), and secondly, in the process of meticulously selecting which samples to incorporate into the dataset I^s.

Although adding a generative model introduces additional complexity to the pipeline, SAIL benefits from this approach in two significant ways: (i) intrinsic encoding of environment

physics, by updating the weights of the generative model $\mathcal{G}$ using Equation 9.2, SAIL allows the loss function $\mathcal{L}_{\mathcal{G}}$ to update the policy π_θ in a way that facilitates the generation of correct state transitions, aligned with the observed transitions. This process helps π_θ encode some aspects of the environment's dynamics, such as physics. Consequently, the generative model becomes a forward dynamics model that aids in the proper conditioning of $\mathcal{G}$ by the actions predicted by π_θ. And (ii), by updating π_θ via Gradient Flow from $\mathcal{D}$. The generative model's weights are updated through the gradient flow derived from the discriminator $\mathcal{D}$, which indirectly updates the policy π_θ. This joint update mechanism assists π_θ in avoiding local minima and promotes exploration, as updating the weights of π_θ with multiple objectives helps prevent the policy from getting stuck.

While it is possible to consider updating only the weights of $\mathcal{G}$ when π_θ accurately predicts a self-supervised label, this approach introduces certain challenges. Firstly, the correctness of π_θ in predicting $\hat{a}$ may not accurately reflect its ability to condition $\mathcal{G}$ properly. The correctness of $\hat{a}$ predictions does not guarantee the correctness of the resulting state transitions. Secondly, if $\mathcal{M}$ stops predicting certain actions due to being stuck in local minima, $\mathcal{G}$ will not receive updates for state transitions associated with those actions. Consequently, $\mathcal{G}$ will update π_θ less frequently, reducing exploration. By updating π_θ with two different objectives, SAIL helps prevent the policy from becoming trapped in local minima and promotes effective exploration.

$$\mathcal{L}_{\mathcal{G}} = -\frac{1}{N}\left[\sum_{i=1}^{N} s_{i+1} \cdot log(\mathcal{G}(s_i, \pi_\theta(s_i))\right] \tag{9.2}$$

The second benefit of incorporating a generative model into SAIL arises from its adversarial training mechanism. In this process, the discriminator $\mathcal{D}$ directly updates the weights of the policy π_θ through gradient flow when it correctly discriminates between the teacher and the student (as described in Equation 9.1). This aspect is useful because updating π_θ via $\mathcal{D}$ provides the agent with a direct temporal signal, indicating where it deviates from the teacher's observations. Consequently, by combining the original behavioral cloning techniques with this update mechanism, SAIL enables the creation of agents that more accurately mimic the trajectories of the teacher while maintaining high performance.

Both moments of generative model updates in SAIL contribute to the generation of more precise trajectories (see Section 9.2) and an increased number of trajectories that closely resemble their source (discussed in Section 9.3). Having trajectories closer to the teacher's is beneficial because it helps avoid undesirable biases from previous methods that rely solely on performance metrics, which lack behavioral meaning [13].

Table 9.1 – SAIL and baselines results for all environments.

Algorithm	Metric	CartPole	MountainCar	Acrobot	LunarLander
Random	AER	$21,92\pm$	-200 ± 0	$-499.36\pm$	$-170.47\pm$
	$\mathcal{P}$	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	$\mathbf{500 \pm 0}$	-200 ± 0	-128.20 ± 15.88	$\mathbf{200 \pm 29.95}$
	$\mathcal{P}$	**1**	0	0.85	**0.86**
IUPE	AER	$\mathbf{500 \pm 0}$	-166.97 ± 18.34	$\mathbf{-75.65 \pm 12.85}$	-81.34 ± 74.5
	$\mathcal{P}$	**1**	0.32	**1**	0.21
SAIL	AER	$\mathbf{500 \pm 0}$	$\mathbf{-99.35 \pm 1.84}$	-78.84 ± 0.41	183.62 ± 5.63
	$\mathcal{P}$	**1**	**0.99**	0.99	0.83

9.2 Experimental Results

In order to evaluate the quality of SAIL, we conducted experiments using four different environments: CartPole, MountainCar, Acrobot, and LunarLander. These environments, which can be accessed through OpenAI Gym [6], are detailed in Chapter 4. Also, following the methodology proposed in this thesis, we measure our experiments using two main metrics: Average Episodic Reward (AER), and *Performance* ($\mathcal{P}$) [17] metrics. It is important to mention that 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 adopt the implementation in [13] for our agents. Specifically, our policy network, denoted as π_θ, is constructed as a Multi-Layer Perceptron (MLP) model. It consists of two hidden layers, each containing 32 neurons, and incorporates two self-attention modules after each layer. We employ an MLP denoted as $\mathcal{M}$ to describe our model for the value function approximation. This MLP also consists of two hidden layers with 32 neurons each and includes two self-attention modules and two Normalization layers. For the generator network $\mathcal{G}$, we utilize an MLP architecture with two hidden layers. The number of neurons in each hidden layer is set to $2 \times (\mid s \mid +1)$, where $\mid s \mid$ represents the size of the environment state vector. Unlike the policy and value networks, the generator network does not include self-attention or normalization layers. Lastly, we utilize a Long Short Term Memory (LSTM) network [16] for our discriminator $\mathcal{D}$. The LSTM network consists of two layers, each containing 32 neurons, and incorporates a dropout rate of 50% to mitigate overfitting.

9.2.1 Results

Table 9.1 presents the results of different baselines and SAIL across four distinct environments. In most environments, such as CartPole, MountainCar, and Acrobot, SAIL's performance closely aligns with the rewards achieved by the teacher, establishing it as the superior method across all environments. However, it exhibits comparatively poorer performance in the LunarLander environment.

When comparing SAIL to other methods, it is evident that SAIL demonstrates a lower standard deviation ($\leqslant 1$) in the first three environments. In contrast, the LunarLander environment displays a higher standard deviation of 5.63. We attribute these lower deviations to the gradient flow from $\mathcal{D}$ into π_θ, as elaborated in Section 9.1.1, and the intrinsic encoding of each environment's physics into π_θ by $\mathcal{G}$, as discussed in Section 9.1.2.

By effectively encoding the physics within its policy, SAIL achieves behavior that enables the agent to generate outcomes akin to the teacher's. This is primarily due to its understanding of how s_{t+1} should be derived from a and s_t. Furthermore, the presence of $\mathcal{D}$ empowers π_θ with a temporal signal within its trajectory, enabling it to rectify any undesired or divergent behavior, thus assisting $\mathcal{D}$ in discriminating between the teacher and the student.

In contrast, SAIL does not achieve optimal results for the Acrobot and LunarLander environments. In the case of the Acrobot environment, SAIL's accumulated reward drops below that of IUPE by 3.19. However, SAIL exhibits a remarkably lower standard deviation than BC, which had access to labeled snapshots during training. This can be attributed to the fact that the Acrobot environment rewards aggressive behavior from the agent, which IUPE benefits from due to its exploration mechanism, while $\mathcal{D}$ incentivizes SAIL to produce more consistent trajectories. We hypothesize that in cases where SAIL has a higher exploration ratio, reducing the gradient from its adversarial phase would be beneficial, preventing it from entering an exploitative phase prematurely.

In the LunarLander environment, SAIL achieves a result similar to GAIfO, with a difference of 16.38 in accumulated reward but a significantly higher standard deviation of 24.32. This can be attributed to SAIL learning proficient behavior more closely than other imitation learning counterparts. When comparing SAIL's Performance to BC, we observe a drastic change in results, similar to BC. For the LunarLander environment, IUPE performs the worst and exhibits a negative result. This can be attributed to LunarLander's optimal behavior highly depends on the agent and goal initialization, which IUPE lacks mechanisms to comprehend from its proficient source.

Furthermore, SAIL outperforms other baselines in environments with stronger state relationships, such as MountainCar. While most baselines yield policies similar to or only marginally better than a random one, SAIL achieves a performance close to 1. Through experimentation, we observed that most other methods require the agent to be in a specific

Table 9.2 – 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

state, such as stationary or minimal movement force. However, SAIL does not exhibit similar behavior due to the information it receives from its discriminator model.

9.3 Discussion

We explore two aspects of our results: (i) we analyze how SAIL learns with varying sample sizes, aiming to understand the trade-off between the number of trajectories and the learning speed. This investigation helps us determine the optimal balance, allowing SAIL to learn with as few samples as possible while achieving faster convergence. This understanding is crucial in optimizing the learning process; (ii) we investigate how the policy π_θ behaves in terms of its performance and similarity to the expert policy π_e. While imitation learning methods typically utilize performance metrics and adversarial evaluation rewards, they often overlook the extent to which the learned policy approximates the teacher policy. By examining this aspect, we gain insights into the quality of the learned policy and its alignment with the expert's behavior.

By considering these factors, we aim to enhance our understanding of SAIL's learning process and evaluate the effectiveness of the learned policy compared to the expert's behavior.

Table 9.3 – Results obtained from SAIL's modules, presenting the accuracy of the Discriminator ($\mathcal{D}$), the loss of the Generator ($\mathcal{G}$), and the performance of the Policy (π_θ) across different environments.

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

9.3.1 Sample Efficiency

The SAIL method exhibits two distinct characteristics inherited from behavioral cloning approaches. Firstly, it needs a larger sample size than a single trajectory to acquire a comparable performance to its teacher. This outcome is unsurprising, considering that a single trajectory lacks sufficient information for SAIL to effectively encode diverse states and generalize well across different environments [25]. Secondly, SAIL suffers from scalability issues as the number of samples increases due to compounding errors [46]. As the sample count grows, the policy deviates less from the observed trajectories, resulting in a decline in agent performance. Hence, behavioral cloning methods must determine the optimal number of trajectories. In contrast, as demonstrated in Table 9.2, SAIL achieves near-optimal performance with only 25 episodes, progressively reducing the standard deviation for each subsequent row, followed by an increase at 100 trajectories. We attribute this achievement to the updates derived from $\mathcal{G}$ and, consequently, $\mathcal{D}$. By employing an alternative objective, such as generating trajectories closer to those of the teachers, SAIL requires fewer episodes compared to prior methods, as π_θ relies less on its LfO objective.

However, it should be noted that SAIL accomplishes these results by employing a smaller sequential discriminator model, which comes at the cost of reduced parallelization. Our hypothesis suggests that in order to maintain a small sample size, SAIL cannot use larger sequence models like Transformers [49]. The substantial number of parameters in such models increases the risk of overfitting the discriminator model on the limited dataset. To further enhance the efficiency of SAIL, additional experimentation is required regarding the augmentation of observations. The modification of teacher samples must be approached with domain knowledge to avoid augmenting tuples that result in impossible or undesired transitions.

9.3.2 Imitation Behavior

The evaluation of IL metrics typically centers on analyzing the agent's cumulative reward, which serves as a crucial indicator to measure the policy's success in reproducing the behavior exhibited by its teacher. However, this approach, illustrated by the commonly used $\mathcal{P}$ metric, overlooks certain problems inherent to IL agents, such as their trajectory. By solely relying on the reward signal for evaluating these agents, it is insufficient to conclude that the agent truly exhibits behavior akin to its teacher. This limitation arises due to the potential existence of proficient behaviors that are not explicitly encoded within the reward function or the presence of stochastic behavior not apparent in the observations. Consequently, the $\mathcal{P}$ metric might fall short in capturing trajectory divergence since the accumulated reward can be equal. To illustrate this point, consider a maze environment where two trajectories always have the same length, but one path carries a slight risk of the floor breaking. In this scenario, a cautious agent may consistently avoid the trajectory with the faulty floor. Conversely, a more aggressive agent may overlook this risk while navigating the maze. An imitation learning agent trained on the conservative policy might inherit the bias of never stepping onto the potentially faulty floor. However, the performance metric fails to account for this behavior, even if the student policy increasingly resembles the aggressive policy in other aspects.

To address this concern, we propose an analysis that incorporates not only the performance of the policy π_θ, but also the accuracy of the discriminator $\mathcal{D}$ and the error of the generator $\mathcal{G}$. By considering these additional metrics, we can gain insights into the agent's behavior beyond just its reward. When π_θ achieves higher performance, accompanied by high accuracy in $\mathcal{D}$ and a lower error in $\mathcal{G}$, it indicates that although π_θ has successfully encoded proficient behavior, its trajectory significantly deviates from that of the teacher. This misalignment becomes apparent due to the high accuracy of $\mathcal{D}$ and the low error of $\mathcal{G}$. Furthermore, if π_θ achieves a performance close to 1, but $\mathcal{D}$ exhibits lower accuracy and $\mathcal{G}$ experiences higher error, it implies that π_θ produces the same reward as its teacher, yet its generator has learned to manipulate the encoding of next states to "fool" $\mathcal{D}$, without following the observations correctly.

Consequently, we are interested in identifying scenarios where π_θ exhibits $\mathcal{P}$ close to 1, $\mathcal{D}$ demonstrates an accuracy around 50%, and its associated $\mathcal{G}$ yields a lower error during the BEHAVIOURALCLONNING phase (specifically, Line 9 in Algorithm 9.1). For a comprehensive presentation of the results, Table 9.3 displays the values of all three metrics for the four environments employed in this study.

Upon observation of all the environments, we noticed a consistent pattern where the best results achieved by π_θ coincided with $\mathcal{D}$ producing results similar to that of a random model. In other words, $\mathcal{D}$ could not discriminate between trajectories originating from the

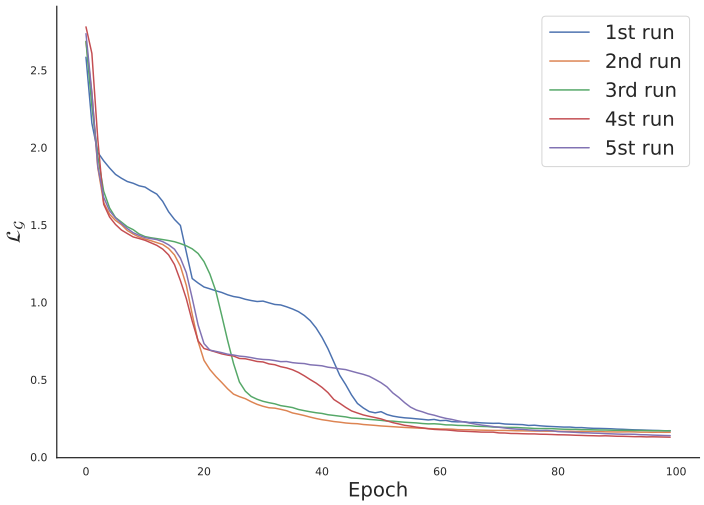


Figure 9.2 – Error rate obtained from SAIL for 5 different runs in the Acrobot environment.

teacher or the student policy. Additionally, we observed lower error rates when comparing the error of $\mathcal{G}$ during its initial learning phase, where it attempts to recreate the next state based on the teacher's trajectory. This suggests that $\mathcal{G}$ effectively learned how to encode state transitions accurately. These findings highlight that SAIL generates a policy that not only aligns with proficient rewards but also maintains a trajectory consistent with its learned source. It is important to note that the interpretation of the error rates is highly dependent on the context of each environment's state encodings.

For instance, in the case of Acrobot, its state is represented as a vector with six values, where four values range from $[-1, 1]$ and the remaining two values range from $[-12.57, 12.57]$ and $[28.27, 28.27]$, respectively. Figure 9.2 illustrates different error margins for five executions of SAIL in the Acrobot environment. It is evident that while the absolute error of $\mathcal{G}$ may be higher for Acrobot compared to other environments, the initial error value (approximately 2.5) is elevated due to the specific characteristics of the environment's encoding. Therefore, it is important to consider the higher error rate in Acrobot (compared to other environments) within its contextual framework, where it could be perceived as a lower margin rate.

9.4 Final Remarks

In this chapter, we presented a novel LfO approach that leverages an adversarial module to imitate the behavior of teachers without direct access to their actions. Our approach, named Self-Supervised Adversarial Imitation Learning (SAIL), demonstrates state-of-the-art performance and efficiency. We conducted extensive evaluations by comparing our model against various baselines from the literature, using different amounts of sampled

behavior. The results showed that SAIL achieved significant improvements in both performance and efficiency with fewer samples.

Our approach's success can be attributed to two main contributions. Firstly, we incorporated an adversarial mechanism into our end-to-end model, enabling a better approximation of the teacher's behavior. Through iterative learning driven by a loss error, our model achieved higher returns. Secondly, SAIL employs an exploration technique that strategically selects the most valuable data for each interaction, leading to faster model convergence. Additionally, we discussed the metrics for measuring the fidelity of policy imitation and conducted an ablation study. This study revealed that SAIL attains proficient rewards while effectively tricking its discriminator, further supported by a detailed analysis of the results.

10. CONCLUSION

In this thesis, we presented four distinct Imitation Learning algorithms inspired by the Behavior Cloning from Observation framework [47], as well as a study that investigates the effectiveness of imitation learning methods when exposed to sub-optimal samples.

First, we introduced a method called ABCO, which enhances the original BCO framework by incorporating two key components: (i) an improved sampling method and (ii) a self-attention mechanism. The first component determines the extent to which policy samples are used in each iteration, while the second component helps mitigate weight updates that frequently change due to dataset shifts. By introducing these novel mechanisms, we successfully solved the issue of negative feedback loops observed in our experiments. However, ABCO, similar to BCO, is susceptible to encountering deadlocks between states, as we found this example while applying our algorithm in a maze-like environment. We conducted an in-depth efficiency analysis to gain further insights into the effectiveness of the ABCO method and compare its performance with different Reinforcement Learning approaches. The experiments outlined in Section 7 demonstrate that, when applied without prior knowledge, the ABCO framework exhibits sub-optimal performance with limited expert samples. However, when combined with a pre-trained Inverse Dynamics Model (see Section 5.1), ABCO achieves superior results using fewer samples.

To mitigate the issues encountered and further enhance the performance of our models, we developed a new method named IUPE. IUPE stands out by introducing an exploration mechanism that aims to prevent getting trapped in a loop of states (deadlock) while assisting in the search for improved decision-making. By sampling examples using the softmax probabilities, the proposed exploration mechanism suppose to act randomly at first, but eventually, as the model improves, it starts to explore less and exploit more since the probabilities of a specific action should stand out from others. While successfully mitigating the deadlock issue in state transitions, there is still a significant need for a considerable volume of data to train a policy that exhibits significant performance. The greater the complexity of a given task, the more data we require to learn how to perform it effectively. This becomes a concern when addressing real-life problems, as data collection can be expensive and time-consuming.

To tackle the challenge of the data requirement, we dedicated our efforts to developing a new framework called CRIL (see Section 8). CRIL combines the strengths of Reinforcement and Imitation Learning, focusing on achieving significant performance while minimizing the need for a large volume of data. The process updates policy weights based on experiences gained while constructing $\mathcal{I}^{pos}$. As demonstrated in Section 8.3, CRIL matches the outcomes achieved by ABCO and IUPE, albeit with a smaller number of expert samples. We adopted a classic version of the DQN methodology for the reinforcement learning

component to explore how IUPE functions when adjusting the policy weights outside the initial training cycle. However, applying more sophisticated reinforcement learning techniques could yield improved results.

Finally, the last method we designed was named SAIL, which utilizes an adversarial component to emulate the conduct of expert teachers without needing to access their specific actions. Remarkably, SAIL is able to yield cutting-edge results in both performance and efficiency. Two main factors contribute to SAIL's noteworthy performance and AER results using fewer samples. Initially, we employed an adversarial mechanism to better align our model's conduct with the teacher's. This mechanism, integrated into our end-to-end model, evolves iteratively through a loss error, enabling the model to achieve superior returns. Additionally, SAIL integrates an exploration technique that aids the model in amassing the most optimal data from each interaction, thereby expediting the model's convergence. To this effect, we conducted an ablation study in Section 9.3 demonstrating that SAIL can consistently achieve high rewards while successfully 'deceiving' its discriminator.

10.1 Limitations

Despite the persistent dedication and concerted efforts invested in developing each study presented in this thesis and the persistent focus on refining the ABCO framework that served as our inspiration, each study individually possesses its unique limitations. Although the initial two studies, ABCO and IUPE, demonstrated very good results, they were noticeably constrained by efficiency limitations. They also encounter the deadlock issue, previously discussed in prior chapters, due to a deficiency in state exploration capability. This deficiency prevents the policy from becoming ensnared in a specific state transition (e.g., $s_1 \rightarrow s_2, s_2 \rightarrow s_1$). While CRIL has shown an enhancement in efficiency, it grapples with effectiveness when the state space of the problem is not considerably large. This situation triggers the proposed framework to activate its exploration mechanism unnecessarily. Another limitation open to further investigation is the application of more robust reinforcement learning algorithms to augment the method's effectiveness.

A significant challenge during this research was the time-intensive nature of the computational processing required for each experiment. Several experiments spanned weeks before reaching completion, primarily due to the iterative design of the framework under study. Each iteration necessitated the agent to deliberate a specified number of times, inevitably prolonging the duration of the experiments. As the complexity of an environment escalated, the learning curve for the agent to master an appropriate function became steeper. This complexity resulted in sub-optimal agent performance, with the agent operating within an environment only up to the maximum number of predetermined timesteps. During several experiments, an early termination mechanism was employed in the policy implementa-

tion due to the inherent nature of the ABCO and IUPE methods, which typically disregard episodes failing to achieve the set environmental goal.

In contrast, CRIL utilizes each experience to advance its learning process, rendering this approach ineffective. Another hurdle that emerged during our work was the lack of readily available source code from IL methods and well-documented environments. Despite these being frequently published as conference papers of significant stature, they often lack adequate community or creator support. Several of these environments are formulated within the OpenAI Gym [6] framework, simplifying the process of adapting an agent to learn within a new domain. However, some environments do not conform to any framework and lack the necessary documentation to understand the essential components. Moreover, OpenAI Gym environments are traditionally Reinforcement Learning (RL) environments employing RL metrics. This setup introduces two key issues: (i) the need to incorporate IL metrics; and (ii) a predominant focus on vector-state. The first necessitates the addition of new code and subsequent testing for each new environment. In contrast, the second presents challenges when transitioning from vector to image-state.

Lastly, it is crucial to note that the current metrics used for evaluating IL algorithms exhibit certain limitations, particularly in accurately assessing an agent's performance. Metrics such as Performance and Average Episodic Reward primarily depend on the final reward achieved by an agent, neglecting its actual capacity to mimic. This approach enables an agent to learn a different trajectory from the expert yet achieve the same outcome. For instance, consider a maze with two paths of equal length, where the agent and the expert choose different paths. Although the final rewards are identical, the agent has not genuinely learned to imitate the expert. Suppose the primary goal of imitation learning is to develop an agent matching the expert's proficiency. In that case, this disparity may not pose a problem. However, complications arise when a domain fails to incorporate a significant action, such as collecting an item within the maze; these standard metrics would not reflect this issue. Consequently, both Performance and Average Episodic Reward metrics appear overly reliant on the reward function designated for each environment, potentially leading to skewed results.

In response to these limitations, there is a need to create a new metric that considers the distance between the teacher and the learner, aiming to reflect differences in their behavior accurately. However, the computational cost of calculating such a distance, largely dependent on the state representation, could lead to significantly slower training times.

10.2 Future Work

As we understand and find limitations in Imitation Learning from Observation, several improvements and experiments were identified for our future work.

Firstly, the impact of various reinforcement learning mechanisms on the iterative nature of the CRIL framework warrants further investigation. Using more recent algorithms could lead to improved results. However, it may also introduce greater complexity due to the multiple optimization functions involved, a larger number of hyperparameters to manage, and a different approach to combining reinforcement and imitation learning may be necessary. Exploring the use of Inverse Reinforcement Learning to achieve similar results could also be a fruitful line of inquiry. Given that GAIL already employs an IRL approach to approximate its policy from the expert, combining this approach with IL could yield positive outcomes.

While the development of an IL metric was not the primary focus of this research, we understand that there may be a need to create a new method for evaluating the effectiveness of IL algorithms. The final goal of IL methods should extend beyond merely achieving expert rewards. Instead, IL agents should aim to perform trajectories that closely mirror those of the expert samples. This could lead to the creation of more robust and efficient IL algorithms, thereby pushing the boundaries of what is currently achievable in the field of imitation learning. The need for a new metric that captures imitation in its essence represents a potential area for further research and development.

10.3 Published Work

This section provides an organized list of the academic publications that have emerged from this thesis. These published works jointly represent the breadth, depth, and the novel contributions of this study. Furthermore, they serve to emphasize the unique methods implemented, the breakthroughs achieved, and their overall implications on the larger research community. Additionally, Figure 10.1 visually portrays the contributions I have made during my time as a PhD student. This illustrative representation presents both my substantial endeavors in the realm of Imitation Learning (highlighted in green), as well as achievements in diverse areas (represented in gray).

• **Monteiro, Juarez**; Gavenski, Nathan; Meneguzzi, Felipe; and Barros, Rodrigo C. Self-Supervised Adversarial Imitation Learning. In Proceedings of the 36th International Joint Conference on Neural Networks (IJCNN), Queensland, Australia, 2023.
(Qualis A2)

• Gavenski, Nathan; **Monteiro, Juarez**; Medronha, Adilson; and Barros, Rodrigo C. How Resilient Are Imitation Learning Methods to Sub-optimal Experts? In Proceedings of the 11th Brazilian Conference on Intelligent Systems (BRACIS), Campinas, Brazil, 2022.
(Qualis A4)

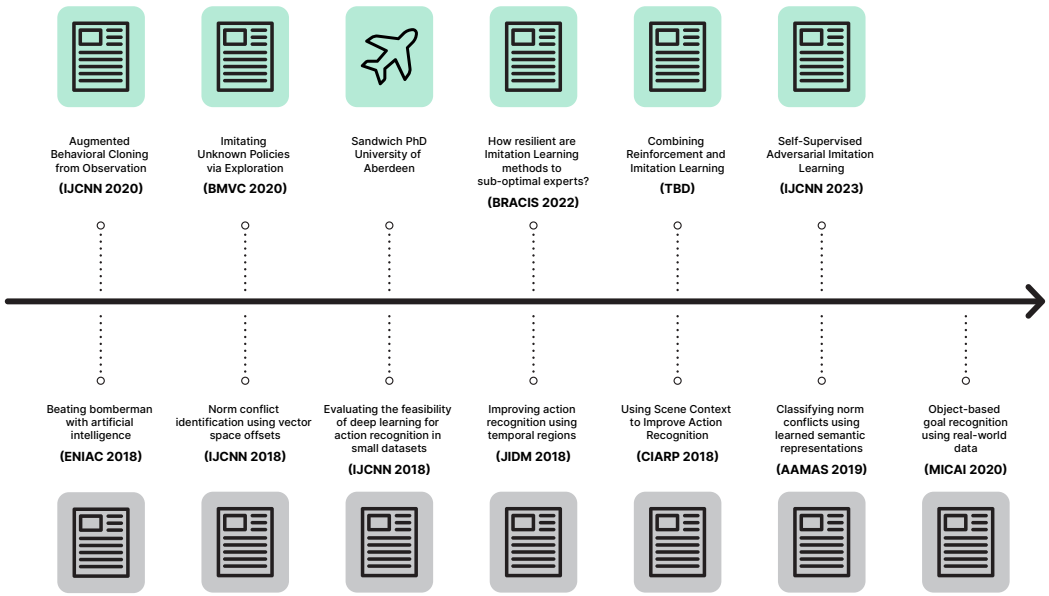


Figure 10.1 – Contributions during my PhD journey.

• Gavenski, Nathan; **Monteiro, Juarez**; Granada, Roger; Meneguzzi, Felipe; and Barros, Rodrigo C. Imitating Unknown Policies via Exploration. In Proceedings of the 31st British Machine Vision Conference (BMVC), Manchester, UK, 2020.
(Qualis A1)

• **Monteiro, Juarez**; Gavenski, Nathan; Granada, Roger; Meneguzzi, Felipe; and Barros, Rodrigo C. Augmented Behavioral Cloning from Observation. In Proceedings of the 33rd International Joint Conference on Neural Networks (IJCNN), Glasgow, Scotland, 2020.
(Qualis A2)

10.4 On Going Work

Here, we enumerate two pivotal works-in-progress: The first primarily centers around an insightful exploration of IfO focused on continuous problems, while the latter is committed to the design and development of a versatile and challenging maze environment to test different RL and IL algorithms.

• Continuous Imitation learning from Observation. Conference/Journal: TBD;

• Adaptive Maze: an Environment for Imitation and Reinforcement Learning algorithms. Conference/Journal: TBD;

REFERENCES

[1] Abbeel, P.; Ng, A. Y. “Apprenticeship learning via inverse reinforcement learning”. In: Proceedings of the 21st International Conference on Machine learning, 2004, pp. 1.

[2] Argall, B. D.; Chernova, S.; Veloso, M.; Browning, B. “A survey of robot learning from demonstration”, *Robotics and Autonomous Systems*, vol. 57–5, May 2009, pp. 469–483.

[3] Arjovsky, M.; Bottou, L. “Towards principled methods for training generative adversarial networks”, *arXiv preprint*, vol. 1701.04862, Jan 2017, pp. 1–17.

[4] Auer, P.; Cesa-Bianchi, N.; Fischer, P. “Finite-time analysis of the multiarmed bandit problem”, *Machine learning*, vol. 47, May 2002, pp. 235–256.

[5] Barto, A. G.; Sutton, R. S.; Anderson, C. W. “Neuronlike adaptive elements that can solve difficult learning control problems”, *IEEE transactions on systems, man, and cybernetics*, vol. 1–5, Sep 1983, pp. 834–846.

[6] Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. “Openai gym”, *arXiv preprint*, vol. 1606.01540, Jun 2016, pp. 1–4.

[7] Ciosek, K. “Imitation learning by reinforcement learning”, *arXiv preprint*, vol. 2108.04763, Aug 2022, pp. 1–15.

[8] Clegg, J. M.; Legare, C. H. “Instrumental and conventional interpretations of behavior are associated with distinct outcomes in early childhood”, *Child Development*, vol. 87–2, Dec 2016, pp. 527–542.

[9] Coates, A.; Abbeel, P.; Ng, A. Y. “Learning for control from multiple demonstrations”. In: Proceedings of the 25th International Conference on Machine Learning, 2008, pp. 144–151.

[10] Degris, T.; White, M.; Sutton, R. S. “Off-policy actor-critic”. In: Proceedings of the 29th International Conference on Machine Learning, 2012, pp. 179–186.

[11] Edwards, A. D.; Sahni, H.; Schroecker, Y.; Isbell, C. L. “Imitating latent policies from observation”. In: Proceedings of the 36th International Conference on Machine Learning, 2019, pp. 1755–1763.

[12] Gatys, L. A.; Ecker, A. S.; Bethge, M. “Image style transfer using convolutional neural networks”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2414–2423.

[13] Gavenski, N.; Monteiro, J.; Granada, R.; Meneguzzi, F.; Barros, R. C. “Imitating unknown policies via exploration”. In: Proceedings of the 31st International British Machine Vision Virtual Conference, 2020, pp. 1–8.

[14] Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. “Generative adversarial networks”, *Communications of the ACM*, vol. 63–11, Oct 2020, pp. 139–144.

[15] Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. “Generative adversarial nets”. In: Proceedings of the 27th International Conference on Neural Information Processing Systems, 2014, pp. 2672–2680.

[16] Graves, A.; Graves, A. “Long short-term memory”, *Supervised sequence labelling with recurrent neural networks*, vol. 385, Jan 2012, pp. 37–45.

[17] Ho, J.; Ermon, S. “Generative adversarial imitation learning”. In: Proceedings of the 30th Conference on Neural Information Processing Systems, 2016, pp. 4565–4573.

[18] Hussein, A.; Gaber, M. M.; Elyan, E.; Jayne, C. “Imitation learning: A survey of learning methods”, *ACM Computing Surveys (CSUR)*, vol. 50–2, Jun 2017, pp. 1–35.

[19] Kaelbling, L. P.; Littman, M. L.; Moore, A. W. “Reinforcement learning: A survey”, *Journal of artificial intelligence research*, vol. 4, May 1996, pp. 237–285.

[20] Kaiser, L.; Babaeizadeh, M.; Milos, P.; Osinski, B.; Campbell, R. H.; Czechowski, K.; Erhan, D.; Finn, C.; Kozakowski, P.; Levine, S.; Mohiuddin, A.; Sepassi, R.; Tucker, G.; Michalewski, H. “Model-based reinforcement learning for atari”, *arXiv preprint*, vol. 1903.00374, Mar 2020, pp. 1–28.

[21] Kidambi, R.; Chang, J.; Sun, W. “Mobile: Model-based imitation learning from observation alone”, *arXiv preprint*, vol. 2102.10769, Jan 2022, pp. 1–28.

[22] Kingma, D. P.; Ba, J. “Adam: A method for stochastic optimization”, *arXiv preprint*, vol. 1412.6980, Jan 2014, pp. 1–15.

[23] Konda, V.; Tsitsiklis, J. “Actor-critic algorithms”, *Advances in neural information processing systems*, vol. 12, Jun 1999, pp. 1008–1014.

[24] Lattimore, T.; Szepesvári, C. “Bandit algorithms”. Cambridge University Press, 2020, 536p.

[25] Le, H. M.; Yue, Y. “Imitation learning tutorial”. ICML Presentation, Source: https://sites.google.com/view/icml2018-imitation-learning, 2021-06-22.

[26] Levine, S.; Finn, C.; Darrell, T.; Abbeel, P. “End-to-end training of deep visuomotor policies”, *The Journal of Machine Learning Research*, vol. 17–1, Apr 2016, pp. 1334–1373.

[27] Liu, Y.; Gupta, A.; Abbeel, P.; Levine, S. “Imitation from observation: Learning to imitate behaviors from raw video via context translation”. In: Proceedings of the International Conference on Robotics and Automation, 2018, pp. 1118–1125.

[28] Maisto, S. A.; Carey, K. B.; Bradizza, C. M. “Social learning theory.”, *Psychological Theories of Drinking and Alcoholism*, vol. 1, Jul 1999, pp. 106–163.

[29] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. “Playing atari with deep reinforcement learning”, *arXiv preprint*, vol. 1312.5602, Dec 2013, pp. 1–9.

[30] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al.. “Human-level control through deep reinforcement learning”, *Nature*, vol. 518–7540, Feb 2015, pp. 529–533.

[31] Monteiro, J.; Gavenski, N.; Granada, R.; Meneguzzi, F.; Barros, R. C. “Augmented behavioral cloning from observation”. In: Proceedings of the International Conference on Neural Networks, 2020, pp. 1–8.

[32] Moore, A. W. “Efficient memory-based learning for robot control”, Ph.D. Thesis, University of Cambridge, 1990, 42p.

[33] Nevmyvaka, Y.; Feng, Y.; Kearns, M. “Reinforcement learning for optimized trade execution”. In: Proceedings of the 23rd International Conference on Machine learning, 2006, pp. 673–680.

[34] Pomerleau, D. A. “Alvinn: An autonomous land vehicle in a neural network”. In: Proceedings of the 1st International Conference on Neural Information Processing Systems, 1988, pp. 305–313.

[35] Raza, S.; Haider, S.; Williams, M.-A. “Teaching coordinated strategies to soccer robots via imitation”. In: Proceedings of the IEEE International Conference on Robotics and Biomimetics, 2012, pp. 1434–1439.

[36] Rizzolatti, G.; Sinigaglia, C. “The functional role of the parieto-frontal mirror circuit: Interpretations and misinterpretations”, *Nature Reviews Neuroscience*, vol. 11–4, Mar 2010, pp. 264–274.

[37] Ross, S.; Gordon, G.; Bagnell, D. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, 2011, pp. 627–635.

[38] Russell, S. J. “Artificial intelligence a modern approach”. Pearson Education, Inc., 2010, 1132p.

[39] Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. “Prioritized experience replay”, *arXiv preprint*, vol. 1511.05952, Nov 2015, pp. 1–21.

[40] Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. “Trust region policy optimization”. In: Proceedings of the 32nd International Conference on Machine Learning, 2015, pp. 1889–1897.

[41] Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; Abbeel, P. “High-dimensional continuous control using generalized advantage estimation”, *arXiv preprint*, vol. 1506.02438, Jun 2015, pp. 1–14.

[42] Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. “Proximal policy optimization algorithms”, *arXiv preprint*, vol. 1707.06347, Jul 2017, pp. 1–12.

[43] Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; Batra, D. “Grad-cam: Visual explanations from deep networks via gradient-based localization”. In: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 618–626.

[44] Sutton, R. S. “Generalization in reinforcement learning: Successful examples using sparse coarse coding”. In: Advances in neural information processing systems, 1996, pp. 1038–1044.

[45] Sutton, R. S.; Barto, A. G. “Reinforcement learning: An introduction”. MIT press, 2018, 552p.

[46] Swamy, G.; Choudhury, S.; Bagnell, J. A.; Wu, S. “Of moments and matching: A game-theoretic framework for closing the imitation gap”. In: International Conference on Machine Learning, 2021, pp. 10022–10032.

[47] Torabi, F.; Warnell, G.; Stone, P. “Behavioral cloning from observation”. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018, pp. 4950–4957.

[48] Torabi, F.; Warnell, G.; Stone, P. “Generative adversarial imitation from observation”. In: Proceedings of the International Conference on Machine Learning Workshop on Imitation, Intent, and Interaction, 2019, pp. 1–8.

[49] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; Polosukhin, I. “Attention is all you need”. In: Proceedings of the 31st Conference on Neural Information Processing Systems, 2017, pp. 5998–6008.

[50] Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al.. “Grandmaster level in starcraft ii using multi-agent reinforcement learning”, *Nature*, vol. 575–7782, Oct 2019, pp. 350–354.

[51] Watkins, C. J.; Dayan, P. “Q-learning”, *Machine learning*, vol. 8–3-4, May 1992, pp. 279–292.

[52] Wooldridge, M. “An introduction to multiagent systems”. John wiley & sons, 2009, 488p.

[53] Wu, Y.; Mansimov, E.; Grosse, R. B.; Liao, S.; Ba, J. “Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation”, *Advances in neural information processing systems*, vol. 30, Dec 2017, pp. 5285—5294.

[54] Zhang, H.; Goodfellow, I.; Metaxas, D.; Odena, A. “Self-attention generative adversarial networks”. In: Proceedings of the 36th International Conference on Machine Learning, 2019, pp. 7354–7363.

[55] Zhu, Z.; Lin, K.; Dai, B.; Zhou, J. “Off-policy imitation learning from observations”, *Advances in Neural Information Processing Systems*, vol. 33, Dec 2020, pp. 12402–12413.

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Pesquisa e Pós-Graduação
Av. Ipiranga, 6681 – Prédio 1 – Térreo
Porto Alegre – RS – Brasil
Fone: (51) 3320-3513
E-mail: propesq@pucrs.br
Site: www.pucrs.br