

Utvärdering av autonom lager- navigering genom generella förstärknings- och imitationsinlärningsalgoritmer

*En jämförande studie av PPO, BC och GAIL metoder för
autonom lager-navigering*

Evaluation of Autonomous Warehouse Navigation through General Reinforcement and Imitation Learning Algorithms

*A Comparative Study of PPO, BC and GAIL Methods for
Autonomous Warehouse Navigation*

Bachelor Degree Project in Information Technology
Basic Level 30 ECTS
Spring 2025

Iilir Ruka

Handledare: András Marki
Examinator: Richard Senington

Acknowledgment

I give my thanks to András Márki at the University of Skövde for support during this thesis project.

This thesis utilizes an example environment from *Godot RL Agents Examples* repository by Edward Beeching, available at https://github.com/edbeeching/godot_rl_agents_examples. The repository is licensed under the MIT License (© 2022 Edward Beeching). The original license text is included in Appendix B.

Abstract

As reinforcement learning (RL) algorithms advance and warehouse automation becomes increasingly important for efficient logistics operations, developing autonomous navigation for robots is a key interest. This study evaluates two machine-learning paradigms within a simulated warehouse environment. First, an RL algorithm called Proximal Policy Optimization (PPO) is evaluated against combined methods that are pre-trained via imitation learning (IL) algorithms and subsequently fine-tuned with PPO (IL + RL). Second, two IL algorithms called Behavioral Cloning (BC), and Generative Adversarial Imitation Learning (GAIL) are evaluated against each other to assess their standalone and combined navigation performance. Together, these experiments show both the benefit of combining IL with RL fine-tuning versus standalone RL, and the comparative value of IL algorithms when used independently (BC versus GAIL) and combined (BC + GAIL) for robot navigation.

The autonomous agent is controlled by a neural network, specifically a multilayer perceptron (MLP). Performance metrics, namely mean reward and sample efficiency are tracked at multiple training milestones. The results show that one method combining BC + PPO (IL + RL) consistently outperforms the PPO (RL) method, even with a low amount of demonstration data used. Also, for the standalone IL evaluations, it shows that BC performs overall better than GAIL for this given game-engine-based environment and MLP complexity. These findings give insight into the generalizability of PPO, GAIL, and BC algorithms outside of domain-specific simulators and the advantages and limitations of both standalone and sequential training methods in autonomous warehouse navigation.

Keywords: Artificial Intelligence, Godot Agents, Reinforcement learning, Imitation learning, Autonomous navigation, Proximal Policy Optimization, Behavioral Cloning

Contents

| | |
|---|-----------|
| Acknowledgment | 1 |
| Abstract | 2 |
| 1 Introduction | 5 |
| 2 Background | 7 |
| 2.1 Machine Learning and Neural Networks | 7 |
| 2.2 Deep Reinforcement Learning | 7 |
| 2.2.1 Proximal Policy Optimization | 8 |
| 2.3 Imitation Learning | 8 |
| 2.3.1 Behavioral Cloning | 9 |
| 2.3.2 Generative Adversarial Imitation Learning | 9 |
| 2.4 Gradient-based Optimization | 11 |
| 2.5 Godot Engine and RL Agents | 11 |
| 2.5.1 Reward Space | 12 |
| 2.5.2 Observations | 13 |
| 3 Problem | 14 |
| 3.1 Aim | 14 |
| 3.2 Related Works | 14 |
| 3.3 Research Questions | 16 |
| 3.4 Hypotheses | 16 |
| 3.4.1 RQ1 Hypotheses | 16 |
| 3.4.2 RQ2 Hypotheses | 16 |
| 3.4.3 RQ3 Hypotheses | 16 |
| 3.5 Objectives | 16 |
| 4 Methodology & Approach | 18 |
| 4.1 Method | 18 |
| 4.1.1 Experimental Approach | 18 |
| 4.1.2 Alternative Methods | 18 |
| 4.2 Environment | 19 |
| 4.3 Training Method | 20 |
| 4.3.1 Stable Baselines3 | 21 |
| Hyperparameters | 21 |
| 4.3.2 Performance Metrics and Evaluation | 22 |
| 5 Results | 23 |
| 5.1.1 PPO | 23 |
| 5.1.2 Behavioral Cloning | 24 |
| 5.1.3 GAIL | 25 |
| 5.1.4 BC + GAIL | 26 |
| 5.1.5 BC + PPO | 27 |
| 5.1.6 BC + GAIL + PPO | 28 |
| 5.2 Result | 29 |
| 5.3 Algorithm Comparison | 30 |
| 6 Discussion | 33 |
| 6.1 Summary | 33 |

| | | |
|----------|---|-----------|
| 6.2 | General Discussion | 34 |
| 6.2.1 | Limitations | 34 |
| 6.3 | Ethics | 35 |
| 6.3.1 | Societal Aspects | 35 |
| 6.3.2 | Research Ethics | 36 |
| 6.4 | Threats to Validity | 37 |
| 6.4.2 | External Validity | 37 |
| 6.4.3 | Internal Validity | 38 |
| 6.4.4 | Construct Validity | 38 |
| 7 | Conclusion | 39 |
| 7.1 | Research Contribution | 39 |
| 7.2 | Future Work | 40 |
| 8 | References | 41 |
| | Appendix A – Raw data of mean rewards | 43 |
| | Appendix B – Godot RL Agents MIT License | 44 |

1 Introduction

Autonomous agents are software-based systems that operate independently in an environment, making decisions to achieve predefined goals. This is central to Artificial Intelligence (AI), a field dedicated to building such systems capable of learning and decision-making. Within AI, Machine Learning (ML) has emerged as an approach, enabling these agents to learn patterns from data. Reinforcement Learning (RL) is a learning paradigm within ML that has become an increasingly popular tool for enabling autonomous agents to learn complex behaviors by collecting data through trial and error. However, RL’s main limitation is that it often requires millions of training timesteps to develop a reliable policy, which is the decision-making rule that maps observations to actions (a function). Thus, the goal in this study is to train a policy network to learn an effective policy that gives the highest cumulative mean reward for RL, or that best imitates an expert for IL.

In this study, the policy is implemented as a multilayer perceptron (MLP), referred to as the policy network. The policy network’s weights are iteratively adjusted during training to refine its ability to perform actions given observations. The goal of training BC, GAIL, and PPO is to optimize this policy network, enabling the agents to achieve their respective objectives efficiently. Essentially, the policy network is the neural network (MLP) that implements the policy, and the MLP refers to the specific architecture used to structure the policy network.

Imitation Learning (IL) utilizes demonstrations to guide the learning process and can operate without RL, referred to as IL-only algorithms in this study. The first IL algorithm used is BC, which trains the policy network by directly mimicking the expert actions through labeled data (supervised learning). However, BC struggles with compounding errors and poor generalization; thus, the second IL algorithm, GAIL, is employed. GAIL uses an adversarial framework that provides continuous reward signals to guide the learning. GAIL encourages the agent to align with the distribution of expert action distributions rather than mimicking the data directly, potentially improving generalization beyond the demonstration data and better performance compared to BC. BC and GAIL tend not to reach an effective policy unless a large amount of demonstration data is used, thus, IL algorithms can also be used as a pre-training step before RL is used to fine-tune the policy, referred to as IL + RL methods. Studies have shown that IL + RL methods can greatly reduce the timesteps needed to reach an effective policy by using IL as a warm start. These methods have shown promise in modern warehouse operations, where autonomous robots have successfully learned to navigate complex environments (Pfeiffer et al., 2018).

The study compares three approaches: PPO as the RL algorithm, and BC and GAIL as the IL algorithms. PPO was selected as the RL algorithm due to being a well-balanced algorithm that is generally applicable to a wide variety of tasks, such as in simulated robotic locomotion tasks and Atari game environments and has been shown to consistently outperform other RL algorithms in terms of final performance and sample efficiency (Schulman et al., 2017). BC was chosen as the first baseline IL algorithm due to it also being used in various fields, originally introduced by Pomerleau (1989), which showed that a neural network trained on human-driven data could autonomously steer a vehicle along roads with no RL feedback. The second IL algorithm, GAIL, aims to evaluate whether learning a reward function from demonstration data can overcome BC’s limitations, and it has also been shown to outperform BC in widely used RL benchmarks (Ho and Ermon, 2016).

By comparing these algorithms in the same game-engine-based setup using the Godot RL Agents simulation framework (Beeching et al., 2021), this study pursues two objectives. First, it aims to understand how each baseline method performs and evaluate to what extent combining IL with RL can enhance the autonomous agent's performance in the warehouse environment. Second, it seeks to determine whether the results align with previous work evaluating PPO, BC, and GAIL in domain-specific simulators, thereby providing insights into the generalizability of these algorithms when applied to a general-purpose simulator that provides reproducibility, ease of use, and is open-source. Consequently, this will guide researchers to use general-purpose simulators rather than relying on simulators that are either difficult to obtain, to reproduce, or generalize outside of the simulator.

To achieve these objectives, an evaluation framework within Godot RL Agents is established to ensure that each algorithm is tested under identical conditions. Next, an experiment is conducted that systematically compares the algorithms in the warehouse environment. First, the RL-only algorithm (PPO) is compared against two combined IL+RL methods, namely BC + PPO and BC + GAIL + PPO. Secondly, the baseline algorithms BC and GAIL, as well as the combined method BC + GAIL, are compared to evaluate the performance of standalone and combined IL algorithms.

2 Background

2.1 Machine Learning and Neural Networks

ML is a branch of artificial intelligence that enables systems to learn from data, identify patterns, and make decisions without explicit programming. ML algorithms can achieve this by continuously adjusting internal parameters based on input data. A common approach to this is called deep learning, involving neural networks and weights, which consist of nodes (artificial neurons) arranged in layers, where weights control how strongly signals pass between them. By iteratively updating these weights, a neural network gradually improves its ability to interpret new inputs and make accurate decisions.

This study trains multiple agents using IL and RL algorithms, each representing a distinct training method. For example, an agent trained only with PPO represents one training method, while an agent trained with BC + PPO represents another. All training methods use the same neural network, which is the policy network (an MLP). However, all training methods that are not trained exclusively with BC also share a second MLP, the value network. This value network estimates the expected cumulative rewards given an observation or observation-action pair, using an advantage function. The advantage function combines environment rewards (for PPO) or adversarial rewards (for GAIL) with the value network's predictions to update the policy network. Both the policy (actor) and value (critic) MLP networks have two layers, with 64 nodes each.

2.2 Deep Reinforcement Learning

RL is a trial-and-error process in which an autonomous agent interacts with an unknown environment. The main challenge in RL is to balance the agent exploring new actions to discover new rewards, with exploiting known actions that yield high rewards. A timestep is one complete cycle where the agent observes the environment's state, selects an action based on its policy, and receives a reward (positive or negative) for that action, which is also referred to as one RL cycle, as shown in Figure 1. An episode is a full sequence of interactions that starts from an initial state and ends when a terminal condition is met, for example, reaching a goal or failing the task. Over a predefined number of timesteps, the policy network's weights are adjusted to maximize the agent's mean reward. Thus, all the learned behaviors the agent acquires are captured in the MLP's parameters. In the chosen environment for this study, the agent initially collides with walls or falls off edges while exploring, and over time, it discovers actions leading to better outcomes.

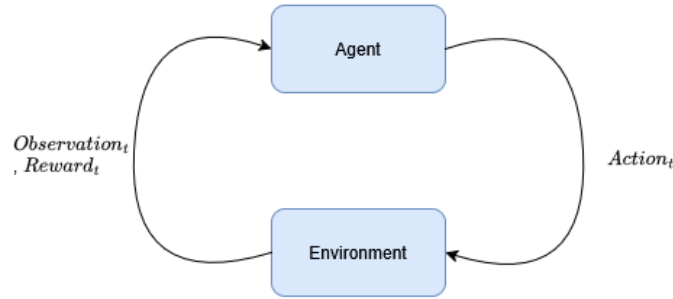


Figure 1: The RL cycle. The agent acts, and the environment outputs observation and reward for the current timestep t .

2.2.1 Proximal Policy Optimization

Proximal Policy Optimization (Schulman et al., 2017) is an on-policy RL algorithm, meaning it uses the current policy’s data to update itself. It is also an actor-critic algorithm, due to using both an actor (the policy network) which collects observations, actions, and rewards, and a critic (the value network) that evaluates a reward for a given observation (scalar value). PPO’s update cycle is shown in Figure 2. First, the actor defines a stochastic policy, meaning that instead of choosing a single action for the agent, it outputs a probability distribution of actions. The actor samples an action from the distribution, and simultaneously, the critic will predict a scalar value used to compute the advantage function in PPO, which measures how much better an action is compared to the average action in that observation. PPO enforces stable learning through its objective function called the clipped surrogate objective, which bounds policy updates and prevents overly large parameter changes.

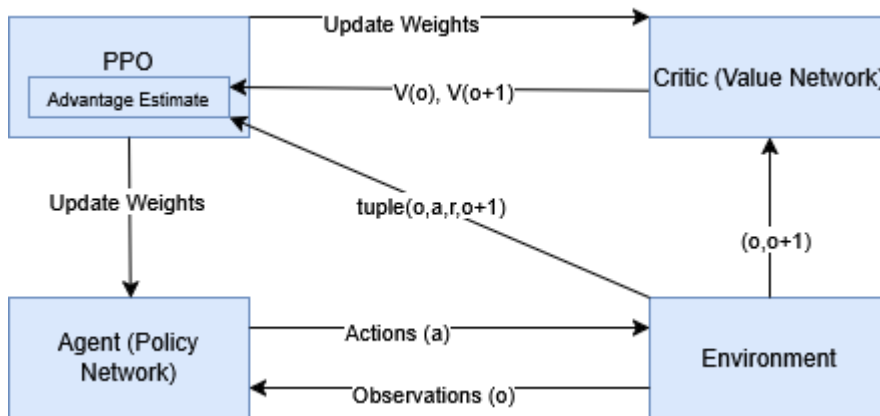


Figure 2: The PPO cycle. The policy network acts, receives an observation, while the value network calculates a value estimate for the observation and the next one. The value estimates are sent to PPO’s advantage estimate function alongside the current observation (o), the current action (a), the reward for that observation (r), and the next observation ($o+1$), consequently updating both the policy and the critic network’s weights.

2.3 Imitation Learning

Imitation Learning (Hussein et al., 2017), also known as learning from demonstration, is a method of RL that changes how the agent’s policy is updated by observing and replicating recorded demonstrations. In this study, expert demonstrations are recorded as observation-

action sequences, which are then used to train the agent to make similar decisions in the same situation. Two chosen imitation learning methods are Behavioral Cloning (BC) and Generative Adversarial Imitation Learning (GAIL), each of which employs a different strategy to incorporate expert demonstration data into the learning process.

2.3.1 Behavioral Cloning

In Behavioral Cloning (Pomerleau, 1991), the agent's policy network is trained using a supervised learning approach with expert demonstration data, making it an off-policy method as it does not learn from the environment directly. As shown in Figure 3, the process involves sampling observations in batches from the demonstration data, which then predicts actions based on those observations. A loss function is used to measure the error between the policy network's predicted actions and the expert's actual actions. The training process is iterated over a predefined number of epochs through the demonstration dataset, an epoch being a single complete pass through the entire training dataset. Over successive epochs, the policy network gradually learns to predict the correct action for each observation and gradually imitates the expert's behavior. However, the main limitation of BC is that it assumes the demonstration data distribution covers all relevant observations; thus, without a reward function to guide the learning, it cannot correct its own mistakes when it drifts into unseen observations.

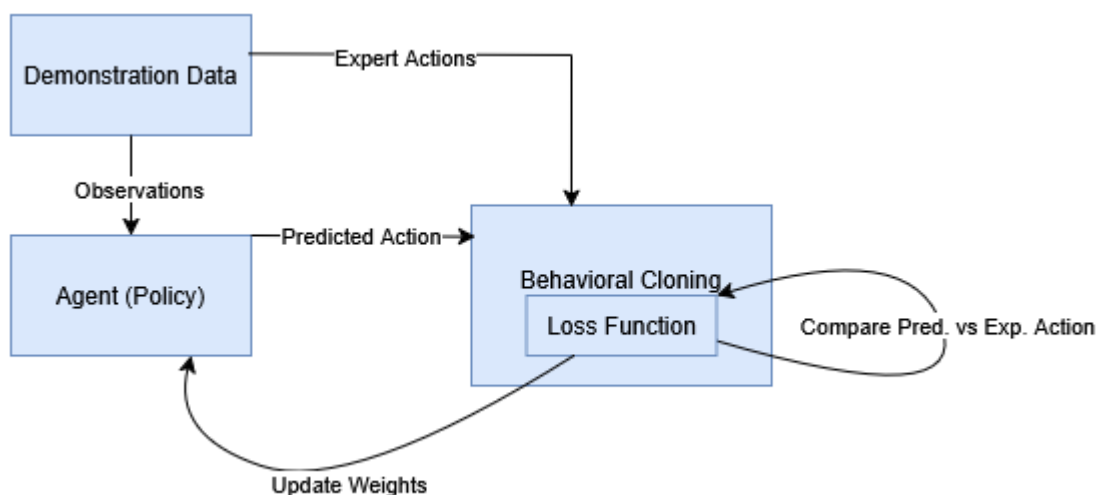


Figure 3: The BC cycle. The policy network is trained on expert observation-action pairs. For each observation, the policy network predicts an action, and a loss function compares the action to the expert's true action. Gradients update the policy network weights, turning it into a policy that imitates the expert.

2.3.2 Generative Adversarial Imitation Learning

In GAIL (Ho & Ermon, 2016), an adversarial framework is used, where a “generator” and a “discriminator” are used to train the policy network to mimic expert behavior. GAIL's cycle is shown in Figure 4. At each training step, the policy network (acting as a generator) receives an observation from the environment and outputs a distribution over actions, from which it stochastically samples and interacts with the environment to collect (o, a) pairs. The discriminator is a separate MLP created for GAIL, which takes both these policy-generated (o, a) pairs and expert demonstration pairs and learns via binary classification to distinguish them. It outputs a probability that each pair is expert-generated, which is then converted into

a reward signal (g_r) to encourage the policy network to fool the discriminator. The value network (critic) uses the GAIL rewards (g_r) to estimate a value function, which PPO uses in an advantage function to update the policy network, consequently generating new action distributions, ensuring that the actor's behavior becomes more expert-like without overshooting. GAIL uses PPO as the RL algorithm to optimize the policy network based on the reward signals given by the discriminator, however, it is still classified as an IL algorithm, as its fundamental learning signal comes from expert demonstrations and its purpose is imitation and not environmental reward maximization. The discriminator network architecture differs from the policy and value networks (32 nodes instead of 64), due to the design choices of the imitation learning library's default configuration.

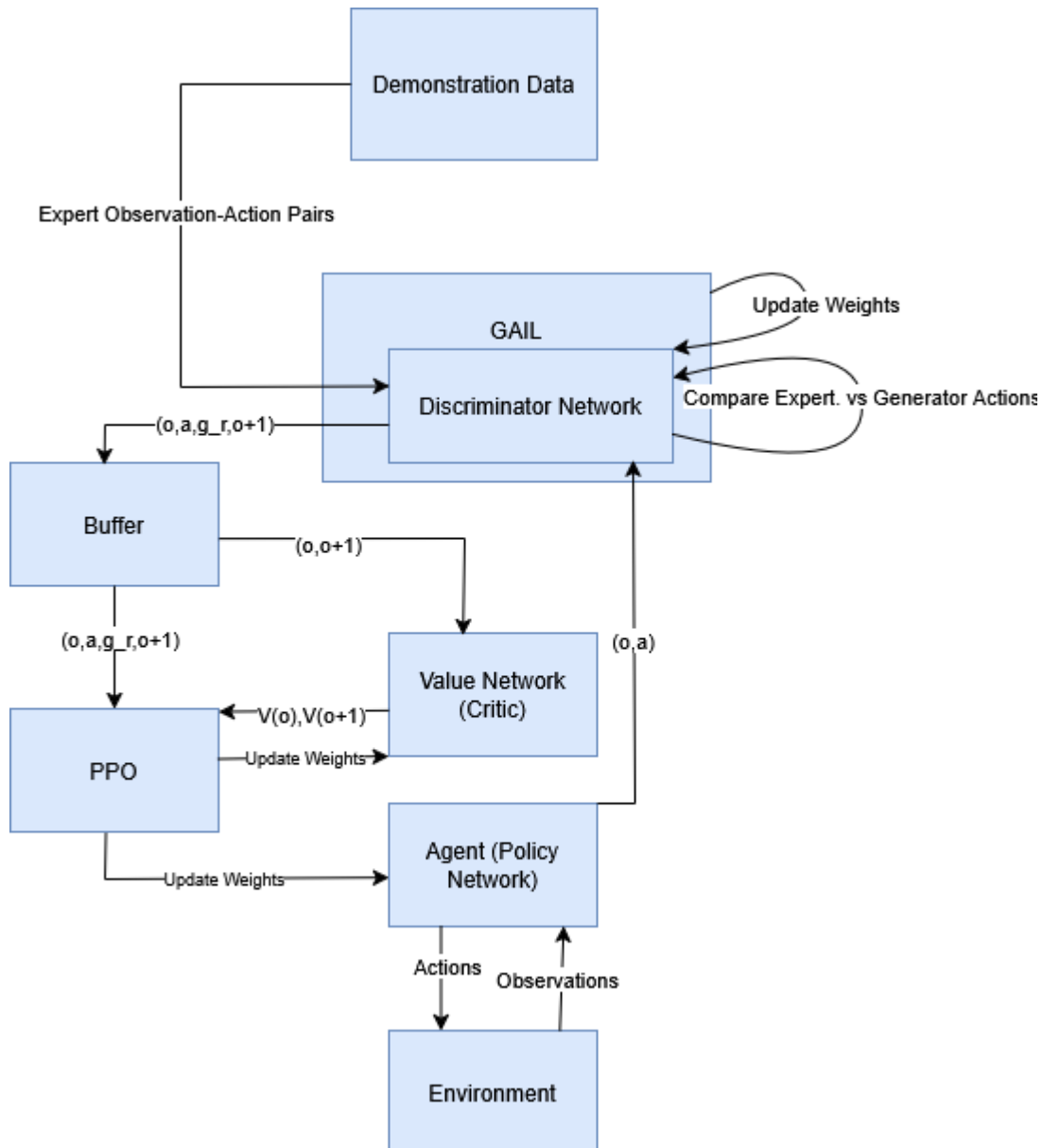


Figure 4: The GAIL cycle. Expert observation-action pairs are fed into the discriminator, alongside observation-action pairs sampled from the policy network (generator) interacting with the environment. The discriminator network learns to distinguish expert from agent behavior, and its output is a reward (g_r) stored in a buffer alongside observation, action, and

the next observation before being used in PPO to calculate advantage estimates and update weights of both the value and policy network.

2.4 Gradient-based Optimization

While all IL and RL algorithms in this study share the same policy network architecture (an MLP), the training objectives and loss functions differ between IL and RL algorithms. The objective of training the policy network is to minimize a loss function, which is achieved via backpropagation. BC minimizes a categorical cross-entropy loss function that measures the difference between the policy network's predicted action probabilities and expert demonstrations. As for PPO, it minimizes the negative of the clipped surrogate objective function, whose goal is to maximize expected cumulative rewards (the total sum of rewards an agent expects to receive until the end of the episode). GAIL uses two loss functions in its framework. The discriminator minimizes a binary cross-entropy loss function to distinguish expert from agent observation-action pairs, the output of which is used to derive a reward signal for the policy network. PPO then uses this discriminator-derived reward to compute an advantage estimate. This advantage is used in the clipped surrogate objective of PPO, which serves as the policy network's loss function within GAIL. Gradients of these losses are computed with respect to the network parameters using the chain rule, indicating how each weight should be adjusted to minimize the loss.

2.5 Godot Engine and RL Agents

Godot Engine (Linietsky et al., 2014) is a game engine that supports 2D and 3D development. It contains an environment scene in which you can create static and dynamic objects. Within the engine, a reinforcement learning framework has been integrated to support the study of various RL algorithms, called Godot RL-Agents. Similarly to Unity ML-Agents, it provides tools and communication protocols that allow RL algorithms to interact with the Godot environment, specifically for training algorithms through Python, which is depicted in Figure 5. In this case, the autonomous agent will be a yellow robot with two wheels, interacting with an environment resembling a warehouse, called *MultiLevelRobot*, which is included by default as an example environment in the official Godot RL-Agents GitHub repository. Figure 5 presents an architectural view of Godot Engine and how components communicate with each other in an RL framework.

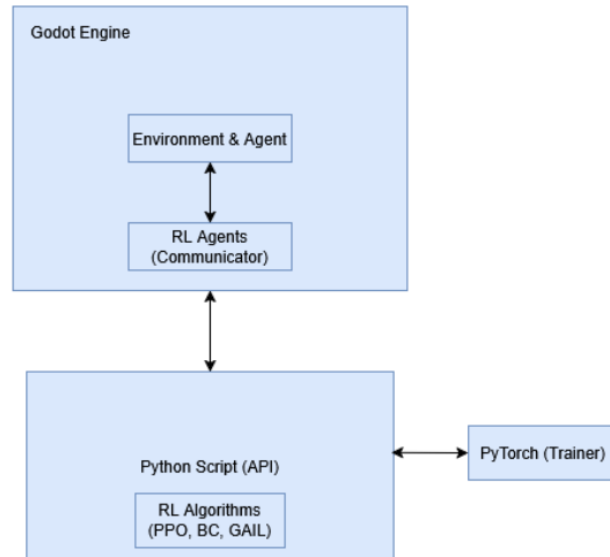


Figure 5: Presents a high-level overview of the general architecture, where the Godot engine simulates the environment and communicates observations to the Python-based RL training pipeline. The pipeline, leveraging IL (via BC and GAIL) and RL (via PPO), is implemented in PyTorch, which trains the policy network.

2.5.1 Reward Space

A reinforcement learning environment can either contain a dense or sparse reward space. During training of the PPO agent, a dense reward space provides consistent rewards for the agent to guide it toward the goal, while a sparse reward space contains minimal or no rewards until it reaches the goal. Since IL agents such as BC and GAIL get reward signals from mimicking expert demonstrations, the reward space only affects them during the evaluation phase, while PPO uses the reward space during both the training and evaluation phases.

The *MultiLevelRobot* environment contains a dense reward space, as indicated by the number of different types of rewards the agent can get throughout its training and evaluation. These rewards were defined by the creator of the environment and were unchanged so that there are constant reward signals for the PPO agent, but also because they “mimic” what could be potentially useful for training an autonomous robot in real-world practice. For example, coins in an environment could be seen as intermediate checkpoints that the robot must go through before going to the next level, and collision with enemies could mimic moving objects in the real world, which must be avoided.

1. Coin pickup: Positive reward (+1.0) upon collecting a coin.
2. Collision with the enemy: Negative reward (-1.0) and episode end.
3. Falling off: Negative reward (-1.0) and episode end.
4. Level completion: Positive reward (+1.0) and episode end.
5. Distance-to-goal: Whenever the agent’s distance to the goal drops below its previous best distance, it earns a small reward proportional to how much closer it is.
6. Time-out penalty: If the agent exceeds a predefined number of steps, the episode ends, and the agent receives a (-1.0) penalty.

2.5.2 Observations

An observation is the information that an agent perceives from its environment at a given timestep, while a state represents all the information needed to predict future rewards. In cases like this study, an observation is used as an approximation of the state. An observation-action pair is a tuple (o, a) , where o is the observation and a is the action chosen by the agent. All positional observations are converted from their world-space `Vector3` into the robot's local space, which will return a `Vector3` indicating direction and distance relative to the robot. A `Vector3` consists of three floats (x,y,z) used to represent any triplet of numeric values in 3D spaces, such as positions and directions. Beyond 30 units (30 meters), the `Vector3` observations will instead be set to zero, so that distant objects do not dominate the observation vector.

Each timestep, the agent receives seven observations:

1. `n_step / reset_after`: The ratio between the current step count and a reset-after value, indicating how far into the episode the agent is.
2. Goal position: The position of the current level's goal in the agent's local reference frame.
3. Closest coin position: The local position of the nearest coin in the current level.
4. Closest enemy position: The local position of the nearest enemy.
5. Enemy movement direction: A vector representing the enemy's velocity, enabling the agent to predict the enemy's trajectory.
6. Agent velocity: The agent's current speed and direction of movement.
7. Coin collection status: A binary flag (0 or 1) indicating whether all coins in the current level have been collected.

3 Problem

Autonomous robot navigation often requires algorithms that balance sample efficiency with high performance. While RL enables adaptive decision-making through environment interaction, its high sample complexity, often requiring millions of training steps, limits scalability. IL offers an alternative by training policies directly on expert demonstrations, but its performance often plateaus due to dependence on the quantity and quality of demonstration data. Combining these paradigms through IL pre-training followed by RL fine-tuning could drastically increase RL’s sample efficiency while overcoming IL’s data limitations. However, existing studies often evaluate IL and RL algorithms in specialized simulators such as robotics manipulation or physics engines, leaving their applicability to generalized frameworks such as game-engine-based simulators understudied. Secondly, previous research reports inconsistent performance for BC and GAIL and understanding how they perform in a sequential training pipeline with PPO remains unclear.

3.1 Aim

This study aims to shed light on how suitable BC, GAIL, and PPO algorithms are in a game-engine-based simulated warehouse navigation task. First, focusing on how sequential training methods perform compared to baseline IL and RL algorithms. Secondly, comparing these results against prior results, the study also aims to show how these algorithms perform outside of domain-specific simulators, to give insights into the generalizability of these algorithms. Thirdly, determine whether a BC + GAIL + PPO training method mitigates GAIL’s instability and shows higher performance compared to PPO.

3.2 Related Works

A study by Whiteson et al. (2004) highlighted an important issue in RL research, which is the tendency to evaluate algorithms in domain-specific simulators, leading to “method overfitting”. As the algorithms are tailored to specific constraints of the emulators, such as neural network size and hyperparameter configuration, this can result in poor generalization of the algorithm’s performance in other domains. The author proposes a parameterized problem generator that systematically varies environmental factors. In this context, Godot RL Agents provides a foundational solution, as its flexible framework supports the creation of diverse environments with controllable parameters (such as obstacle density and reward structures). While this study employs fixed observation spaces and neural networks, Godot’s design avoids domain-specific biases and enables future work to test generalization under systematic variations, aligning with Whiteson’s vision.

In robotics navigation, pre-training a policy with expert demonstrations via IL and then fine-tuning with RL has proven highly effective at reducing sample complexity. For example, Pfeiffer et al. (2018) introduced this sequential training approach for map-less navigation, which led to a significant enhancement in training efficiency, reducing the required environment interactions by roughly 80% compared to pure RL. Their policy network uses a large-scale MLP consisting of three hidden layers, with 1,000 nodes in the first layer, 300 in the second, and 100 in the third. This raises a question of whether similar performance can be achieved with an MLP with two layers and 64 nodes used in this study, being a “low complexity” neural network relative to this study.

Similarly, Hester et al. (2018) highlighted that Deep Q Network (an RL algorithm) has poor initial performance and extreme sample demands, thus demonstrating their IL+RL algorithm *Deep Q-learning from Demonstrations (DQfD)* which showed massive sample efficiency gains from leveraging limited demonstration data compared to a pure RL algorithm, which needed 83 million steps to catch up to DQfD. However, the time taken to create the demonstration data is frequently excluded in papers, such as in the *DQfD* paper, because they are often gathered from existing logs or a scripted controller that automatically generates “perfect” expert actions. Because it only needs engineer time to write the code, the time cost of creating demonstration data is negligible, and thus is also an excluded variable in this study. Lastly, the neural network architecture used in the paper was not described; thus, only a comparison of their IL + RL algorithm against the ones in this study can be used, with no regard for neural network differences.

The method of combining IL and RL has been extended beyond the field of robotic navigation. Silver et al. (2016) also demonstrated the power of IL+RL by introducing AlphaGo, the first program to defeat a human professional in the game of Go. Their approach used supervised learning of a policy network trained on 30 million expert human moves, followed by RL to further improve it. They also reduced the broad search space of Go and substantially accelerated learning, which is a motivation for this study to explore if the same method can be used to cut the training timesteps needed for PPO in a robot navigation scenario.

This work is also motivated by two contrasting studies, the first one by Ho and Ermon (2016), who proposed GAIL as an alternative to BC, noting that behavioral cloning requires extensive demonstration data to be effective. Their work demonstrated that GAIL could perform better than BC in high-dimensional, physics-based tasks. They also briefly discuss that pre-training with BC and fine-tuning with GAIL could significantly improve learning speed for GAIL, however, they do not report the results of this method. By contrast, Silva and Calvo (2023) found that in a simulated mobile-robot navigation benchmark, BC produced more efficient trajectories and lower path error than GAIL, showing a conflict with the first work. While the results between the papers are not directly comparable with each other, Silva and Calvo (2023) used an MLP architecture with two hidden layers and 32 nodes, which is very similar to the one used in this study. Thus, despite the paper using a domain-specific simulator, it does motivate this research that seeks to determine whether BC or GAIL can achieve a high-performing policy for robot navigation in this simulator with a similar MLP architecture, and whether the combined BC + GAIL method can offer improved performance, aligning with Ho and Ermon (2016).

The final method used in this study, specifically BC + GAIL + PPO, seeks to find out whether fine-tuning a BC + GAIL training pipeline with PPO can help mitigate GAIL’s inherent instability, as noted by Luo et al. (2024). A similar sequential pipeline has been evaluated previously by Wang et al. (2024), focusing on the performance of BC + PPO + GAIL (GAIL and PPO order switched) in construction work, and showing better performance for tower crane lift path planning compared to PPO. Thus, this study seeks to find out if BC + GAIL + PPO, compared to PPO, is comparable to their results and shows generalizability outside of their environment, adding to further research in these algorithms and how they perform sequentially. The neural network architecture in their study is also an MLP, with three layers with 512 nodes each, making their MLP considerably more complex compared to the one used in this study.

3.3 Research Questions

RQ1: Given the relatively low-complexity neural network and limited demonstration data, does BC or GAIL achieve a higher mean cumulative reward on the warehouse navigation task?

RQ2: Does pre-training with BC reduce the sample complexity (timesteps) required for PPO or GAIL to reach a target performance threshold in the same task?

RQ3: Does the sequential BC + GAIL + PPO method show higher performance compared to PPO in the same task?

3.4 Hypotheses

3.4.1 RQ1 Hypotheses

1. H1.0: There is no significant difference in the final mean cumulative reward between standalone BC and GAIL
2. H1.1: There is a significant difference in the final mean cumulative reward between standalone BC and GAIL

3.4.2 RQ2 Hypotheses

1. H2.0: Pre-training with BC does not reduce the timesteps required for GAIL or PPO to reach the target performance threshold.
2. H2.1: Pre-training with BC reduces the timesteps for GAIL or PPO to reach the target performance threshold.

3.4.3 RQ3 Hypotheses

1. H3.0: Sequential BC + GAIL + PPO training does not reduce GAIL's reward variance (instability) or improve the final mean cumulative reward.
2. H3.1: Sequential BC + GAIL + PPO training reduces GAIL's reward variance (improves stability) and achieves a higher final mean cumulative reward compared to PPO.

3.5 Objectives

The following objectives are designed to address the research questions outlined in this study:

1. Set up a multi-level warehouse environment in a game engine utilizing reinforcement learning agents.
2. Ensure the agent can be trained with PPO, GAIL, and BC.
 - PPO for Reinforcement Learning
 - BC and GAIL for Imitation Learning
3. Define and execute a training protocol for each algorithm across all levels.
4. Log performance metrics
 - Mean Reward: After training, the agent's performance is evaluated over 200 episodes, and the mean reward is computed by averaging the total rewards across these episodes.
 - Sample efficiency: How many training timesteps or epochs are needed for each algorithm to reach an effective policy?
5. Perform statistical tests to discover performance differences among the methods.
6. Generate visualizations to highlight observations.

7. Consolidate findings to determine how each algorithm performs.

4 Methodology & Approach

4.1 Method

The primary goal of the experiment is to compare several RL and IL algorithms in a warehouse environment. By systematically evaluating the agents' performance across the levels, the performance metrics of both individual and sequentially combined methods can be assessed. To ensure that differences in reward curves could be attributed solely to algorithmic design rather than environmental noise or random-seed variability. A controlled simulation methodology is used, with each training method being evaluated on the same environment with seed-paired training runs, and results aggregated over multiple trials using a nonparametric test to avoid distributional assumptions (Demšar., 2006). To draw cause-effect conclusions about baseline IL and RL algorithms and their sequentially trained methods impact navigation performance, all conditions are standardized except the learning method.

4.1.1 Experimental Approach

In this study, the selected algorithms are the chosen independent variables:

- PPO
- BC
- GAIL
- BC+GAIL
- BC+PPO
- BC+GAIL+PPO

The dependent variable is the agent's performance in each level of the environment, measured by mean reward.

First, the baseline algorithms, PPO, BC, and GAIL, are run to evaluate their policies. Second, the algorithms are run in a sequential training pipeline, in which each policy is pre-trained via IL methods (either BC or GAIL) and then fine-tuned with RL (PPO).

4.1.2 Alternative Methods

Systematic Literature Review

A systematic literature review (SLR) examines previous studies to consolidate existing evidence. SLRs rely on detailed and comparable studies; however, in fields like deep reinforcement learning, specific hyperparameter settings and environment configurations are frequently excluded, which complicates direct comparisons. Since this study involves experimentally evaluating multiple algorithms in a novel environment, with specifically selected algorithms and hyperparameters, an SLR would be insufficient to generate the empirical data.

Survey

A survey could collect qualitative data from industry experts on their experiences with various robot navigation algorithms. However, this method would not generate quantifiable data that can be used to strictly evaluate performance differences in a controlled experimental setup.

Field Observation

Observational studies in real-world settings, like in warehouse environments, can provide insights into practical performance. However, the lack of control makes it challenging to isolate the impact of specific variables, such as the choice of learning algorithm or differences in hyperparameter settings. However, field observations do have the potential to identify practical deployment issues of autonomous robots.

4.2 Environment

This environment consists of eight levels that the agent must complete in sequence. Some levels require the agent to collect all coins before advancing, while the final level introduces mobile red robots to avoid. The agent must move to the spawned green goal. At runtime, there will only be one existing green object for each level, where the starting position for the object is randomly chosen.

Level 1 & 2

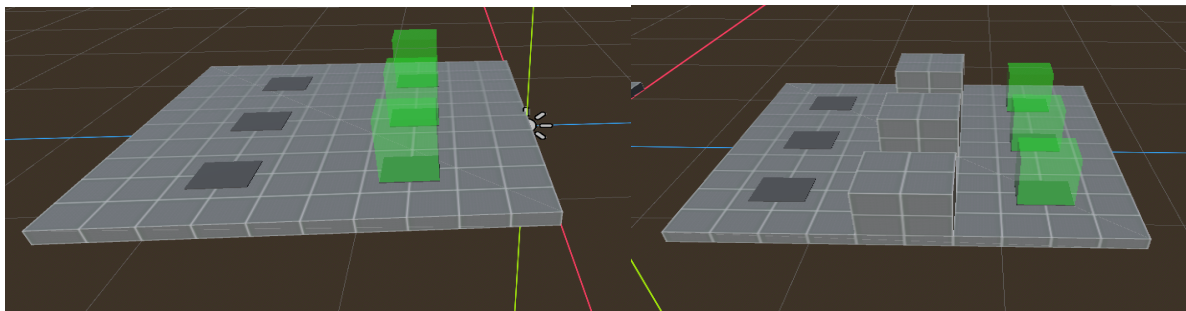


Figure 6: First and second level of the environment.

In Figure 6, the agent must first navigate a short distance to the green object. In the second level, the agent is blocked by 3 objects in the middle and must maneuver its way to the goal.

Level 3 & 4

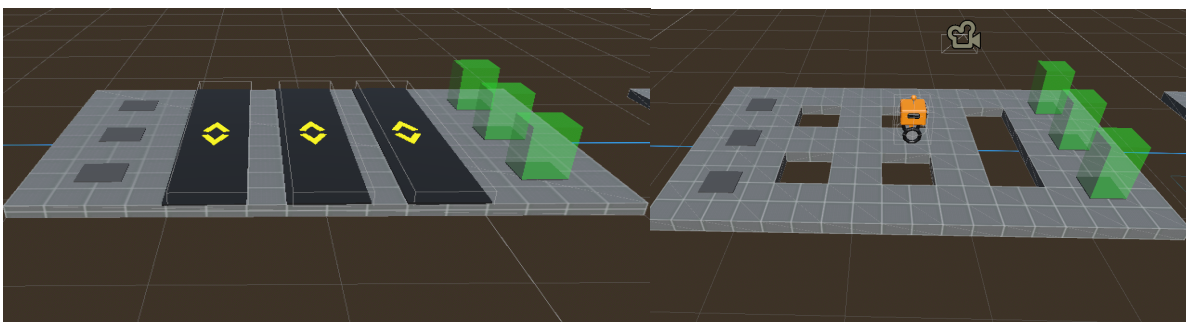


Figure 7: Third and fourth level of the environment.

In Figure 7, there are three tracks in the middle pushing the agent in one direction in level 3. The agent must counteract this by going in the opposite direction while also moving towards the goal. In level 4, the level now has multiple holes in the area that the agent must maneuver around.

Level 5 & 6

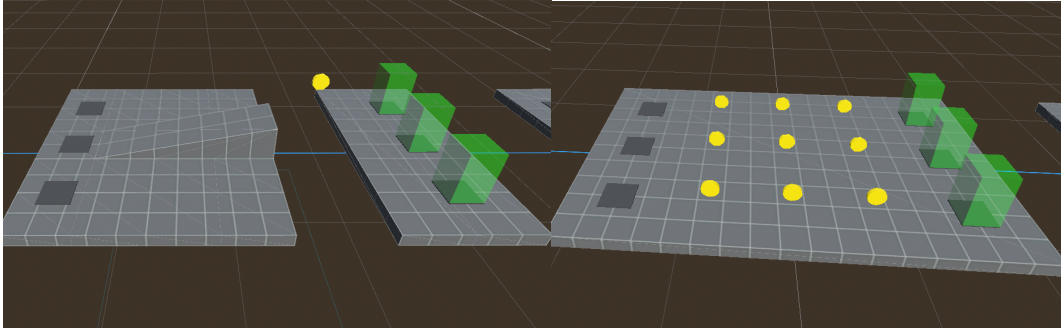


Figure 8: Fifth and sixth level of the environment.

In Figure 8, level 6 has a ramp and a gap in the middle. The agent must go across the ramp and collect the gold coin to land on the other side and activate the goal. In level 6, the level contains nine gold coins that must be collected before moving towards the goal.

Level 7 & 8

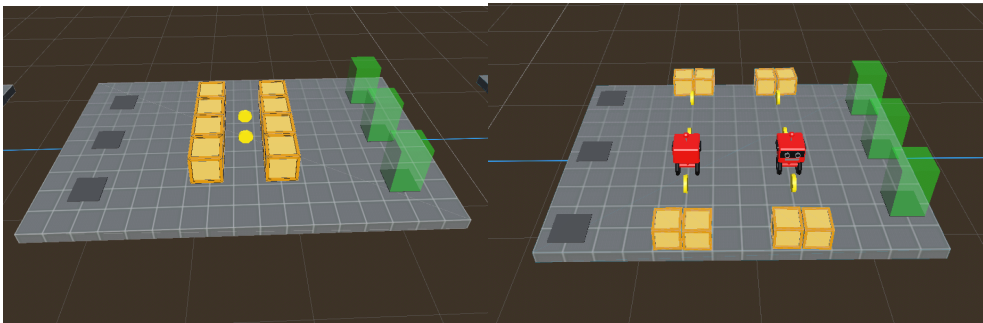


Figure 9: Seventh and eighth level of the environment.

In Figure 9, level 6 has gold coins blocked by boxes, and the agent must maneuver its way around the boxes to collect the gold coins before moving towards the goal. In level 8, six gold coins are protected by enemies. The agent must maneuver around the enemies whilst collecting them before moving towards the goal.

4.3 Training Method

Six different learning approaches are trained in the environment. For any method that includes imitation learning algorithms (BC and GAIL), 30 demonstrations were recorded for each level in the environment and fed into the agent for training. This small demonstration size was chosen to test whether even limited expert data can significantly improve sample efficiency when combined with RL, and whether the results align with the related works.

Proximal Policy Optimization (PPO)

- Group 1-4: 500k/1M/1.5M/2M timesteps

Behavioral Cloning

- Group 1-4: 50/100/150/200 epochs

GAIL

- Group 1-4: 500k/1M/1.5M/2M timesteps

BC + GAIL

- Group 1-4: Fixed 150 BC epochs, 500k/1M/1.5M/2M timesteps

BC + PPO

- Group 1-4: Fixed 150 BC epochs, 500k/1M/1.5M/2M timesteps

BC + GAIL + PPO

- Group 1-4: Fixed 150 BC epochs, 500k/1M/1.5M/2M timesteps (timesteps between GAIL and PPO each split equally).

4.3.1 Stable Baselines3

Stable Baselines3 (SB3) is an open-source library for reinforcement learning built on top of PyTorch. PyTorch is the deep learning framework used under the hood by SB3 to handle all neural network operations, such as defining the MLPs, computing gradients, and updating weights. Stable Baselines3 has been integrated into the Godot RL-Agents framework and provides implementations of several RL algorithms, such as PPO. In this study, SB3 is used to train and evaluate PPO in the *MultiLevelRobot* environment.

Imitation Library

To train the imitation learning methods BC and GAIL, the Imitation library by HumanCompatibleAI (Gleave et al., 2022) is used. The Imitation library is an open-source Python package that is built upon PyTorch and SB3.

Hyperparameters

Hyperparameters are the variables in RL algorithms that determine how the learning performs. These are tuned by the user, which in this case has been kept to the recommended settings based on the documentation, as they had been designed specifically for this environment.

Behavioral Cloning

- `batch_size=32`: Number of demonstration samples processed per training batch.
- `ent_weight=0.001`: A scalar that multiplies the policy's entropy bonus in the loss.

Generative Adversarial Imitation Learning

- `demo_batch_size=256`: Number of demonstration samples used per discriminator training batch.

- `n_disc_updates_per_round=16`: Determines how many times the discriminator is updated for each round of generator updates.

Proximal Policy Optimization

- `batch_size=256`: The size of each batch used in every policy update.
- `ent_coef=0.007`: Encourages exploration by keeping the policy from becoming too deterministic.
- `learning_rate=0.0002`: The magnitude of weight adjustments made during training.
- `n_steps=64`: Number of timesteps collected before performing a policy update.
- `target_kl=0.02`: The target maximum divergence between the new and old policies to maintain stable updates.
- `n_epochs=5`: Number of epochs over the collected data during each update cycle.

4.3.2 Performance Metrics and Evaluation

To measure performance, the mean cumulative reward is the primary metric, since it directly reflects success in clearing all levels and efficiency in navigation. As noted during training, a mean cumulative reward of around 4.5 shows that an agent is consistently clearing all levels efficiently and thus has been set as the target performance threshold that all algorithms should aim to achieve. To ensure that the comparisons reflect algorithmic differences rather than chance, each algorithm and timestep group is trained three times using fixed random seeds (0, 1, and 2). After training, each run is evaluated over 200 episodes to produce a single mean reward per seed, giving three paired mean rewards per group.

1. Mean Cumulative Reward
2. Sample Efficiency

In deep reinforcement learning, inherent randomness arises from multiple sources within the training pipeline, such as the weight initialization from PyTorch, any stochastic logic from Python, and the simulated environment itself. This is important because, according to a study by Henderson et al. (2018), deep reinforcement learning outcomes show high variability due to random seed initialization, environment stochasticity, and exploration noise, making single-run point estimates unreliable for evaluating an algorithm's performance. One key point made in the study is to mitigate random seed initialization, which, in their experiment, shows that without random seed selection, averaging results across different random seeds can lead to misleading results if not enough runs are performed. Therefore, the mean rewards are paired with their respective seeds. Specifically, the first mean reward in any timestep group has seed 0, the second mean reward has seed 1, and the third mean reward has seed 2. Thus, the mean rewards can be compared across different timestep groups and algorithms, and because they are paired, statistical significance can be assessed using the Wilcoxon signed-rank test, which is a nonparametric rank-based test for median shifts in matched samples, and recommended by Demšar, (2006) in their study “*Statistical comparisons of classifiers over multiple data sets*”.

5 Results

5.1.1 PPO

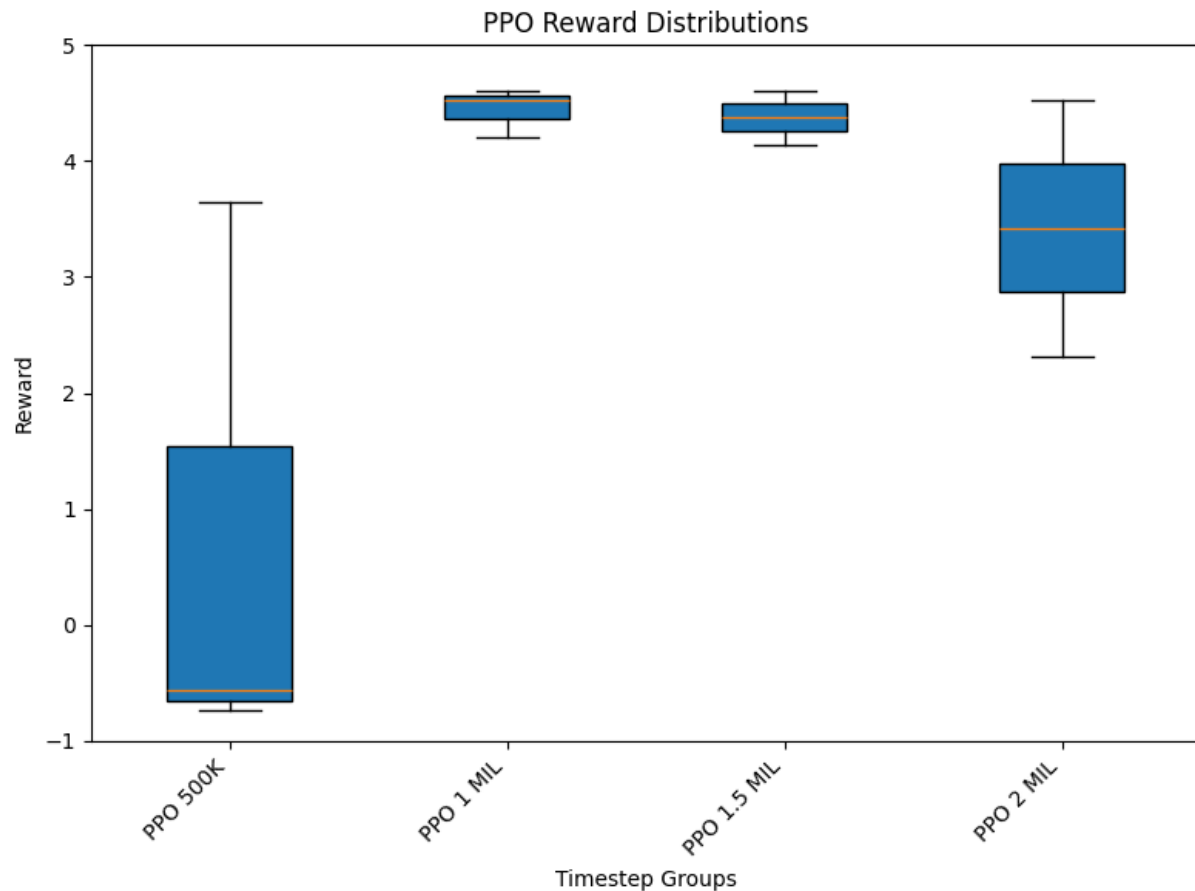


Figure 10: Reward distribution of PPO.

For the results in Figure 10, PPO shows an overall low reward when trained to 500k timesteps, as expected due to RL naturally needing a lot of timesteps to exploit the dense reward signals. As training progresses to 1 M and 1.5 M timesteps, the policy has already reached the target mean reward of 4.5, and the reward distribution shifts upwards as high as 4.5. When observing the agent's policy with a mean reward of 4.5, it showed being able to consistently clear all levels of the environment with minor flaws. Thus, a mean reward of 5.0 has been set as the upper limit of what an algorithm can achieve. Training for longer than 1.5 M did show slightly lower performance, which could be related to the variance in the three runs performed and already having converged, potentially causing a longer duration to influence it negatively. However, the most likely reason for the slight shift downward in performance is due to the sample size of the runs being too small relative to most studies in RL.

5.1.2 Behavioral Cloning

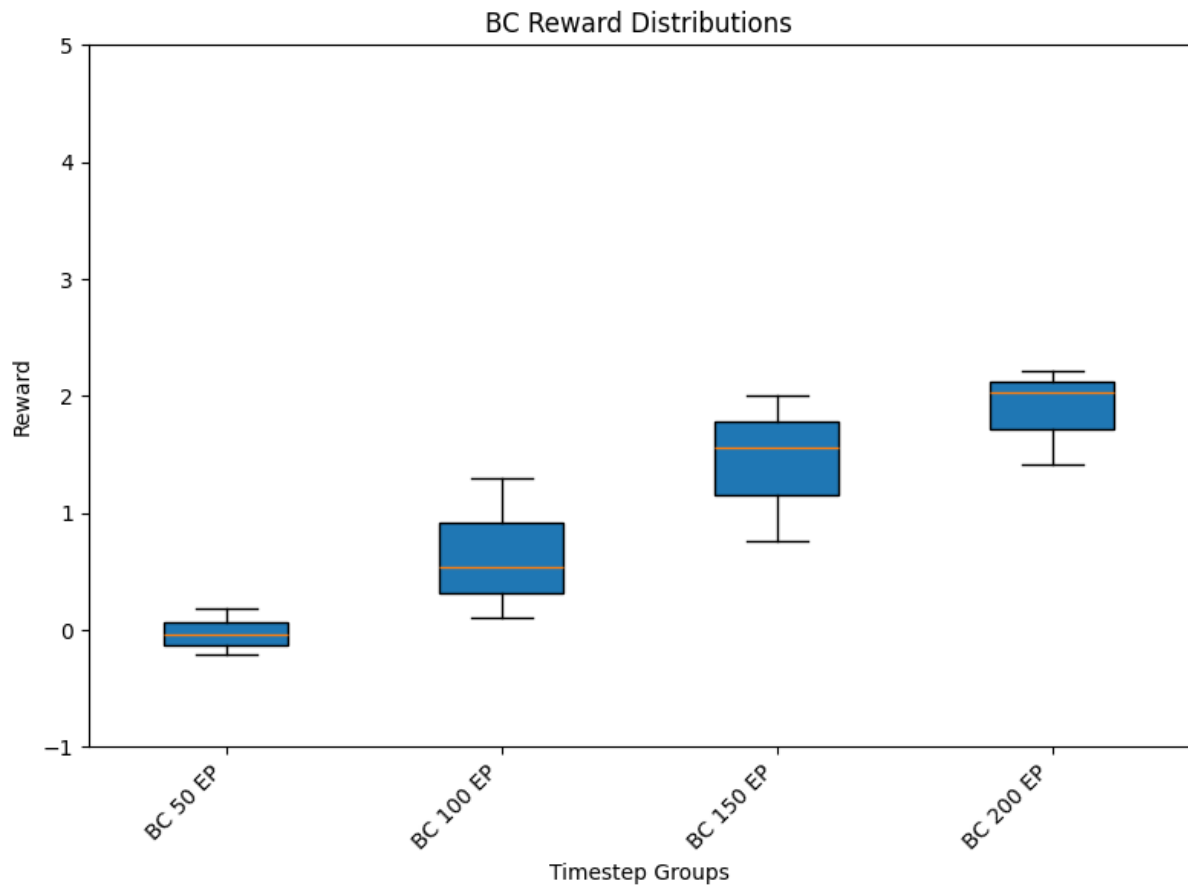


Figure 11: Reward distributions of BC.

The results in Figure 11 show a linear progression in the performance of BC as training epochs increase. The 200 epochs BC method achieved the highest performance, with a median reward approaching 2.0. Thus, even with low demonstration data, the amount of training duration influences BC positively.

5.1.3 GAIL

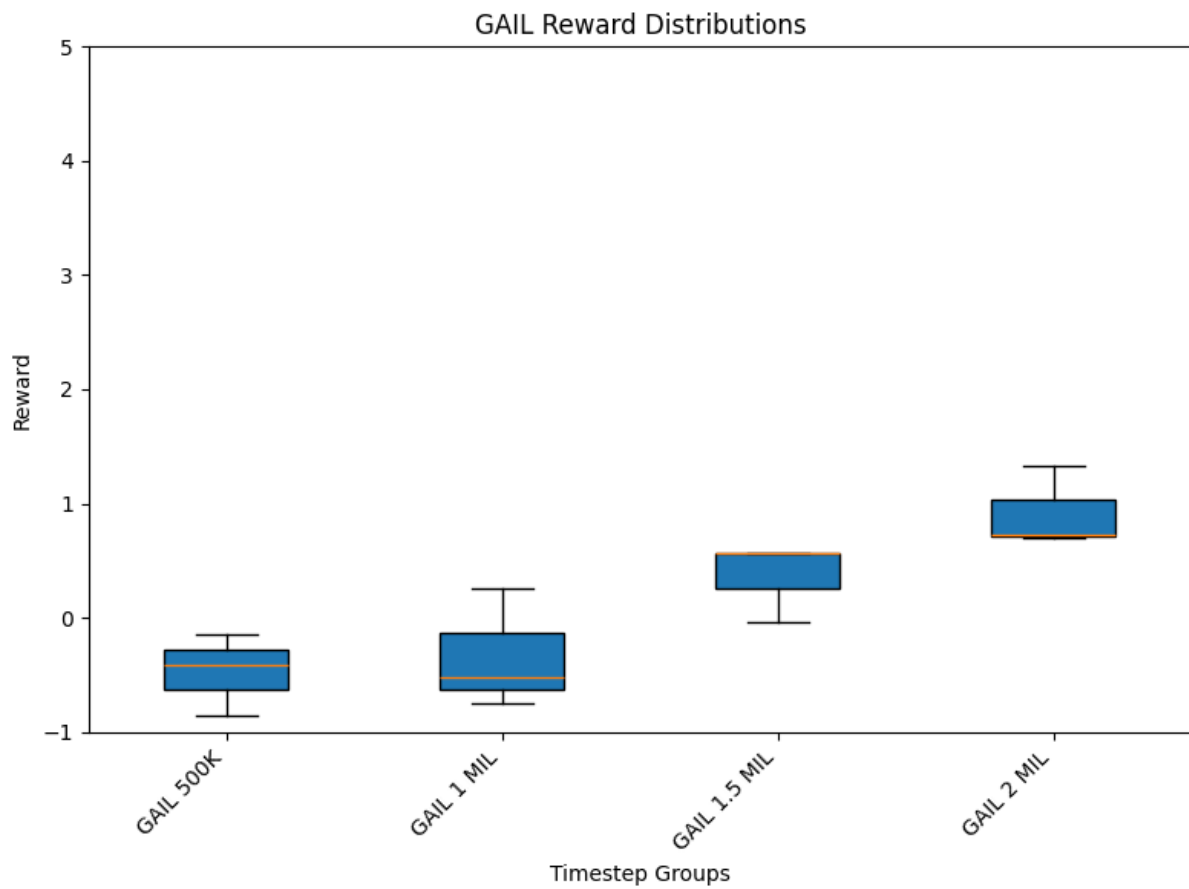


Figure 12: Reward distribution of GAIL.

Figure 12 shows that GAIL is struggling to reach the target mean reward of 4.5 at every timestep group. However, as training progresses to 1 M and 2 M timesteps, the reward distribution shows further improvement. Considering the baseline IL algorithms, BC performs on average better than GAIL in this environment, aligning with the study by Silva and Calvo (2023).

5.1.4 BC + GAIL

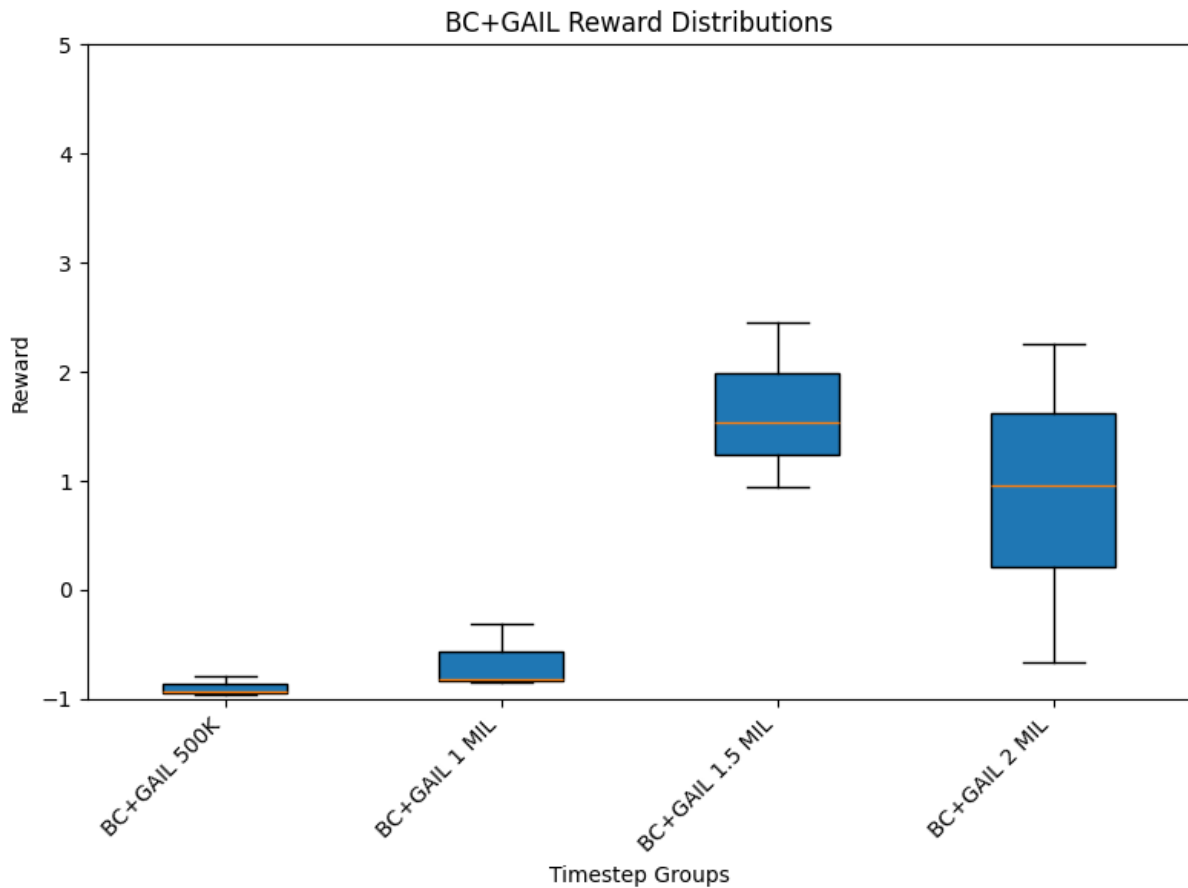


Figure 13: Reward distribution of BC + GAIL

Figure 13 shows that BC + GAIL does not reach the target performance of 4.5 across all timesteps groups. By 2 M timesteps, the median reward shifts slightly lower than at 1.5 M with a high variance in the runs, indicating that the training is oscillating and not converging to an effective policy. These results align with the insights in the study by Jena et al. (2021), who found that pre-training with behavioral cloning and fine-tuning with GAIL causes the agent to learn a sub-optimal policy compared to running GAIL by itself, and contrasts with the original paper of GAIL, Ho and Ermon. (2016), which highlighted that BC + GAIL can significantly improve GAIL's learning speed.

5.1.5 BC + PPO

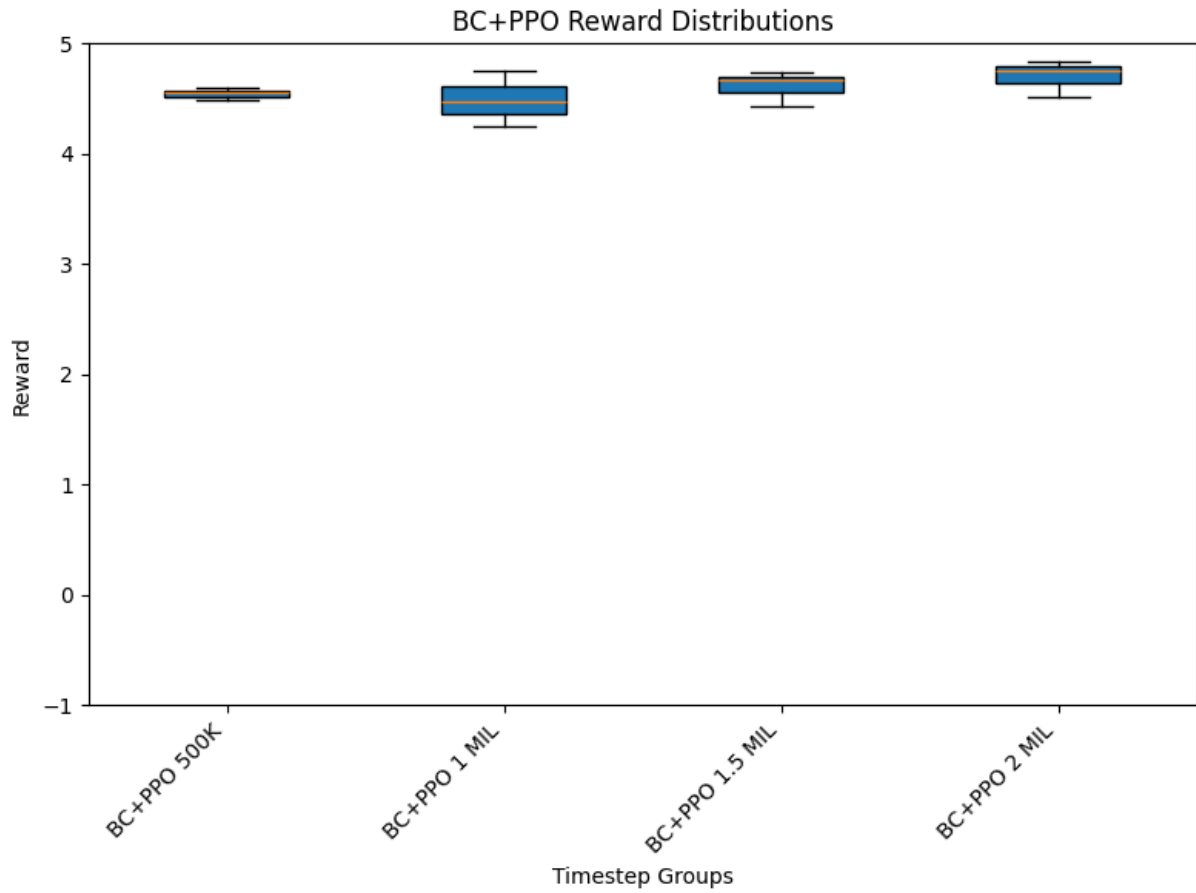


Figure 14: Reward distribution of BC + PPO

For Figure 14, BC + PPO shows the best performance overall, reaching the target mean reward of 4.5 with only 500 K PPO timesteps. This shows that the combination of IL + RL with the right algorithms can achieve a much higher sample efficiency compared to RL-only methods while matching or potentially averaging a higher reward.

5.1.6 BC + GAIL + PPO

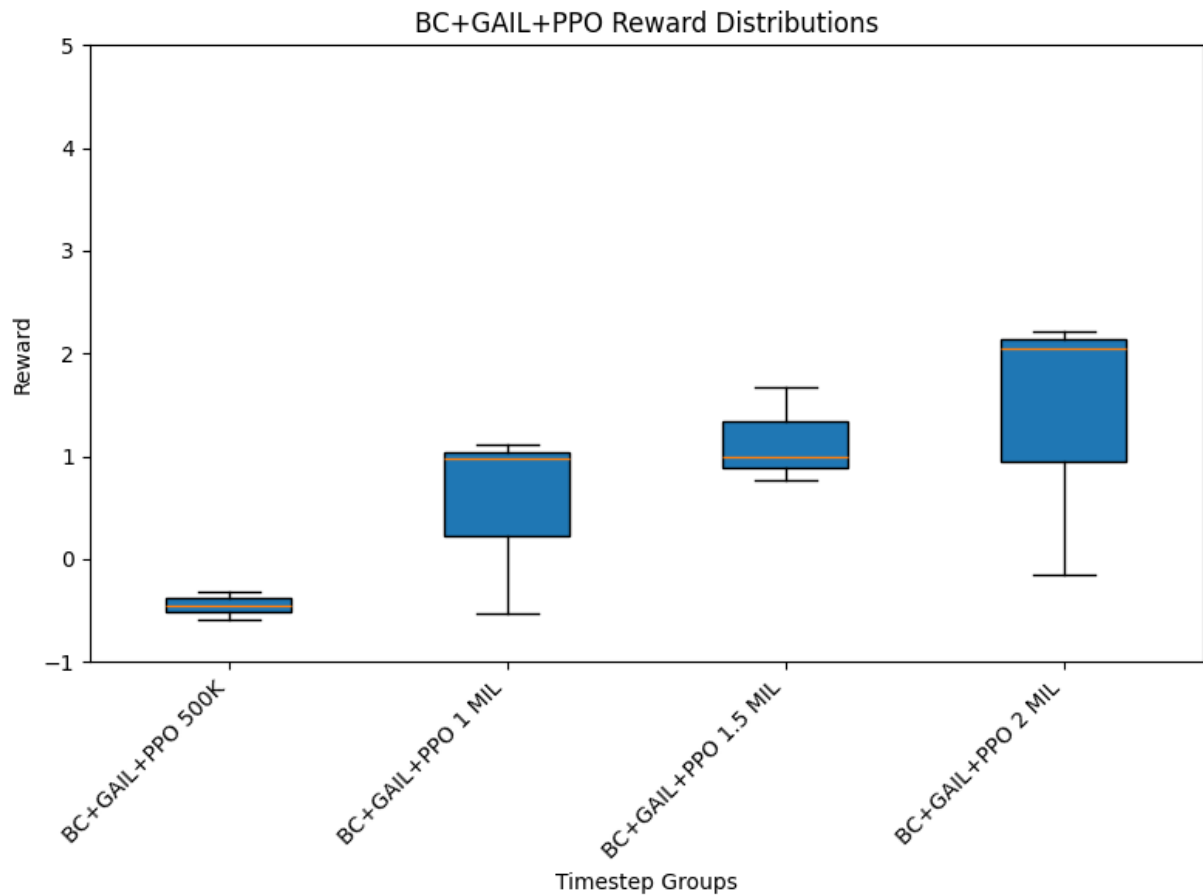


Figure 15: Reward distribution of BC + GAIL + PPO.

Figure 15 shows that BC + GAIL + PPO has an average reward of 2.0 at 2 M timesteps. Compared to PPO and BC + PPO, the performance is much worse. The worse performance could be attributable to several factors. First, fine-tuning the BC pre-trained policy with GAIL could negatively affect the overall method for the same reason as described in section 5.1.4 BC + GAIL. Consequently, training the policy afterward with PPO does not help the algorithm substantially, unlike the BC + PPO method, due to the policy having learned behavior from GAIL that does not translate well to PPO. In the discussion section, this theory is discussed further.

5.2 Result

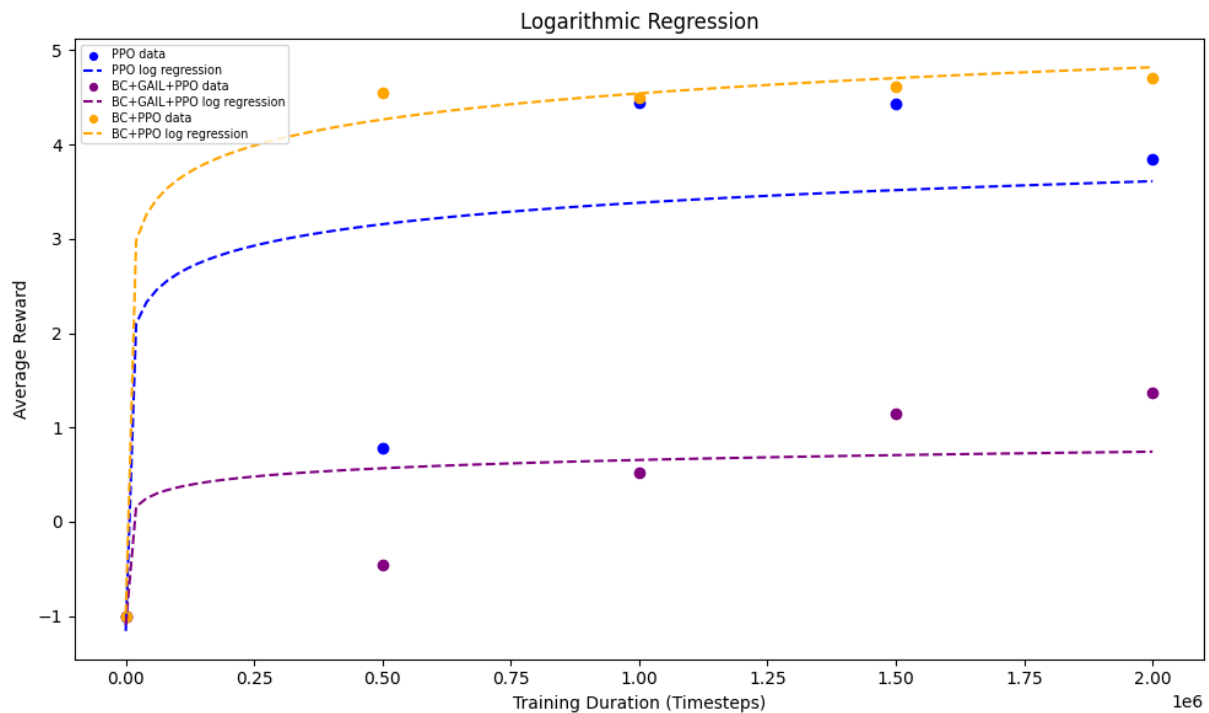


Figure 16: Logarithmic regression of average reward across RL and IL+RL algorithms.

Figure 16 shows how the average reward evolves across training timesteps for each RL algorithm. The top performer was BC + PPO, showing that pre-training with BC helps PPO later rapidly converge to an optimal policy with a mean reward of around 4.5, meaning it can consistently clear all levels. As for PPO, a statistical comparison must be made to see if there is a significant difference between BC + PPO and PPO, as it also performs consistently well, as expected due to RL being guided by dense reward signals and experiencing environment behavior that does not appear in the demonstration data. The BC + GAIL + PPO method was the worst performer and does not come close to contending with BC+PPO or PPO by itself.

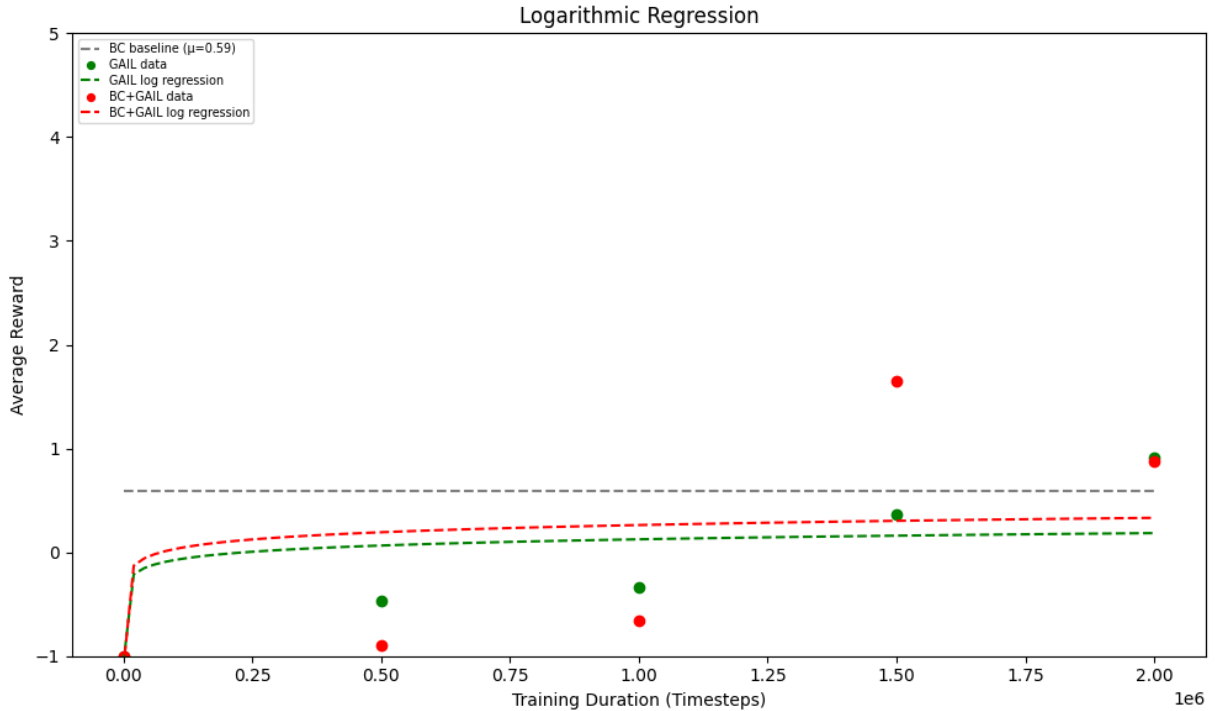


Figure 17: Logarithmic regression of average reward across IL algorithms.

Figure 17 shows how the average reward evolves across training timesteps for each IL algorithm. The IL-only methods (BC, GAIL, and BC+GAIL) show an average reward far below 4.5 and thus have a sub-optimal policy for navigating the environment. The best performing one was the BC baseline method, however, it is important to note that because BC uses epochs while GAIL uses timesteps, it is not a direct fair comparison and will be discussed further in the discussion. The next part will compare algorithms and timestep groups, which are difficult to statistically evaluate differences between visually, namely BC + PPO 500 K timesteps versus BC + PPO 2 M timesteps and PPO 1 M timesteps versus BC + PPO at 1 M timesteps.

5.3 Algorithm Comparison

To statistically evaluate whether the minor differences in performance between timestep groups and algorithms are relevant, the Wilcoxon signed-rank test is used at a significance level of $\alpha = 0.05$. There are a few key outputs of the Wilcoxon test. The first one is the number of paired observations (in this case, the number of seed-matched runs, or $n=3$). The differences between the seed-matched mean rewards are converted into T -positive and T -negative ranks. The T value is the minimum of the sums of positive and negative signed ranks. Values of T near zero indicate that most differences favor one condition, while larger T values indicate no statistical difference. T can be converted into a standardized Z -score, and the p -value is then compared against $\alpha = 0.05$ to decide significance. To highlight the magnitude of the median shift, the variable effect size $r = \frac{Z}{\sqrt{n}}$ is calculated as well.

As seen before in Figure 14, BC+PPO already reached optimal performance at 500k timesteps, thus, it is relevant to compare it against itself at 2M timesteps, to show signs of statistical significance, shown in Figure 18. For the same reason, it is also relevant to compare PPO at 1M timesteps against BC+PPO at 1M timesteps, to show if IL+RL also leads to higher mean

reward on average compared to RL-only in Figure 19. Note that the y-axis starts at 4.0 to show an easier visualization of how the lines cross.

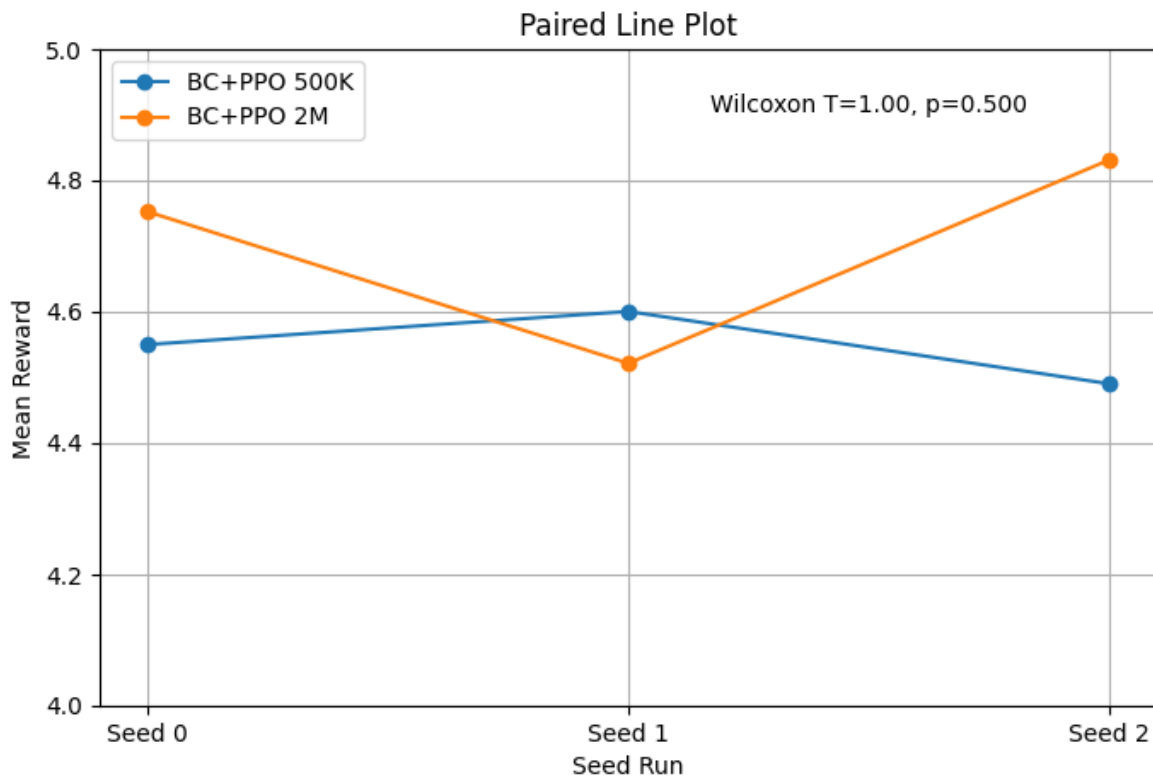


Figure 18: Paired Line Plot of BC+PPO at 500k timesteps vs BC+PPO at 2 M timesteps.

The Wilcoxon signed-rank test found no significant median difference between BC+PPO 500 K timesteps (median 4.5502) and BC+PPO 2 M timesteps (median = 4.7524), which gave $T = 1.00$, $p = 0.50$, and $r = -0.62$, indicating no statistically significant median difference. The $r = -0.622$ shows a magnitude of the differences but given the small sample size ($n = 3$) and the algorithms crossing seed-by-seed, there is no evidence of an advantage to either algorithm. This variability also shows why deep RL benchmarks typically use a much higher number of runs, as with so few samples, it is not possible to get a statistical significance either way.

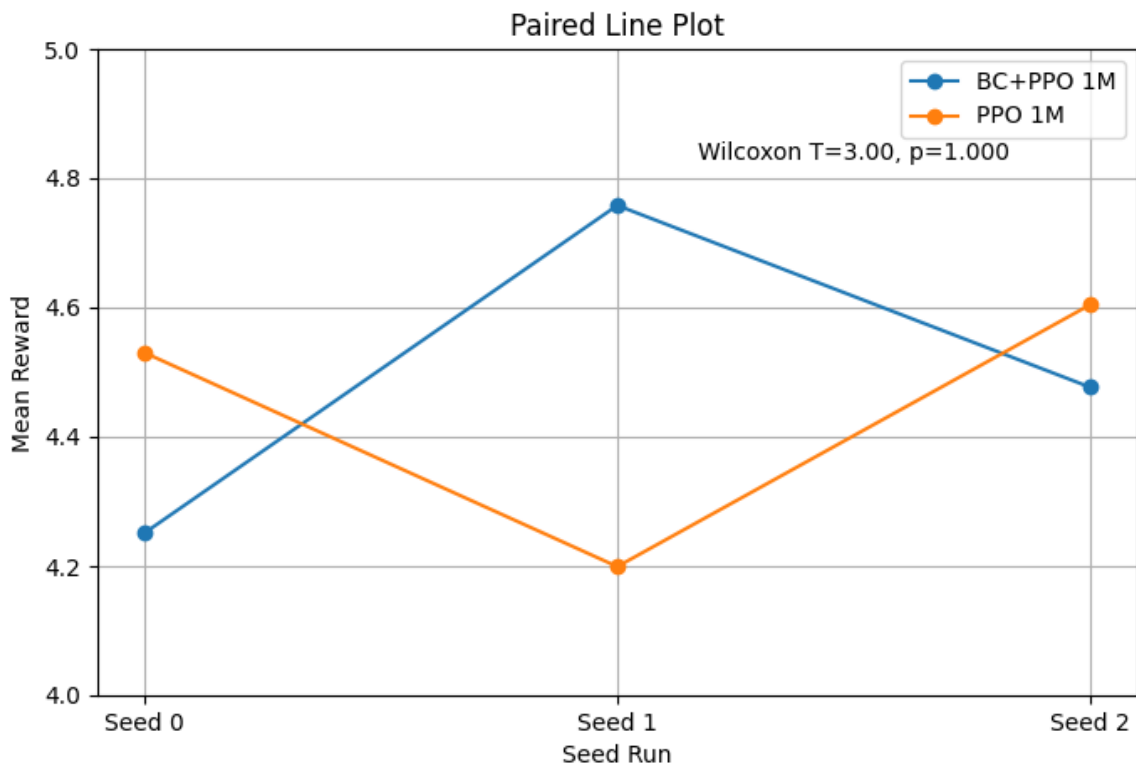


Figure 19: Paired Line Plot of BC+PPO at 1 mil vs PPO at 1 mil timesteps.

The Wilcoxon test returned $T = 3.0$ and $p = 1.00$, meaning the signed ranks sum in such a way that there is no evidence of any median shift between A and B. The medians themselves are almost identical ($A = 4.5300$ vs. $B = 4.4769$), and the effect size $r = 0.00$ confirms there is effectively no difference.

6 Discussion

6.1 Summary

In this experiment, pre-training with Behavioral Cloning (BC) and then fine-tuning with PPO halved the training duration needed, reaching the target performance threshold at 500 K timesteps versus 1 M timesteps for PPO alone, thus showing a 50% reduction in timesteps while only needing 30 demonstrations per level. This mirrors the 80 % reduction in RL training time achieved by Pfeiffer et al. (2018) in map-less navigation when pre-training on just 10 demonstrations before RL fine-tuning. Likewise, AlphaGo used supervised pre-training on 30 M human moves before RL to then beat professional human players in Go, demonstrating that even in vastly different domains, IL guidance can substantially reduce the amount of RL exploration needed. Thus, this study shows that the generalizability of the IL + RL method further extends to game-engine-based simulators.

Similarly, as Hester et al. (2018) showed, DQfD (IL+RL) outperforms pure RL and IL algorithms by leveraging small demonstration sets, and this study showed BC alone plateaued without combining with PPO, indicating that limited expert data leaves BC-only policies stuck far below pure RL’s performance. These findings align with the insights of Hester et al. (2018), that demonstration-based pre-training must be paired with sufficient RL fine-tuning to be successful.

The GAIL-only runs showed low mean reward and high variance, preventing a stable performance increase. Recent analysis by Luo et al. (2024) confirms that GAIL inherits Generative Adversarial Network-style training instability and thus proposes “C-GAIL” to enforce stability. This analysis explains why standard GAIL struggled to reach consistently increasing rewards without any algorithm adjustments. Thus, the findings in this study do not align with a different yet similar sequential training method by Wang et al. (2024), who found that BC + PPO + GAIL training method shows increased performance over PPO in their simulated crane operating task. The results of BC + GAIL + PPO in this study could in theory be due to “catastrophic interference” (Zhang et al., 2023), since each algorithm involves updating the policy network based on different objectives, GAIL’s inherent instability causes these successive updates to interfere with previously learned behavior and lead to performance degradation. However, despite having a similar sequential training pipeline, the order of algorithms does seem to have an importance, and these contrasting results also show different hyperparameters, neural network structures, and simulator variables cause the algorithms to perform widely differently, thus making this study important, as it also sheds light on how the algorithms generalize to more general simulators such as Godot Engine.

Silva and Calvo (2023) directly compared BC and GAIL in mobile-robot path planning, finding that BC had overall smoother trajectories and lower path error. However, going into more detail, they also showed that GAIL reached “satisfactory learning” and was able to drive along the same path as the expert at just 80 K timesteps. Compared to the results of GAIL in this environment, similar results cannot be drawn for GAIL up to 2 M timesteps and a similar MLP architecture. Some reasons can be argued for this; firstly, their environment featured simpler dynamics (flat terrain, static obstacles), while this environment featured dynamic obstacles (mobile enemies), hierarchical objectives (coin collection before goal access), and ramps and gaps. Secondly, despite the similar MLP architecture, the observation space, hyperparameters,

variables, and the overall environmental complexity that the agent must consider are all different from those being used in this study, further strengthening the importance of a generalized domain for RL research.

To summarize, IL and RL methods have shown their applicability in a wide range of fields, from mastering board games to guiding autonomous robots. For that reason, it is important to research how RL and IL algorithms such as PPO, BC, and GAIL perform in general-purpose simulators such as Godot, where currently the domain-specific simulators dominate the field and make it unclear whether the algorithms and sequentially trained methods will generalize towards new environments or simulators. Consequently, this study gave insights into how three common IL and RL algorithms perform in a robot navigation scenario, and how comparable the results were to previous work, showing that some methods are highly generalizable while others need a lot more effort to make them work. Specifically, the results show that applying the BC + PPO method in this scenario, is a highly generalizable method that has been proven to be valuable across different domains, while other methods utilizing GAIL, despite using a low-complexity neural network relative to other studies, potentially needs a lot more training timesteps or hyperparameter tuning to make it a practical.

6.2 General Discussion

6.2.1 Limitations

Environment-specific Limitation

The limitation of this study is that, most commonly, robot navigation research utilizes specifically customized frameworks and environments designed to closely mimic real-world robot settings, whereas this experiment is conducted in a controlled, game engine-based setup. Thus, even if this allows for easier reproducibility and experimentation, it does not capture all the complexities of a real-world warehouse navigation scenario. For example, a promising extension is provided by Zhou et al. (2024), who introduce an IL + RL method for autonomous driving path planning, where BC is first used to learn expert path-tracking, and then PPO fine-tunes the policy to nudge trajectories around static obstacles. Their experiments show that this approach significantly reduces collisions in simulation benchmarks.

Sequential Approach versus Combined Approach

Since this experiment was based on running each algorithm sequentially combined, a combined approach that runs algorithms in tandem should also be evaluated in future work, as seen in the Cycle-of-Learning framework (Goecks et al., 2020), which resulted in the algorithm outperforming modern approaches to RL in both sparse and dense reward environments, due to adopting a unified BC + RL objective throughout the fine-tuning process. If this method were to be adopted in this study, a sparse reward environment would be feasible to evaluate and consequently be more generalizable and applicable in the real world, as most commonly, robot navigation studies make use of very few reward signals, such as: “*did the robot reach the goal?*”

Limited Demonstration Data

Since the performance of imitation learning algorithms like BC and GAIL is highly dependent on the quality and number of expert demonstrations, limited demonstration data hinders the performance of the IL-only methods. Increasing demonstration data is very effective, as seen in Silver et al. (2016) with AlphaGo being able to beat professional players by having 30 M of human expert moves recorded before RL-training, however collecting enough demonstrations that would substantially improve performance of IL-only methods compared to IL+RL could potentially be an even more expensive approach, depending on how the demonstration data has to be gathered, or be a potential case of overfitting.

6.3 Ethics

For the ethical considerations, several factors were considered, namely the industry impact of AI robot advancement, safety, reliability, and transparency of the robot's decision-making process, as well as the societal implications and research ethics.

Safety and Reliability

Ensuring that an autonomous agent performs safely in all scenarios is important. In this study, the agent is trained in a controlled simulated environment, but when such systems are deployed in the real world (such as in warehouses), safeguards must be in place to prevent accidents and unintentional behavior. The reason for this is that training an RL algorithm produces a policy that behaves like a "black box". As an example, IL methods such as BC and GAIL rely on expert demonstrations. If these demonstrations favor a particular strategy that is not effective in all scenarios, the resulting agent can inherit these biases, and the behavior could lead to inefficiencies or safety issues. Thus, even if the hyperparameters used are known and all training data can be accessed, the neural network's internal logic cannot be inspected in a way that guarantees it will act predictably when faced with novel scenarios. Small shifts in the environment or inherent randomness that the neural network never saw during training can lead it to choosing actions that produce unintended behavior, and this is presently a major practical challenge of RL in real-world settings (Dulac-Arnold et al., 2021).

6.3.1 Societal Aspects

This research has important societal implications as it explores methods for training autonomous robots for warehouse navigation. Improved robot navigation can lead to more efficient warehouse operations by reducing routine human tasks and increasing overall productivity. However, it also raises questions about workforce displacement as automation becomes more prevalent. At the same time, there is also potential for new jobs that oversee autonomous robots, analyze their behavior, or continuously improve their designs. Whether the outcome of improved robot navigation has a positive or negative effect on society depends on how societies invest in human labor and redesign jobs to manage the transition, so that displaced workers are not left behind. Furthermore, optimized path planning in robotic navigation could lower energy consumption, contributing to ecological sustainability through lower carbon emissions in logistic warehouse operations.

From an economic standpoint, Graetz and Michaels (2018) analyze how modern industrial robots affect manufacturing output and labor markets, finding that each additional robot per 1,000 workers contributed 0.36 percentage points to annual labor productivity. Importantly, their findings show no statistically significant decline in overall employment, even though the

share of low-skilled workers fell, suggesting that productivity gains from automation can coexist with stable employment levels.

6.3.2 Research Ethics

This study was conducted with respect to ethical research standards. No data has been fabricated, falsified, or omitted, thus made transparent in Figure 20, Appendix A, where the raw data of the mean rewards are found. All code, analysis, and written content in this report are either original work or properly cited, and no unattributed text, figures, or data have been used. This research involves only a controlled simulated experiment and synthetic demonstration files; it does not involve any living subjects. No personal, sensitive, or identifying information has been collected, stored, or processed. Experiment configurations, hyperparameters, and environment versions are fully documented to enable reproducibility. Any third-party libraries used are referenced.

Furthermore, the choice of learning method might influence and bias the findings in this study. For example, BC can inherit blind spots in the demonstration data, such as consistently favoring certain paths, which can also lead to overfitting due to dataset biases, which damages its generalizability outside of the simulated environment, consequently misinforming real-world applications if deployed without awareness of this issue. Secondly, GAIL has a couple of issues that several papers try to address, such as Luo et al. (2024), who propose “C-GAIL” as an alternative algorithm to GAIL that aims to address its instabilities. Since the generator and discriminator in GAIL continually chase each other’s errors, this will often lead to oscillations rather than smooth convergence, which was made apparent in Figure 20. As the mean reward shows a high variance between runs, this reflects the oscillation seen in the training data. Thus, trying to evaluate GAIL’s performance in any scenario will not show meaningful or applicable results in the real world unless steps are taken to address the oscillation. In such a case, one would have to run GAIL for many more timesteps than done in this study, do extensive hyperparameter tuning, or implement “C-GAIL” instead, to increase the stability of this algorithm and potentially get meaningful results and support the research in this field. To address the research ethics of this issue, it was made apparent early on that GAIL has inherent instability that could either cause conflicts with previous research, or show results that lack significance, thus was only tested to evaluate whether a low complexity neural network (64 nodes with 2 layers) in a game-engine based setup would address its limitations and how it compares to previous works.

Lastly, comparing IL and RL algorithms directly raises ethical concerns about methodological fairness, as their operational constraints are fundamentally different. BC is inherently limited by the quantity of demonstration data, making this algorithm more useful as a pre-training step before running RL, unless a large amount of data is used to support a BC-only method. RL can theoretically improve indefinitely through environment interactions, at the cost of computational and environmental resources. Instead of directly comparing IL to RL, the two paradigms were grouped. PPO was compared against BC + PPO, while BC, GAIL, and BC + GAIL were compared against each other. Despite this, one can argue that comparing BC at a fixed number of epochs against GAIL timesteps, which can also run for an indefinite number of timesteps like PPO, is not a fair comparison. However, as they are both IL algorithms, they share the same limitations (demonstration data).

6.4 Threats to Validity

6.4.1 Conclusion Validity

Wohlin et al. (2012) describe threats to conclusion validity as factors that undermine the ability to correctly infer a relationship between the experimental treatment and its observed outcomes.

Due to hyperparameter settings heavily influencing the performance of RL and IL algorithms, and poor hyperparameter settings harming the validity of comparisons against baseline algorithms (Henderson et al., 2018), the hyperparameters in this experiment were kept to the recommended ones based on the documentation to strengthen the validity of the comparisons. Even with this in mind, there might be hyperparameter settings that will perform better, and the amount and quality of demonstration data will still heavily influence the performance of IL-only methods, thus unfairly favoring RL-only methods in dense reward spaces.

The performance metrics in this study (mean reward and sample efficiency) are repeatable measures and often used in RL studies, and the experiment implementation is standardized with the same hyperparameters across the methods, and evaluation protocols (200 episodes and fixed timestep groups) to minimize variability.

In a paper by Strens (2002), the author argues that running each policy on the same fixed random-number seeds controls stochastic variability and thus shows real performance differences. They also conclude that when the distribution of paired differences is irregular or deviates from normality, which is common with small sample sizes, the Wilcoxon test's rank-based approach is an appropriate choice. By seed-pairing the runs and using the Wilcoxon signed-rank test, the variance is reduced due to random seeds and avoids reliance on normality assumptions. However, a key threat to conclusion validity arises with very small n , as for a two-tailed Wilcoxon test at $\alpha=0.05$, no paired sample sizes below 6 have critical values that allow rejection of the null hypothesis unless all paired differences go in one direction (giving $T=0$ when $n=6$). In other words, with $n = 3$, even extreme median shifts cannot achieve $p<0.05$. This means it is necessary to increase the number of seed-matched runs to attain sufficient statistical significance for nonparametric paired comparisons.

6.4.2 External Validity

External validity concerns the extent to which the causal relationships observed in one study hold across different settings, specifically whether a treatment–outcome effect identified under specific experimental conditions can be generalized to industrial practice (Wohlin et al., 2012).

Since this navigation experiment was conducted in a “toy” environment, as Wohlin et al. (2012) describe, instead of a physical warehouse with more complex obstacles and sensors, this experiment may overestimate algorithm performances in real-world settings and thus not generalize well. To mitigate these threats, some insights can be generalized into similar navigation tasks and thus into the real world. First, the high sample efficiency and performance of BC + PPO are likely to extend to other simulated scenarios where an agent is pre-trained with IL and subsequently fine-tuned with RL, showing that even a small amount of demonstration data can effectively speed up learning in new environments. This is also

made clear in the study by Lu et al. (2023), who introduced a combined approach of IL and RL and decreased safety-critical failures by over 30% in autonomous driving benchmarks. Thus, even if this experiment does not directly generalize to real-world capabilities, the approach has been validated in more realistic settings.

6.4.3 Internal Validity

In this study, internal validity is mainly affected by the limited demonstration data of IL methods and the hyperparameter settings. As all algorithms share the same policy network (actor MLP), receiving the same input from the observation vectors, the algorithms operate over an equivalent observation space. This means that the only variable is the learning approach, as each algorithm updates the same policy network using different functions and gradient signals. Even with that in mind, it is difficult to scale the performance of IL algorithms without more demonstration data or more extensive hyperparameter tuning, thus resulting in a major difference in performance between RL-only and IL-only algorithms.

On another point, due to RL being inherently noisy between each training run, multiple runs per timestep group and algorithm must be executed. Even if in this study three runs per algorithm group were performed, the potential risk of algorithm performance being too varied still exists. To mitigate these internal threats, hyperparameters were kept to the recommended documentation settings, seeds were kept distinct for each run and paired across algorithms, and evaluation protocols were standardized to ensure that any remaining performance differences reflect the algorithms themselves.

6.4.4 Construct Validity

Construct validity in this study depends on whether the chosen metrics and experimental design in this study reflect the theoretical constructs of “best performance” in terms of navigation capability. One important design threat is inadequate preoperational explication, that is, if there is no exact definition of what better performance means, for example, faster completion or safer trajectories, then these measurements of mean reward and sample efficiency may not fully capture the intended concept. Thus, in this study, the point was made early that better performance means getting high rewards consistently, translating to being able to clear all levels in the environment efficiently.

The chosen performance metrics, namely mean reward and sample efficiency could however not be the concept of performance in autonomous warehouse navigation. While these metrics are commonly used in RL research, they do not capture all dimensions of performance relevant to practical applications, such as safety, reliability, or energy efficiency.

Next, mono-operation bias is avoided because six different algorithms are evaluated, however, only a single policy architecture is used (MLP), which introduces mono-operation bias on that dimension. Similarly, mono-method bias occurs when the experiment exclusively relies on scalar reward signals without complementary measures such as path collision frequency, thus, the risk in this study lies in that it only measures one part of navigation quality. Confounding construct levels is also possible, as this study treats sample efficiency as binary (efficient versus inefficient) and ignores how it varies continuously with demonstration size or in a sparse reward space.

7 Conclusion

In this study, three algorithms and three combined approaches were evaluated and compared: PPO, BC, and GAIL for autonomous warehouse navigation. The findings indicate that pre-training with BC and subsequently fine-tuning with PPO reached the best-performing policy with the highest sample efficiency. The limited amount of demonstration data constrained the scalability of BC and GAIL; however, despite GAIL training for 2 M timesteps, BC still showed better performance compared to GAIL. GAIL’s instability persisted even with a low-complexity neural network, which suggests algorithmic and not environment limitations, consequently showing that GAIL’s results in Silva and Calvo (2023) could not be replicated with a similar environment and neural network complexity. Lastly, pre-training with BC and fine-tuning with GAIL showed worse performance than training the IL baselines by themselves, aligning with the insights of Jena et al. (2021) and conflicting with Ho and Ermon (2016).

Research Question 1 was answered by comparing the results of BC (Figure 11) and GAIL (Figure 12). GAIL performed worse than the BC baseline, even at 2 M of GAIL timesteps, showing that despite having a low complexity neural network, GAIL most likely needs much more training timesteps than used in this study.

Research Question 2 was answered by comparing the results of Figures 10 and 14. PPO needed at least 1 M timesteps to reach the target performance threshold of 4.5 mean reward, while BC + PPO only needed 500 K timesteps, showing a 100% sample efficiency improvement over PPO.

Research Question 3 was answered by comparing the results of Figures 10, 12, and 15. BC + GAIL + PPO does not show improved performance over PPO by itself, thus not aligning with the results of Wang et al. (2024) which evaluated a similar method (BC + PPO + GAIL against PPO).

7.1 Research Contribution

Firstly, the study demonstrated that combining BC + PPO in a game-engine-based simulation significantly improves sample efficiency, achieving target performance with 50% fewer timesteps compared to PPO, though no significant difference in the mean cumulative reward was found. This validates the generalizability of IL + RL methods beyond domain-specific simulators and shows that, despite using only 30 demonstrations per level, even limited demonstration data can dramatically reduce RL training costs, which has practical implications for real-world applications where collecting demonstration data is difficult or expensive to obtain. Secondly, by evaluating standalone IL methods, the study shows that GAIL’s learning inefficiency (prior work needing at least 5-10 M timesteps) persists even with a low-complexity neural network, suggesting algorithmic limitations. In addition, GAIL’s results contrasting with an argument made by Ho and Ermon (2016), being that pretraining with BC and fine-tuning with GAIL can significantly improve learning speed for GAIL, which was not the case for BC + GAIL results in this study, instead confirming the results of Jena et al. (2021). Thirdly, the study reveals sequential BC + GAIL + PPO training degrades performance compared to PPO, due to GAIL’s poor performance at 2 M timesteps and conflicting optimization objectives. As the papers referenced in this study evaluate GAIL with significantly more timesteps and more complex neural networks, it is thus important to

determine whether a less complex neural network can aid in the learning speed of GAIL so that it can become a viable pre-training algorithm like BC, but with potentially improved generalization.

7.2 Future Work

If IL-only methods were to be improved further through more demonstration data, this could lead to RL-training being redundant and thus save training and energy costs, however, this is determined by the complexity of the environment the robot must navigate through. One study by Ramrakhya et al. (2023) found that as the size of the demonstration data increases, the improvements from RL fine-tuning are smaller, showing that most of the performance gains from IL+RL policy can be achieved with a low amount of demonstration data. Thus, it would be interesting to see if demonstration data can be scaled to a point where RL fine-tuning is not needed at all.

On another point, applying a framework like Zhou et al. (2024) to the warehouse environment and introducing more variables, such as collision rate, would improve policy generalization and, consequently, would be able to evaluate the safety of autonomous vehicles, thus being more applicable to the real world. The limited generalization of the current environment could also be helped by introducing a sparse reward environment and leveraging Goecks et al. (2020) work to combine algorithms by having a unified IL+RL objective throughout training, as RL-only training in sparse reward environments is too sample-inefficient. By having a sparse reward environment, the generalizability of the policies would potentially be improved due to not being constrained by the specific reward space, thus supporting exploration.

8 References

- Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587), pp.484-489.
- Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I. and Dulac-Arnold, G., 2018, April. Deep q-learning from demonstrations. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 32, No. 1).
- Gleave, A., Taufeeque, M., Rocamonde, J., Jenner, E., Wang, S.H., Toyer, S., Ernestus, M., Belrose, N., Emmons, S. and Russell, S., 2022. imitation: Clean imitation learning implementations. *arXiv preprint arXiv:2211.11972*.
- Pomerleau, D.A., 1991. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1), pp.88-97.
- Ho, J. and Ermon, S., 2016. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29.
- Beeching, E., Debangoye, J., Simonin, O. and Wolf, C., 2021. Godot reinforcement learning agents. *arXiv preprint arXiv:2112.03636*.
- Silva, K.L.R.M. and Calvo, R., 2023. Mobile Robot Navigation Strategies Through Behavioral Cloning and Generative Adversarial Imitation Learning. In *ICEIS (1)* (pp. 517-524).
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B. and Wesslén, A., 2012. *Experimentation in software engineering* (Vol. 236). Berlin: Springer.
- Luo, T., Pearce, T., Chen, H., Chen, J. and Zhu, J., 2024. C-gail: Stabilizing generative adversarial imitation learning with control theory. *arXiv preprint arXiv:2402.16349*.
- Pfeiffer, M., Shukla, S., Turchetta, M., Cadena, C., Krause, A., Siegwart, R. and Nieto, J., 2018. Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations. *IEEE Robotics and Automation Letters*, 3(4), pp.4423-4430.
- Goecks, V.G., Gremillion, G.M., Lawhern, V.J., Valasek, J. and Waytowich, N.R., 2019. Integrating behavior cloning and reinforcement learning for improved performance in dense and sparse reward environments. *arXiv preprint arXiv:1910.04281*.
- Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. and Meger, D., 2018, April. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 32, No. 1).
- Demšar, J., 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan), pp.1-30.
- Zhou, M., Wang, B., Tan, T. and Sun, X., 2024. Developing Path Planning with Behavioral Cloning and Proximal Policy Optimization for Path-Tracking and Static Obstacle Nudging. *arXiv preprint arXiv:2409.05289*.
- Jena, R., Liu, C. and Sycara, K., 2021, October. Augmenting gail with bc for sample efficient imitation learning. In *Conference on Robot Learning* (pp. 80-90). PMLR.

- Zhang, T., Wang, X., Liang, B. and Yuan, B., 2022. Catastrophic interference in reinforcement learning: A solution based on context division and knowledge distillation. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12), pp.9925-9939.
- Strens, M.J. and Moore, A.W., 2002. Policy search using paired comparisons. *Journal of Machine Learning Research*, 3(Dec), pp.921-950.
- Hussein, A., Gaber, M.M., Elyan, E. and Jayne, C., 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2), pp.1-35.
- Lu, Y., Fu, J., Tucker, G., Pan, X., Bronstein, E., Roelofs, R., Sapp, B., White, B., Faust, A., Whiteson, S. and Anguelov, D., 2023, October. Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 7553-7560). IEEE.
- Dulac-Arnold, G., Levine, N., Mankowitz, D.J., Li, J., Paduraru, C., Gowal, S. and Hester, T., 2021. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9), pp.2419-2468.
- Graetz, G. and Michaels, G., 2018. Robots at work. *Review of economics and statistics*, 100(5), pp.753-768.
- Ramrakhya, R., Batra, D., Wijmans, E. and Das, A., 2023. Pirlnav: Pretraining with imitation and rl finetuning for objectnav. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 17896-17906).
- Linietsky, J., Manzur, A. & Vershelde, R. 2016., *Godot* (Version: 4.4) [Computer Software]. GitHub. Available at: <https://github.com/godotengine/godot> (Accessed: 23 April 2025).
- Wang, Z., Huang, C., Yao, B. and Li, X., 2024. Integrated reinforcement and imitation learning for tower crane lift path planning. *Automation in Construction*, 165, p.105568.
- Whiteson, S., 2009. Generalized domains for empirical evaluations in reinforcement learning.

Appendix A – Raw data of mean rewards

| Algorithm | Trainir | Run 1 (Seed 0) | Run 2 (Seed 1) | Run 3 (Seed 2) |
|--------------------------|------------|----------------|----------------|----------------|
| BC | 50 Epochs | -0.2116 | -0.0386 | 0.1784 |
| BC | 100 Epochs | 0.5320 | 1.2964 | 0.1031 |
| BC | 150 Epochs | 2.0047 | 0.7612 | 1.5527 |
| BC | 200 Epochs | 1.4086 | 2.0333 | 2.2131 |
| GAIL | 500K | -0.1461 | -0.8570 | -0.4133 |
| GAIL | 1M | 0.2511 | -0.5153 | -0.7555 |
| GAIL | 1.5M | 0.5618 | 0.5586 | -0.0411 |
| GAIL | 2M | 0.6968 | 0.7246 | 1.3318 |
| PPO | 500K | 3.6444 | -0.5616 | -0.7415 |
| PPO | 1M | 4.5300 | 4.1992 | 4.6041 |
| PPO | 1.5M | 4.6113 | 4.1389 | 4.5540 |
| PPO | 2M | 4.5282 | 2.3159 | 4.7063 |
| BC+GAIL (150 epochs) | 500K | -0.9363 | -0.7944 | -0.9606 |
| BC+GAIL (150 epochs) | 1M | -0.8197 | -0.8417 | -0.3035 |
| BC+GAIL (150 epochs) | 1.5M | 2.4553 | 0.9501 | 1.5307 |
| BC+GAIL (150 epochs) | 2M | 1.4068 | -0.6564 | 0.5010 |
| BC+PPO (150 epochs) | 500K | 4.6006 | 4.6006 | 4.4905 |
| BC+PPO (150 epochs) | 1M | 4.2515 | 4.7580 | 4.4769 |
| BC+PPO (150 epochs) | 1.5M | 4.6732 | 4.4345 | 4.7322 |
| BC+PPO (150 epochs) | 2M | 4.7524 | 4.5216 | 4.8312 |
| BC+GAIL+PPO (150 epochs) | 500K | -0.5939 | -0.3217 | -0.4475 |
| BC+GAIL+PPO (150 epochs) | 1M | 1.1118 | 0.9763 | -0.5290 |
| BC+GAIL+PPO (150 epochs) | 1.5M | 1.0003 | 1.6661 | 0.7607 |
| BC+GAIL+PPO (150 epochs) | 2M | 2.0454 | -0.1524 | 2.2189 |

Figure 20: Raw data of mean rewards across all timestep groups and algorithms used in this study.

Appendix B – Godot RL Agents MIT License

MIT License

Copyright (c) 2022 Edward Beeching

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.