Investigation into the Effect of Replay Buffer Diversity on Generalizability MSc Thesis

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Felix Kaubek
born in Vienna, Austria

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Investigation into the Effect of Replay Buffer Diversity on Generalizability MSc Thesis

Author: Felix Kaubek
Student id:
Email:

Abstract

In reinforcement learning, the ability to generalize to unseen situations is pivotal to an agent's success. In this thesis, two novel methods that aim to enhance the generalizability of an agent will be introduced. Both of the methods rely on the idea that the diversity of a replay buffer increases an agent's ability to generalize. The first utilizes the agent's exploration strategies to reach interesting states. The second aims to reach further using an additional goal-conditioned agent. Both methods demonstrate improved adaptability without relying on domain-specific knowledge and show promising results. The code is available through GitHub [1].

Thesis Committee:

Chair:	Wendelin Böhmer, TU Delft
University supervisor:	Wendelin Böhmer, Sequential Decision Making, TU Delft
Committee Member:	David Tax, Pattern Recognition & Bioinformatics, TU Delft

[1] https://github.com/felixk29/thesis

Preface

During my master's, many people helped make it a memorable and enjoyable period of my life. They helped me throughout my studies, especially when I wrote my thesis. I have to thank the friends I made along the way, especially Monica, Konrad, and Albin. I wish them all the best in their future endeavors and will support them whenever they need me. I would also like to thank my supervising professor, Dr. Wendelin Böhmer, and the PhD candidate, Max Weltevrede, for not only helping me find a suitable topic but also for their support throughout this thesis. Last but by no means least, I would like to thank my family for their continuous support.

Felix Kaubek
Delft, the Netherlands
July 30, 2024

Contents

List of Figures

Chapter 1

Introduction

With the increased focus on machine learning and artificial intelligence in general, Reinforcement Learning (RL) has also gained interest. Unlike classical machine learning, RL can neither be described as supervised nor unsupervised learning. Instead, it learns through interactions with a given environment. While it has been a research subject for a long time and advancements have been made, the potential it bears is much greater [33] [23]. If fully successful, RL could be applied to many real-world use cases, such as self-driving cars, robotic applications, as well as sustainable management and energy control. However, in order to achieve that potential, many challenges need to be overcome, such as poor sample efficiency and lack of scalability. Among them, the ability to generalize is of high priority.

1.1 Motivation

Generalization allows an RL agent to apply learned knowledge to different contexts, which is crucial for real-world applications like autonomous vehicles or control systems. This is even more difficult in multi-context learning due to the increased variability of contexts. For example, a vacuum robot should still navigate rooms effectively despite the furniture being moved, or self-driving cars should be able to handle unmarked intersections and unexpected road works [19].
Just as leaving their comfort zone helps humans adapt more easily to unfamiliar situations, this principle should also apply to RL agents. This could be achieved by increasing the number of starting positions from which an agent begins an episode. This should further lead to a higher diversity in the agent's replay buffer, which are the experiences from which it learns, as that has been shown to lead to better generalization[37]. Therefore, the aim of this thesis is to find general and scaleable approaches that increase the diversity of the agent's replay buffer by first bringing the agent into interesting positions.
The broadly applicable methods introduced aim to improve the generalization capabilities of RL agents. They focus on strategies that do not require specialized domain knowledge and are not limited to niche applications. We utilize standard reinforcement learning algorithms and environments with implicit goals and incorporate insights from goal-conditioned learning research to enhance agent adaptability and performance.

1.2 Research Questions

Q1: Does increasing the diversity of starting positions within the reachable set increase an agent's ability to generalize to unreachable states?

We hypothesize that agents' generalization capabilities, specifically in multi-task learning, improve with a higher diversity in starting positions within the reachable states. This hypothesis stems from the intuition that training the agent to navigate from non-starting positions, which are, however, still within the reachable set, towards the goal will achieve a broader and more general understanding of the environment, which would lead to a better ability to generalize as well as higher robustness.

Q2: Does the diversity of the replay buffer impact the agent's ability to generalize?

Additionally, we hypothesize that the composition of the experience replay buffer influences the agent's generalizability, particularly that the diversity of said buffer during training correlates with the agent's ability to generalize in evaluation to unseen contexts.

Q3: What are possible ways to achieve an increase in diversity of starting positions without altering the environment?

Further, we want to investigate how to increase the diversity in starting positions without altering the environment. As we postulate, this would lead to a higher generalizability. These methods should enable implementation without domain knowledge and, therefore, lead to a higher applicability.
Furthermore, we aim to provide applicable methods to achieve such a replay buffer. Through comparative analyses and other analytical experiments, we strive to empirically investigate the impact an increase in the diversity of starting positions has on agents' replay buffer diversity and further on the agent's ability to generalize. We further propose two different methods to achieve a diverse replay buffer. By investigating these fundamental parts of reinforcement learning and their connection, this thesis seeks to contribute a deeper understanding of how agents can effectively generalize across contexts and adapt to unseen scenarios in real-world applications while also aiming to make these possible improvements accessible.

1.3 Thesis Structure

The remainder of this thesis is organized as follows: The next chapter (Chapter 2) provides the necessary background information, followed by a review of related works (Chapter 3). Subsequent chapters describe the methodologies (Chapter 4) developed and experiments (Chapter 5) evaluating those. After presenting the results (Chapter 6), the final chapter (Chapter 7) summarizes the findings, discusses the benefits and limitations of the approaches introduced, and recommends potential future research. The appendix (Chapter A) contains further experiments and the specific hyperparameter used for the main comparisons.

Chapter 2

Background

This chapter will cover the necessary background to understand this work. This introduction explains most core concepts, whereas the remainder of this chapter further elaborates on the details.
Reinforcement Learning is a powerful tool for training agents to make sequential decisions in complex environments. At its core lies the Markov Decision Process, a 6-tuple-based foundational framework used for theoretical reasoning within the field. Generalization in RL refers to an agent's ability to apply learned knowledge to unseen situations, which is crucial for adapting to diverse contexts efficiently. Techniques like Deep Q-Networks utilize replay buffers and neural networks to learn policies from even high-dimensional raw data. Exploration strategies, including ε-greedy and utilizing intrinsic reward, play pivotal roles in balancing the trade-off between exploiting known strategies and exploring novel states. Further, goal-conditioned agents coupled with Hindsight Experience Replay enhance adaptability, given an environment that gives explicit goals, by enabling agents to learn from more than just the intended goal, promoting robustness and making learning more efficient.

2.1 Markov Decision Process

As a starting point, we can assume a standard setup for reinforcement learning. We denote our Markov decision process (MDP) as a 6-tuple $M = (S, A, T, R, p_0, \gamma)$, where S and A are the state and action space respectively, T represents the transition probability function, R is a reward function, p_0 is the initial state distribution and $\gamma \in (0, 1]$ is a discount factor [30]. To model multi-task reinforcement learning, we will utilize a contextual MDP (CMDP, [13]) where $S = S' \times C$, for which C is a context space and S' is the underlying state space. The context $c \in C$ cannot be changed during an episode and can also be considered the task of the agent. A context can be understood as a variable of the environment that can differ but cannot be changed by the agent. An example would be furniture placement in a room for the vacuuming robot. The robot cannot change the placement, but it can differ between various deployments.

Environments are practical implementations of the theoretical MDP. They simulate the MDP with which an agent interacts and from which it learns [34]. In most cases, the environment provides an observation and action space in which the agent interacts. The observation space can entail anything from a simple binary vector (BitFlip [1]), the position and velocity of the agent (MountainCar [25]), to raw image data. The action space consists of any action the agent can access and can take many forms, such as discrete, continuous, or even a combination of both. Furthermore, in multi-task learning, oftentimes, there are so-called "dead" actions, which means they are available to the agent but do not have any effect. As they are not mapped to an interaction with the environment, optimal agents should learn not to use those dead actions.
Environments can be either stochastic, meaning that either the reward or transition function involves some kind of randomness, or deterministic, meaning that they do not. Further, they can be fully observable, meaning that the whole state is visible to the agent, or only partially observable, meaning that some parts of the state are unknown to the agent. The environment used for the experiments is both deterministic and fully observable.

2.2 Generalization

Generalization in reinforcement learning refers to the ability of an agent to apply its knowledge to unseen situations. These unseen situations could mean that the goal state is in a different position than during training or that the context c differs from any context it has encountered before.
A key challenge of RL is the trade-off between exploration and exploitation, as higher exploration leads to a better understanding of the environment but can decrease performance. On the other hand, too little exploration can lead to overfitting. During training, an agent should learn which actions lead to the goal and a policy that is not only the given but any possible goal. Otherwise, overfitting can occur, characterized by a high success rate on training data but a very low success rate on test data. Conversely, if exploration is too high, the learning efficiency of an agent is severely reduced [17] [34].
While many general applicable techniques to counteract overfitting exist, such as dropout layers in neural networks [20] or batch normalization, the choice of architecture and training algorithm also has a strong influence on the generalizability of an agent [6].

Formally, let π_θ be a policy parameterized by θ, and let $M_{train} = (S_{train}, A, T_{train}, R_{train}, p_{0,train}, \gamma)$ and $M_{test} = (S_{test}, A, T_{test}, R_{test}, p_{0,test}, \gamma)$ be the MDPs representing the training and testing environments, respectively. As this thesis focuses solely on zero-shot generalization, the agent is only trained in the training environment and uses the test environment only during evaluation. The generalization performance of π_θ can be defined as the expected return in the testing environment, further denoted as $J_{test}(\pi_\theta)$.
The generalization gap can then be measured by comparing the expected return in the training environment, $J_{train}(\pi_\theta)$, to the expected return in the testing environment, $J_{test}(\pi_\theta)$:

$$\Delta J(\pi_\theta) = J_{train}(\pi_\theta) - J_{test}(\pi_\theta)$$

A smaller value of $\Delta J(\pi_\theta)$ usually indicates higher robustness, as the policy performs similarly in both the training and testing environments.

2.3 Reachability

In multi-task reinforcement learning, reachability refers to the set of states that can be encountered through some sequence of actions [37]. Every state is either reachable or unreachable, given another state. This is important for understanding and improving generalization. A state s_r is considered reachable if a policy exists that has a non-zero probability of encountering s_r when deployed in the training MDP. S^0_{train} is the set of starting positions of the training set. The set of reachable states, $S_r(M|S^0_{train})$, includes all states that can be reached from the initial training states S^0_{train} in a given MDP M. This leads to two properties. If a state s^* is reachable from any state $s \in S_r(M|S^0_{train})$ then s^* itself must also be in $S_r(M|S^0_{train})$. Further, it can be concluded that interaction with the $M|S^0_{train}$ cannot lead to states outside the reachable set $S_r(M|S^0_{train})$.
For Zero-Shot Generalization, this means that if the testing states S^0_{test} are part of the reachable states $S_r(M|S^0_{train})$, an optimal policy on the training environment will also perform well during testing, even when the agents generalization ability to other context is low. Therefore, to properly evaluate an agent's ability to generalize to different unreachable contexts, it is important to include an additional test set, which is unreachable from the training set.

2.4 On-Policy or Off-Policy

Reinforcement learning algorithms are categorized as either on or off-policy based on how exactly they learn from the environment. On-policy algorithms learn the policy directly, meaning they aim to learn exactly what to do given a certain state. On the other hand, off-policy algorithms learn the value an optimal policy would evaluate a state as, independently from the agent's action given a state. This allows the agent to learn from data that is collected through a different policy. Q-learning and all algorithms that are based on it are classic examples of off-policy algorithms.

2.5 Replay Buffer

In reinforcement learning, the concept of a replay buffer, also known as experience replay, has emerged as a fundamental tool for off-policy algorithms [22]. Essentially, it can be viewed as the agent's memory from which it learns. The agent's trajectories within an environment are saved as tuples of the form $(\text{state}, \text{action}, \text{reward}, \text{next state})$ and are sampled during the learning process.
The standard method for filling the buffer is first-in-first-out, but other variations exist, such as prioritizing more significant experiences [14]. Additionally, the sampling process, which is usually uniformly random, can employ more sophisticated methods [32]. The primary

aim of a replay buffer is to stabilize learning and enhance sample efficiency. Most off-policy algorithms rely on experience replay to learn efficiently.

Formally, let $\mathcal{D}$ denote the replay buffer containing tuples of the form (s_t, a_t, r_t, s_{t+1}). At each time step t, the agent's experience gained from taking that specific action at that specific state is stored in $\mathcal{D}$. Until a predetermined amount of tuples has been stored, then the oldest experiences are overwritten by new ones.
During training, a batch of experiences is sampled from $\mathcal{D}$ to update the agent's policy. The replay buffer helps to break temporal correlations, ensuring that the learning process is more stable and efficient.

2.6 Deep Q-Networks

Deep Q-Networks (DQN) [24] represent a significant advancement in reinforcement learning, capable of handling environments with high-dimensional raw data. DQN is a deep learning-based reinforcement learning algorithm that uses neural networks to learn the optimal action-value function [23]. By integrating deep learning techniques with the Q-learning algorithm [15], DQN enables efficient learning of optimal policies from raw data, such as images or other types of observational data.
The two main innovations in DQN are the replay buffer, which was previously explained, and target networks. Target networks are added to the online network. While the online network is updated at every iteration, the target network is a delayed copy. This addresses the issue of moving targets in Q-learning by periodically updating a separate network with the parameters of the main Q-network. This process helps stabilize training, preventing instabilities and especially divergence.

Formally, in DQN, the Q-value function is approximated by a neural network $Q_\theta(s,a)$, where θ are the parameters of the network. The target value for the Q-learning update is gained through

$$y_t = r_t + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a')$$

where θ^- are the parameters of the target network. The network parameters θ are then updated by minimizing the loss:

$$L(\theta) = E_{(s,a,r,s')\sim\mathcal{D}}\left[(y_t - Q_\theta(s,a))^2\right]$$

An improvement on the standard DQN, known as Double Deep Q-Network (Double DQN), aims to fix the inherent overestimation bias in Q-learning [36]. This is achieved by decoupling action selection and evaluation between the two different networks, the online part selects the action, while the target network evaluates it.

In Double DQN, the target value is modified to reduce possible overestimation bias:

$$y_t = r_t + \gamma Q_{\theta^-}(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$$

2.7 Exploration

Exploration refers to the strategy by which an agent interacts with the environment to discover new states and actions, thereby improving its understanding and capabilities [34]. Effective exploration is crucial in reinforcement learning, as it enables the agent to understand the environment and find the optimal policy. A common exploration strategy is ε-greedy, where the agent chooses the action dictated by the current policy with a probability of $(1-\varepsilon)$ and otherwise samples a random action from the action space [21].
As ε-greedy represents a shallow exploration method, meaning that it is unguided and does not take prior experiences into account, other techniques, such as intrinsic motivation, have been developed.
In these methods, the agent receives an additional intrinsic reward r_{int} that encourages exploration. The total reward r_t at time step t is then given by $r_t = r_{ext} + \beta r_{int}$, where r_{ext} is the extrinsic reward from the environment and β is a scaling factor that has to be manually set. This encourages the agent to explore through internal curiosity or novelty detection, improving exploration, especially in high-dimensional or hard-exploration environments, such as the MountainCar problem [4].

2.8 Random Network Distillation

Random Network Distillation (RND) [5] is an exploration strategy in reinforcement learning that aims to enhance exploration. RND operates by initializing two neural networks with the same architecture: the predictor and the target. The target network is fixed and does not change, while the predictor network is trained to match the output of the target network. The distance between the outputs of the two networks approximates the state's novelty, with higher values indicating states that have been visited less frequently [29]. In many implementations of this algorithm, however, not only the state is included in the reward evaluation, but also the action. This approach emulates curiosity and is often used as an intrinsic reward-based exploration method [21].
In RND, let $f_{target}(s,a)$ be the target network and $f_{predictor}(s,a)$ be the predictor network. The intrinsic reward r_{int} for a state s is given by the prediction error:

$$r_{int}(s) = \|f_{target}(s) - f_{predictor}(s)\|^2$$

This intrinsic reward encourages the agent to explore states with high prediction errors, thereby enhancing the agent's exploration capabilities. The predictor network is trained to minimize this prediction error on the states the agent has encountered. The data can be

sampled in various ways, from training on the fresh data the agent encounters, or the data can be the same batch the agent trains on.

2.9 Goal-Conditioned Agents & Hindsight Experience Replay

In many environments the objective never changes or the objective is implicitly given through the observation, for example, the target square is marked. A third case is environments that explicitly give the objective ("goal") as an additional observation. Goal-conditioned agents are a class of reinforcement learning techniques to enhance generalization in goal-conditioned environments [31]. These agents learn a policy that is conditioned on a goal, allowing them to understand similarities between different goals and share parameters across different goals. This facilitates faster learning and better adaptation to new contexts and goals, as knowledge gained from previous goals can be readily applied.

Formally, let g represent a goal, and let the policy $\pi_\theta(s,g)$ and value function $Q_\theta(s,a,g)$ be parameterized by the goal. The goal-conditioned value function approximates the expected return given a state s, action a, and goal g:

$$Q_\theta(s,a,g) = E\left[\sum_{t=0}^{\infty} \gamma r_t \mid s_0 = s, a_0 = a, g, \pi\right]$$

Hindsight Experience Replay (HER) is a technique designed to improve generalization and facilitate faster learning in goal-conditioned reinforcement learning, particularly in environments with sparse and binary rewards [1]. HER enhances learning by not only using actual outcomes of actions but also by learning from virtual goals based on "hindsight," similar to humans who think about what they could have done differently. While goal-conditioned agents enable the learning of a general value function across different goals, HER allows the agent to learn from a broader range of experiences, enhancing efficiency and effectiveness, especially in sparse reward scenarios.

Formally, in HER, for each trajectory $T = (s_{0:t}, g, a_{0:t}, r_{0:t}, s_{1:t})$, a new goal g^* is created using a state sampled from the trajectory. T is then copied, g is replaced with g^* and the rewards are recalculated. The new trajectory $T^* = (s_{0:t}, g^*, a_{0:t}, r^*_{0:t}, s_{1:t})$ is then added to the replay buffer. This can be done multiple times per trajectory with different goals. This approach allows the agent to learn from one set of experiences in multiple ways, which is particularly beneficial in sparse environments. Specifically, it increases the agent's success rate during training and, because of that, facilitates faster learning.

Chapter 3

Related Work

Reinforcement Learning has gained attention due to its success in solving complex decision-making problems, such as Atari games. However, achieving robust performance across diverse environments and contexts remains a challenge. Zero-shot generalization is a central focus of current research, with extensive efforts being made in various facets of this problem. This chapter examines previous work that has inspired this thesis and attempts to tackle similar challenges.

3.1 Exploration Strategies

This thesis aims to increase an agent's ability to generalize by increasing the diversity of the replay buffer, which can also seen as a goal of good exploration. While some of the following papers were used as inspiration, they are not directly comparable to our proposed methods, as they alter the agent directly.

3.1.1 First Return, Then Explore

"First Return, Then Explore" [9] by Ecoffet et al. introduces GoExplore, a family of novel exploration methods in RL. The authors identify "derailment," the inability to reach previously visited states, and "detachment," the inability to return to an interesting state to explore from, as key problems in deep exploration.
Their algorithm, which significantly outperformed other state-of-the-art results in various Atari benchmarks [3], works in two phases. The first phase, the "Exploration Phase," involves the agent starting each episode at the most promising spot encountered in previous episodes and exploring its surroundings. This is repeated as necessary to explore the state-action space thoroughly. In the second, optional "Robustification Phase," the concatenated trajectory is utilized as a demonstration, and the agent is trained on the resulting trajectory to become more robust. This achieved previously unseen scores in Atari games like "Montezuma's Revenge." While the paper does not focus on generalization, we took inspiration from the exploration phase to increase the replay buffer diversity of the agent.

3.1.2 BeBold: Exploration Beyond the Boundary of Explored Regions

The BeBold algorithm, introduced by Zhan et al. [39], presents an innovative approach to enhance exploration in RL. Traditional methods like count-based and state-difference approaches often struggle with issues such as short-sightedness, which means only considering the near future, and detachment. The BeBold algorithm addresses these challenges by combining count-based and state-difference into one criterion to encourage agents to explore beyond known boundaries. Unlike GoExplore, the algorithm does not rely on human-designed down sampling of images, and it has a lower dependence on hyperparameter tuning. This algorithm significantly improves performance on environments like MiniGrid and NetHack without requiring extensive tuning or meta-learning techniques such as curriculum learning.

3.1.3 Efficient Self-Supervised Data Collection for Offline Robot Learning

Endrawis et al. [10] introduce a data collection method for robotic RL applications. The agent fully explores an environment in simulation to collect exhaustive data for an offline robot. The algorithm first explores, then selects a target from its replay buffer and tries to reach it. The target selection is based on an RND module, which determines the state with the highest attributed uncertainty as a target. The aim of this paper is to exhaust the state-action pairs of an environment and use those to gather a complete training set for an offline-trained agent. While the paper is mostly aimed at offline RL, the way the exploration works is similar to our proposed solution and is therefore included. However, this algorithm does focus purely on exploration and not on exploitation and is therefore not a feasible solution for online learning.

3.2 Generalization Methods for RL

As the primary research objective is to increase the Zero-Shot Generalization performance of reinforcement learning agents, we will now introduce some other works in this field to put ours into context properly. These papers showcase that exploration and generalization are interlinked and that diverse experiences can improve generalization.

3.2.1 The Role of Diverse Replay for Generalisation in Reinforcement Learning

”The Role of Diverse Replay for Generalisation in Reinforcement Learning” by Weltevrede et al. [37] explores the importance of diverse training data on an agent’s ability to generalize to unseen contexts. The concept of reachable states is utilized to demonstrate that a diverse buffer increases generalizability, especially in Zero-Shot situations. In an experiment, two different ways of sampling from the buffer are compared to showcase the importance of diverse state-action pairs in the training data. While this paper lays the groundwork for this research, it mainly investigates the importance of the replay buffer on Zero-Shot generalization, providing the simple solution of increasing the replay buffer size to fit every transition

experienced into it. This approach, however, tends to be suboptimal in practice in multiple ways. Firstly, hardware limitations are a concern, as memory becomes an issue with large replay buffers. Further, the data in the replay buffer can become stale, meaning that either the experiences do not reflect the current policy anymore and are, therefore, of less use or that experiences have been sampled enough times already and are no longer impactful.

3.2.2 On the Importance of Exploration for Generalization in Reinforcement Learning

"On the Importance of Exploration for Generalization in Reinforcement Learning" by Jiang et al. [16] explores the importance of exploration on generalizability and proposes "Exploration via Distributional Ensemble" (EDE), a novel exploration method. The paper investigates disentangling reducible epistemic uncertainty from non-reducible aleatoric uncertainty, which is caused by an uncertainty of the current context. EDE uses an ensemble of networks that are trained to estimate the uncertainty of a state, as they believe this will better discern between epistemic and aleatoric uncertainty. They argue that putting the agent into sub-optimal states during training helps improve its performance during testing when such under-visited sub-optimal states may be unwillingly reached. While their motivation and proposed goal are similar to this thesis's, the use of ensembles differs strongly from our solution. While their improvements compared to ε-greedy exploration or Noisy Net [12] are significant, they do not compare against state-of-the-art exploration methods such as such as intrinsic reward-based exploration. Nonetheless, they exemplify that better exploration not only improves an agent's generalizability but is vital to its generalization performance.

3.2.3 Zero-Shot Task Generalization with Multi-Task Deep Reinforcement Learning

"Zero-Shot Task Generalization with Multi-Task Deep Reinforcement Learning" by Oh et al. [28] tackles a specific subgenre of generalization, as they focus on sequential tasks. The aim is to improve an agent's zero-shot task generalization capabilities with a focus on sequences of parameterized sub-tasks, meaning that the agent's immediate goal changes whenever it reaches its previous goal. It does so by learning variations of the sub-tasks and then, through meta-learning, adapts the agent to understand previously unseen combinations and orders of parameterized subtasks.

3.2.4 Towards Robust Bisimulation Metric Learning

"Towards Robust Bisimulation Metric Learning" by Kemertas and Aumentado-Armstrong [18] builds on the use case of a bisimulation metric introduced by Zhang et al. [38]. It aims to improve the robustness of learned representations under complex conditions. Traditional deep RL algorithms often struggle with noisy or distracting environments, leading to reduced performance. To combat this, the authors propose improvements such as generalizing value function approximation bounds and incorporating norm constraints to mitigate instability issues regarding the embedding. This means the agent learns to disregard irrelevant information in the observation, increasing efficiency and robustness. They empirically

validate the work and demonstrate that the proposed methods not only increase the stability of the learning process but also work well in sparse reward conditions.

3.3 Task and Learning Enhancements

While many traditional approaches in RL focus on optimizing an agent's policy or its ability to generalize through behavioral changes, other strategies exist. This section discusses strategies that, for example, focus more on the training loop than the actual agent, aiming to improve an agent's qualities through different means. This is, in some ways, similar to our approach, as the underlying agent has not been changed. These methods could further be used together with our proposed methods, therefore they are not used as comparisons.

3.3.1 Meta Learning

Meta-learning involves training models on various tasks to improve the performance and generalizability of the agent [2]. Meta-learning tries to find similarities across tasks and infers useful representations. Further meta-learning aims to find an optimal set of hyperparameters for a given agent. Unlike many other methods, this does not focus on zero-shot generalization, as the knowledge gained through prior tasks should facilitate quick adaptation to new tasks. Prominent versions of meta-learning are Reptile [27] and MAML [11], but other research in that area has also been fruitful.

3.3.2 Domain Randomization

"Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World" by Tobin et al. [35] explores domain randomization, a technique where the simulation environment is varied randomly during training to improve the robustness and generalization of the RL agents. This approach has successfully trained agents in simulated environments that can effectively transfer their skills to real-world scenarios. Further, some of the methods in this thesis can be considered an alternate version of domain randomization, in which only the starting position of the agent gets changed and not the context, which is also controlled by the agent and not the environment.

3.3.3 Curriculum Learning

Curriculum learning involves altering the training cycle so that the tasks or data become progressively more demanding, starting with simpler or more familiar instances and gradually increasing in complexity. By providing a structured learning curriculum, agents can build basic skills first, avoid overfitting to specific tasks, and then further enhance performance and generalization capabilities by incrementally increasing the diversity and difficulty of the tasks provided[26].

Chapter 4

Methodology

This section will explain the methodological framework employed in our experiments. This thesis focuses on proposing novel techniques to enable RL algorithms to generalize more effectively across contexts. As previously mentioned, this is achieved by increasing the diversity present in an agent's replay buffer. To do so, every method introduced consists of two phases. The first brings the agent into a new "interesting" state, meaning one that is under-explored and, therefore, somewhat outside of the agent's comfort zone. In the second phase, the agent continues with its off-policy algorithm. This enables all the methods to be utilized as extensions of any off-policy reinforcement learning algorithm; however, we chose DQN for the purpose of this thesis. This chapter explains the concept behind each algorithm and provides pseudo-code.

4.1 Teleportation

The Teleportation method is a proof of concept, partly inspired by the first stage of the Go-Explore algorithm. The aim is to demonstrate that learning to reach the goal from various states assists the agent at better generalizing to unseen situations and enhances understanding of the environment. To do this, this method takes the approach of "teleporting" the agent to a new position, which is usually impossible without modifying the environment, but makes the impact of the diversifying start states stronger and more noticeable. Pseudocode for this method is provided in Algorithm 1.

At the start of an episode, the teleportation is activated with a user-set probability of c. The reason for setting c lower than 1 could be to ensure a stable replay buffer diversity. States are sampled from the replay buffer of the agent and are evaluated with an RND module to select state g with the highest uncertainty associated. The before-mentioned RND module is, however, strictly used for uncertainty evaluation and not for exploration, as the ε-greedy strategy is used. The sampling of states from the replay buffer is necessary to use this method without domain knowledge, as otherwise teleportation states could be invalid or outside of the reachable set $S_r(M|S^0_{train})$. The environment then loads the current context but sets the agent's position to be the same as in g, initiating an episode from the given po-

sition. The agent then tries to reach the goal while continuously employing the exploration strategy, as he would in standard DQN learning.

The strength of this method lies in its ability to teleport, similar to the first phase of the original version of GoExplore [8]. However, it relies on state-loading, a feature most environments do not provide. Even then, further modifications to the environment and, therefore, high domain knowledge are still required. Following this algorithm is merely a proof of concept and is meant as an upper bound of the potential of the following methods.

Algorithm 1 Teleportation Algorithm

Hyperparameters: teleportation chance c, sample size t, Loss function $\mathcal{L}$
Initialize agent $\mathcal{A}$ with replay buffer $\mathcal{D}$
Initialize random network distillation function RND(x)
Initialize $done = False$
Initialize environment $\mathcal{E}$
$s \leftarrow \mathcal{E}$
while Training not finished **do**
    while Rollout not finished **do**
        if $done = True$ **then**
            Sample a random number $r \sim$ Uniform $(0,1)$
            if $r < c$ **then**
                $b^t \sim \mathcal{D}$ ▷ t samples are drawn
                $s_{max} = \arg\max_{s \in b^t} \text{RND}(s)$
                $p, c = s$ ▷ every state consists of (state'×context)
                $p', c' = s_{max}$
                $s = p', c$
            end if
        end if
        action $a \leftarrow \mathcal{A}(s)$
        reward r, next state s', $done \leftarrow \mathcal{E}(a)$
        store transition $(s, a, r, s', done)$ in $\mathcal{D}$
        $s \leftarrow s'$
    end while
    if enough experiences in $\mathcal{D}$ **then**
        $b \sim \mathcal{D}$
        update $\mathcal{A}$ with $\mathcal{L}(b)$
        update RND with RND(b)
    end if
end while

4.2 Pure Exploration

As the Teleportation method is not always applicable, we aim to emulate the same procedure in a generally applicable way while keeping the complexity low. As exploration strategies naturally tend to move to novel states, they are also helpful for our purpose. Our novel method, "Pure Exploration," can be understood as an extension of traditional exploration, combined with the aim of diversifying the starting positions of an agent and, therefore, also aims to increase the diversity of the replay buffer. This thesis will test Pure Exploration with a shallow exploration strategy, ε-greedy, and a deep exploration strategy, intrinsic reward based on RND.

Algorithm 2 Pure Exploration Algorithm

Hyperparameters: exploration duration d, Loss function $\mathcal{L}$
Initialize exploration policy exp
Initialize agent $\mathcal{A}$ with replay buffer $\mathcal{D}$
Initialize $done = False$
Initialize environment $\mathcal{E}$
Initialize $c = 0$ ▷ c is an episodic step counter
$s \leftarrow \mathcal{E}$
while Training not finished **do**
    while Rollout not finished **do**
        if $done = True$ **then**
            $c \leftarrow 0$
        end if
        if $c < d$ **then**
            $a \leftarrow \exp(s)$
            $s \leftarrow \mathcal{E}(a)$
        else
            $a \leftarrow \mathcal{A}(s)$
            reward r, next state s', $done \leftarrow \mathcal{E}(a)$
            store transition $(s, a, r, s', done)$ in $\mathcal{D}$
            $s \leftarrow s'$
        end if
        $c \leftarrow c + 1$
    end while
    if enough experiences in $\mathcal{D}$ **then**
        $b \sim \mathcal{D}$
        update $\mathcal{A}$ with $\mathcal{L}(b)$
    end if
end while

The algorithm splits every episode in the training cycle into two phases: a pure exploration phase and a goal finding or main phase. The length of the pure exploration phase is determined by the hyperparameter d. Pseudocode is provided in Algorithm 2. During

the exploration phase, the agent strictly(”purely”) follows its own exploration strategy, for example, ε-greedy(1-greedy in this use case), while ignoring its currently trained policy. These trajectories should not be added to the replay buffer as they would be strongly off policy and hinder efficient training. After the set amount of actions d has been taken, the agent switches to the main phase. In this stage, the agent continues as usual by trying to find the goal with the help of its policy while continuing to utilize its original exploration strategy.

An immediate advantage of this method is its simplicity and the fact that it does not rely on any particular methods of the environment or any modifications to it. However, the higher d is set, the less time the agent has available to reach the goal, as the steps taken during the exploration phase count towards the environment's timeout limit. While this parameter can usually be set by the user, this also means the algorithm has a lower sample efficiency. Further, the algorithm does not provide as much control over the state distribution the agent encounters as Teleportation does.

4.2.1 ε-greedy based Exploration

The first choice of exploration strategy for testing is the standard ε-greedy, as it is a widely used exploration strategy and relies on uniform sampling from the action space and choosing the selected action. During the usage in Pure Exploration, it can be understood as being set to $\varepsilon = 1$, and furthermore, the trajectory generated can also be seen as a Random Walk. Its shortcomings are scenarios in which the wrong action leads to terminal states, as well as scenarios that require a specific sequence of action to escape.

4.2.2 Intrinsic reward-based exploration

Intrinsic Reward has become a widely used deep exploration strategy, with many implementations being based on RND [5]. Instead of randomly sampling from the action space, the RND module guides the agent towards uncertainty. During normal usage, the output of the RND module is scaled with a β value. During the first phase of Pure Exploration, it can be understood as a beta value of ∞. After d actions, the agent should be in a completely new position, and the main phase of Pure Exploration should start. This, as is also the case with ε-greedy, leads to an increased variance in starting positions. However, it should be both more evenly distributed as well as consist of more ”interesting” states.

4.3 GoExploit

The GoExploit method is the most advanced solution presented in this paper with pseudocode provided in Algorithm 3. It also aims to replicate the Teleportation method but without domain knowledge or modifications to the environment. It is comparable to the first phase of GoExplore [9], but after reaching the state selected, we want to reach the actual goal (”exploit”) instead of exploring the surroundings. It employs the use of not only the main agent but also a secondary agent, which is a goal-conditioned agent, henceforth

called GC, that is trained as a universal value function approximator [31]. Both agents have their own architecture and replay buffer, which gets only filled by their own actions. This is necessary to avoid causing instability in the learning process by including data that shows strongly off-policy behavior.

Like Pure Exploration, GoExploit works in two phases. The first aims to reach an interesting starting position for the main agent to continue from in the second stage. As we have to find reachable states to move to, at the start of an episode, states are sampled from the replay buffer of the main agent and, similar to Teleportation, evaluated with an RND module. As we aim to move the agent out of its "comfort zone" and increase the diversity of both the starting state and of those in the replay buffer, the state with the highest uncertainty attributed to it is selected as an interim goal. This interim goal is given to the GC, which aims to reach this state within a maximum of d actions, where d is a hyperparameter set by the user. After reaching either the interim goal or the maximum allowed trajectory length, the GC stops, and the main agent continues to reach its goal.
The latter case of ending the GC phase is expected to be common, as the interim goal from the replay buffer cannot be guaranteed to be within the same context and topology as the current environment. This also leads to the importance of using Hindsight Experience Replay in the training of the GC, as it enables the agent to learn from its interactions with the environment even when only rarely reaching its goal.

The drawbacks of this algorithm are the fact that an additional agent is necessary, as well as greatly increased wall clock run time and further an increased computational demand as two agents have to be trained at the same time. However, the advantages of this method are its close approximation of the Teleportationmethod and, therefore, a greatly increased generalizability of the agent without relying on domain knowledge or altering the environment in any way.

Algorithm 3 GoExploit algorithm

Hyperparameters: exploration duration d, Loss functions $\mathcal{L}_{main}$ and $\mathcal{L}_{gc}$
Initialize agent $\mathcal{A}_{main}$ with replay buffer $\mathcal{D}_{main}$
Initialize agent $\mathcal{A}_{gc}$ with replay buffer $\mathcal{D}_{gc}$
Initialize random network distillation function RND
Environment $\mathcal{E}$
Initialize $done = False$
Initialize $gc\_done = True$
Initialize $c = 0$ ▷ c is an episodic step counter
$s \leftarrow \mathcal{E}$
while Training not finished **do**
    while Rollout not finished **do**
        if $done = True$ **then**
            $c \leftarrow 0$
            $gc\_done \leftarrow False$
            $b \sim \mathcal{D}_{main}$
            $g = \arg\max_{s \in b} \text{RND}(s)$
            sample goal state $g \sim D_{main}$
        end if
        if $c >= d$ or gc_done **then**
            action $a \leftarrow A_{main}(s)$
            reward r, next state s', $done \leftarrow \mathcal{E}(a)$
            store transition $(s, a, r, s', done)$ in D_{main}
        else
            action $a \leftarrow A_{gc}(s, g)$
            reward r, next state s', $done \leftarrow \mathcal{E}(a)$
            store transition $((s, g), a, r, s', done)$ in D_{gc}
            $gc\_done \leftarrow s' == g$
        end if
        $s \leftarrow s'$
        $c \leftarrow c + 1$
    end while
    if enough experiences in D_{main} **then**
        sample $b \sim D_{main}$
        update A_{main} with $\mathcal{L}_{main}(b)$
        update RND with RND(b)
    end if
    if enough experiences in D_{gc} **then**
        update $\mathcal{D}_{gc}$ with HER algorithm
        sample $b \sim D_{gc}$
        update A_{gc} with $\mathcal{L}_{gc}(b)$
    end if
end while

Chapter 5

Experiments

This chapter will cover the setup of the experiments and the architecture used for the agent. It will also explain the environment and the different configurations of environments used. Further, it will list the experiments done to evaluate the performance of the methods proposed in chapter 4 as well as to answer the research questions asked in 1.2. The reasoning behind the experiments' inclusion will also be explained.

5.1 Architecture

The agent has the same architecture in every experiment and method used. As the observation is an image, the data is first passed through 3 CNN layers and, after this, through incrementally shrinking fully connected layers. A ReLU activation function is used between each layer. Exact hyperparameters are in the appendix A.2.

5.2 Environment

The environment used in all experiments is a fully observable FourRoom adaption of the popular Minigrid environment. It is a 2D grid-world scenario consisting of four connected rooms, separated by walls with doorways (Fig.5.1). The agent's task is to navigate from a starting position to the goal. Upon reaching the goal, a reward of 1 is given to the agent. No other rewards, neither positive nor negative, are given. The contexts differ by starting position, goal position, and doorway position. Furthermore, five different sets of configurations are used in this thesis. The corresponding validation and test sets each utilize identical topologies, meaning the door positions are the same but with different starting and goal positions.

- **Training set:**
  The training set consists of 40 different contexts.
- **Validation/Test Reachable set:**
  The reachable set consists of 40 contexts that are the same as the **Training** ones, but

with different starting positions. The name comes from the fact that every context in this set is reachable from one of the contexts of the training set.

- **Validation/Test Unreachable set:**
  The unreachable set consists of 40 contexts that differ in starting position, wall position, and goal position from the training and both reachable sets. No context of this set can be reached from any context of prior sets through interaction with the environment.

The average reward over the evaluated environments, 40 for each set, has been chosen as the metric for evaluation. Another metric originally included was the average steps taken to reach the goal, but as this is just an inverted version of the reward average, it has been excluded from the graphs. Further, most graphs display the values achieved on the y-axis alongside the training steps taken on the x-axis, often denoted in "Timesteps (1e3)". However, some graphs solely focus on the results achieved at the end of training to simplify comparison.

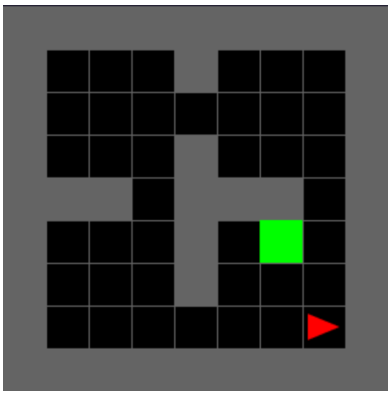

Figure 5.1: Visualization of the environment with the agent's position marked by a red arrow and the goal given by the green square. The picture does not represent the observation the agent receives (see A.1).

5.3 Hyperparameter tuning

Every method was hyperparameter-tuned on the validation sets to achieve optimal performance. These include the duration (step-count) for which the proposed methods are active, as well as the chance of activation for both the Teleportation method as well as GoExploit. Further, as GoExploit consists of two different agents, various architecture choices were evaluated for the goal-conditioned agent. Another experiment was also done to determine whether the training length is enough for the goal-conditioned agent to reach its potential. Some of the more interesting results have been added to the appendix A.3. Also, for a list of the hyperparameters that were used further, see A.2.

5.4 Generalization performance comparison

The core experiment of this thesis evaluated the methods proposed in a quantitative comparison. Every method was assessed on 50 seeds and compared against each other as well as a baseline regarding their generalization performance. Each method was first hyperparameter tuned on the validation sets, and the best performance was compared on both test sets. As Pure Exploration depends on an agent's exploration strategy, it was tested with ε-greedy and intrinsic reward-based exploration, with both being compared against their respective baseline. The hyperparameters that were used for the architecture and the comparison of the agents can be found in the appendix (A.2).

5.5 Starting Positions

An experiment evaluated the number of starting positions each method generates. Every method proposed splits an episode into two parts. The first is the phase in which various methods steer the agent to a new position. Therefore the start of the second phase, in which the traditional DQN agent acts, is considered a starting position for this purpose. This was done to showcase the impact of more starting positions, for which we have to confirm that the methods indeed increase the number of starting positions an agent experiences. Every method has been evaluated on 5 different seeds.

5.6 Diversity

The aim of all methods proposed is to increase generalizability through an increase in diversity in the replay buffer. To showcase this connection, the impact of the method on the replay buffer's diversity had to be evaluated. This was also necessary to show a link between an increase in starting positions and the diversity of the replay buffer.

To do so, 10 seeds were run in each method and on both baselines. During these training runs, the replay buffer was evaluated every 1000 steps, and the unique state-action pairs and unique states were counted. As this is a small environment, there are only 6240 different states reachable from the training set environments. However, to fully exhaust the possibilities of experience, the action that was taken in a given state has to be taken into account as well. With an action space size of 3 this means that a total of 18720 of state-action pairs are reachable from the training set in theory.

5.7 GoExploit ablation

As GoExploit offered many design decisions, we decided to investigate those further. This includes but is not limited to evaluating the impact that adding the experiences of the main agent to the goal-conditioned agent would have. Further, the importance of the tournament-size, the number of possible goals that are sampled from the replay buffer, was also put into question.

As the GoExploit method also depends on many design decisions of the second, the goal conditioned agent, we also investigated those choices, however only included them in the experiment section of the appendix A.3, as most likely vary between environments and applications.

Chapter 6

Results

This chapter will cover the most notable experimental results. Further results that are not included here are attached in the appendix A.3.

6.1 Generalization Perforamnce

The core experiments of this thesis are quantitative comparisons. The methods will be iteratively added to a plot and compared against a baseline and each other.

6.1.1 Teleportation

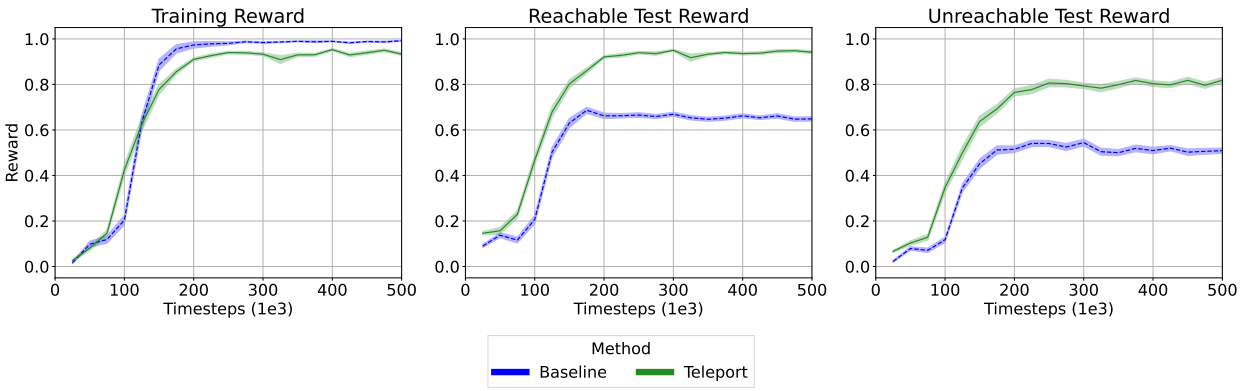

Figure 6.1: Results of Teleportation methods with a buffer size of 50,000.

The Teleportation method served as a proof of concept and demonstrated promising results across the board. Even though the performance was below the baseline in the training set, it significantly outperformed it in both the reachable, as well as the unreachable test and validation set with both a buffer size of 50,000 (Fig. 6.1) as well as 500,000 (Fig. 7.1). As the aim of these methods is to improve generalizability, we care more about the performance in the test and validation sets, as these are important to evaluate an agent's

ability to generalize. Therefore, the slight reduction in performance on the training set is acceptable. The performance will be used as an upper bound of expectation regarding generalization for the following methods. However, it again has to be stressed that this method is enabled by strong modifications to the environment, without which it would not be feasible to implement. Therefore it is only usable as a proof of concept and an idealised implementation of the basic idea of all methods proposed in this thesis.

6.1.2 Pure Exploration

As Pure Exploration heavily relies on the exploration strategy employed by the agent, we tested it both with ε-greedy exploration and an RND-based intrinsic reward exploration strategy.

ε-greedy Exploration

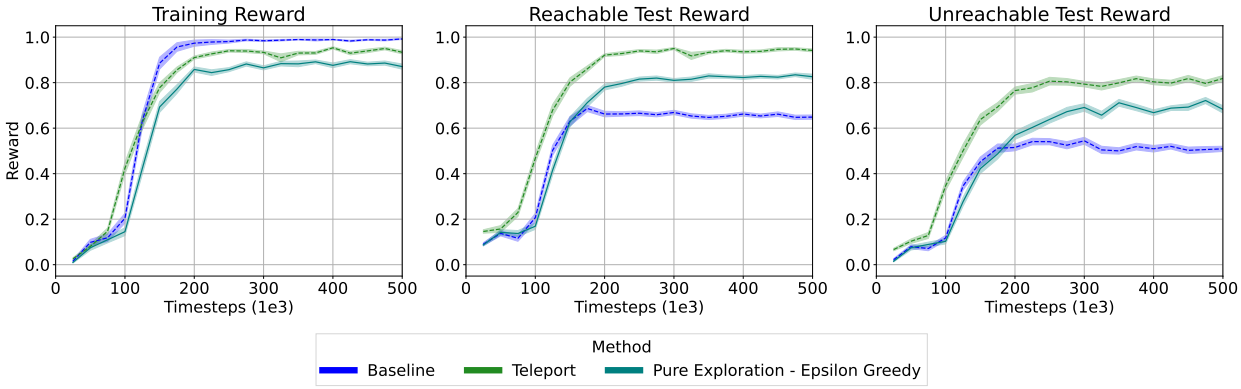

Figure 6.2: Results of Pure Exploration method using ε-greedy with a buffer size of 50,000.

Despite its relatively simple implementation, the Pure Exploration method utilizing eps-greedy exploration exceeded expectations with a disproportional success. With both a buffer size of 50,000 (Fig. 6.2) as well as 500,000 (Fig. 7.1), the improvements on both the reachable as well as the unreachable test set are significant.
However, the amount of randomly sampled actions compared to an agent that purely utilizes ε-greedy is not to be understated. This reduces the sample efficiency, which explains the delayed learning visible in the performance on the training set. Another problem of this approach is the sampling of the action space, as both continuous action spaces and discrete action spaces, with many dead actions (for example ProcGen [7]), pose a challenge if not properly adjusted. Regardless of these complications, the results of this method on this environment were above expectations.

Intrinsic Reward Exploration

When utilizing the Pure Exploration method with the intrinsic reward-based exploration, we included both the standard baseline, utilizing ε-greedy exploration, as well as the baseline for intrinsic reward exploration. The results can be seen in figure 6.3.

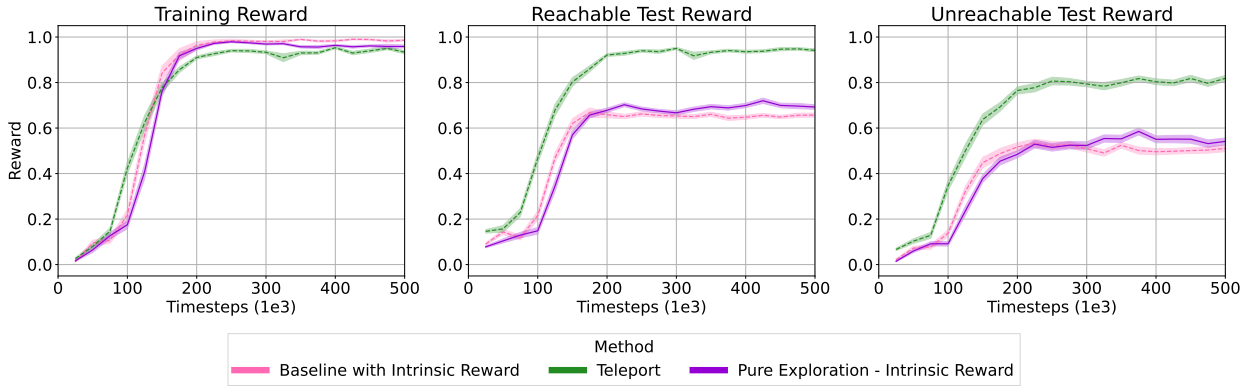

Figure 6.3: Results of Pure Exploration method using intrinsic Reward Exploration with a 50,000 buffer.

Unfortunately, this method performed below expectations, which might have been caused by improper implementation and tuning or by an unfortunate combination with this specific environment.

Therefore, despite being based on a usually smarter exploration strategy, this method's performance showcases the importance of selecting a valid exploration strategy for this algorithm. However, as this phenomenon might be environment-based, we assume that smarter action sampling improves this method's chances in complex and especially continuous environments.

6.1.3 GoExploit

The most intricate method in this chapter is the GoExploit approach, which relies on training two agents simultaneously, resulting in a multitude of hyperparameters to tune as the second agent is not a direct copy of the first one, therefore it has to be designed independently from the ground up. Further, it is also the most complex method to implement.

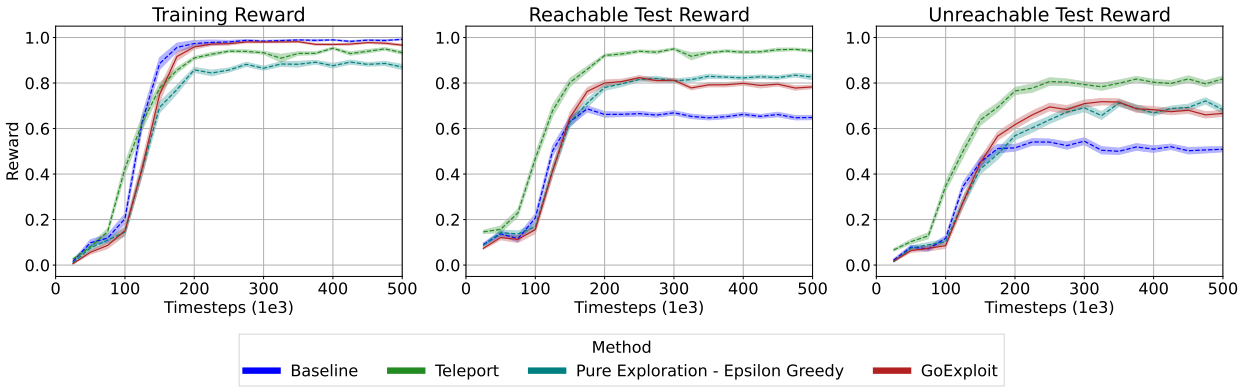

Figure 6.4: Results of the GoExploit method with a 50,000 buffer

The quantitative experiment revealed mixed results. Improvements to the baseline are significant in both reachable as well as unreachable test sets, with both a buffer size of 50,000 (Fig. 6.4) as well as 500,000 (Fig. 7.1). It performed approximately the same in all evaluation cases as Pure Exploration with ε-greedy did. However, after 300,000 steps, the agent's performance started to deteriorate slightly, which could be caused by various reasons. One of which will be further discussed in Subsection 6.2.2. Overall, it performed very well, though somewhat below expectations, particularly considering the complexity of the implementation compared to its performance equivalent Pure Exploration.

6.2 Analytical Experiments

Other experiments aside from comparing test results of the proposed methods were conducted to analyze the methods further and investigate their effect. They will be presented in this section. Further experiments that are either inconclusive or not impactful were not included here and were added to the appendix A.3.

6.2.1 Starting Positions

The experiment to evaluate the number of starting positions an agent starts from throughout a training run showed expected results. The baseline has 40 different starting positions in the training set. The next lowest was Pure Exploration, using intrinsic reward-based exploration, with an average amount of 463 unique starting positions. The next was the GoExploit method with an average of 1,889 unique starting positions. Pure Exploration, using ε-greedy exploration, achieved an average of 2,690. Teleportation more than doubled that with 5,698, showcasing one of the reasons behind its impressive performance. To put these numbers into proper context, the total amount of states reachable from the training set is 6,240. That means the percentage of possible states that were started from, from the lowest to the highest: Baseline (0.6%), intrinsic reward-based Pure Exploration (7.4%), GoExploit (30.3%), ε-greedy Pure Exploration (43.1%), Teleportation (91.3%).

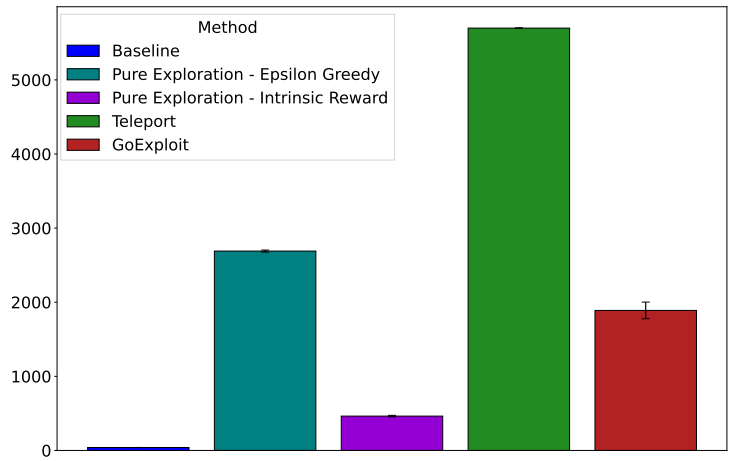

Figure 6.5: Comparison of the number of starting positions an agent starts from in a training run. All agents used a buffer size of 50,000.

6.2.2 Diversity

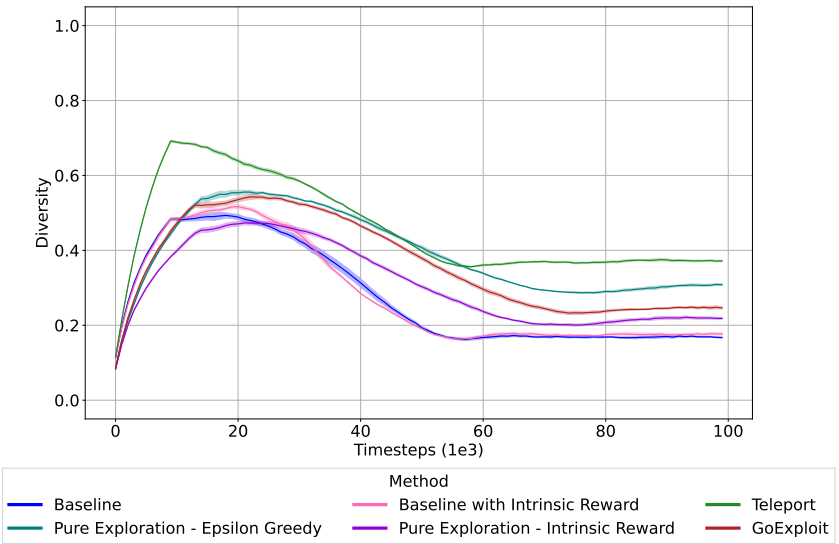

Figure 6.6: Comparison of the diversity of state-action pairs in the replay buffer for a buffer size of 50,000. Normalized with a maximum possible value of 18,720 state-action pairs.

To simplify the graphs, they were normalized following the maximum number of state-action pairs reachable from the training set ($\frac{\text{\# of unique state-action pairs}}{18720}$) and the maximum number of reachable states reachable from the training set ($\frac{\text{\# of unique states}}{6240}$).
For the state-action pair diversity, it can be seen in the graph (Fig. 6.7) that the methods achieving a higher generalization performance also achieved a higher diversity. Notably,

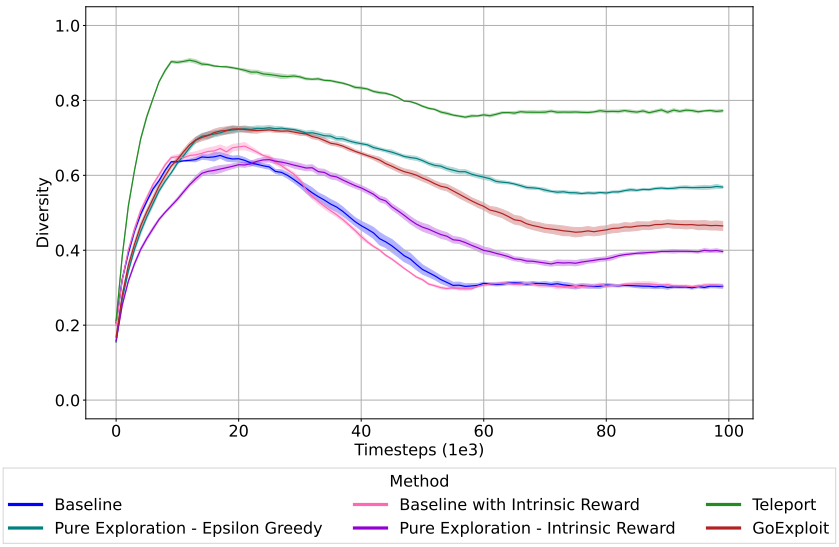

Figure 6.7: Comparison of the diversity of state-action pairs in the replay buffer for a buffer size of 50,000. Normalized with a maximum possible value of 6,240 states.

the only crossing of lines is GoExploit with Pure Exploration using ε-greedy. This almost directly lines up with their performance on both the unreachable test set and the validation set. This mostly also holds for the state diversity graph (Fig. 6.6). However, the difference between methods is more pronounced. This can be explained by the fact that the proposed methods do not influence the agent's decisions after ending their corresponding phase. This means the methods proposed in this thesis have a stronger influence on state diversity than state-action diversity.

6.3 GoExploit ablation

As the GoExploit method is the most intricate proposed in this thesis, it also required more experiments regarding its qualities. Especially as multiple agents lead to a multitude of implementation decision and hyperparameter choices. As the secondary agent differs both in architecture as well as use case the choices are up to the implementation and have to take the environment at hand into account. The following experiments were done to verify choices made and to showcase potential options.

6.3.1 Extended Replay Buffer

An additional experiment was done to explore the effect that off-policy data would have on the goal-conditioned agent. As both agents act independently and have independent replay buffers, the training data, especially of the secondary agent, is fairly limited. To combat this, the trajectories controlled by the main agent were added to the secondary buffer. Under normal circumstances, this would be strongly off policy and lead to deteriorating performance.

However, as the agent learns mostly through the effect of Hindsight Experience Replay, it could lead to an improvement. As is shown in Figure 6.8, the effect is very slight and not a signifcant improvement. Further, it seems that the extended replay buffer leads to a higher level of instability.

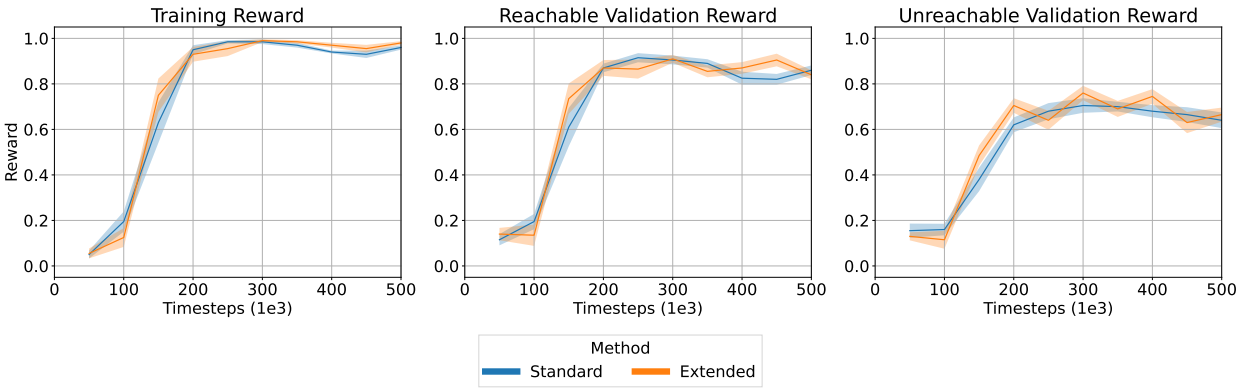

Figure 6.8: Results of adding main agent buffer to goal conditioned agent's buffer compared to the standard way.

6.3.2 Effect of tournamentsize

Another experiment was done to investigate the impact of tournamentsize in the algorithm. This size determines how many states are sampled from the replay buffer, which will then be evaluated through the RND module to select the most uncertain ones. A tournamentsize of 1 would mean that the first state that was sampled will be used. As a comparison, 150 was chosen. As can be seen, the larger tournamentsize leads to improved performance. A smaller tournamentsize would reduce computational costs. As shown in Figure 6.9 the performance difference is visible, even if not particularly significant. A size of 3 was utilized for further experiments.

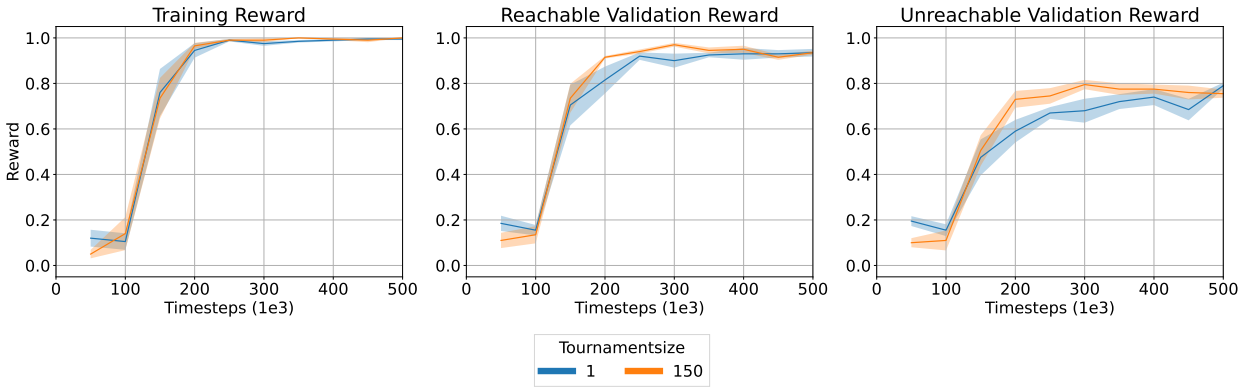

Figure 6.9: Comparison of different tournamentsize used for GoExploit with a buffer size of 500,000.

Chapter 7

Discussion & Conclusion

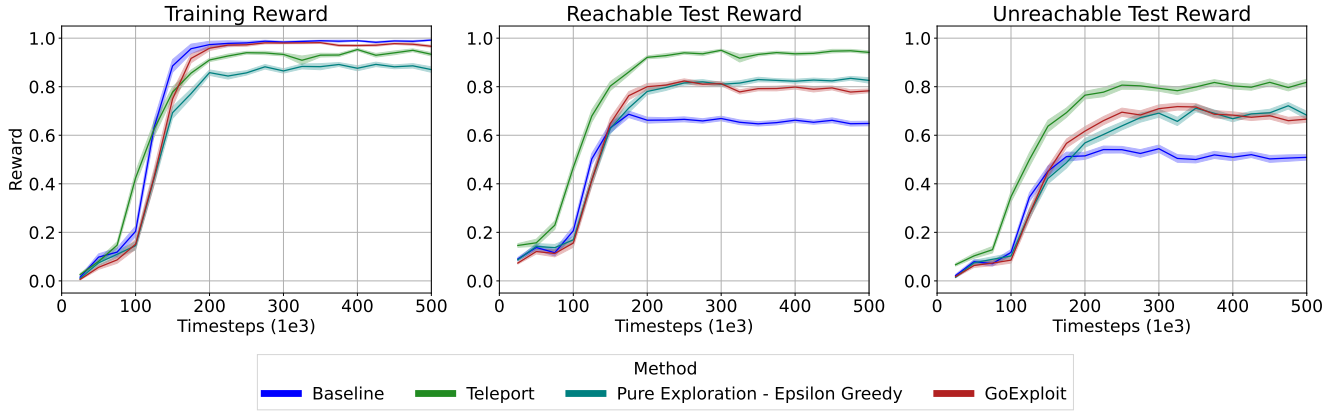

Figure 7.1: A comparison of all methods proposed used in this thesis, using a replay buffer size of 50 000

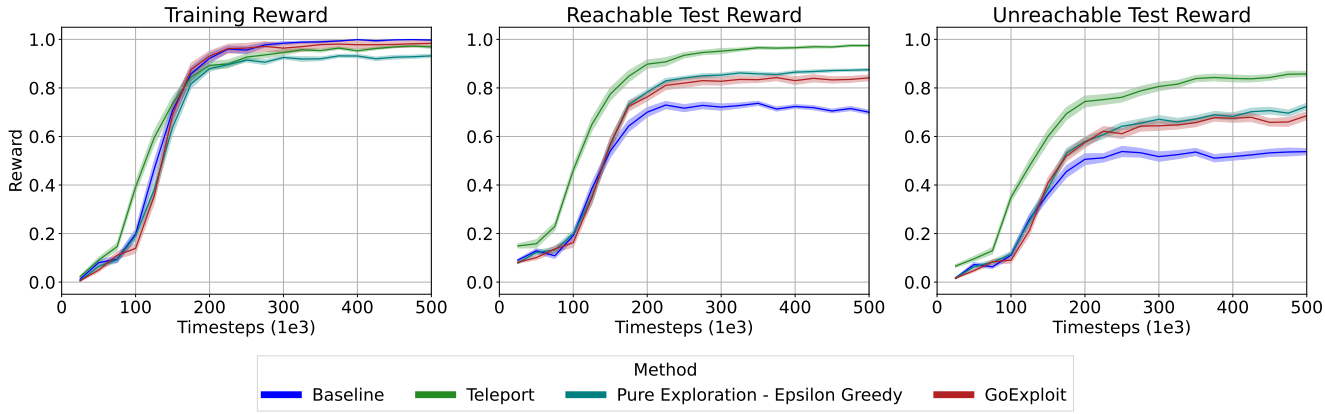

Figure 7.2: A comparison of all methods proposed used in this thesis, using a replay buffer size of 500 000

In this thesis, we introduced and analyzed three novel methods aimed at improving the generalizability of reinforcement learning agents. However, only two of these are actually usable. Each method offered advantages and disadvantages, bringing us further insight and understanding of how RL agents adapt to diverse environments and contexts. This chapter will focus on discussing these findings. For reference, see the comparative results of all novel methods against a baseline using ε-greedy as an exploration strategy utilizing a replay buffer size $50,000$ (Fig. 7.2) and $500,000$ (Fig. 7.1) respectively. Alternatively, the plots that also include results from the validation set have been added to the appendix A.3.1.

7.1 Teleportation

It is clear from the Teleportation experiment results that it serves as a good synthetic upper bound for determining generalization performance, as the results across all evaluation sets and on both buffer sizes exceed any other method. Its limitations are severe, as it can only be used when a valid MDP can be generated on demand to meet the requirements of the Teleportation method employed. As this is not the case in any environment we are aware of, this method's applicability is limited. Despite this, the method functions as a proof of concept and showcases that increasing the number of starting positions, especially in such a way that selects interesting states to start from, is a viable solution to closing the generalization gap.

7.2 Pure Exploration

Pure Exploration as a method is more of a concept than an actual specific implementation. As it depends strongly on the exploration strategy employed by the agent, the difficulty of implementation mirrors that. For both use cases at hand, the implementation only has a few lines of code. The success of both versions of this method differs strongly depending on the exploration strategy used, as you can see further in the following sections. This, however, again showcases both the importance of selecting the right exploration strategy for an environment, as well as the dependency of this method on said strategy. While the performance of intrinsic reward-based Pure Exploration was disappointing, it did not harm the agent's performance in any way. Therefore, at minimum, a small Pure Exploration phase should be strongly considered.

7.2.1 ε-greedy exploration

ε-greedy based Pure Exploration performed above expectations. It was originally intended as a lower bound, but it prevailed and outperformed any other method than the upper bound of Teleportation. This might be caused by the amount of starting positions it generates as can be seen in section 6.2.1. With its simplicity, its downfall might be terminal states, as it cannot learn to avoid them.

7.2.2 Intrinsic reward exploration using RND

Despite its surprisingly poor performance compared to the ε-greedy implementation, the intrinsic reward exploration-based Pure Exploration still outperformed its baseline (see Fig. 6.3). This showcases that even when using this exploration strategy, a Pure Exploration phase can benefit the agent's ability to generalize. This is especially beneficial as intrinsic reward exploration is commonly used in more complex environments and should be able to perform even better when the underlying intrinsic reward exploration properly improves exploration.

7.3 GoExploit

GoExploit was the original aim of this thesis, as it mirrors the first phase of "First return then explore" [9] with a focus on exploitation instead of exploration.
While GoExploit exhibits promising results in all evaluation sets, it also reaches about the same level of generalization that ε-greedy based Pure Exploration does. As experiments (Section **??**) showed, this is not only a problem of training as even with longer training time, the performance did not increase significantly. Furthermore, its plethora of hyperparameters poses a tuning challenge, and the architecture of the secondary agent also has to be designed in a fitting way. While these statements seem to diminish the results at hand, they just showcase the need for improved investigation of Hindsight Experience Replay or even better alternatives as the secondary agent strongly relies on this algorithm.
A niche use case could be situations in which a perfect goal-conditioned agent can be utilized as a plug-in. The secondary agent does not aim to generalize to non-training environments, so it can replace the teleportation part without worrying about overfitting.

7.4 Conclusion

To return to the original questions asked in Section 1.2.

Does increasing the diversity of starting positions increase an agent's ability to generalize?
Yes, all methods introduced showcased an increase in starting positions, as can be seen in Figure 6.5. Further, the correlation between starting positions and generalization performance is visible, as all of the methods led to higher performance on the test sets and overall a smaller generalization gap, as can be seen in both Figure 7.2 and 7.1.

Does the diversity of the replay buffer impact the agent's ability to generalize?
Yes, as can be seen in the experiment comparing the diversity of replay buffers, the results seem to be correlated to the results of the agent on the test environments (see Fig. **??** and 7.2 respectively).

What are possible ways to achieve an increase in diversity of starting positions without altering the environment?

Pure Exploration (Alg. 2) is an easily implementable solution to achieve a diverse buffer without relying on domain knowledge and does not rely on domain knowledge. GoExploit (Alg. 3) is a more intricate solution that could achieve more success in complex environments. While Teleportation is not applicable in real-world scenarios, it underlines the finding that all these methods increased the diversity of starting positions and the replay buffer and further increased the generalization performance of the agent.

In total, this work not only conclusively answered the questions asked but also provided methods and algorithms to achieve the performance and improvements that were investigated and aimed for.

7.5 Future Work

While working on this thesis, various details would have been strongly out of scope, but this showed us the many possible avenues for future research in this direction.
Starting with GoExploit, which again relies strongly on the use of HER, which, however, limits the usage of this method to Off-Policy algorithms such as DQN. Therefore, a potential future work would be to adapt HER to On-Policy algorithms, as these seem to outperform in many use cases. This, in turn, would lead to a potential implementation of GoExploit to On-Policy algorithms.
The success of Pure Exploration opens the possibility of designing exploration strategies specifically for this use case, as well as investigating the effects of different exploration strategies.
Furthermore, the again-shown direct link between an agent's ability to generalize and the diversity of state-action pairs present in the replay buffer would lead to the assumption that an overall focus on increasing said diversity should be a priority for further research into generalization. Possibilities here range from using the diversity of the replay buffer as a benchmark during training to aiming to find novel ways to increase said diversity. This includes specific replay buffer implementations or further research into exploration strategies.

Bibliography

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017. URL http://arxiv.org/abs/1707.01495.

[2] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning, 2023.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013. ISSN 1076-9757. doi: 10.1613/jair.3912. URL http://dx.doi.org/10.1613/jair.3912.

[4] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. Unifying count-based exploration and intrinsic motivation. *CoRR*, abs/1606.01868, 2016. URL http://arxiv.org/abs/1606.01868.

[5] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation, 2018.

[6] Karl Cobbe, Oleg Klimov, Christopher Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. *CoRR*, abs/1812.02341, 2018. URL http://arxiv.org/abs/1812.02341.

[7] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *CoRR*, abs/1912.01588, 2019. URL http://arxiv.org/abs/1912.01588.

[8] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *CoRR*, abs/1901.10995, 2019. URL http://arxiv.org/abs/1901.10995.

[9] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, Feb 2021. ISSN 1476-4687. doi: 10.1038/s41586-020-03157-9. URL https://doi.org/10.1038/s41586-020-03157-9.

[10] Shadi Endrawis, Gal Leibovich, Guy Jacob, Gal Novik, and Aviv Tamar. Efficient self-supervised data collection for offline robot learning, 2021.

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL http://arxiv.org/abs/1703.03400.

[12] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. *CoRR*, abs/1706.10295, 2017. URL http://arxiv.org/abs/1706.10295.

[13] Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes, 2015.

[14] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. *CoRR*, abs/1803.00933, 2018. URL http://arxiv.org/abs/1803.00933.

[15] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667, 2019. doi: 10.1109/ACCESS.2019.2941229.

[16] Yiding Jiang, J. Zico Kolter, and Roberta Raileanu. On the importance of exploration for generalization in reinforcement learning, 2023.

[17] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996. URL https://arxiv.org/abs/cs/9605103.

[18] Mete Kemertas and Tristan Aumentado-Armstrong. Towards robust bisimulation metric learning. *CoRR*, abs/2110.14096, 2021. URL https://arxiv.org/abs/2110.14096.

[19] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *J. Artif. Int. Res.*, 76, may 2023. ISSN 1076-9757. doi: 10.1613/jair.1.14174. URL https://doi-org.tudelft.idm.oclc.org/10.1613/jair.1.14174.

[20] Alex Labach, Hojjat Salehinejad, and Shahrokh Valaee. Survey of dropout methods for deep neural networks. *CoRR*, abs/1904.13310, 2019. URL http://arxiv.org/abs/1904.13310.

[21] Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. Exploration in deep reinforcement learning: A survey. *Information Fusion*, 85:1–22, September 2022. ISSN 1566-2535. doi: 10.1016/j.inffus.2022.03.003. URL http://dx.doi.org/10.1016/j.inffus.2022.03.003.

[22] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.*, 8(3–4):293–321, may 1992. ISSN 0885-6125. doi: 10.1007/BF00992699. URL https://doi.org/10.1007/BF00992699.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

[25] Andrew Moore. Efficient memory-based learning for robot control. 06 2002.

[26] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *CoRR*, abs/2003.04960, 2020. URL https://arxiv.org/abs/2003.04960.

[27] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018. URL http://arxiv.org/abs/1803.02999.

[28] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. *CoRR*, abs/1706.05064, 2017. URL http://arxiv.org/abs/1706.05064.

[29] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *CoRR*, abs/1705.05363, 2017. URL http://arxiv.org/abs/1705.05363.

[30] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994. ISBN 0471619779.

[31] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schaul15.html.

[32] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015. URL http://arxiv.org/abs/1511.05952. cite arxiv:1511.05952Comment: Published at ICLR 2016.

[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[34] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[35] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

[36] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30 (1), Mar. 2016. doi: 10.1609/aaai.v30i1.10295. URL https://ojs.aaai.org/index.php/AAAI/article/view/10295.

[37] Max Weltevrede, Matthijs T. J. Spaan, and Wendelin Böhmer. The role of diverse replay for generalisation in reinforcement learning, 2023.

[38] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *CoRR*, abs/2006.10742, 2020. URL https://arxiv.org/abs/2006.10742.

[39] Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E. Gonzalez, and Yuandong Tian. Bebold: Exploration beyond the boundary of explored regions. *CoRR*, abs/2012.08621, 2020. URL https://arxiv.org/abs/2012.08621.

Appendix A

Appendix

A.1 Libraries

Aside from the altered FourRoom adaption of MiniGrid, other libraries were also utilized. A visualization of the environment can be seen in Figure A.1. However, the observation the agent has access to is set up differently. It is a 3-dimensional array consisting of $\{0, 1\}$, with 4 $9x9$ arrays. Further, the observation of the agent is centered around the agent. Layer 0 and Layer 1 represent the agent's position and viewing direction, respectively. Layer 2 displays the walls, and Layer 3 displays the goal position.
For implementation, the library stable_baselines3 was used. For the most part, the algorithms were used as is. However, the DQN implementation has been extended to a Double DQN version. Further, as the HER implementation of this library has a bug, we changed it to assert "done" flags correctly for virtual trajectories.
The code utilized for this project is public, and the link can be found in the abstract.

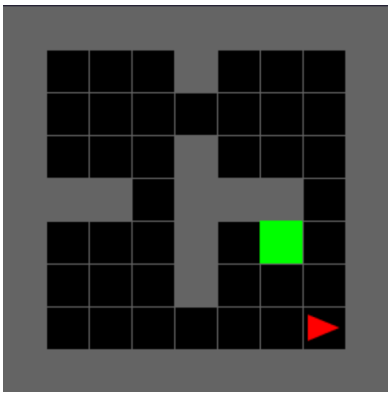

Figure A.1: Visualization of the environment with the position of the agent given by the arrow and the goal given by the green square.

A.2 Hyperparameter

Hyperparameter	Value
Total timesteps	500,000
DQN	
Buffer size	500,000
Batch size	256
Discount factor γ	0.99
Max. gradient norm	1
Gradient steps	1
Train frequency (steps)	10
Target update interval (steps)	10
Target soft update coeffcient τ	0.01
ε-greedy exploration	
Exploration initial ε	1
Exploration final ε	0.1
Exploration fraction	0.5
Intrinsic reward exploration	
Beta	0.1
Adam	
Learning rate	1×10^{-4}
Weight decay	1×10^{-5}
CNN	
Kernel size	3
Stride	1
Padding	1
Padding mode	Circular
Channels	32

Table A.1: Hyperparameters used if not stated otherwise.

Hyperparameter	Value
Teleportation	
”Teleportation” chance	1.0
Pure Exploration	
Maximum steps	30
GoExploit	
Maximum steps	30

Table A.2: Hyperparameters of the proposed methods used in the main generalization comparison.

A.3 Experiments

A.3.1 Method Comparison

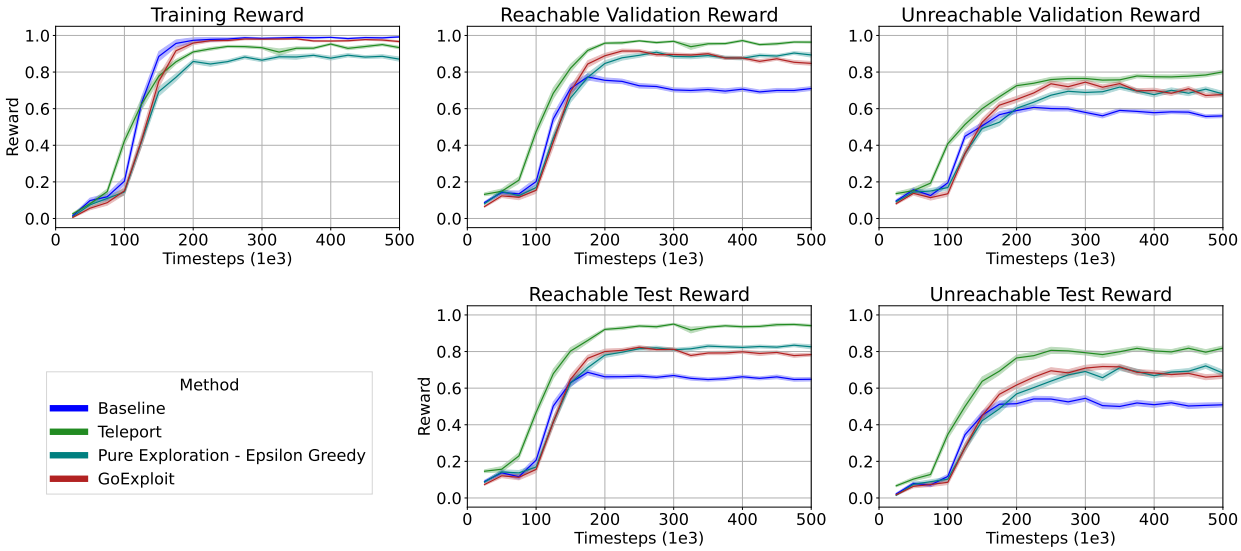

Figure A.2: A comparison of all methods proposed used in this thesis, using a replay buffer size of 500,000

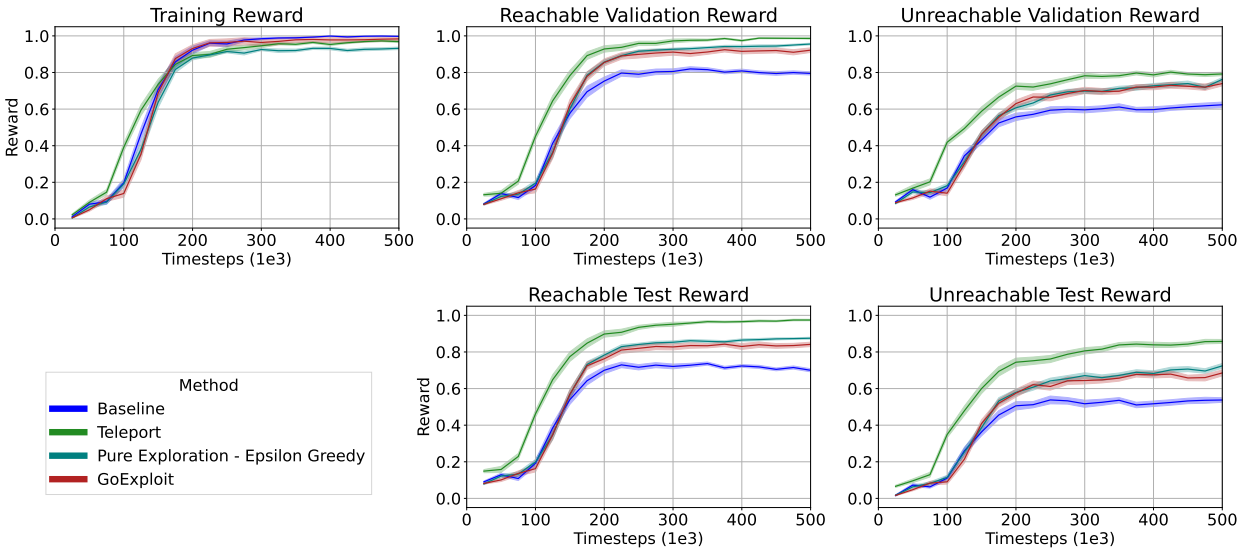

Figure A.3: A comparison of all methods proposed used in this thesis, using a replay buffer size of 50,000

A.3.2 Impact of more actions

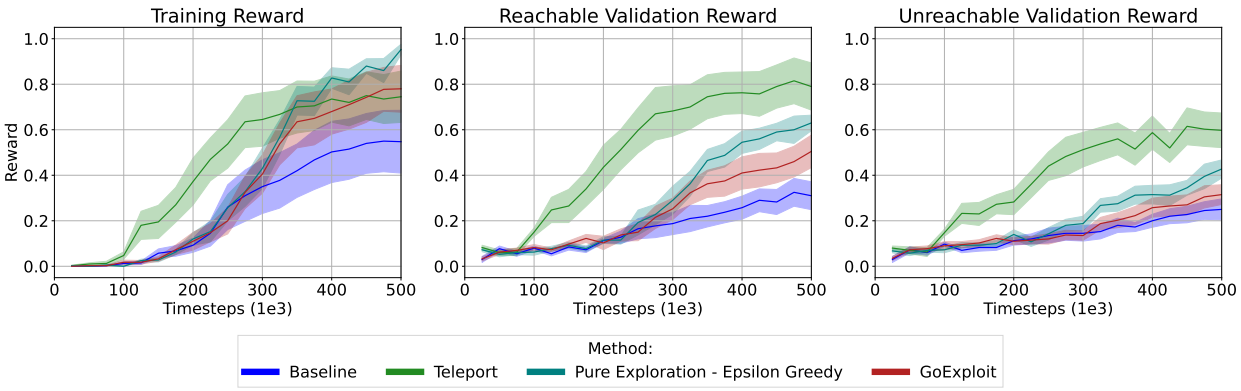

Figure A.4: Comparison of the agent's performance in an environment with increased action space of 10 actions using a replay buffer size of 50,000.

Another experiment investigated the impact of dead actions on the proposed methods. This was done by extending the action space from 3 to 10 actions and mapping the additional actions to "Do nothing". None of the hyperparameters were changed between the original main experiment and this. As we originally assumed that especially ε-greedy based Pure Exploration would struggle in this case, we were surprised to find it still outperforming all other methods except for Teleportation, aside from a slight lead in the training environment. Furthermore, GoExploit performed somewhat worse than expected. We assume this is caused by the overhead of training two networks to ignore the majority of possible outputs.

A.3.3 Effect of Teleportation chance on the Teleportation method

One experiment was done to investigate whether a value exists from which onward the teleportation is more of a hindrance than an improvement. This was investigated on both the 50,000 Buffer (Fig. A.5) as well as the 500,000 Buffer. Values of 70% and 100% performed the best with a significant difference to other teleportation chances values on the unreachable validation set. Further, on the smaller Buffer sizes higher teleportation chance also increased the training performance and made training more stable. This might come from the fact that otherwise, the buffer is filled with many duplicate entries, which the teleportation combats. This leads to a higher diversity in training data and, therefore, to higher performance.

A.3.4 Effect of ε-greedy Pure Exploration duration

The importance of the Pure Exploration phase's duration was investigated for the ε-greedy based Pure Exploration method (Fig. A.6). However, it shows that the value is less impactful than expected, influencing the reachable test set results more than the unreachable ones. However, the trend is still visible in both and reaches a soft peak at 30 in the reachable set

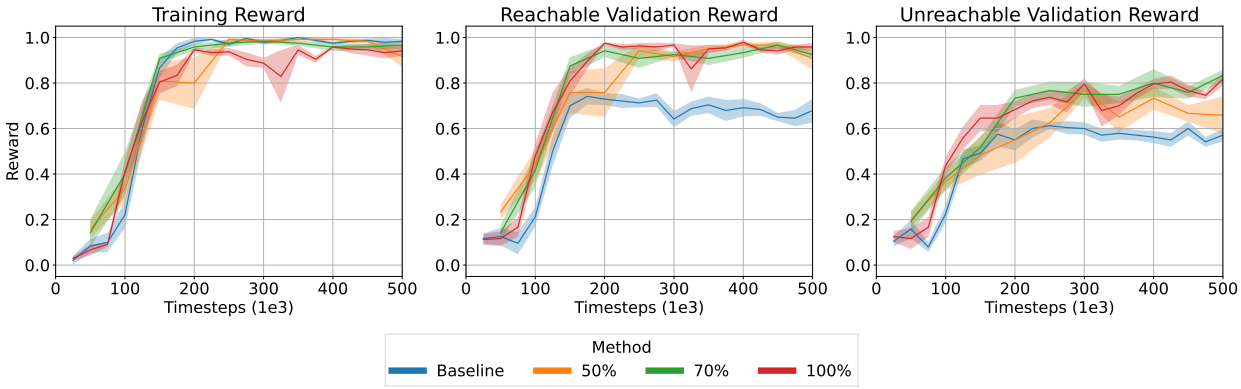

Figure A.5: Comparison of different teleportation chances on the performance of the agent, using a buffer size of 50,000.

and 40 in the unreachable set. After that, the stability of the agents appears to deteriorate again, as the number of steps the agent has left over to reach the goal is reduced too much. Further, a higher value also decreases the sample efficiency of the agent. This showcases that the parameter should be carefully selected and heavily depends on the chosen environment, especially the maximum number of steps allowed. 30 has been chosen for further experiments.

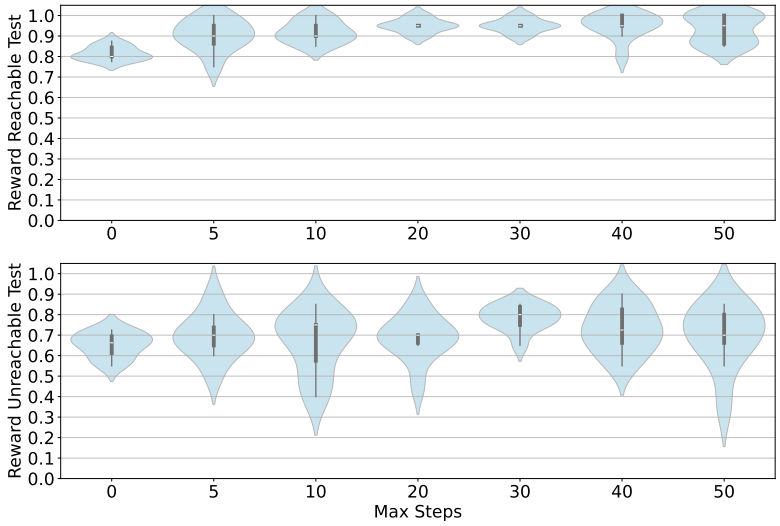

Figure A.6: Comparisons of lengths of the Pure Exploration phase using ε-greedy and a buffer size of 50,000.

A.3.5 GoExploit

As the GoExploit method is the most intricate proposed in this thesis it also has the most experiments regarding its qualities.

Effect of MaxStep

The first one was a simple investigation into the optimal value of allowed maximum steps of the goal-conditioned agent (A.7). The value is of even higher significance in this method than for the Pure Exploration, as it controls not only the duration available to reach the interim goal but also the number of transitions fed to the secondary agent. However, the actual impact seems to be nonconclusive. The performance reaches a soft peak at a value of $30-40$ in all test categories. After that, the performance decreases again, as the main agent likely has too few actions left to learn and generalize properly. 30 has been chosen as the standard value for other experiments.

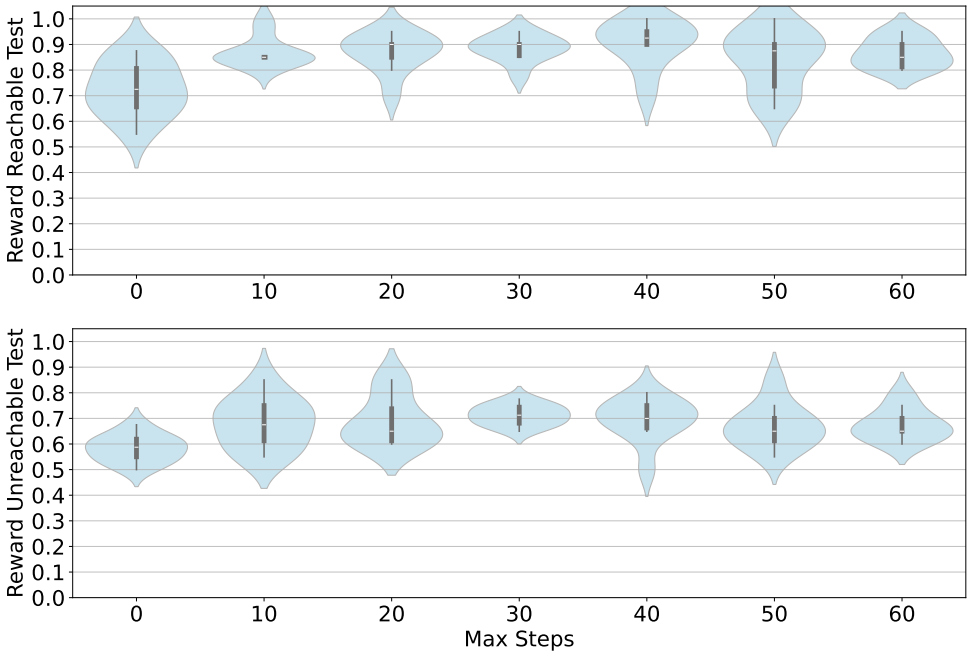

Figure A.7: Comparison of different amounts of steps available to the goal-conditioned agent of GoExploit using a buffer size of 50,000.

Effect of longer training time

Another experiment was done to discern whether the agent suffers from too little training time. This could be caused by the split of replay buffers. As both agents have their own replay buffer, it could be that they need longer in total to reach their full potential. To show that this is not the case, an experiment was carried out in which the results were compared against an agent that had double the time steps (1,000,000) (Fig. A.8). The experiment results indicate that 500,000 time steps sufficed to reach the agent's full potential.

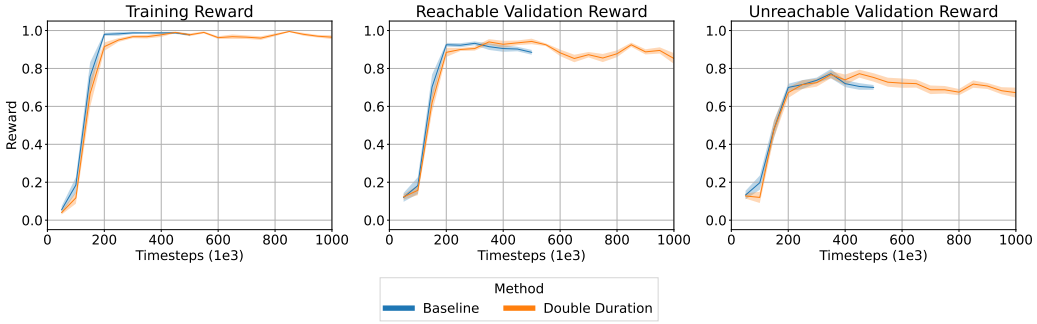

Figure A.8: Comparison of different training durations, using a buffer size of 50,000.

Architecture Choice

The last experiment was done to verify which architecture to choose for the goal-conditioned agent. However, as this is a highly environment-dependent choice, the implications of this choice do not extend further than the FourRoom Environment. We compared both stacking observations on top of each other, as well as utilizing a hypernetwork (Fig. A.9), and came to the conclusion that the hypernetwork architecture achieved slightly better results. Therefore, hypernetwork was chosen as the architecture for further experiments.

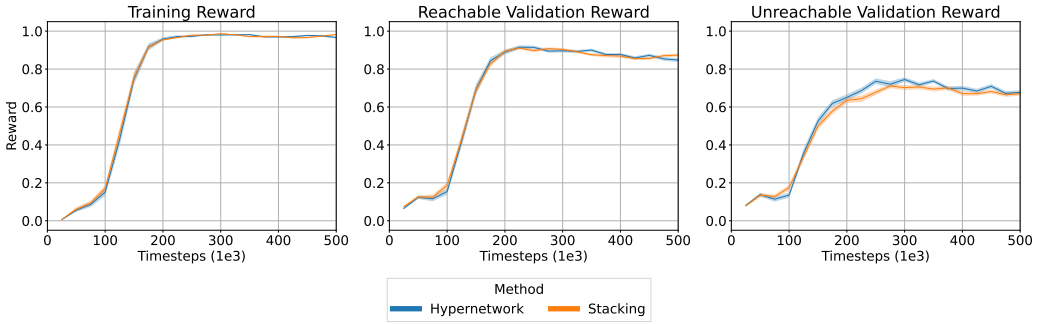

Figure A.9: Comparison of using a hypernetwork or stacking observations for the goal conditioned agent of the GoExploit method, both using a buffer size of 50,000.