Using Machine Learning Classifiers as Shields in Game Environments to Enable Safer Exploration while training with Reinforcement Learning Algorithms

Paul Burrow-Newton

MSc in Artificial Intelligence

The University of Bath

2025

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Using Machine Learning Classifiers as Shields in Game Environments to Enable Safer Exploration while training with Reinforcement Learning Algorithms

Submitted by: Paul Burrow-Newton

Copyright

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the author unless otherwise specified below, in accordance with the University of Bath's policy on intellectual property (see

https://www.bath.ac.uk/publications/university-ordinances/attachments/Ordinances_1_October_2020.pdf).

This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of MSc in Artificial Intelligence in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

This project investigates the use of logical prediction and machine learning classifiers on the safety of agents acting under the control of reinforcement learning algorithms in simulated environments.

The focus of reinforcement learning is often the training of an agent to navigate an unknown environment. Due to the expense of this being done in a real-world environment, simulations and games are often used. Shields can be used as a safety mechanism to make these agents perform exploration in a safer manner by restricting the actions the agent may take. They are largely created either by using a priori information or making an off-line model of the environment. While these methods are effective, they require either a large amount of information about the environment, or a two-stage training in the case of model-based shielding.

The current work aims to overcome these issues by creating a dynamic online shield that will learn about the hazards as the agent explores and updates the shield each time a new hazard is encountered in real-time, without the need to undertake multiple training runs. This is tested on two game environments: Frozenlake and Pacman using tabular and deep learning methods respectively.

In the Frozenlake environment using a q-learning algorithm, a predictive shield, which works to simulate the transition dynamics of the environment, performs best but only in a deterministic environment. When faced with stochasticity the performance of the shield significantly declined.

In the Pacman environment using a double deep q-learning algorithm, a neural network based shield performed best, as this allowed complex relationships between variables and associated correlations to be learnt. This allowed the agent to safely navigate the environment with a relatively high accuracy in most cases.

The promising results in this work could lead towards the development of shields for use in real world environments.

Research questions

Hypotheses

The current research focuses on demonstrating the utility of dynamic shielding within game environments and is split into the following hypotheses:

H1: An agent with a dynamic shield will give better results than an unshielded agent. Better results meaning a higher reward per episode on average than an unshielded agent.

H2: A dynamic shield will result in fewer unsafe states being visited when exploring an environment than without a shield.

Contents

Chapter 1: Literature Review 1
An Overview of Reinforcement learning 1
Main Algorithms of Reinforcement Learning 3
Safety in Reinforcement Learning 5
Changing the Reward Function 6
Adding a Second Cost Function 7
Shielding in Reinforcement Learning 7
Overview 7
Model-Based Shielding 8
Model-Free Shields 12
Adaptive Shielding 15
Discussion 15
Chapter 2: Research Methodology 17
Environments 17
Frozenlake 17
Pacman Environment 18
RL Algorithms 19
Tabular RL Methods in the Frozenlake Environment 19
Deep Q-Learning and Double Deep Q-Learning for the Pacman Environment 22
Existing Shielding Techniques for Comparison 24
Catastrophic Avoidance Shield 24
Intrinsic Punishment 24
Comparison of Baseline Shields in the Frozenlake Environment 24
Shielding Methods to be Applied 26
Learning a Predictive Shield with Online Shielding 26
Dynamic Shielding 26
Tabular Shield Methodology 27
Deep Learning Methodologies 27
Chapter 3: Experiments 31
Tabular Shield Experiments – Frozenlake Environment 31
Deep Learning Experiments – Pacman Environment 31
Chapter 4: Results and Discussion 32
Tabular Shield Results – Frozenlake Environment 32

Standard 8x8 Frozenlake map 32
Standard 8x8 Frozenlake map with slippery attribute 32
Extended 10x10 Frozenlake Map 33
Deep Learning Shield Results – Pacman Environment 34
Simple Pacman map 34
Advanced Pacman Map 35
Discussion 35
Tabular Shields – Frozenlake Environment 35
Deep Learning Shields – Pacman Environment 37
Chapter 5: Conclusions and Future Work 39
Conclusion 39
Future Work 39
References 40
Appendix A: Compute Environment 46
Appendix B: Implementation Scripts 46
Appendix C: Frozenlake Additional Data 46

List of Figures

Figure 1: Example of a Markov chain showing states (represented by circles) and probability of transitions (represented by arrows).
Figure 2: Results of the median scores for different DQN variants (Taken from Hessel., 2018).
Figure 3: Shield data flow (Taken from Bloem et al., 2015).
Figure 4: Workflow of the Shield Construction (Taken from Jansen et al., 2020).
Figure 5: ADVICE Shield training (Bethell et al., 2024).
Figure 6: Frozenlake environment for the (a) 8x8 map and (b) 10x10 map (Brockman et al., 2016).
Figure 7: Simplified Pacman Map.
Figure 8: Advanced Pacman Map.
Figure 9: Q-learning pseudocode (Sutton and Barto., 1998).
Figure 10: Double Q-learning pseudocode (Hasselt, H., 2010).
Figure 11: Dyna-Q pseudocode (Sutton and Barto., 1998).
Figure 12: Results of Q-learning and variants on Frozenlake environment.
Figure 13: DQN and DDQN rewards and safety violations in the simplified Pacman environment.
Figure 14: Rewards for several shielding methods (a) without and (b) with the slippery attribute turned on.
Figure 15: Flowchart of data flow in dynamic shielding.
Figure 16: Rewards received in an 8x8 Frozenlake environment (10 run average) for (a) all episodes and (b) a close-up for the first 650 episodes.
Figure 17: Rewards received in a slippery 8x8 Frozenlake environment (10 run average).
Figure 18: Rewards received in a 10x10 Frozenlake environment (10 run average) for (a) all episodes and (b) a close-up for the first 650 episodes.
Figure 19: Reward and cumulative safety violations on the Simple Pacman map.
Figure 20: Reward and cumulative safety violations on the Advanced Pacman map.

List of Tables

Table 1: Comparison of DQN and DDQN safety violations in the simplified Pacman environment.
Table 2: Multinomial Naïve Bayes accuracy and recall for different under sampling ratios.
Table 3: Gaussian Naïve Bayes accuracy and recall for different under sampling ratios.
Table 4: Multinomial Naïve Bayes grid search highlights.
Table 5: Gaussian Naïve grid search highlights.
Table 6: Neural Network grid search highlights (hyperparameters used highlighted in bold).
Table 7: Results for the standard 8x8 Frozenlake environment.
Table 8: Results for a slippery 8x8 Frozenlake environment (10 run average).
Table 9: Results for a 10x10 Frozenlake environment (10 run average).
Table 10: Top five lowest safety violation results in a 10x10 Frozenlake environment.
Table 11: Safety violations on the Simple Pacman Map for the Naïve Bayes and Neural Network Shields.
Table 12: Episode Wins on the Simple Pacman Map for the Naïve Bayes and Neural Network Shields.
able 13: Safety violations on the Advanced Pacman Map for the Naïve Bayes and Neural Network Shields.
Table 14: Episode Wins on the Advanced Pacman Map for the Naïve Bayes and Neural Network Shields.

List of Equations

Equation 1: Bellman equation (Bellman, 1966).
Equation 2: Loss function of DQN (Mnih at al., 2013).
Equation 3: Double DQN secondary network update (Hasselt et al., 2016).
Equation 4: Q-learning state-action pair value update function (Sutton and Barto., 1998).
Equation 5: Bayes theorem (Stuart & Ord., 1994).
Equation 6: Recall Equation.
Equation 7: Binary Cross Entropy (Goodfellow et al., 2016).
Equation 8: Sigmoid Function (Goodfellow et al., 2016).

Chapter 1: Literature Review

This research considers finding safer ways for an agent to explore in a Reinforcement Learning (RL) environment by utilising a dynamic shield. The current chapter provides an overview of the foundations and history of RL followed by the detailed consideration of modern algorithms that have advanced the field in a significant way.

Safety methods in RL, i.e. the use of various constructs to aid the learning algorithm, is then introduced, followed by an in-depth discussion of shielding. Special attention is given to current forms of shielding and dynamic shielding to give an overview of the current progress in this sub-field of safety within RL. The aim of this chapter is to summarise the current state of the art in the RL field and highlight the potential gaps in the current research related to dynamic shielding.

An Overview of Reinforcement learning

"The more we study the information-processing aspects of the mind, the more perplexed and impressed we become. It will be a very long time before we understand these processes sufficiently to reproduce them." (Bellman, 1966). Almost 60 years later, this quote still resonates. Although there have been significant advances in AI, an AI with human level intelligence (Artificial general intelligence) has not yet been achieved (Bubeck et al., 2023).

The quote is by Richard Bellman, whose equation (given in Eq. 1) is at the heart of many modern RL techniques. The goal of RL is for an agent to learn a policy that will maximise the expected reward when implemented (Sutton, 2018). The policy selects the action that the agent takes for each state in the environment.

These problems are generally modelled as Markov Decision Processes (MPDs) (Puterman, 2014), which are a popular modelling process for decision making under uncertainty (White, 1985). The Markov decision process was built on the idea of Markov chains by Andrey Markov (Meyn et al., 2012). The principle is that the transition to the next state only depends on the decision made in that state and not in any prior states. This property is called the Markov property and means there is no memory in the chain of state transitions. See Figure 1 for an example of a Markov chain, with each state having possible transitions with probabilities that sum to one.

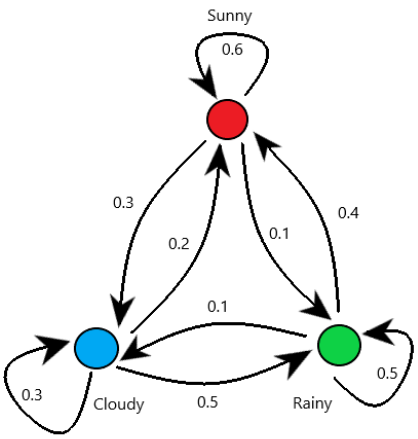


Figure 1: Example of a Markov chain showing states (represented by circles) and probability of transitions (represented by arrows).

The first practical implementation of RL was implemented using dynamic programming using the Bellman equation (Bellman, 1966) by Watkins (1989) in the form of Q-learning. The Bellman equation,

$$V(x_t) = \max_{a_t}\{F(x_t a_t) + \beta V(T(x_t a_t))\}$$

Equation 1: Bellman equation (Bellman, 1966).

defines the value of a state, x_t, where the subscript, t, denotes the timestep. The value of the state is calculated as the maximum reward, F, when action, a_t, in the current state, x_t, is taken, plus the discounted future value, V, of the value of the new state $T(x_t, a_t)$ modified by a discount factor, β.

As the value of each state includes a discounted value of the best possible future state, the equation is recursive. This is a dynamic programming method that uses Bellman's principle of optimality defined as follows: "An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision" (Bellman, 1957).

This has been improved upon by several methods including Dyna-Q (Sutton and Barto, 1998), which learns a model of the environment and then simulates experiences to reach an optimal policy in fewer iterations and double Q-learning (Hasselt, 2010) which uses two q-tables to limit the overestimation of the state-action value, which stems from the max function in the Bellman equation (Eq. 1). Q-learning, Dyna-Q and double Q-learning are all tabular methods, which means that a value must be stored for every action, in every state. While effective in smaller settings (i.e. those with a few hundred thousand state-action pairs), this is unsuitable in large, or continuous environments; as is also the case for other methods such as brute force or the Monte Carlo approach.

Independent to RL, neural networks were developed and deep learning came into the forefront of AI research (LeCun, et al., 2015). Instead of tabular methods, which are limited by space constraints due to the size of the q-table, a deep neural network can generalise and learn abstractions from data.

Mnih et al. (2015) used deep learning in conjunction with RL to successfully learn policies play Atari video games (in some cases at a human level or higher). They recognised that the ability of deep neural networks to learn concepts and object abstractions from visual data was suited for the task of learning the appropriate action for each state. It was established that a convolutional neural network (CNN) (LeCun et al.,1998) was the best method available at the time for visual analysis.

In 2016 RL had another major success when an RL agent AlphaGo beat a world champion Go player (BBC, 2016). This used Monte Carlo tree search in conjunction with a neural network. In addition, as Go is a symmetric game (any piece can be moved in any direction), data augmentation was used. This meant that each state could be used eight times when flipped and rotated to provide more training data. Google DeepMind was also able to beat the most powerful chess engine Stockfish, with just nine hours of training with a new more advanced version, AlphaZero (Silver et al., 2018).

Dota 2, a team-based strategy game, provided the next challenge for the RL community, with attempts made to learn the game. This started with a single agent created by OpenAI defeating one of the best players in a one vs. one game (Signal Processing Society., 2017) and cumulated with a multi agent team called OpenAI Five defeating the current best Dota 2 professional 5-person team in the world (Berner et al., 2019). They did this using the proximal policy optimisation (PPO) (Schulman et al., 2017) algorithm utilising a large network of network 256 GPUs and128,000 CPUs. A Long Short Term Memory (LSTM) (Hochreiter, 1997) was used for each character to learn long term strategies, as each character has different abilities and will contribute to the team's performance in a specific way.

Following the success in Dota 2, Google DeepMind turned to the computer game StarCraft 2, a real-time strategy game, as the next challenge for RL developing an AI under the name AlphaStar (Vinyals et al., 2019). AlphaStar used self-play whereby multiple agents play against each other (similar to the method used by

OpenAI Five) and human vs. human games to obtain a high level of play. The human vs. human games were learnt from initially and imitated to narrow down the vast array of moves that can be taken. The limiting of the available moves demonstrates that RL has limitations, as if the exploration space is so vast, even with massive computational resources, a reward signal may never be found.

Methods such as human imitation in real-world applications such as autonomous driving (Lu et al., 2023) have been developed to take advantage of the wealth of knowledge that can be gained through from humans. Applying this principle to RL for gaming proved to be very effective, with the AlphaStar AI achieving grandmaster status in StarCraft 2 (Vinyals et al., 2019) noting that Grandmaster status has been granted to only 733 people as of 2024, out of the six million copies of the game sold, showing the prestige of the title.

RL has also proven effective in poker (Brown and Sandholm., 2019) where an AI developed by Meta won an average of $1,000 per hour against five professional poker players; this was significantly better than the humans over 10,000 hands. This AI was so effective that the source code was not released, for fear it would be used be online players to cheat.

Outside of the gaming world, RL has recently been used to improve the performance of a Large Language Model (LLM). The application of RL from Human Feedback (RLHF) starts with a pre-trained LLM which is able to form sentences but is then fine-tuned as follows. A human ranks answers from prompts from one or more LLMs. This is used to create a reward model, which is an LLM designed to send a ranking signal to the original model. This is trained by the comparison dataset, so the best and most relevant answers give the highest reward. This reward model is combined with the original LLM that can take the reward signal as an input to adjust its policy. This makes an LLM that will give useful answers and react correctly to specific prompts such as “write an email that…”.

OpenAI found that a model with 1.3 billion parameters outperformed a model with 175 billion parameters after RLHF was applied (Ouyang et al., 2022). RLHF can also make the model safer, as impolite and rude response can be ranked lower and therefore filtered out in the final model.

Main Algorithms of Reinforcement Learning

There are many different modern algorithms used for RL. Some have had more impact on the community than others. Below are the ones that are most used today and that have significantly contributed to the field of RL.

Deep Q-network (DQN)

The Deep Q-network (Mnih et al., 2013) was created to take advantage of recent advances in training deep neural networks in the RL domain to play 49 different Atari games (Mnih et al., 2015). The network used a CNN to process the pixel input from Atari games. Using this hierarchical system of tiled convolutional filters (described above) mimics the effect of visual receptor fields in animals. This was inspired by the work on the feedforward process in the visual cortex of cats (Hubel and Wiesel., 1963). This was then fed through a neural network to estimate the action values at that state.

They sought to overcome the training instability of other RL methods. This was done firstly by using experience replay (Lin, 1992) that captures an experience, $e_t = (s_t, a_t, r_r, s_{t+1})$ (i.e. state, action, reward and next state) at each timestep. These are stored in the replay buffer $D_t = \{e_1, e_2, \ldots e_t\}$. Sampling from the replay buffer randomises the data, which removes correlations in observed sequences and smoothing over changes in the distribution of the data. The smoothing over the distribution is especially important, as this can drastically change as the agent learns. Sampling from the older experiences ensures that the network does not forget these and override their knowledge with that of new experiences. Minibatches of samples are drawn uniformly from the replay buffer. New methods such as Prioritised Experience Replay (Schaul et al., 2015) have sought to improve on this by taking samples from the experiences with the lowest predicted action values and are

therefore important to learn from (in the hope that the predicted values will be closer to the correct ones when next sampled).

The DQN loss function, $L_i(\theta_i)$, is described as:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$$

Equation 2: Loss function of DQN (Mnih at al., 2013).

where an update is performed at each iteration, *i*. E is the expected value, *r* is the reward given, *γ* is the discount factor, θ_i are the parameters of the Q-network at iteration *i* (online network) and θ_i^- is the network parameters used to compute the target at iteration *i*. The target network is only updated every pre-defined number of steps and not changed between these updates. This periodic updating reduces correlation with the target.

The DQN algorithm scored more than 75% of a professional human score across on 29 out of the 49 Atari games it was tested on and outperformed the best existing RL algorithms on 43 of these. Additionally, there was no a priori knowledge incorporated into this technique, unlike other methods that used hand crafted reward functions specific to certain games. This algorithm became the standard in the field and has been improved upon many times since (e.g. Hessel., 2018), but the core idea of using a CNN as the input, minibatches and a replay buffer to store experiences are common among all of them.

Double Deep Q-network (DDQN)

One of the first improvements made to the DQN algorithm was the Double-DQN (DDQN) network (Hasselt et al., 2016). Building on the Double Q-learning algorithm (Hasselt, 2010), this work sought to utilise the double network architecture that stop overestimation in a deep learning environment. This is done, not by adding an extra network, but by modifying the value function. The DQN (described in Eq. 2) already has two networks, the target network and the online network. The weights of the online network (θ_i) are replaced with those of the target network (θ_i^-) for the analysis of the current policy, with the target network update formula remaining unchanged from that of DQN, as follows:

$$Y_t^{DoubleDQN} \equiv R_{t+1} + \gamma Q\left(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \theta_t), \theta_t^-\right)$$

Equation 3: Double DQN secondary network update (Hasselt et al., 2016).

The above change is very small, with no extra computation, but gives the benefit of Double Q-learning. In testing the DDQN, Hasselt (2016) kept the rest of the algorithm and methodology the same to enable a fair comparison of the methods. This led to an improved mean score of 330.3% for DDQN vs. 241% for DQN, showing that the overestimation is in fact causing poorer policies to be produced.

DDQN, prioritised replay and four other improvements were combined in the Rainbow method (Hessel., 2018), which showed that different improvements could be combined to create a more effective architecture. As shown below in Figure 2 this combination of methods was able to match DQNs best score with 7 million frames and surpass all other DQN variants by 44 million training frames. This paper is very motivating, as it shows that while the core of an algorithm can be very effective, it possible to build on the foundation algorithm with many modifications to create incremental improvements for years.

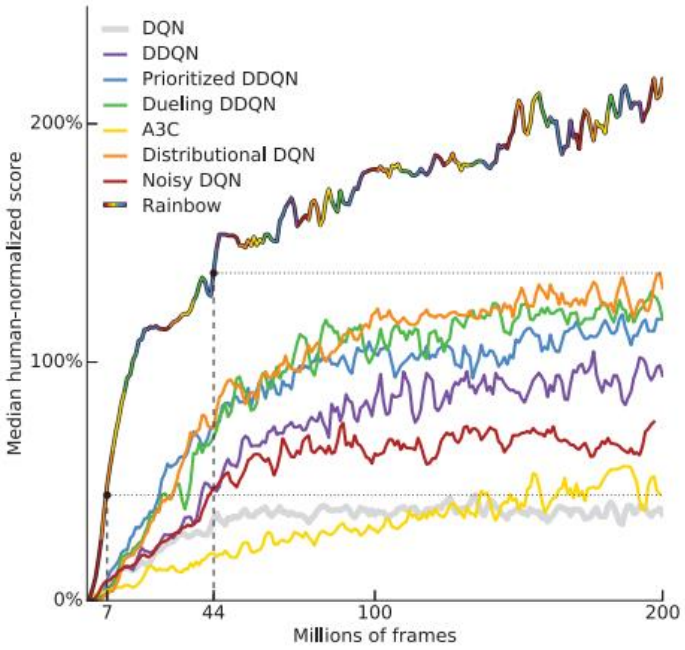


Figure 2: Results of the median scores for different DQN variants (Taken from Hessel., 2018).

Trust Region Policy Optimisation (TRPO)

A year after DQN was released, a team of researchers created an algorithm called Trust region policy optimisation which is a policy gradient method (Schulman, 2015). It avoids excessively updating the network parameters with a Kullback-Leibler divergence constraint (Kullback et al., 1951). The KL divergence is a measure of how much a models probability distribution differs from the true probability distribution.

This is based around maximising the advantage rather than state-action pair values, where the advantage is the difference between the value of the action taken and the value of the on-policy action. Exploration is done via the stochastic on-policy method of action choice. The actions typically become less random as the policy leans towards exploiting rewards that have previously been experienced, however, it can also cause the algorithm to get trapped in a local optima. This algorithm has been demonstrated to work very well in control tasks, including robotics (Sunilkumar et al., 2024).

Proximal Policy Optimisation (PPO)

Proximal policy optimisation (Schulman et al., 2017) is an effective simplification of TRPO. Instead of using the KL divergence to constrain the policy update, clipping is used, which would simply clip the update to be a maximum of 0.8-1.2 of the old values (if an epsilon of 0.2 was used, as was shown to be effective in Schulman et al. (2017)). The minimum in the function ensures a minimal safe update, to keep the policy stable. This algorithm has been used to perform a number of impressive tasks such as playing Dota 2 (Berner et al., 2019) as discussed above and is one of the main algorithms used in RL research today.

Safety in Reinforcement Learning

While learning the optimal, or near optimal policy, exploration must be done. This is to ensure that every state-action pair is visited many times, so that their values can be accurately calculated. The standard RL algorithms such as q-learning make the ergodicity assumption. This is that any state is eventually reachable from another state. This is true in most simulated cases, as if a terminal state is reached, the episode ends and will start again, so any other state can be reached. The issue of safe exploration has been labelled as one of the five major issues in AI Safety (Amodei et al., 2016) and so is an important issue for further research.

In real life situations, however, this is not the case. For example, the Curiosity Mars rover (Lockwood, 2006) would not be able to restart if it reached a terminal state (falling over or getting trapped in a crater with steep sides, for example). This gives rise for the need for safe exploration. This was explored by Moldovan and Abbeel (2012), who used an optimization formulation using constraints to restrict the action space to actions which preserve ergodicity. As there is an element of stochasticity in most MPDs, the policies were deemed safe if the safety level was above a safety probability. Simulated exploration on Mars was effectively done using safe policies generated by this method.

Safety in RL can also refer to the safety of states visited by the agent in isolation (rather than their ability to transition to other states). This is useful in the case of simulations, where no state would be deemed unsafe by the previous definition, due to the ergodicity of all states. A state is defined as either safe or unsafe by some criteria. These could be hand labelled (e.g. Saunders, 2017) or labelled automatically by using a set of logical criteria (Könighofer et al., 2023). Safety in RL can also refer to the probability of reaching a large negative reward (Thomas, 2015) as would be the case if an unsafe space were reached.

Various methods have been used to research the topic of safe exploration. These include human oversight, where a human is utilised for checking or labelling actions and states (Saunders, 2017), but this approach is subject to scalability problems, as well as timing issues, as a human operator may not be fast enough to identify a safe action. Simulated exploration is common in robotics, as it is not only safer, but much cheaper to simulate environments rather than test in real life. There is a simulated to real gap, however, that means models trained in simulations have tended to perform poorly in real life (Wagenmaker et al., 2024). There has been research on how to address this problem such as RL CycleGAN (Rao et al., 2020), using scene consistency loss, which offered good performance in reality, based on simulations from a small number of images of the scene to be trained on.

On the other side of safety, there is the concept of risk. This has term has traditionally been used to mean the variability of returns from a policy (Coraluppi & Marcus.,1999). This term has also been used as the likelihood of reaching fatal or undesirable states (Geibel, 2001). Geibel (2005) used this likelihood of risk (probability of entering an error state) as a secondary value function to constrain the actions of an agent to create a model-free online algorithm which performed well, treating the MPD as a constrained MPD.

Standard exploration algorithms such as e-greedy (Sutton, 2018) and r-max (Brafman and Tennenholtz., 2002) do not take safety into account when choosing actions to be taken, so it is highly likely many unsafe states will be visited during exploration.

There are three main methods of enforcing safety in RL (Carr et al., 2023):

1. Changing the reward function (“engineering”)
2. Adding a second cost function (“constraining”)
3. Blocking unsafe actions (“shielding”)

The focus of this work is shielding (covered in the next Section), but for a complete picture of safety in RL, the first two points of the above list are briefly covered here.

Changing the Reward Function

The standard Q-learning algorithm has been built upon to allow safer exploration. The Hamilton-Jacobi reachability safety analysis (Mitchell et al., 2005) was utilised by Fisac et al. (2019) to modify the Q-learning algorithm with the aim of creating a safer version. Instead of maximising the expected future reward, they sought to maximise the minimum reward (have the best worst-case scenario). This is done by specifying a set of failure states (which can be in the form of constraints, e.g. a leg of a robotic agent is not going below a certain height). A priori knowledge of the unsafe states is needed for this to work, meaning detailed environmental

knowledge may be needed if this was to be applied to exploration problems. The experiments of Fisac et al. (2019) revolved around control problems, as this method is based in the field of robotics. This meant the safety constraints involved the position of the agent or the agents' limbs in relation to the environment (e.g. the half-cheetah was constrained by not being able to touch the ground with its head). The value function they created calculates the minimum payoff if the action is applied at each timestep. Their Safety Q-learning algorithm has convergence guarantees and when tested, shows the number of violations decreases as learning episodes increases. They do not give comparisons to any baseline or other methods however, making it hard to quantify if it is performing well. Additionally, the authors state that the method will ensure the agent acts safely, but due to the maximisation of the minimum payoff, will not seek a high reward, which in turn means the agent will not perform another task while staying safe. This gives the method limited practical application but does give insight into an alternative way of thinking about safety.

Adding a Second Cost Function

Bounded exploration is another method of ensuring safety. That is, constraining the agent to an area that is defined as safe. This could be for instance keeping an unmanned drone above a certain height within a specified area, to ensure that there are no buildings to collide with at that height. This uses the ergodicity definition of safety (Moldovan and Abbeel., 2012), as every safe state can visited via any other. The success of this method, however, is limited, as exploring environments containing hazards will be inherently dangerous (e.g. avoiding buildings does not guarantee not colliding with birds), so absolute safety guarantees cannot always be made.

Constraining by adding a second cost function, which will constrain the action of the agent is not to be confused with the similarly named Constrained MPD (Altman, E., 1999). The CMPD also involves a second cost function, but can only by solved by linear programming, not dynamic programming. CMPDs have multiple costs on each action, which causes the constraint. This is unlike the bounds of safety for the agent by Moldovan and Abbeel (2012).

Adding a second cost function has been utilised in areas such as finance (Tamar et al., 2012), where policy gradient methods were used to predict the trajectory variants of a bond portfolio. Using these to determine different policies resulted in a low variance (low risk) portfolio when the Sharpe ratio (Sharpe, 1966) was used as a constraint. The Sharp ration being the expected value of asset return less the risk-free return over the standard deviation of the asset excess return. When maximising profit (reward) was the objective, the resulting risk was much higher, showing that constraint-based methods do effectively result in safer policies.

Shielding in Reinforcement Learning

Overview

The final category of safety mechanics in RL is shielding, the focus of the current work. Shields have been used by soldiers, and more recently police forces for centuries to protect themselves against projectile and melee weapons. These have been made from a variety of materials including, wood, leather, metal, and plastic. The earliest known shield was found in Germany, dating back to the late Neolithic period (Vencl, 1979), approximately 3400 - 2800 BC.

However, shielding as a named concept in programming only dates back to 2015 (Bloem et al., 2015). In this work, a shield was created for use in a traffic light system to modify the output if it did not fit a specified set of critical properties, as shown in Figure 3. The shield was designed to only interfere in the minimum possible way, so it would only interact if necessary and the deviation from the original output was kept to a minimum. This ensured correctness in the system by ensuring all the critical properties are always satisfied.

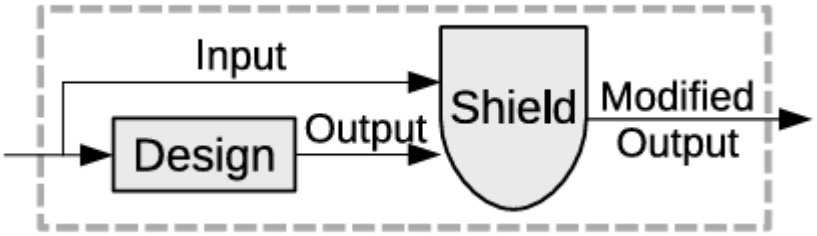


Figure 3: Shield data flow (Taken from Bloem et al., 2015).

In general, the set of critical properties that would need to be met, even in a complex system, tends to be small and easy to specify meaning that this type of shield is scalable to much larger systems. This method of specifying critical properties to comply with safety criteria using logic has been achieved in various ways (e.g. (Carr et al., 2023)) since the seminal work of Bloem et al. (2015), showing that not only was the shield a significant idea, but the execution using logic was well thought out.

Shielding in RL refers to the use of a shield to act as a filter (Alshiekh et al., 2018). This filter allows actions that the shield identifies as safe and will block actions that are unsafe. This allows the agent to move around the environment in a safer manner, in some cases avoiding unsafe actions altogether.

There are three safety levels of shields in RL (Brunke et al., 2022):

- **Soft Constraints - Safety Level 1:** These are constraints with which the agent is encouraged to comply. There are no safety guarantees.
- **Probabilistic Constraints - Safety Level 2**: These shields give safety up to a maximum probability. This is needed in stochastic environments where the effects of the agent on the environment are not purely deterministic. Reaching the goal state cannot always be done with a one hundred percent guarantee of safety.
- **Hard Constraints - Safety Level 3:** These are constraints that the agent must comply with at all times. This is the highest level of safety and will ensure the agent will never be harmed.

Model-Based Shielding

The principle of model-based shielding was first introduced in RL in 2018 (Alshiekh et al., 2018). The principles of correctness and minimal inference were applied using temporal logic (Wirsing., 1990). This was tested in two ways, pre-emptive shielding and post-posed shielding. Pre-emptive shielding gives the agent a list of safe actions to choose from after taking the observation from the environment, whereas post-posed shielding takes place after the agent makes the choice. In this case, the agent gives a list of actions in the preferred order. The shield will then choose the highest valued action choice that is available. This has the advantage of speeding up learning, as in the case where three actions are ranked, the agent receives feedback for each unsafe action until a safe action is found (which will also receive feedback). The post-posed shielding was found to be the most effective and is the style of shielding that is used in other shielding literature. The correctness of the system (ensuring all constraints are always maintained) gives this shield a safety level of three.

Probabilistic Logic Shields

In Jansen et al. (2020) an offline shield built with probabilistic temporal logic constraints (Baier and Katoen., 2008) was used to limit the actions that would reach an unsafe state in the game Pacman. Using this method, an offline shield is created before the training of the agent begins and is run while the agent is exploring the environment.

This was achieved in several stages, as outlined in Figure 4. Firstly, adversaries (ghosts) are observed in one training phase, to capture their behaviour to create a behavioural model; this is done on small arenas to speed up the training. Secondly, training is undertaken in the main arena to capture the dynamics of the arena. The output of these stages is then used to train a shield offline; in the set-up used by Jansen et al. (2020) training for the full Pacman game took roughly six hours.

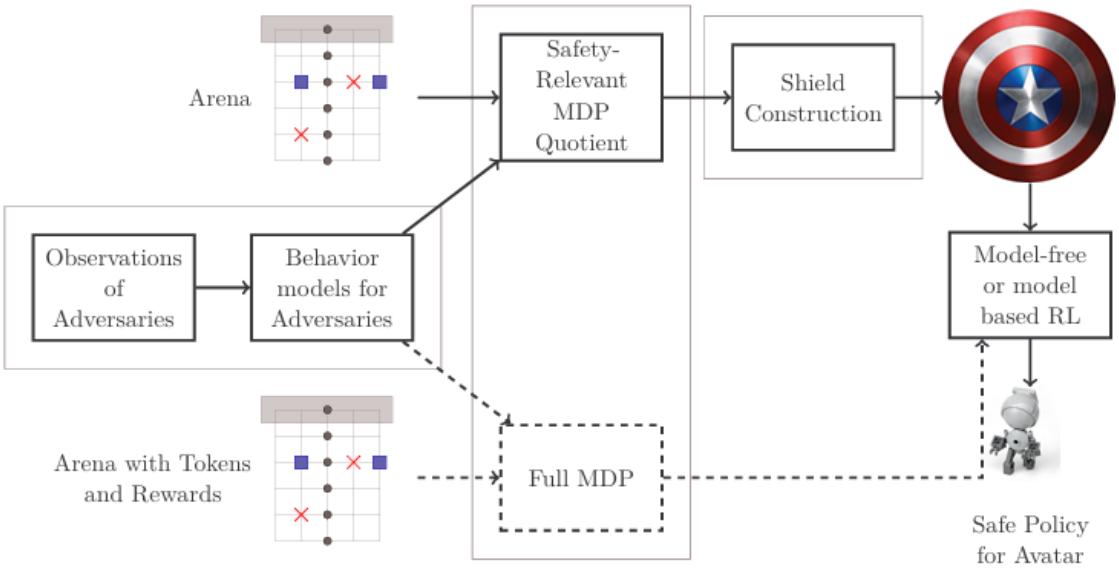


Figure 4: Workflow of the Shield Construction (Taken from Jansen et al., 2020).

After the creation of the shield, the training to create the policy is started. During this, some optimisations were used to speed up the process, such as only using the shield in some critical states where shielding is needed. In addition, the shield is adaptive which means it is iteratively weakened if no progress is being made by the agent (e.g. if the agent is trapped in a loop). This allows additional, potentially unsafe actions to be made with the aim of allowing progress.

Overall, Jansen et al. (2020) found that higher scores and win rates are obtained by the shielded agent across all the benchmarks and that the use of the shield sped up learning of the optimal policy, as fewer episodes are needed in the training process. However, while this is true in an isolated sense, this needs to be considered in balance with the time taken to train the shield offline.

In addition, the importance of shielding versus freedom is noted, as a shield which is minimally invasive will give the agent the most freedom to explore the arena. This is balanced by the shield that gives the most freedom is also the one which allows the agent to reach unsafe situations the most.

Könighofer et al. (2023) expanded on this work on probabilistic shields that use temporal logic constraints (Jansen et al., 2020) by modifying the shield to be online rather than offline for a two-player game called Snake. Meaning that safety calculations are done during the running of the software, rather than calculated in advance. This approach is suitable to discrete problems where computation can be done in the time between states and decisions, but is less suited to a fast real-world environment.

While Snake has many corridors and a moving adversary, there are only two safety specifications to be met. First the “collision heads specification” ensures there are no head on collision between the agent and the adversary. Secondly, the “collision bodies specification” ensures that the head of the agent does not collide with the body of either the agent or the adversary. These conditions, expressed by temporal logic are always held globally. The objective of the shield is to ensure unsafe actions are avoided by using a probabilistic model

that checks if any safety violations will occur (above a pre-defined probability threshold) within a number of timesteps, *k*, which is referred to as a *k*-step lookahead.

In addition to the use of an online shield, Könighofer et al. (2023) tested an informed shield. This meant that as well as the shield blocking the action, a negative reward is given whenever an unsafe action is chosen. This was found to speed up learning and lead to a higher average reward in training (compared to using an uninformed or shield). However, when the policies derived from the agents that were shielded during training were tested, it was found that the unshielded agent performed best. It was necessary to shield the other agents during testing for them to slightly surpass the standard agent. This was surprising, as the agent with an informed shield is receiving extra feedback on the negative outcomes, but the end policy ends up worse than without this. This shows that although shielding during training is effective, it must also be used during testing to truly be effective. This suggest that although a policy is created during training with shielding, it is not the optimal policy. This is reinforced by the agent trained without a shield performing slightly better than the agent trained with shielding (when both have a shield during testing). It is perhaps better to train the model without a shield, and then apply it at runtime to achieve the highest return.

Yang et al. (2023a) also created a probabilistic logic shield using two neural networks to calculate the action probabilities (policy network) and the state representation probabilities (state network). The state network output are probabilities of atomic facts and would be represented as {0.8: obstacle left, 0.5: obstacle right, 0.1: obstacle front}. These are fed to the logic program, which is differentiable to allow for backpropagation. This worked in a similar manner to Könighofer et al. (2023), using a lookahead technique (with higher safety levels reached with a higher lookahead). Noisy sensors were introduced to allow for the idea of imperfect information that can happen in real life scenarios.

Bounded Prescience Shielding

A lookahead technique called Bounded Prescience Shielding (BPS) (Giacobbe et al., 2021) assumes black-box model of the environment is available to test actions that are available to be taken. This is available in most simulated RL environments, so is not unreasonable. This takes the future states and labels them as safe or unsafe based on 43 hand crafted properties (across 30 games, so there are only 1 or 2 per game). These are labelled either via the system output from the black-box, or by hand from the pixel output. The shield looks ahead *k*-steps and if a path exists where each state is safe (i.e. it does not violate any of the safety properties), that path is taken. There are two types of commonly used properties in this technique as follows:

- *Shallow properties*: A property is shallow if a safety violation is caused by a recent action, for example, when shooting a gun in the game Assault, would lead to overheating the gun. These violations are at most within 10 frames or less. It is reasonable to assume a human would not intentionally violate these properties, so it is also reasonable to ensure an agent does not either. These properties do not require long term planning to achieve.
- *Minimal Properties*: These properties are required to score at least 10% of the human average. Violating these properties would result in no score or a negative score. These may be longer term, such as missing pins with a thrown bowling ball, which could have happened hundreds of frames beforehand.

This technique has the disadvantage of having a polynomial component to the method. For example, if *n* is the number of actions and *k* is the number of steps in looked ahead, the worst-case scenario will have to check n^k paths. This renders the technique only useful for the shallow properties. However, the technique did raise the satisfaction level for three shallow properties to 100%, so is effective in this reflexive capacity.

Approximate Bounded Prescience Shielding

This was proceeded by the creation of Approximate BPS (He et al., 2021). This technique does not assume a black-box model of the world is available. It only requires a labelling function for each state (to be safe or

unsafe). This requires less a priori information than the hand-crafted safety properties approach that other methods use. Syntactically co-safe Linear Temporal logic (Kupferman and Vardi., 2001) is used to specify the criteria, that when broken, means a state is unsafe. This is a weaker assumption that the world model, and so the algorithm may lose its correctness property (or at least cannot be guaranteed, due to the approximation of safety). Instead of a black-box, a world model is used. This is learnt as the first stage of training, to learn the dynamics, transitions and rewards of the environment. The safety of states is labelled during this stage.

The world model is then integrated into the shield. As opposed to BPS, a fixed number of trajectories is sampled (rather than a maximum of n^k trajectories). This allows the size of the k-step lookahead to be higher. The paths that are more likely to be taken are sampled to get the fixed number, so actions that are unlikely to be taken are not tested. This allows the computation costs to be significantly cut compared to the brute force approach of BPS.

He et al. (2021) also uses the concept of *intrinsic punishment*. This means that when the agent chooses an action that would violate a safety constraint, although the action is blocked by the shield and a safe action chosen, a negative reward is given. This concept of punishment when safety has been violated had been used before this (Alshiekh et al., 2018) and is used to speed up policy convergence. This comes under the category of reward engineering but shows the two safety methods (shielding and engineering) can be used in conjunction to potentially produce better result than when they are used independently.

Approximate Model-Based Shielding

Approximate BPS was built upon by Goodall and Belardinelli (2023a), who utilised Probabilistic Computation Tree Logic (PCTL) to define the safety and reachability properties for the world models they used. This technique was called approximate model-based shielding. They built on the work of He et al. (2021) by using safety critics to bootstrap imagined trajectories, without having to increase the lookahead-horizon (which is computationally expensive). PCTL was used as the safety specification language as the probabilistic aspect of PCTL allows a trade-off between safety and exploration by specifying the probability threshold to which the temporal logic formula is satisfied.

Satisfaction of the temporal logic formula will guarantee safety. While this approach did yield a lower amount of safety violation when trained on two Atari games (8,579 vs. 11,726 [Assault] and 4,889 vs. 15,697 [Seaquest]), the number is still high, and although the violations are reduced substantially, so is the best score achieved on (e.g. 11,400 vs. 7,040 [Seaquest]). This shows that the either the probabilistic safety threshold is too high and cutting off potentially useful and safe actions to lead to a better score, or the leant world model is incorrect, leading to incorrect state transition assumptions. A weakness of Goodall and Belardinelli (2023a) is that it only compares their own shield vs. non-shielded; a useful addition would have been to compare their shield to the baseline latent shield that they built upon from He et al. (2021).

The work of Goodall and Belardinelli (2023a) was further built upon by Goodall and Belardinelli (2023b) who used Approximate Model-Based Shielding (AMBS). In this work, the method was tested against the current state of the art techniques for Atari video games, including Rainbow (Hessel., 2018) and a shielding technique. AMBS outperformed the algorithms in three of the five chosen games, with the least safety violations in four of these. This technique is useful, as only expert labelling of states is needed, rather than the full safety dynamics being given as a priori knowledge. This technique was extended to continuous environments (Goodall and Belardinelli., 2024), where it was also effective. These probabilistic shields all fall under safety level 2, as the constraints are probabilistic.

The use of temporal logic has also been used in partial observability problems (Carr et al., 2023) where the agents are in Partially Observable MDPs (POMDPs). This method employed temporal logic constraints to build a shield using reach-avoid specifications. This requires knowledge of all potential transitions in the model. The

rewards and probabilities of these transitions is not known. This technique, unlike the others described above, eliminates safety violations when the shield is turned on. It also has a much larger degree of a priori knowledge, which should be taken into consideration when comparing the two temporal logic-based methods. Carr et al. (2023) also experimented with bootstrapping. This involves improving the convergence rate by using the shield and then disabling the shield, to enable a better policy to be learned faster. This was found to be most effective using intrinsic punishment during the gradual disabling of the shield (the shield was disabled with increasing probability during each action, with intrinsic punishment used when it was enabled). When the shield was abruptly disabled many more safety violations occurred (2,369 vs. 33). Whether this shows the benefit of gradually turning off the shield is debateable, as the introducing intrinsic punishment changes the reward signal, which is a safety mechanism in itself. While the shield is active, the hard constraints ensure safety for the agent, so this is algorithm is safety level 3.

Another interesting approach to safety using temporal logic was by Hasanbeig et al. (2020) who developed a system for predicting unsafe state-action pairs, to provide a safe padding for the agent. The task specification, and in turn the reward structure, were implemented with linear temporal logic (Pnueli, 1977). This was built upon by a method they called Cautious RL. This used an optimistic learner and a pessimistic leaner to guide the agent, in what we would call pre-shielding, as the pessimistic learner would output a set of potential actions for the optimistic learner to potentially take. This has a mechanism whereby if the uncertainty around a state was high (the next state did not have a high chance of being predictably correct) then the agent would employ a further lookahead for the temporal logic, thus increasing the knowledge of the agent. Doing this only in uncertain times increases the computational efficiency of the algorithm. When experimenting, it was shown that the agent did take a much longer, but safer route when in a slippery grid world, giving clear visualisation of the effect of the pessimistic learner of the agent's behaviour.

Bayesian Methods

Bayesian inference has also been used in safe exploration (Mitta et al., 2024). This technique used a priori information about the safety of states, which is put into Dirichlet distributions (Kotz et al., 2019). These are updated with Bayesian inference each exploratory episode to better learn the transition dynamics of the model. This was combined with confidence bounds to give levels of safety confidence which could be preset by the user. This method does require knowledge of the unsafe states in advance, to ensure they are not reached by the agent during exploration. This a priori information combined with the learnt dynamics is an effective combination. The confidence bounds for the agent to comply with do not guarantee safety, so this model falls under safety level 2.

Robotics has also been a focus for shielding with methods such as statistical model predictive shielding (Bastani et al., 2021) focusing on robot safety. This uses safe equilibrium points as backups to try to transition to in the case that the robot enters the boundary of the "safe region" using a backup policy. This entry of the boundary is checked with a shield using sampling of the simulated model dynamics. If the boundary is predicted to be reached, then the recovery policy will be used, which will guide the robot towards an equilibrium point.

Model-Free Shields

While most of the work in shielding has been based on models of the environment using logic-based specifications or learning a world model before creating a shield, there has been some work into shielding where no a priori information about the environment is used. This is sometimes referred to as a black-box environment. This is challenging, as you must enforce constraints on behaviour in complex environments while the dynamics of the environment are unknown, as are the states where safety violations occur. These safety violations and the dangers that cause them must be learnt through experiences.

Shperberg, Liu and Stone (2022) created an online shield which learns from mistakes. This shield treats the environment as a black-box and learns as the agent explores. Their system uses an extended form of POMDP, notated as POMDP-CA. The CA stands for catastrophic actions and denotes an action in a state which leads to catastrophic results, i.e. a driverless car crashing. These state-action pairs are (s, a) ∈ C. Learning the whole set of C is the objective, as they can then be avoided, and safe exploration will be achieved. This is done by using a tabular method within a discrete environment. This has several limiting issues including the amount of space required to store the results of every un-safe (s, a) pair which restricts the model to small environments. Shperberg, Liu and Stone (2022) also trial a parametric version of this method, which, while it is less precise (as it can result is false positives, so mistakes may be repeated), it can work for continuous environments and uses much less memory.

The shield of Shperberg, Liu and Stone (2022) proved to be effective and results in high returns in the LavaGrid RL environment when compared to other safe-RL based agents included constrained policy optimisation methods (Achiam et al., 2017). Safety violations will occur while the shield is learnt, so this falls under safety level 1. The main disadvantage to this shield is that it suffers from the "cold-start" problem. That is, until the shield is trained by the agent entering all of the unsafe states, it will not be fully effective. This may take some time to do on larger environments.

Recent work by Bethell et al. (2024) also assumes a black-box environment. Supervised contrastive learning (Hadsell et al., 2006) is used to create a contrastive autoencoder. Contrastive learning is a process with the aim of distinguishing between similar and dissimilar pairs of data. A contrastive loss function is used to put the points in one category close to each other (by Euclidean distance or cosine similarity) and dissimilar points at a further distance. Initially no shield is used, and the agent acts unshielded to gather data to train the shield with the contrastive autoencoder (named ADVICE), as shown in Figure 5.

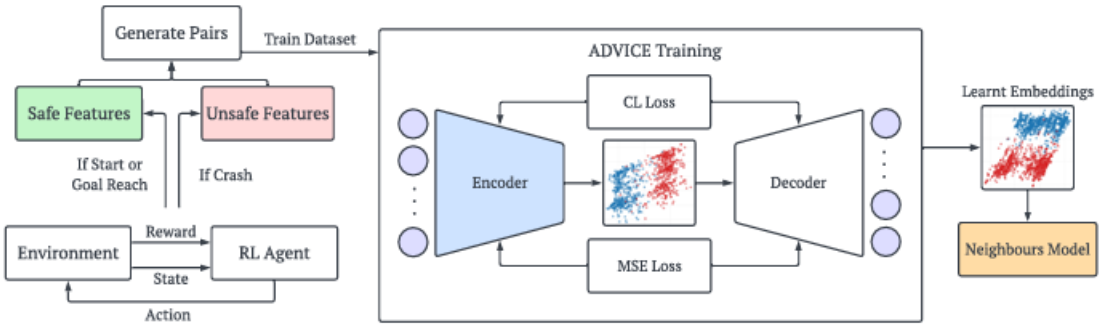


Figure 5: ADVICE Shield training (Bethell et al., 2024).

New points are then classified using the autoencoder and classifier. The autoencoder will encode the state and use the k-nearest neighbours (Fix, 1985) to check k-nearest neighbours' classifications. If the number of safe neighbours is above a given threshold, given as the safety threshold, then the action is taken. The performance of the agent is monitored over time, with the shield being adjusted to have a higher K safe value if the performance is considered to be degrading. This strengthens or relaxes the cautiousness level of the agent.

The benefits of the method include the following. The generalisation using the autoencoder allows new states to be classified as safe or unsafe; this means unlike CA, not every unsafe state has to be visited. The method is shown to be effective compared to other model-free techniques, as although the routes taken by the shielded agent are longer, they are safer, again demonstrating the trade-off between safety and reward maximisation.

However, there are also disadvantages to the method. Bethell et al. (2024) does not state the accuracy of the autoencoder, so it is not known how accurate it is. False-positives (safe state-action pairs being classified as unsafe) could slow down the progress of the agent. Another disadvantage is the "cold-start" issue although this

is somewhat mitigated by the fact that not every state has to be visited, and generalisation will stop unsafe states being visited after some training has occurred.

Safety Critics

Another method of model-free shielding that has been tested are safety critics (Srinivasan et al., 2020). These are secondary policies that take control when the main policy is in danger of constraint violation in the near future. These originally had the goal of creating a secondary policy that could be transferred to other tasks and assist with safety in those. They consider that using a shield to modify the main policy can cause a sub-optimal policy to be created (Ray et al., 2019). This method does not use a priori data about the safety constraints unlike other methods previously mentioned. It does however involve using an offline data set to train the safety critic policy. This means that the learning takes place in two stages: one to learn the safety critic policy, and the second to learn the main policy. The data was collected by having human-defined policies to direct the agent towards the obstacles and dangerous states (Thananjeyan et al., 2021). This is in effect using a priori knowledge of the environment, just not explicitly.

This method has its advantages, as the offline pre-training data can be collected under human supervision (via simulation) and mean that the agent does not have to experience as many violations while performing the online training. The disadvantages are that it is a two-stage learning process and the offline data trained from will have to be collected using a hand-trained policy, with states labelled as safe or unsafe. States having to be labelled as safe or unsafe is common to all methods however, whether this is via a binary function, or by a logical constraint.

Bharadhwaj et al. (2020) tests a safety critic approach in the context of Conservative Q-learning, first implemented by Kumar et al. (2020). Conservative Q-learning is a method of offline RL that aims to learn a policy that lower bounds the true value, which prevents any overestimation of the value of the state-action pair, which may happen in the case of actions that result in rewards that are outside the normal distribution of expected returns (e.g. if an agent won the lottery after buying five lottery tickets this would wildly skew the expected return for this action, despite the result having a one in ten million chance of occurring). Conservative Q-learning does not assume a known dynamics model, so safety cannot be guaranteed and therefore this method falls under safety level 2.

Bharadhwaj et al. (2020) uses the method of Kumar et al. (2020), but instead of using a lower bound for the states' value, it is reversed to give an over-estimation of the probability of failure when training the safety critic. This method resulted in up to 50% less safety violations than other safe exploration methods. An additional feature of this algorithm is that when testing the algorithm, one hundred rollouts were done before every action to ensure the safest action was taken; while effective, the time taken to do this is not mentioned in the paper. Compared to other shielding methods that will either query the shield once per action this is likely to be a rather slow method. The provable safety guarantees could make this worthwhile, however, as the method converges to a much lower failure rate than other comparable methods (Bharadhwaj et al., 2020).

Neural Barrier Certificates

Work by Zhao et al. (2021) created an implicit safe state algorithm to aid in collision avoidance for mobile robots in 2D planes. The safe set being the states that the robot is safe in, in this case, this is a certain distance away from a hazard. This is created by synthesizing a safety index which is a subset of the total set of states. The agent's presence in the safe subset of states is forced by forward invariance, which means the robot will never leave the set if it starts in it. This falls under the category of safe control, which is the field of robotic control. This mechanism is referred to as the Neural Barrier Certificate (NBC).

The dynamics of the agent are modelled by data-driven learning in the form of either a digital-twin or a neural network. This digital twin does not have explicit white-box model knowledge but will be able to predict

outcomes to a reasonable degree of accuracy. This means the knowledge of the agent and environment is implicit, not explicit, unlike the many model-based algorithms.

Yang et al. (2023b) expanded on the concept of NBC. In their work, they do not require a digital twin during training. They show their results and safety violations after convergence, and the author states that there are safety violations during training. These safety violations, however, are an implicit result not producing a digital twin, but comes with the advantage of not having an additional training phase to produce the digital twin. The work also extends the invariance from single step to multi-step to further the effectiveness of the NBC. While training (with PPO as the RL training algorithm), both the policy and barrier certificate are updated. The barrier certificates acts as a shield in preventing actions that would cause violations. When tested the learnt policies and NBC achieve virtually zero safety violations when compared to other methods, however, the policy does not have as high a reward as the unrestricted PPO algorithm policy.

Adaptive Shielding

Shields that can adapt to a changing environment have not been deeply researched in RL. Most shields are constructed before runtime, either from models of the environment or pre-collected environmental observations. Some methods in the model-free shielding category such as the CAS (Shperberg et al., 2022) and Adaptive Shielding (Bethell et al., 2024) update the shield during exploration but this is different to adapting to a changing environment. The physical world is not static, so a policy that can adapt is necessary in some situations, for example, for a robot that will explore outside of controlled environments. Testing in changing environments is not the focus the current work, but a discussion is included here as the adaptive element is akin to a shield that learns, which is the purpose of the current work and potentially in the future, the current work could be extended to include an adaptive shield.

Adaptive shielding has been tested in RL, not for assisting in learning an optimal policy, but to give better overall results for a system. Pranger et al. (2021) tested the use of an adaptive shield where the controller was trained via RL in a traffic control system (akin to the work of Bloem et al. (2015) discussed earlier). This system performed an online estimation of the transition probabilities to form a model of the infinite state-MPD. This infinite-MPD is created at runtime, as is the MPD, using the controller (agent), whose policy is already known and trained. Observations on the current state of the environment are also factored into this. A smaller finite-state MPD was created for the shield by abstracting details away and merging less travelled to states. Both the MPDs were updated every *t* timesteps with the observations that had happened in that time. The shield was also updated every *t* timesteps, using the two updated MDPs. Past observations were discounted, to give weight to more recent observations. The shield takes the abstracted state and the action as an input and will output the original action if safe, or a new action if not. The resulting shield resulted in a lower waiting time for the cars in the traffic system, so was successful.

The authors state that adaptive shield should work in any case where it is possible to create an abstraction to a finite set of states. This gives the method use in any discrete environments but could also be used in continuous environments by creating bins at consecutive non-overlapping intervals for the continuous variables.

Discussion

There have been many advances in shielding for use in RL. Many of these are model-based shields that use either temporal logic or probabilistic logic. The use of hard coded logic, while effective in small environments, becomes limiting in an open world scenario due to the number of variables which need to be factored in. It also means that for each new problem or environment, a massive amount of work has to be done in understanding the environment and coding the hazards. This is counterintuitive to the RL algorithms such as DQN and PPO, which are effective in any suitable environment in which they are deployed with no additional parameters being

required, just a reward signal. Additional reward functions can be hand crafted to suit the environment to make the model perform better, but it is not strictly necessary for the agent to complete the task.

Lookahead techniques, such as BPS (Giacobbe et al., 2021), which looked at the potential future states given certain actions, are a powerful and effective tool, however, they have the disadvantage that they require e^n states to be checked, where n is the number of available actions (resulting in significant processing time being required, which can be restrictive in real-time scenarios). This can be offset to some extent in various ways, such as pruning actions, to only have to check a smaller subset or using a heuristic to select the potentially best actions.

Model-free shields are better suited to the principles of creating a RL algorithm. The environment is treated as unknown black-box to be explored, with no information about hazards or unsafe states given prior to training commencing. These shields all suffer from the "cold-start" problem. That is the issue where the unsafe states must be entered to determine that they are unsafe. Generalisations can be made however, so it is not necessary to enter every unsafe state in the environment. This results in lower safety guarantees than in model-based shielding. This major disadvantage could be offset by using simulated environments for training, to lessen the risk of the agent reaching an unsafe state.

Comparatively little work is available in the literature on model-free shielding. However, in the current work it is considered that this type of shield has the potential to perform well in some environments. The current work aims to tests the performance of these type of shields in two game environments with increasing complexity. Several shield methods will be tested including using a classifier as an online shield. The author believes this machine learning based approach is the first time that such a method has been used in the test environments presented here. It is noted that although safety violations will inevitably occur during training, the shield created will aim to minimise them and eliminate them as throughout the training process. Although tested in a game environment, there is potential for shielding methods to be transferred to real-world problems. The current research aims to present a first step towards use of these shielding techniques in real-world environments.

Chapter 2: Research Methodology

This chapter explains the RL environments and algorithms and used in the current work. The justifications for using these is provided, along with comparisons to other potential methods. Note that the compute environment used is detailed in Appendix A.

Environments

All experiments will be undertaken in environments that will terminate after a single safety violation. This reflects a variety of real-world environments where safety violations would require the immediate halting of, for example, a robot or self-driving car due to damage to itself, pedestrians or its surroundings. Safe exploration is concerned with minimising the number of these unsafe states entered by the agent.

In this work, two environments will be used for testing: the first a simple environment to allow the testing of shields in the context of a tabular RL algorithm (e.g. Q-learning) and the second to allow the testing of shields in conjunction with a deep-learning algorithm (e.g. DDQN). The environments to be used are outlined below.

Frozenlake

The Frozenlake test environment (Brockman et al., 2016), as shown in Figure 6, will be used as the initial test environment. Note that in addition to the 8x8 map, a 10x10 map, which is initially randomly generated but kept the same for each of the 10 runs, is also tested. In this environment the agent (the elf, marked B) must explore the map from the starting point (stool, marked A) to find the goal state (present, marked D), avoiding hazards (melt ponds, marked C). Therefore, this environment requires the agent to explore as well as avoid hazards during this exploration. If the agent enters an unsafe state the agent must restart the level (and hence not gain any reward), therefore, making it in agent's best interest to avoid these hazards where possible.

The simplicity of the Frozenlake environment make it ideal as an initial test environment. The RL algorithm and associated shielding will be tested on different size maps, and it is noted that this will increase the complexity of the environment.

It is possible to make the environment stochastic by turning on the "slippery" attribute. This means the agent will move in the intended direct with a 1/3 chance and move in either perpendicular direction with a 1/3 chance. This makes the movement unpredictable and gives a danger in moving forward when a lake is above or below (as you could fall in and cause the episode to terminate). This makes training much more difficult, as even the "correct" actions can lead to safety violations.

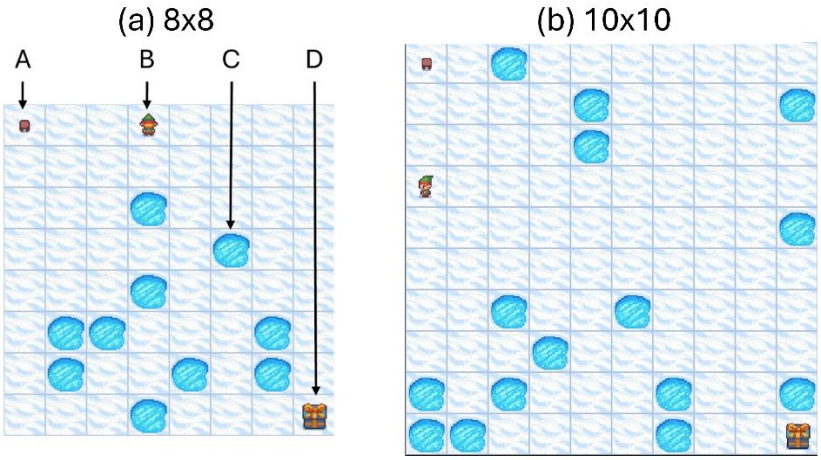


Figure 6: Frozenlake environment for the (a) 8x8 map and (b) 10x10 map (Brockman et al., 2016).

Pacman Environment

A custom-made Pacman environment is used at the more complex environment. Two test environments have been generated using Grok 3.0 (xAI, 2025), a simple case shown in Figure 8 and the advanced case shown in Figure 9. This environment is more challenging than the Frozenlake environment due to the following factors:

- Need for exploration: To complete the level, the Pacman (marked as A in Figure 7) must obtain all the food (marked as C in Figure 7). This is much more difficult than simply getting to a certain location, as in the simpler Frozenlake environment.
- Danger: In the Frozenlake environment the hazards (melt ponds) are fixed obstacles. In Pacman, there are stationary hazards (firepits, marked as C in Figure 8) as well as ghosts (marked as B in Figure 7) which constantly move around the environment. This means active hazard avoidance must take place to prolong the agent's safe exploration of the environment. Shielding in this environment could therefore play a critical role in improving the learning rate and success rate of the agent.

The simplified variant, see Figure 7, is used for initial testing of the shields. It contains two ghosts and one piece of food to collect. The ghosts have a wall following movement pattern, so will circle the islands they are next to. They will move with 90% probability to introduce slight uncertainty to the movement pattern, while remaining predictable.

The advanced variant, see Figure 8, has a maze of stationary firepit hazards that will also have to be avoided in order to complete the level. It contains two pieces of food to collect: one after the two ghosts, and one after the maze of fire pits.

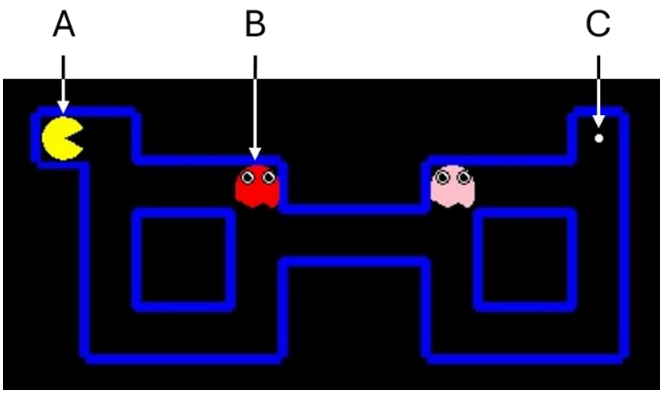


Figure 7: Simplified Pacman Map.

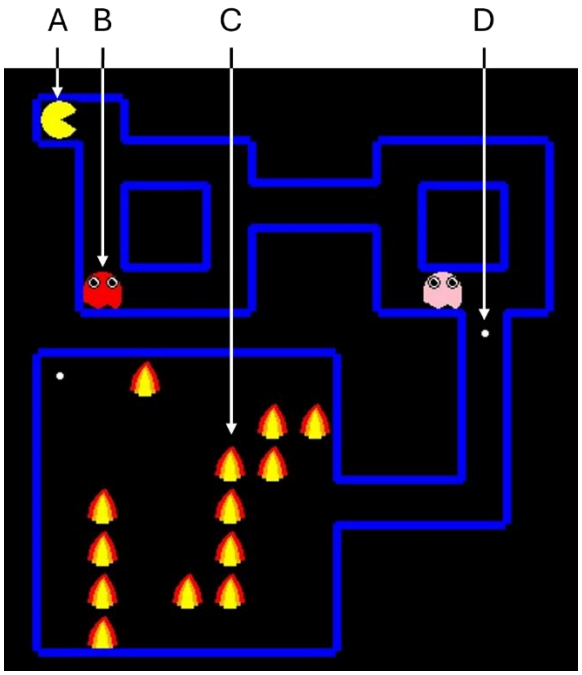


Figure 8: Advanced Pacman Map.

RL Algorithms

Tabular RL Methods in the Frozenlake Environment

For the Frozenlake environment, tabular RL methods are used, as these comparatively quick to run and so are suited for fast iteration and testing of new techniques. Q-learning and two variants of this algorithm are initially tested on the Frozenlake environment to determine which is best suited for further analysis of the performance of the various shields to be implemented.

Q-Learning

Q-learning (Watkins, 1989) is a simple tabular method which is described by the pseudocode given in Figure 9.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state},\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 9: Q-learning pseudocode (Sutton and Barto., 1998).

The core of the algorithm, described in detail in Chapter 1, revolves around the Bellman equation (see Eq. 1) and essentially performs a weighted value iteration update for a state-action pair each step. The values are stored in a q-table, which is initialised at the start of the algorithm.

Using α, the learning rate, the current value is updated to a value somewhere in between the old value and the new estimated value of the state-action pair. The discount factor, λ, is used to propagate the value of the next best action in the newly reached state backwards. For example, a discount factor value of 1 would copy the

exact value and would lead to all state-action pairs that can eventually transition to the goal state having the terminal state reward value. A value of the discount factor slightly lower than 1, such as 0.9 is often used, to allow most of the value to be propagated backwards, but not all, so the best path to the goal state can be found.

Weighting using the learning rate is described by Eq. 4, with the current value and new value separated (as indicated). The agent takes steps until a terminal state is reached by selecting the state-action pair with the highest Q-value when in that state. The whole process is then repeated for each episode.

$$Q^{new}(S_t, A_t) \leftarrow (1 - \underbrace{\alpha}_{\text{learning rate}}) \cdot \underbrace{Q(S_t, A_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{\left(\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(S_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}_{\text{new value (temporal difference target)}}$$

Equation 4: Q-learning state-action pair value update function (Sutton and Barto., 1998).

The Q-learning method is a relatively simple method, but highly effective in small environments, such as the Frozenlake environment. Due to the tabular nature of the algorithm (a value is held for every state-action pair) the size of the q-table will be the number of visitable states multiplied by the number of actions available. This limits the technique to small environments, due to the memory requirements, hence it is not considered application for the Pacman environment used here.

Double Q-learning

A disadvantage of the Q-learning method is that it tends to overestimate the values of state-action pairs due to the use of the max function in the equation. There is potential for this to negatively impact the performance of the algorithm. For example, if all action values were uniformly higher, then the policy would not be any different. On the other hand, if the overestimations were not uniform, this could lead to a sub-optimal policy being produced.

A way to overcome this is to use Double Q-learning (Hasselt, 2010) which initiates two q-tables instead of one. The algorithm then randomly chooses which table to update each time a step is taken. The estimate is done using the q-value from the table that is not being updated and the maximum value of the state-action pair in the q-table that is being updated. This method has been shown to stop the overestimation of state values. In the long run it will converge to the optimum policy in the same way that Q-leaning does. See Figure 10 for the Double Q-learning pseudocode.

1: Initialize Q^A,Q^B,s
2: **repeat**
3: Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
4: Choose (e.g. random) either UPDATE(A) or UPDATE(B)
5: **if** UPDATE(A) **then**
6: Define $a^* = \arg\max_a Q^A(s',a)$
7: $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
8: **else if** UPDATE(B) **then**
9: Define $b^* = \arg\max_a Q^B(s',a)$
10: $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
11: **end if**
12: $s \leftarrow s'$
13: **until** end

*Figure 10: Double Q-learning pseudocode (*Hasselt, H., 2010*).*

Dyna-Q Learning

Another variant of the Q-learning algorithm is Dyna-Q (Sutton and Barto., 1998). As well as learning the state-action pair values, this model also memorises the transitions between states to build a model of the environment. This is done every time an action is taken, so over time more and more is learnt about the model of the environment. Each time the agent takes an action, the model is sampled n-times to further updated the q-table. This enables faster learning, as the reward from the goal can be propagated back through transitions from which nothing was initially learnt. See Figure 11 for the pseudocode for the Dyna-Q learning algorithm.

This algorithm assumes a deterministic environment, as in a stochastic environment the same action in a given state can lead to several different states, however, modifications can be made to the algorithm to suit a stochastic environment. This can involve choosing the most likely transition when an action is taken to learn from to enable an approximation of the model. Alternatively, choosing an outcome with the probability that it has previously been seen, for example, a 25% chance of choosing a transition from a state-action pair that has happened 2/8 times previously. Although such modifications could improve the performance of the algorithm in a stochastic environment, this is not the focus of this current work and hence is not implemented here.

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \epsilon$-greedy(S,Q)
- (c) Execute action A; observe resultant reward, R, and state, S'
- (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
- (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Repeat n times:
  - $S \leftarrow$ random previously observed state
  - $A \leftarrow$ random action previously taken in S
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Figure 11: Dyna-Q pseudocode (Sutton and Barto., 1998).

Comparison of the Q-Learning Algorithms in the Frozenlake Environment

Each of the algorithms described above (Q-learning, Double Q-learning and Dyna-Q) have been implemented in the Frozenlake environment (shown in Figure 6) to determine which should be used as a baseline in the current work, on top of which the various shields will be tested.

Figure 12 shows the results when running each of these variants for 5,000 episodes with the same parameters. Each algorithm was run ten times to reduce the random noise inherent in the training process and reduce the effect of outliers to allow a more robust comparison. It can be seen that Double Q-learning does not converge to high rewards as quickly as standard Q-learning. This is because in the Double Q-learning two q-tables are being learnt, so rewards will not propagate back as quickly. It does converge to the best policy however, showing that the method is highly effective. Dyna-Q converges the fastest, with each step not only learning from the standard q-table update, but also a batch of sixty-four previous actions, therefore it learns much faster. This comes at a cost, however, as the algorithm took six and a half minutes to run, compared to roughly thirty seconds for the other two methods. Dyna-Q also did not converge to the optimal policy within the given episode count. Therefore, although the learnt policy was good, it was not quite as effective as the ones learnt by Q-learning or Double Q-learning.

In terms of safety violations, Q-learning had 1,366 on average, Double Q-learning had almost double at 2,500, due to converging twice as slowly and Dyna-Q had a much lower 735, due to converging on a good, but not optimal policy much faster, using simulated actions rather than taking actions in the environment. As

stochastic environments will be used, and Dyna-Q does not have strong optimal policy convergence with these, I will use standard Q-learning when testing shielding techniques within the Frozenlake environment.

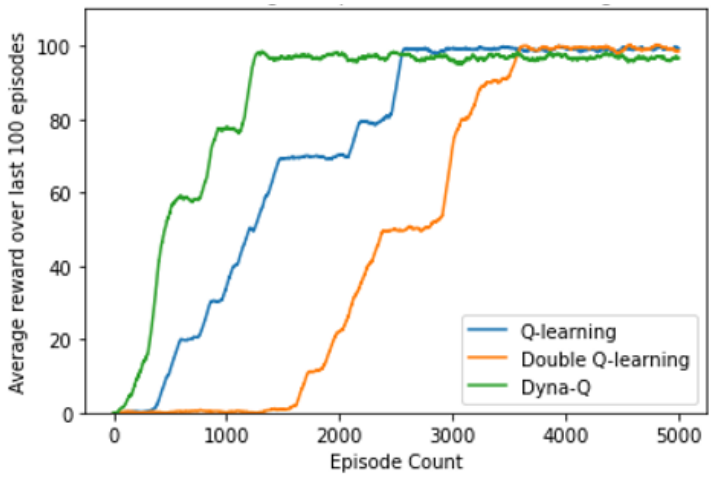


Figure 12: Results of Q-learning and variants on Frozenlake environment.

Deep Q-Learning and Double Deep Q-Learning for the Pacman Environment

With a more complex environment, in this case both the simple and advanced Pacman maps, generating a q-table to hold values for all the state-action pairs is not feasible. For the larger 10x10 Frozenlake environment there are 400 state-action pairs, however, even for the very simplified Pacman game environment (Figure 7) there are roughly 35,000 state-action pairs. This will increase substantially for even a slightly more complex environment.

For this reason, a neural network-based method is used as the RL algorithm for the Pacman environments. Using an algorithm that used a neural network means that the states will be generalised, so any number of potential state-action pairs can be used without issue. The DQN method has been chosen (detailed in the previous chapter), due to its proven effectiveness in similar environments. It will be tested here whether the DQN or DDQN proves to be more effective in the simple Pacman environment before making the choice of which one to use for the shield experiments.

In this case, ReLU (Householder, 1941) will be used to introduce non-linearity to the network. The formula for this is f(x) = max(0, x). This is generally preferred over the Sigmoid activation function due to the Sigmoid function suffering from the vanishing gradients issue (where the gradient becomes so small as to almost vanish as backpropagation is done) (Hochreiter, 1998). It is also very computationally efficient, as only the comparison of two numbers is needed.

An experience replay buffer (Lin, 1992) is used to store the experiences of the agent. The maximum capacity of the buffer is 500,000. The experiences are cycled out, oldest first when this capacity is reached. In this way the actions of the agent when it is more trained are used, as these will be more useful to learn from than the mostly random movements that are performed initially.

In the DQN vs. DDQN testing, the following hyperparameters are used:

- epsilon = 1 initially, 0.05 after episode 20,000.
- epsilon decay = 0.0001 (per episode after episode 10,000 down to a minimum of 0.05)
- discount factor = 0.9
- learning rate = 0.001
- network synchronisation rate = 100

The epsilon is kept high initially to allow the agent to randomly explore for some time to build up a bank of diverse experiences before the algorithm starts to guide it. Initially 10,000 random episodes were tested but this was later reduced to 1,000, as this proved to be equally effective.

The discount factor, learning rate and network synchronisation rate were chosen with a small hyperparameter sweep. The discount factor is relatively high to allow for rewards to be propagated backwards for many steps.

The following are used for the policy and target network structure:

- Input: Vector of length 8
- Hidden Layer 1: 32 nodes
- Hidden Layer 2: 16 nodes
- Output Layer: 4 nodes

Several different network structures were tested, however the 8-32-16-4 structure reached a higher average reward faster than other structures.

To test the DQN vs. DDQN, the agent was trained for 120,000 episodes for three runs. As shown in Table 1 and Figure 13, the DQN performed well initially but remained volatile due to the intrinsic optimism of the algorithm which led to lower average reward. In terms of agent safety the DQN had an average of 85,389 safety violations per run vs. 67,820 for DDQN. For this reason, the DDQN algorithm will be used, as DDQN has more stable training. Using the method with fewer safety violations is preferable as it allows a lower baseline on which to test the shielded vs. unshielded methods.

Run	DQN SV	DDQN SV
1	114,153	98,886
2	82,006	54,601
3	**60,008**	**49,973**
Average	85,389	67,820
Std. Dev.	22,234	22,048

Table 1: Comparison of DQN and DDQN safety violations in the simplified Pacman environment.

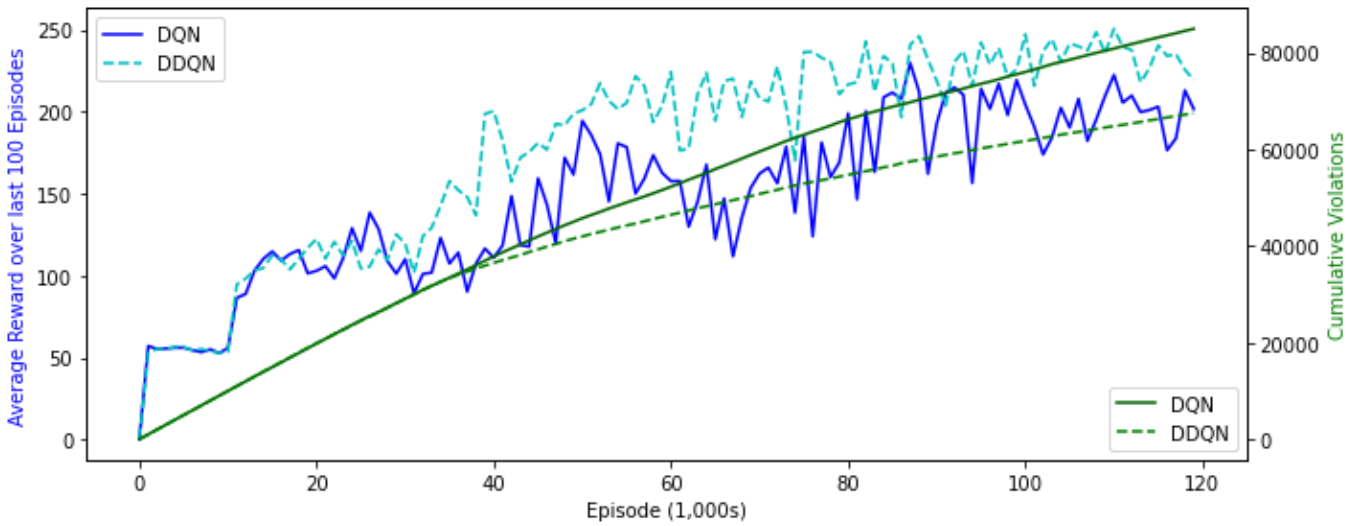


Figure 13: DQN and DDQN rewards and safety violations in the simplified Pacman environment.

Existing Shielding Techniques for Comparison

In the current work, novel shielding techniques will be developed and tested. To understand and quantitatively determine the performance of these novel techniques, baselines are required.

For the simplistic Frozenlake environment, three cases are used as a baseline: no shield, a Catastrophic Avoidance Shield (CAS) and intrinsic punishment. These last two are safe RL techniques that cause the agent to act in a safer manner than when a shield is not used.

For the Pacman environment, the baseline used is no shield as the CAS is not suitable in a non-tabular setting and while intrinsic punishment was trialled, it gave worse results than the no-shield baseline and therefore adds no value to the analysis.

Catastrophic Avoidance Shield

The CAS (Shperberg et al., 2022) as described in Chapter 1, is highly effective as it simply prevents a mistake from being made twice. This can also be used in more advanced environments by using parametric methods to create a classifier to classify the state-action pairs as safe or unsafe. This could be trained and updated every *n* steps to make it more accurate. The disadvantage of this method however would be that every state-action pair would have to be recorded. However, as this is done as part of deep learning methods such as the replay buffer of DQN, it should not be an issue.

Intrinsic Punishment

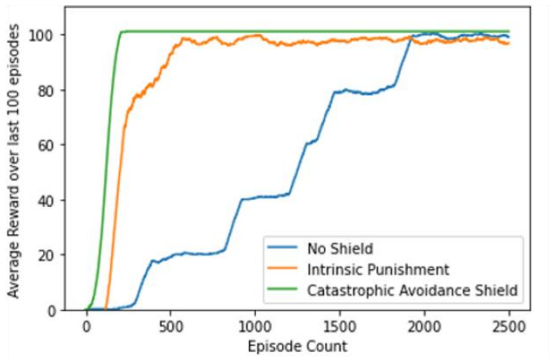
Intrinsic punishment (Alshiekh et al., 2018), as described in Chapter 1, is the stick to the carrot of the reward signal in RL. This method punishes the agent for taking an unsafe action, to discourage it from doing it again. Each time an unsafe action is taken, the agent will be given a small negative reward. This will lower the value in the q-table and mean it should not get chosen again.

Initially a negative reward of -1 (equal to the goal state reward of -1) was tested. This caused a high negative reward from the unsafe states to propagate back and stop progress. The agent chooses actions that do not go anywhere near the unsafe states but also avoided the terminal state and would repeatedly time out after many actions were taken. A small negative reward of -0.1 was then implemented, the agent performed better and started consistently reaching the goal state.

Comparison of Baseline Shields in the Frozenlake Environment

Figure 15 shows a comparison of the different baseline methods to be used. Each was trained with an epsilon-greedy policy where epsilon reduced from 0.5 to 0.02 over the course of the episodes. Meaning in the initial episodes there is a ~50% chance of random action, and a 50% chance of an action being chosen by the policy. This high degree of randomisation is chosen to enable sufficient exploration. Each episode this chance of a random action being chosen is reduced until it reaches 2%. The learning rate and discount factor were both set to 0.9. Each experiment was run ten times to get a more accurate average convergence rate.

a) Comparison of different shielding methods



b) Comparison with slippery attribute turned on

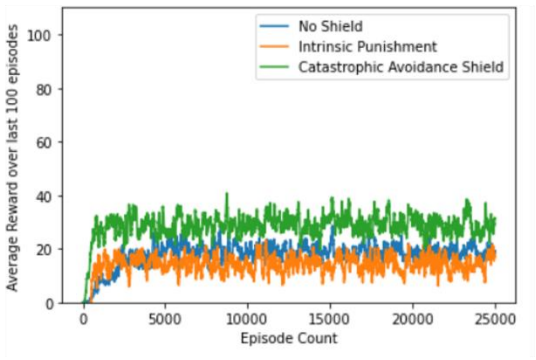


Figure 14: Rewards for several shielding methods (a) without and (b) with the slippery attribute turned on.

Two trials were run: one without the slippery attribute turned off (hence a purely deterministic environment) and one with the slippery attribute turned on (hence there is a high degree of stochasticity in the agents movements, see the earlier description of the environments for further information).

Figure 14a shows the first case where the environment was completely deterministic. There were 33 state-action pairs that ended in an unsafe state (a melt pond) and terminated the episode. The standard Q-learning was the slowest to converge, which is expected, as there is nothing in place to speed up convergence. It had an average of 1,100 safety violations per run. This was done to be a comparable baseline. The CAS worked very well and only had 27.4 safety violations on average per run. This meant that on average it would converge so quickly that not all the 33 unsafe actions were made (as each can only be made once). The intrinsic punishment was also quite effective, with 249 safety violations per run. This is ten times that for the CAS, but less than a quarter of what the standard q-learning did. For intrinsic punishment the modified reward did not converge as highly as standard Q-learning, however. This was because of the negative reward given when an unsafe state was entered. The overall policy therefore ended up as not quite as effective at reaching the goal state as standard Q-learning.

In Figure 14b the slippery attribute was turned on. This gives a 1/3 chance for the agent to perform the intended action and then another 1/3 (x2) for the agent to go in either perpendicular direction. This provides very noisy feedback for the RL algorithms to learn from. It also means that even with an optimal policy, there is still a high chance of failure. The q-learning and intrinsic punishment method performed very similarly, getting an average of 1,184 and 1, 145 violations respectively. The intrinsic punishment method converged on a good policy faster, however, so was still positive in regard to the learning rate.

The CAS did much better, only getting 684 violations on average, and converging on a better policy than the other two methods. It had to be modified to work in this environment, as so many state-action pairs ended in an error state that the agent would freeze during training, as there we no safe actions to take. If a deadlock was reached, where every action has previously ended in a catastrophe (this can happen where there is a lake on two or more of the agents four sides), then the action with the highest value is chosen instead.

This modified CAS method resulted of an average reward of around 30 per 100 runs vs. around 20 for the other two methods. This was surprising, as the q-learning method has convergence guarantees. It did converge, but just not on the optimal policy. The CAS also converged much faster than the other two methods. This is due to the shield blocking the unsafe actions caused by the e-greedy policy (if they have previously been made), which means the training episodes will run for longer, giving a higher chance of reaching the goal state.

Shielding Methods to be Applied

Learning a Predictive Shield with Online Shielding

The core of this research is to create a shield that learns the hazards of the environment and then prevents the agent from reaching those hazards. This means to not take actions that lead to unsafe states. This will be done using a dynamic shield that learns in one online training phase. In this phase, the shield will learn both the environment dynamics and the hazards and combine this knowledge to create a shield that will enable the agent to move safely through the environment.

In environment / agent dynamics modelling, the actions the agent takes will be mapped to the effect on the environment. This will be done by introducing the coordinates of the agent as part of the state knowledge for the Frozenlake environment and for the Pacman environments both the coordinates of the agent and the ghosts will be introduced. The actions taken will be analysed to see the changes that they have made. Using inference, a prediction will then be made whenever the agent takes an action. This prediction will be the state that the agent will be in after the action has been taken.

Dynamic Shielding

The shield will be a combination of the future state prediction and learnt hazards. As the agent explores the environment, catastrophic states will be saved. This will be done in a tabular fashion for the q-learning based learning (for Frozenlake), and in a parametric fashion for the neural network based method (for Pacman).

The online shield will be implemented through the following steps:

1) Have the chosen action by the RL algorithm as an input, along with the other available actions ranked in order of value.
2) Predict the state that the action will reach.
3) Classify the state as safe or unsafe.
4) Allow the action to happen if it is safe to do so – the verified safe action is the shield output.
5) If it is not safe, repeat steps 2 & 3 on the remaining actions until a safe action is found – the new safe action is the shield output.
6) If no safe action is found, mark the state as unsafe and allow the chosen unsafe action to happen. This will prevent deadlocks stopping training from occurring – the initial action is the shield output.

Number 2 and 3 will be the core workings of the shield. The prediction engine and the classifier will be updated regularly as training goes on and adapt to new information.

A safe action is defined as an action in a state that does not lead to a catastrophic event. A catastrophic event is a state that will cause termination of the episode with no reward given. These states are considered unsafe. By marking states that have no safe actions as unsafe in the same way as catastrophic events, these states will also not be visited, as the shield will prevent this.

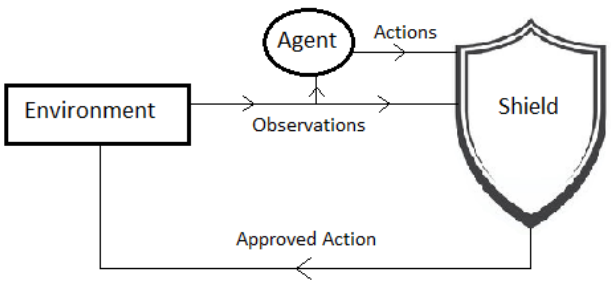


Figure 15: Flowchart of data flow in dynamic shielding.

Unlike the catastrophic action shield (Shperberg et al., 2022) this shield should not only prevent unsafe actions that have already been taken from occurring, but also prevent the unsafe states being visited via other actions.

Tabular Shield Methodology

A tabular shield will first be created with hard-coded logic in the Frozenlake environment. A database will be created to record changes in the x, y position of the agent (which is given as an output, as the state). In addition, when an action is taken a count will be record for each occurrence of each result for each action.

Separately, each time an unsafe state is visited, it will be saved, in a similar fashion to the catastrophic action shield method (the catastrophic action shield records state-action pairs). This will build a picture of the unsafe states in the environment as the agent explores it.

After ten episodes the shield will activate. This is to allow the shield to build up a small amount of data before activating. The shield will then predict the state for the action that the q-learning algorithm has chosen and if it has previously been recorded as an unsafe state, a new action will be chosen. The prediction will be the effect of the x and y change of the agent for the chosen action, which will be the effect with the highest count at the time. This is updated for each action taken. The new action (or actions if the next action also leads to an unsafe state) will be based on the order of descending q-values for that state.

Deep Learning Methodologies

For the tabular shield the prediction of the next state and classification of that state are implemented as two separate stages, however, with non-tabular shields (which will be applied here in the Pacman environment) this does not have to be the case. Given a state-action pair, we can use a classifier to predict the safety of the outcome. For this a machine learning classifier will be chosen which must support training in several stages, to accommodate new training data as the agent explores the environment. Support Vector Machines (SVMs), while a powerful classifier, able to classify non-linear data due to a variety of kernel functions, are not suitable for this as once it is trained, the multidimensional plane is set. It would have to be retrained entirely for each new batch of data and combined with the relatively slow training inherent to SVMs, they are unsuitable for use in dynamic shielding. Therefore, in the current work, the following classifiers are considered.

Naïve Bayes Classifier

The Naïve Bayes classifier is very fast to train and can be partially trained and then have more data added to it and the probabilities updated, so is suitable to be run alongside deep learning algorithms. This classifier is commonly used as text classification in use cases such as spam detection (Sahami et al., 1998) and sentiment analysis of tweets (Goel et al., 2016).

The Naïve Bayes classifier is based on Bayes theorem (Eq. 5). This theorem (Stuart & Ord., 1994) calculates the probability of an event given some prior knowledge. The knowledge needed is the probability of events A and B happening and the probability of B occurring given A also occurring.

$$P(A|B) = P(B|A) * P(A) / P(B)$$

Equation 5: Bayes theorem (Stuart & Ord., 1994).

Where:

P(A|B) is the probability of A occurring given that B has occurred.
P(B|A) is the probability of B occurring given that A has occurred.
P(A) is the probability of A happening.
P(B) is the probability of B happening

For this use case of the Naïve Bayes classifier, we will be looking to find the probability that a state-action pair is safe given the attributes of the state-action pair. There are three different varieties of Naïve Bayes: Bernoulli, Gaussian and Multinomial. Bernoulli Naïve Bayes uses binary input data, so it not suitable in this case. Gaussian Naive Bayes is suited for continuous data and assumes a normal distribution of that data, while multinomial Naïve Bayes uses discrete data and assumes a multinomial distribution, that is, each attribute having its own distinct probability within a given class. Given that the Pacman environment is a grid map, with discrete attributes, the multinomial Naive bayes is the suitable candidate. The Gaussian Naïve Bayes will also be tested, as the distribution of the data is not known.

To evaluate whether the Naïve Bayes classifier will work as a shield it will first be tested on data collected from a training run on the simple Pacman environment. The average accuracy across the 5 sequentially trained batches of data will be recorded, as well as the recall (Eq. 6) for the unsafe instances.

$$\text{Recall} = \text{True Positives} / (\text{True positives} + \text{False Negatives})$$

Equation 6: Recall Equation.

Where:

True Positives: Number of unsafe samples correctly classified unsafe.
False Negatives: Number of unsafe samples misclassified as safe.

The results of this analysis are presented here as they inform the choice of classifier and the hyperparameters of that classifier used in the shield to be tested. Initially, Multinomial Bayes performed poorly, with the recall of the unsafe class being zero and an accuracy of 98.32%. This was due to the dataset being highly unbalanced with 98.22% of the dataset being safe instances. To address this, under sampling was implemented to balance the dataset. This consisted of taking a subset of safe examples to match the number of unsafe examples in a chosen proportion. The initial ratio tested was 1:1 with an equal number of safe and unsafe samples used. This gave a reduced accuracy of 54.86, but an unsafe recall of 76.96%. Other ratios were tested, with the ratio being the amount of safe instance for each unsafe instance, however, none gave better results than the initially tested 1:1 ratio. As seen in Table 2, at ratios lower than 1, recall was 100%, but accuracy dropped to around 2%. This would be not be suitable, as although the unsafe instances would be caught, the shield would cause deadlocks due to being triggered by so many false positives. The Gaussian Naïve Bayes yielded similar results for the under sampling, with the trend of the recall decreasing as the accuracy increased as shown in Table 3.

Safe: Unsafe Under Sample Ratio	Accuracy [%]	Unsafe Recall [%]
None	98.32%	0%
0.5	1.68%	100%
0.75	1.75%	100%
0.85	9.12%	99.78%
1	50.86%	74.96%
1.25	96.48%	4.99%
1.5	98.32%	0%

Table 2: Multinomial Naïve Bayes accuracy and recall for different under sampling ratios.

Safe: Unsafe Under Sample Ratio	Accuracy [%]	Unsafe Recall [%]
None	98.32	0
0.5	38.95	94.91
0.75	46.78	85.14
0.85	49.26	81.98
1	53.12	77.65
1.25	59.30	70.63
1.5	64.90	61.08

Table 3: Gaussian Naïve Bayes accuracy and recall for different under sampling ratios.

The second parameter tested was the threshold for the unsafe class. Initially the default for both classes was 0.5, with the class that has a probability of more than 50% being the chosen as the classifier output. As recall of the unsafe instances is the priority, the threshold for the unsafe class was lowered to raise recall. The threshold for the safe class was raised to provide the inverse effect. Many variations of class thresholds combined with different ratio multipliers were tested to gauge the best fit for the data. The key results are presented in Tables 4 and 5.

Accuracy and unsafe recall are the two metrics that need to be maximised. As they are inversely correlated in the tests a balance must be struck to choose the optimum. The metric used to assess this is accuracy added to unsafe class recall. This simple yet effective metric aims to find a balance between the two. The theoretical maximum possible result for this metric is 200. That would be with 100% accuracy and 100% recall (100+100 = 200).

Safe: Unsafe Under Sample Ratio	Accuracy [%]	Unsafe Recall [%]	Accuracy + Recall
0.9	29.18	**96.67**	125.85
0.95	42	87.42	129.42
1	**50.86**	74.96	125.83
0.9 (safe class threshold of 0.49)	39.57	90.74	**130.51**

Table 4: Multinomial Naïve Bayes grid search highlights.

Safe: Unsafe Under Sample Ratio	Accuracy [%]	Unsafe Recall [%]	Accuracy + Recall
0.5	38.95	94.91	133.87
0.9	50.42	80.55	130.97
0.95	51.96	79.43	131.40
1	**53.12**	77.65	130.77
0.5 (safe class threshold of 0.52)	37.68	**96.56**	**134.25**

Table 5: Gaussian Naïve grid search highlights.

As shown in Tables 4 and 5, changing the safe threshold marginally improves the recall given the same under sampling ratio. The Gaussian Naïve Bayes performed better overall, so will be the chosen classifier for the shield. An under sample multiplier of 0.5 will be used along with a safe class threshold of 0.52. This likely means the accuracy of the shield will be low, around 38%, but the many false negatives should be offset by the high recall of over 96.5% that will catch the vast majority of unsafe actions.

A Neural Network as a Shield

The second method of creating a shield for the Pacman environment will utilise a neural network as classifier. Neural networks are better at learning non-linear relationships between features and can generalize well to classify new data. Unlike Naïve Bayes, the features are not assumed to be independent, which may improve performance over the Naïve Bayes classifier.

The network will have an input vector length of nine, one more than the network used for the DDQN as the action taken is now being used as an additional input feature. The hidden layers will have 64 and 32 nodes respectively. A 32 and 16 node hidden layer architecture was also tested, but is not used here as this performed slightly worse, with no accuracy plus unsafe recall scores surpassing 198, while the 64 and 32 hidden layer architecture had 21 hyperparameter combinations that surpassed this.

Binary Cross-Entropy Loss *(Goodfellow et al., 2016)* will be used as the loss function, as this is a binary classifier. This quantifies the difference between the predicted class probabilities and the actual class of the sample. The formula for the binary cross-entropy is:

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

Equation 7: Binary Cross Entropy (Goodfellow et al., 2016).

Where:

N is the number of observations.
y_i is the actual class label of observation *i*.
p_i is the predicted probability of observation *i* being in class 1 (safe).

A sigmoid function (Goodfellow et al., 2016) will be used on the output of the network to give the probability of the experience being fed through being a member of the safe state class. The sigmoid function is shown below

$$\sigma(x)=1/(1+e^{-x})$$

Equation 8: Sigmoid Function (Goodfellow et al., 2016).

The network will have the structure below:

- Input: Vector of length 9
- Hidden Layer 1: 64 nodes
- Hidden Layer 2: 32 nodes
- Output Layer: 1 node

Dropout (Srivastava et al., 2014) was tested at a probability of 10%, 20% and 30% as this technique can help to prevent overfitting of the data. It works by randomly disabling a percentage of neurons during each training epoch. This reduces the reliance on any one specific neuron and makes the network more robust. It also enhances generalisation on unseen data.

The network was trained with the last 20,000 data points from each of the five batches data recorded from an unshielded DDQN run on the simple Pacman map. Only the last 20,000 out of around 500,000 samples per batch were trained on due to processing constraints. This is still a training set of 80,000 data points and a test set of 20,0000 data points, so it is considered it will give a relatively fair comparison to Naïve Bayes.

Additionally, weighting was done, to give the unsafe class data points a higher weighting, so an equal number of each class will be chosen when training. This is equivalent to using an under sample modifier of 1 in the Naïve Bayes method. Instead of under sampling, the unsafe class data points are oversampled to ensure all data points are sampled at least once.

After multiple safe classification thresholds were tested in combination with the two architectures discussed above and dropout, it was found that the larger architecture in combination with dropout of 0.2 and a safe class threshold of 0.4 gave the best accuracy plus unsafe recall of 198.57. 50 epochs of training for this network have been chosen here; while 100 epochs were also trialled, it was found that they gave no additional benefit, despite doubling the training time. A table of some of the highlight results is shown below in Table 6.

Dropout	Safe Threshold	Accuracy [%]	Unsafe Recall [%]	Accuracy + Recall
No	0.8	98.39%	99.6%	198
0.1	0.2	98.93%	99.6%	198.53
0.2	**0.4**	**98.57**	**100%**	**198.57**
0.2	0.46	98.47%	100%	198.47
0.3	0.5	99.24%	86.80%	186.04

Table 6: Neural Network grid search highlights (hyperparameters used highlighted in bold).

This is substantially better than the best Naïve Bayes classifier result of 134.25 (see Table 5). This will be implemented as a shield, to be trained every 10,000 episodes in the same way the Naïve Bayes classifier has been. The same data flow will be used as in the Naïve Bayes shield, with the list of actions sorted by descending q-values being output by the DDQN. Each action will be checked in turn and if it is deemed to be safe by the classifier, the action will be taken. If all actions are deemed to be unsafe the action with the highest q-value will be used.

Chapter 3: Experiments

Tabular Shield Experiments – Frozenlake Environment

The tabular shield method will be tested in several ways. Firstly, with the standard 8x8 map then with the 8x8 with the slippery attribute turned on.

A larger 10x10 map will then be used in conjunction with the Dyna-Q algorithm to test the effectiveness in a larger environment where more exploration is needed. This has 100 states, whereas the 8x8 map has 64 states, i.e. a 56% increase. All experiments will be run ten times, and the average recorded, as e-greedy based RL algorithms can learn at a very different rate in the same environment due to the randomisation of actions taken. For comparison purposes, the unshielded q-learning, CAS and intrinsic punishment methods described and demonstrated above will also be run with these same environments, also ten times each. The parameters used for each map are given in Table 7.

These experiments will have the following hyperparameters:

- Epsilon: The chance that a random action will be taken with the e-greedy exploration.
- Discount factor: The percentage of the future reward that is given to the state-action pair.
- Learning rate: The rate at which the new q-value overrides the old q-value.

Hyperparameters for the 8x8 map:

- epsilon = 0.5 - This will decay by 0.001 per episode down to a minimum of 0.02
- discount factor = 0.9
- learning rate = 0.9

Hyperparameters for the 10x10 map:

- epsilon = 0.05
- discount factor = 0.9
- learning rate = 0.9

Deep Learning Experiments – Pacman Environment

For the Naïve Bayes shielding experiments, the same parameters are used for the DDQN as in the methodology section. The Naïve Bayes shield was tested with both the 0.5 multiplier and standard thresholds, as well as 0.5 multiplier and the 0.52 safe class threshold on both the standard and advanced Pacman maps.

The neural network shielding experiments will be run in a similar fashion, with the DDQN algorithm used as the RL algorithm in conjunction with the neural network shield. The shield will have a dropout of 0.2 and a safe class threshold of 0.4 used on both the standard and advanced Pacman maps.

Preliminary training was done for 120,000 episodes on the standard map and 200,000 episodes on the advanced map for each shielding technique. Longer training time was given to the advanced map due to the difficulty of the map. In shorter trial training runs it was common for the agent to not collect both pieces of food

and win game in any of the 120,000 episodes. This does confirm however that the advanced environment is sufficiently harder than the simple one.

Chapter 4: Results and Discussion

Tabular Shield Results – Frozenlake Environment

Below are the results of the tabular shields used with the Frozenlake environment for the standard 8x8 map, the standard map with the slippery attribute turned on and the larger 10x10 map. The shields tested here are the predictive shield implemented as part of this work, the CAS (from Shperberg et al., 2022) and intrinsic punishment (Alshiekh et al. (2018)). The full results are included in Appendix C.

Standard 8x8 Frozenlake map

Model	Average Safety Violations per run	Average Steps per run
Predictive Shield	**12.9**	64,145.1
Catastrophic Shield	30.6	64,981.5
Intrinsic Punishment	377.9	**57,547.5**
No Shield	1,055.1	65,448.3

Table 7: Results for the standard 8x8 Frozenlake environment.

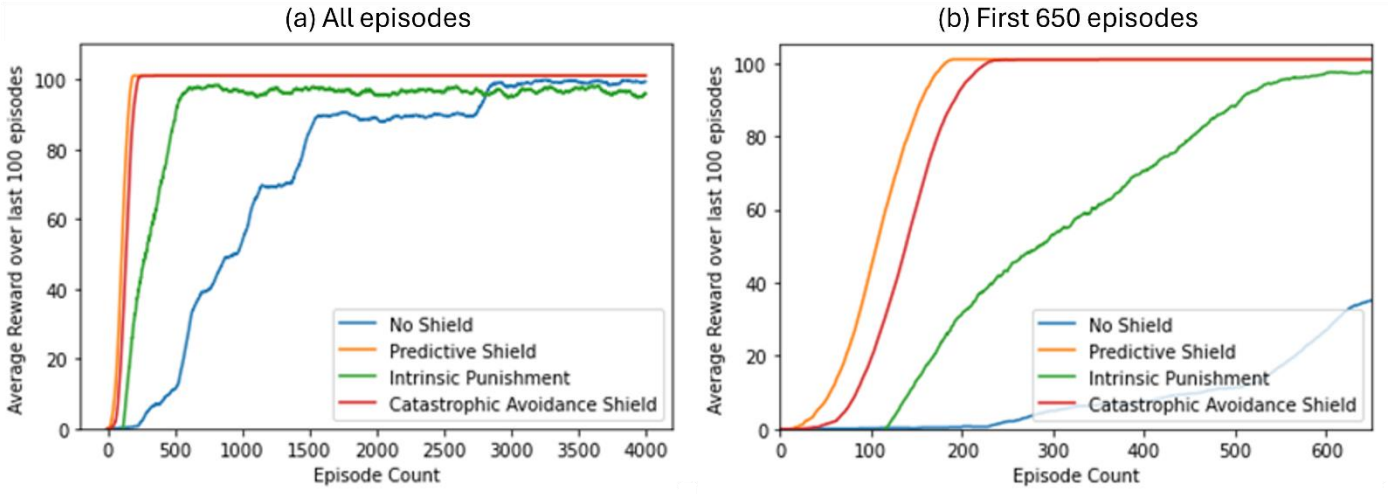


Figure 16: Rewards received in an 8x8 Frozenlake environment (10 run average) for (a) all episodes and (b) a close-up for the first 650 episodes.

Standard 8x8 Frozenlake map with slippery attribute

Model	Average Safety Violations per run	Average Steps per run
Predictive Shield	5,940.9	**613,261.2**
Catastrophic Shield	**2,872.6**	768,058.5
Intrinsic Punishment	4,722.6	692,858.5
No Shield	5,386.7	658,482.8

Table 8: Results for a slippery 8x8 Frozenlake environment (10 run average).

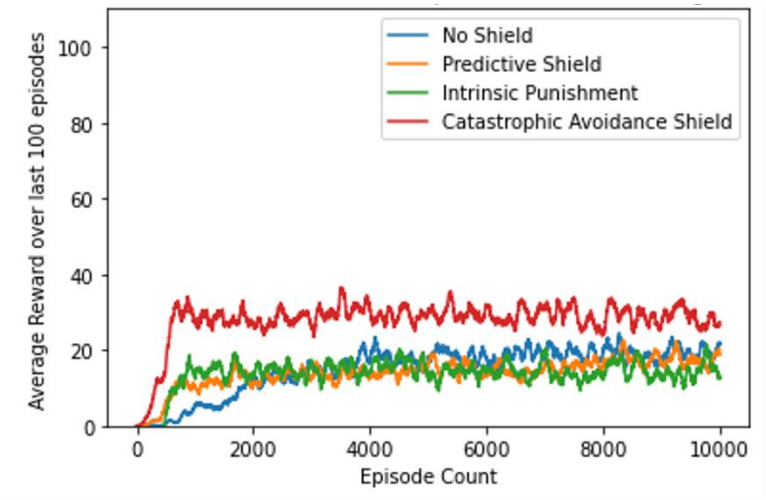


Figure 17: Rewards received in a slippery 8x8 Frozenlake environment (10 run average).

Extended 10x10 Frozenlake Map

Model	Average Safety Violations per run	Average Steps per run
Predictive Shield	302	510,982.8
Catastrophic Shield	**40**	480,578.3
Intrinsic Punishment	1,690	464,461.9
No Shield	5,108	**462,211.1**

Table 9: Results for a 10x10 Frozenlake environment (10 run average).

Model	Average Safety Violations per run	Average Steps per run
Predictive Shield	53	531,995.2
Catastrophic Shield	**38.6**	477,705
Intrinsic Punishment	1,608.6	464,421
No Shield	3,337	**458,224.6**

Table 10: Top five lowest safety violation results in a 10x10 Frozenlake environment.

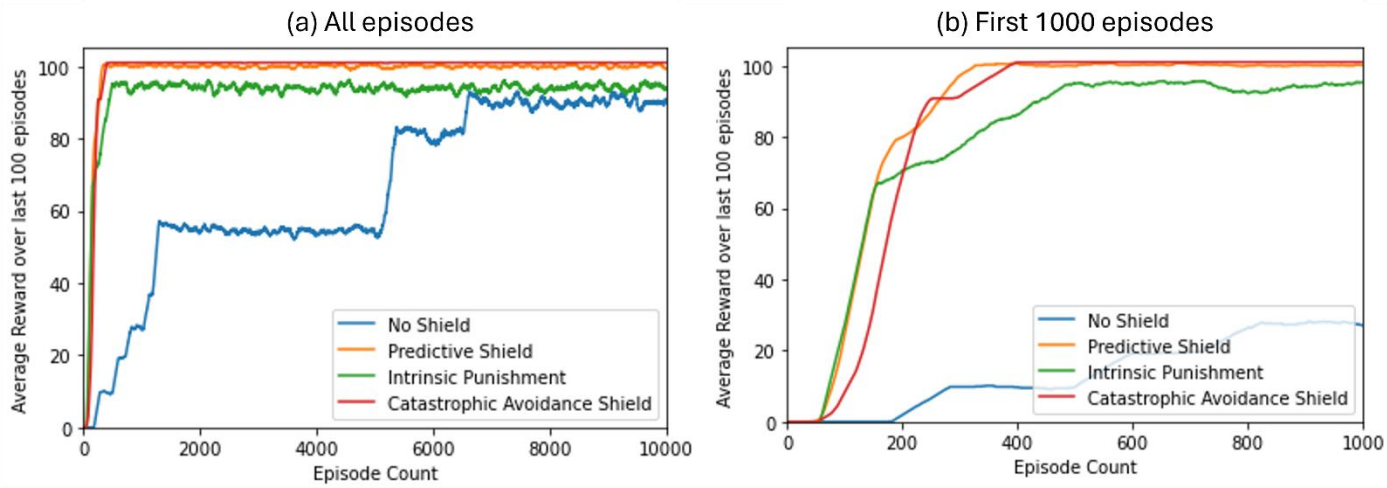


Figure 18: Rewards received in a 10x10 Frozenlake environment (10 run average) for (a) all episodes and (b) a close-up for the first 650 episodes.

Deep Learning Shield Results – Pacman Environment

Simple Pacman map

Below are the results of using the Naïve Bayes and Neural Network shields (both implemented as part of this work) on the simple Pacman map. The results are shown over all 120,000 episodes for three runs.

Run	Unshielded SV	Naïve Bayes Shield SV	Neural Network Shield SV
1	37,079	96,866	34,460
2	95,999	60,138	34,499
3	100,144	97,354	**26,422**
Mean	77,741	84,786	31,794
Std. Dev.	28,801.89	17,429.91	3,798.38

Table 11: Safety violations on the Simple Pacman Map for the Naïve Bayes and Neural Network Shields.

Run	Unshielded Wins	Naïve Bayes Shield Wins	Neural Network Shield Wins
1	82,494	22,506	83,341
2	23,610	59,485	85,020
3	18,433	21,813	**93,320**
Mean	41,512	34,601	87,227
Std. Dev.	29,055.38	17,597.68	4,362.59

Table 12: Episode Wins on the Simple Pacman Map for the Naïve Bayes and Neural Network Shields.

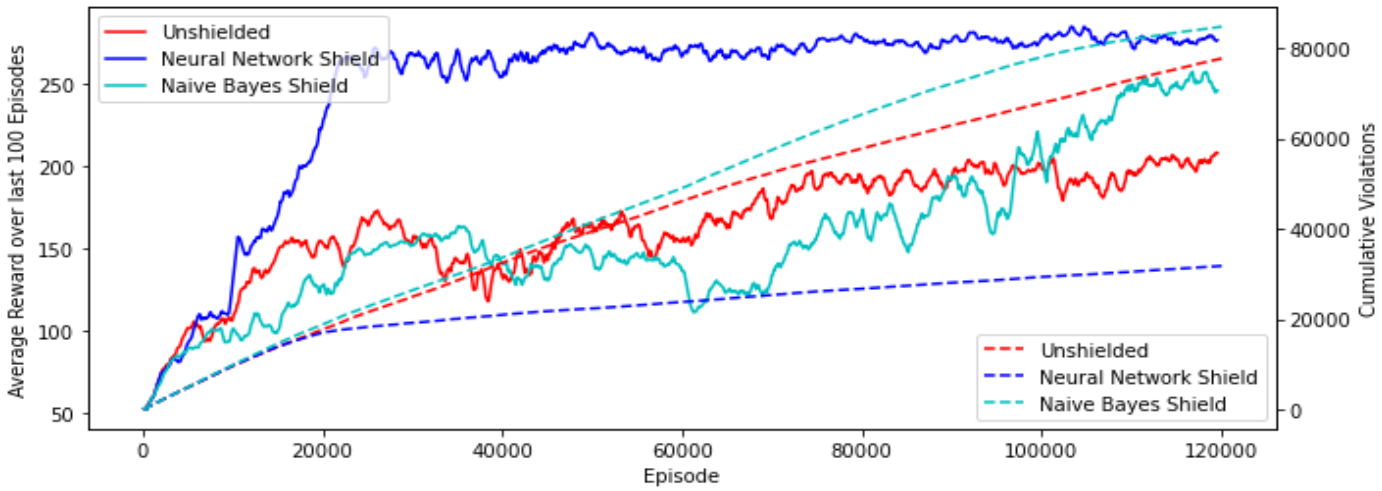


Figure 19: Reward and cumulative safety violations on the Simple Pacman map.

Advanced Pacman Map

Below are the results of using the Naïve Bayes and Neural Network shields on the advanced Pacman map. The results are shown over all 200,000 episodes for three runs.

Run	Unshielded SV	Naïve Bayes Shield SV	Neural Network Shield SV
1	35,721	45,093	**33,973**
2	100,013	42,912	53,210
3	42,098	72,667	68,370
Mean	59,277	53,557	51,851
Std. Dev.	28,921.88	13,541.88	14,075.36

Table 13: Safety violations on the Advanced Pacman Map for the Naïve Bayes and Neural Network Shields.

Run	Unshielded Wins	Naïve Bayes Shield Wins	Neural Network Shield Wins
1	655	2,065	6,325
2	1	642	**19,290**
3	0	0	1
Mean	219	902	8,539
Std. Dev.	308.53	862.90	8,028.77

Table 14: Episode Wins on the Advanced Pacman Map for the Naïve Bayes and Neural Network Shields.

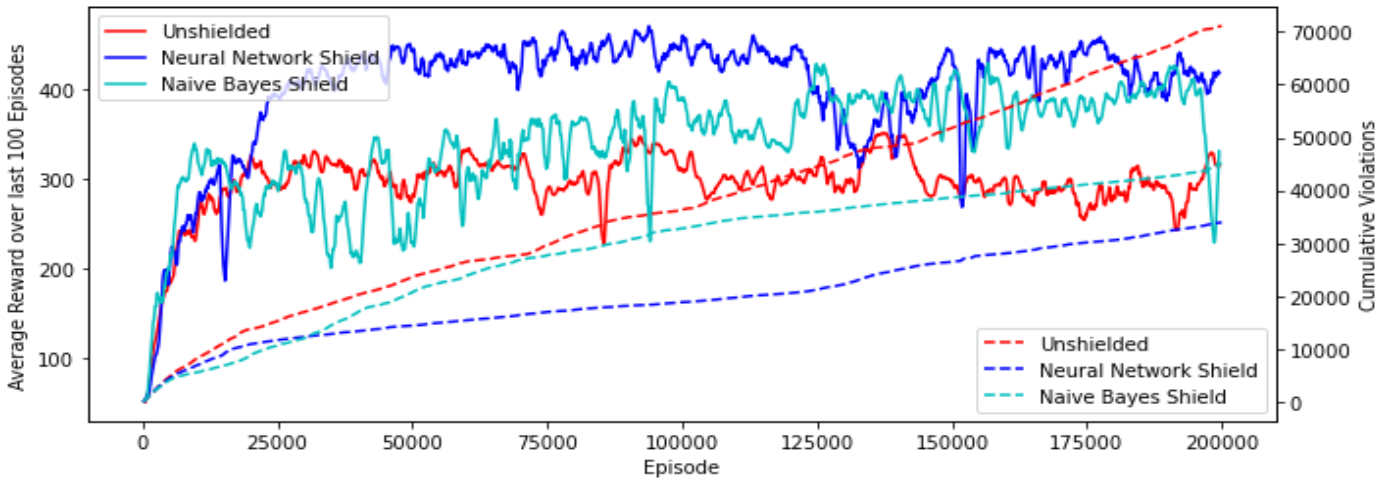


Figure 20: Reward and cumulative safety violations on the Advanced Pacman map.

Discussion

Tabular Shields – Frozenlake Environment

For the Frozenlake environment the predictive shield worked as intended, i.e. reducing the number of times unsafe states were entered, in the standard 8x8 Frozenlake map. The predictive shield achieved 12.9 Average Safety Violations (ASV) per run (see Table 7). This was an improvement on the catastrophic shield (Shperberg et al., 2022) that gave 30.6 ASV. Intrinsic punishment was also relatively effective, cutting ASV by around 70% with 377.9 ASV compared to 1,055.1 ASV for no shielding, however, out the shielding methods it performed by far the worst.

Figure 16 shows the reward over the last 100 episodes for each of the methods (where reaching the goal state receives a reward of 1), with a value 100 representing no unsafe states being reached i.e. there are no safety violations or truncations of the episode due to the maximum number of 200 steps being taken. Due to the e-

greedy exploration policy, random actions are taken with a 2% chance and hence in the un-shielded case, a value of 100 cannot be reached, as even with a fully trained policy, unsafe actions can still be taken. With the predictive and CAS, a value of 100 is achieved as once all the hazard locations (predictive) or unsafe state-action pairs (CAS) are learnt the unsafe states will no longer be visited. With intrinsic punishment a small negative reward is given when an un-safe state is visited which means it converges slightly lower than the unshielded case. Note that the maximum number of steps is not reached for any of the Frozenlake test cases presented here.

All three safety techniques including the two shields converged faster than the unshielded agent as was expected per the work of Jansen et al. (2020). The predictive shield performed better than the other methods tested, in terms of both the ASV and the reward convergence rate, as it correctly estimates the environment dynamics by recording additional data about the absolute location of hazards (melt ponds). This enabled correct prediction as to when the agent would move into an unsafe state and change course accordingly, hence the number of ASVs are reduced compared to other methods.

The reward for the predictive shield converges to 100 faster than the next best technique (the CAS) as the CAS saves unsafe state-action pairs, rather than information about the location of hazards, so will never take the same unsafe state-action choice again but can enter the same state from a different state (i.e. approaches a hazard from a different location). This gives the predictive shield an edge over the CAS, enabling safer exploration for the agent.

In terms of Average Steps per run (AS), which is the total amount of actions the agent takes in each training run, the lower the number the faster the objective is achieved. Using this metric, it appears that intrinsic punishment performed best, with the other methods (including unshielded) have similar performance to one another. These results are somewhat counterintuitive as the shielded methods are expected to have a lower AS.

In the 8x8 slippery environment (see Table 8) the CAS performs the best with 2,872.6 ASV. The predictive shield performed the worst at 5,940.9 ASV, which was even worse than an unshielded agent. This is because of the unpredictability of the agents' movements in the predictive shield made it impossible to consistently predict the next state correctly, which leads not only unsafe states being blocked in some cases, but also potentially safe actions being blocked and unsafe ones being forced to be taken instead. This resulted in the predictive shield being detrimental to the agent. The CAS likely performed better due to the labelling of all state-action pairs that could potentially lead to an unsafe state as unsafe. Therefore, the CAS could be said to be overly cautious, which is supported by it having the highest AS.

As shown in Figure 17, the average reward from the CAS agent was around 30 per hundred episodes, whereas the other methods achieved mostly below 22 per hundred episodes, with the unshielded getting a higher average reward. While CAS has the highest reward it is still very low, especially compared to the non-slippery deterministic environment.

The larger 10x10 map has 100 states, compared to the 64 states of the 8x8 map. Table 9 shows that the CAS performed in the safest manner on average with 40 ASV, with 302 for the predictive shield, 1,690 for intrinsic punishment and 5,108 for unshielded. This was surprising, as although the predictive shield performed well compared to unshielded agent and the agent with intrinsic punishment, it had five times higher ASV than the CAS.

On closer inspection of the predictive shield results for each of the ten runs (see Appendix C) this appears to be due to the shield development. In half of the runs, the shield was at first making incorrect predictions (resulting in around 650 ASV per run), which caused unsafe states to be visited in error. This was corrected as the run progressed and more information was added to the database resulting in correct predictions being made. This

was due to not having any sensors that would account for being at the edge of the map, e.g. a movement to the left would give no change in the state if the agent were at the left edge of the map. The positions of the hazards made this likely to be the case, so the dynamics were incorrectly learnt in some cases.

In the other half of the runs, the average results of which are shown in Table 10, the predictive shield only came out as 37% worse in terms of ASV than the CAS, rather than 655% worse if all ten runs were used. The predictive shield had a best 5 run average of 53 vs. 38.6 for the CAS.

In terms of the reward curves (Figure 18), the predictive shield converges faster than the CAS, however, the CAS has a slightly higher average reward than the predictive shield. This is likely due to the poorly performing runs as discussed earlier.

Deep Learning Shields – Pacman Environment

For the simple Pacman environment, Table 11 shows the safety violations and Table 12 the wins. The neural network shield performed the best in terms of both safety violations and wins. Compared to the baseline of the unshielded run, the neural network has 60% less ASVs and more than twice as many wins. Surprisingly, the Naïve Bayes shield performed slightly worse than the unshielded case.

The superior performance of the neural network shield is also shown in the reward curve (Figure 19). For this environment, picking up a piece of food gives 1 reward and entering an unexplored state gives a reward of 0.1 to encourage exploration. The episode ends when all of the food is collected: in the simple map there is one piece of food and in the advanced map there are two pieces. For the simple map, the neural network shield converges to about 270 while the other methods do not fully converge within the 120,000 training episodes.

It was expected that the neural network would perform better than Naïve Bayes, based on the accuracy of the data analysis undertaken when developing the methodology (see Table 6 for the neural network vs. Table 5 for the Naïve Bayes). In practice, this accuracy level means that the neural network shield did not result in many false positives whereas the Naïve Bayes is likely to be triggering the shield and blocking potentially safe actions. This would hinder the progress of the agent in the environment and result in a lower reward for agent using the Naïve Bayes shield. This reasons for this will be discussed further in the context of the Advanced Pacman Map.

For the advanced Pacman environment, Table 13 shows the safety violations and Table 14 the wins. In the advanced map, with added stationary firepits to be dealt with, the agent struggled greatly, even with an extra eighty thousand training episodes added. In terms of safety violations, the unshielded agent had the highest average (around 60 thousand) with a high variability between the runs. The Naïve Bayes and neural network performed similarly with each having just over 50 thousand safety violations. Whilst there is variability, the shielded agents have a lower standard deviation of safety violations, suggesting that adding the shield adds an element of consistency to the very variable RL training. This was also true for the simple map.

The significant difference between the shielding methods can be seen better in the number of wins. The neural network was the most successful, reaching the goal state in all three runs, winning the episode many thousands of times in two of those runs (with a maximum of 19,290). The Naïve Bayes and unshielded both consistently struggled to achieve this level of performance with the Naïve Bayes completing 2,065 episodes out of 200,000 at best.

The Naïve Bayes shield performed better than the baseline unshielded case, with an average of 902 episode wins to the unshielded agents 219. Although the agent with the Naïve Bayes shield had lower ASVs, when runs with similar win rates are compared, the unshielded agent had a lower violations, suggesting that the Naïve Bayes shield was in fact, not very effective. The same was true for the Neural Network shield with the run which only had one win having sixty eight thousand safety violations.

Figure 20 shows the reward curves for each of the cases. Due to there being several runs in which the win rate was low, some cherry-picking of the data was required for this plot to allow a meaningful analysis. In the cases where there were 0 wins, it indicates the agent was never able to find the goal state, due to the difficult of the advanced map (noting this did not happen for the simple map).

The neural network shield outperforms both the unshielded and Naïve Bayes shield by earning higher rewards and reaching a convergence rate earlier, however, there is a dip in the rewards around 125,000 episodes. This indicates there is likely some instability in the training algorithm. This could potentially be addressed through the use of more runs with more episodes.

The strength of the neural network comes from the multiple hidden layers, as well as the ReLU function enabling non-linear relationships between input variables to be learnt, as well as correlations between these variables. Naïve Bayes on the other hand uses Bayes Theorem (Equation 5), which while effective in classification problems has several weakness. The first is that independence of variables is assumed. This is not the case in this environment, as the x, y coordinates of the Pacman and the Ghosts will have a direct correlation. The second is that complex non-linear relationships between variables are not able to be extracted by the Naïve Bayes algorithm. It was thought that the strength of the algorithm could make up for this shortcoming in the relatively simplistic Pacman environment (in either the simple or advanced map), but this not the case. For these reasons, Naïve Bayes is deemed unsuitable as a tool for shielding purposes in a reinforcement learning environment. Jansen et al. (2020) used iterative weakening in his shield work which was implemented by lowering the safety threshold of the shield when progress of the agent was deemed to have stalled. This could be implemented to improve the performance of the Naïve Bayes shield.

The extra compute to implement both forms of shielding was substantial, with the three training runs with the Naïve Bayes shield taking six hours, the neural network shielded approach taking almost eight hours but the unshielded training runs only taking three hours. This is a clear downside of this technique and any upsides of this must be balanced with the extra training time required.

Chapter 5: Conclusions and Future Work

Conclusion

This work considered two hypotheses: the first (H1) postulated that an agent with a dynamic shield would give better results than an unshielded agent and the second (H2) that the use of a dynamic shield would result in fewer unsafe states being visited by an agent than in the unshielded case. These hypotheses have been tested in game environments (Frozenlake and Pacman) which have been implemented with different base RL algorithms (tabular vs. deep learning) and varying degrees of complexity to allow the testing of the shields under different conditions.

For the tabular reinforcement learning method implemented in the Frozenlake environment, the predictive shield outperformed the unshielded agent and both of the baseline safety methods. However, in the stochastic "slippery" environment the predictive shield performed worse than both the unshielded agent and the baseline safety methods. This indicates that the predictive shield works best in a deterministic environment.

For the deep learning reinforcement method implemented in the Pacman environment, the neural network based shield outperformed the unshielded and the Naïve Bayes shield. This is due to the neural networks ability to capture complex relationships between variables including where variables are correlated.

As all of the tested shields received safety violations during training, they fall under the safety level 1, soft constraints. While the shields in this work help to guide the agent in some instances, there are no guarantees of safety. This is the case with all model-free algorithms due to the "cold-start" problem.

While the test environments used were relatively simple, the neural network shield proved to be effective and could be used in larger, and potentially more complex, environments. This could be in support of simulated agents acting in a digital-twin environment which would prepare the agent for eventual deployment in real life environments.

Future Work

There are many ways this work could be expanded on. While the experiments completed have shown a high potential for the use of the predictive shield in simple environments and indicated that the neural network based shield has potential in more complex environments, further tests could be done both in the Pacman and other environments to evaluate it potential usefulness. Processing power was a limiting factor in this work. Ideally at least ten runs of five hundred thousand episodes would be done on the advanced Pacman map to obtain more informative results and see where the reward curves truly converge for each method.

For Naïve Bayes shield, a potential improvement would be to give more weight to the action taken, perhaps over representing as more than one variable. This could assist in the classification of the safety of the action. This would only be advisable in smaller environments however, due to the discussed limitations of the algorithm.

For the Neural Network shield, a lookahead style prediction (as per the work of Könighofer et al. (2023)) could be implemented that would assess each of the available safe actions and judge them based on the safety of the next state arrive in. This would be done through collection of data from past actions in a similar way to the current neural network shield is implemented.

In conclusion, this work demonstrates the potential of classifiers to be used as dynamic shields. While the Naïve Bayes did not achieve great success, the neural network shield showed promising results worthy of future development. These findings contribute to the field of safety in RL and support the advancement of shielding techniques in game environments.

References

Achiam, J., Held, D., Tamar, A. and Abbeel, P., 2017, July. Constrained policy optimization. In International conference on machine learning (pp. 22-31). PMLR.

Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S. and Topcu, U., 2018, April. Safe reinforcement learning via shielding. In *Proceedings of the AAAI conference on artificial intelligence AAAI-18 (Vol. 32, No. 1) (AAAI-18)*, 2nd to 7th February 2018, New Orleans, USA.

Altman, E., 1999. Constrained Markov Decision processes. Vol 7. CRC Press.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J. and Mané, D., 2016. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*.

Artificial intelligence: Google's AlphaGo beats Go master Lee Se-dol. BBC. 2016.Accessible from: https://www.bbc.co.uk/news/technology-35785875 [Last accessed 3.9.24]

Baier, C. and Katoen, J.P., 2008. *Principles of model checking*. MIT press.

Bastani, O., Li, S. and Xu, A., 2021, July. Safe Reinforcement Learning via Statistical Model Predictive Shielding. In *Robotics: Science and Systems* (pp. 1-13).

Bellman, R. E., 1957. *Dynamic Programming*. Dover

Bellman, R. E., 1966. Dynamic programming. *science*, *153*(3731), pp.34-37.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C. and Józefowicz, R., 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Bethell, D., Gerasimou, S., Calinescu, R. and Imrie, C., 2024. Safe Reinforcement Learning in Black-Box Environments via Adaptive Shielding. Version 1, Submitted 28th May 2024. *arXiv preprint arXiv:2405.18180*.

Bharadhwaj, H., Kumar, A., Rhinehart, N., Levine, S., Shkurti, F. and Garg, A., 2021. Conservative safety critics for exploration. In *Proceedings of 9th International Conference on Learning Representations (ICLR 2021)*, 3rd to 7th May 2021, Virtual Event, Austria.

Bloem, R., Könighofer, B., Könighofer, R. and Wang, C., 2015, April. Shield synthesis: Runtime enforcement for reactive systems. In *International conference on tools and algorithms for the construction and analysis of systems* (pp. 533-548). Berlin, Heidelberg: Springer Berlin Heidelberg.

Brafman, R.I. and Tennenholtz, M., 2002. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, *3*(Oct), pp.213-231.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "OpenAI gym." *arXiv preprint arXiv:1606.01540* (2016).

Brown, N. and Sandholm, T., 2019. Superhuman AI for multiplayer poker. Science, 365(6456), pp.885-890.

Brunke, L., Greeff, M., Hall, A.W., Yuan, Z., Zhou, S., Panerati, J. and Schoellig, A.P., 2022. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, *5*(1), pp.411-444.

Bubeck, S., Chadrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y.T., Li, Y., Lundberg, S. and Nori, H., 2023. *Sparks of artificial general intelligence: Early experiments with gpt-4* [online]

Carr, S., Jansen, N., Junges, S. and Topcu, U., 2023, June. Safe reinforcement learning via shielding under partial observability. In *Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 37, No. 12, pp. 14748-14756) (AAAI-18)*, 7th to 14th February 2023, Washington DC, USA.

Coraluppi, S.P. and Marcus, S.I., 1999. Risk-sensitive and minimax control of discrete-time, finite-state Markov decision processes. *Automatica*, *35*(2), pp.301-309.

Fisac, J.F., Lugovoy, N.F., Rubies-Royo, V., Ghosh, S. and Tomlin, C.J., 2019, May. Bridging Hamilton-Jacobi safety analysis and reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)* (pp. 8550-8556). IEEE.

Fix, E., 1985. *Discriminatory analysis: nonparametric discrimination, consistency properties* (Vol. 1). USAF school of Aviation Medicine.

Geibel, P., 2001, June. Reinforcement learning with bounded risk. In *Proceedings of the Eighteenth International Conference on Machine Learning,* pp. 162-169, 28th June to 1st July 2001.

Geibel, P. and Wysotzki, F., 2005. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, *24*, pp.81-108.

Giacobbe, M., Hasanbeig, M., Kroening, D. and Wijk, H., 2021. Shielding Atari games with bounded prescience. In *Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems, (AAMAS '21)*, 3rd to 4th May 2021, Virtual.

Goel, A., Gautam, J. and Kumar, S., 2016, October. Real time sentiment analysis of tweets using Naive Bayes. In *2016 2nd international conference on next generation computing technologies (NGCT)* (pp. 257-261). IEEE.

Goodall, A.W. and Belardinelli, F., 2023a. Approximate shielding of Atari agents for safe exploration. In *Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '23),* 29th to 30th May 2023, London, UK.

Goodall, A.W. and Belardinelli, F., 2023b. Approximate Model-Based Shielding for Safe Reinforcement Learning. In *ECAI 2023* (pp. 883-890). IOS Press.

Goodall, A.W. and Belardinelli, F., 2024. Leveraging Approximate Model-based Shielding for Probabilistic Safety Guarantees in Continuous Environments. In *Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '24),* 6th to 10th May 2024, Auckland, New Zealand.

Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y., 2016. *Deep learning* (Vol. 1, No. 2). Cambridge: MIT press.

Grok, X., 3. Beta—The Age of Reasoning Agents| xAI, 2025. *URL https://x. ai/news/grok-3*.

Hadsell, Raia, Sumit Chopra, and Yann LeCun. "Dimensionality reduction by learning an invariant mapping." In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, vol. 2, pp. 1735-1742. IEEE, 2006.

Hasanbeig, M., Abate, A. and Kroening, D., 2020. Cautious reinforcement learning with logical constraints. In *Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '20),* 9th to 13th May 2020, Auckland, New Zealand.

Hasselt, H., 2010. Double Q-learning. *Advances in neural information processing systems*, *23*.

Van Hasselt, H., Guez, A. and Silver, D., 2016, March. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 30, No. 1).

He, C., León, B.G. and Belardinelli, F., 2021. Do androids dream of electric fences? safety-aware reinforcement learning with latent shielding. In *Proceedings of the Workshop on Artificial Intelligence Safety 2022 (SafeAI 2022)*, 24th to 25th July 2022, Vienna, Austria.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. and Silver, D., 2018, April. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence AAAI-18 (Vol. 32, No. 1) (AAAI-18)*, 2nd to 7th February 2018, New Orleans, USA.

Hinton, G.E., 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hochreiter, S., 1997. Long Short-term Memory. *Neural Computation MIT-Press*.

Hochreiter, S., 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*(02), pp.107-116.

Householder, A.S., 1941. A theory of steady-state activity in nerve-fiber networks: I. Definitions and preliminary lemmas. *The bulletin of mathematical biophysics*, *3*, pp.63-69.

Hubel, D.H. and Wiesel, T.N., 1963. Shape and arrangement of columns in cat's striate cortex. *The Journal of physiology*, *165*(3), p.559.

Jansen, N., Könighofer, B., Junges, S., Serban, A. and Bloem, R., 2020. Safe reinforcement learning using probabilistic shields. In *31st International Conference on Concurrency Theory (CONCUR 2020)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.

Könighofer, B., Rudolf, J., Palmisano, A., Tappler, M. and Bloem, R., 2023. Online shielding for reinforcement learning. *Innovations in Systems and Software Engineering*, *19*(4), pp.379-394.

Kotz, S., Balakrishnan, N. and Johnson, N.L., 2019. *Continuous multivariate distributions, Volume 1: Models and applications* (Vol. 1). John Wiley & sons.

Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

Kullback, Solomon, and Richard A. Leibler. "On information and sufficiency." *The annals of mathematical statistics* 22, no. 1 (1951): 79-86.

Kumar, A., Zhou, A., Tucker, G. and Levine, S., 2020. Conservative Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, *33*, pp.1179-1191.

Kupferman, O. and Vardi, M.Y., 2001. Model checking of safety properties. *Formal methods in system design*, *19*, pp.291-314.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. Nature, 521(7553), pp.436-444.

Lin, L.J., 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, *8*, pp.293-321.

Lockwood, M.K., 2006. Introduction: Mars Science Laboratory: the next generation of Mars landers. *Journal of Spacecraft and Rockets*, *43*(2), pp.257-257.

Lu, Y., Fu, J., Tucker, G., Pan, X., Bronstein, E., Roelofs, R., Sapp, B., White, B., Faust, A., Whiteson, S. and Anguelov, D., 2023, October. Imitation is not enough: Robustifying imitation with reinforcement learning for

challenging driving scenarios. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 7553-7560). IEEE.

Meyn, S.P. and Tweedie, R.L., 2012. *Markov chains and stochastic stability*. Springer Science & Business Media.

Mitchell, I.M., Bayen, A.M. and Tomlin, C.J., 2005. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on automatic control*, *50*(7), pp.947-957.

Mitta, R., Hasanbeig, H., Wang, J., Kroening, D., Kantaros, Y. and Abate, A., 2024, March. Safeguarded Progress in Reinforcement Learning: Safe Bayesian Exploration for Control Policy Synthesis. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 38, No. 19, pp. 21412-21419).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wisteria, D & Riedmiller, M., 2013. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. nature, 518(7540), pp.529-533.

Moldovan, T.M. and Abbeel, P., 2012. Safe exploration in Markov decision processes. *In Proceedings of the International Conference on Machine Learning*, 26th June to 1st July, 2012, Edinburgh, UK.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A. and Schulman, J., 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, *35*, pp.27730-27744.

Pnueli, A., 1977, October. The temporal logic of programs. In *18th annual symposium on foundations of computer science (sfcs 1977)* (pp. 46-57). ieee.

Pranger, S., Könighofer, B., Tappler, M., Deixelberger, M., Jansen, N. and Bloem, R., 2021, May. Adaptive shielding under uncertainty. In *2021 American Control Conference (ACC)* (pp. 3467-3474). IEEE.

Puterman, M.L., 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Rao, K., Harris, C., Irpan, A., Levine, S., Ibarz, J. and Khansari, M., 2020. Rl-cycleGAN: Reinforcement learning aware simulation-to-real. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 11157-11166).

Ray, A., Achiam, J. and Amodei, D., 2019. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, *7*(1), p.2.

Sahami, M., Dumais, S., Heckerman, D. and Horvitz, E., 1998, July. A Bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop* (Vol. 62, pp. 98-105).

Saunders, W., Sastry, G., Stuhlmueller, A. and Evans, O., 2017. Trial without error: Towards safe reinforcement learning via human intervention. *In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS'18)*, 10th to 15th July, 2018, Stockholm, Sweden.

Schaul, T., Quan, J., Antonoglou, I. and Silver, D., 2015. Prioritized Experience Replay. arXiv e-prints, art. *arXiv preprint arXiv:1511.05952*.

Schulman, J., 2015. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:1889-1897*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sharpe, W.F., 1966. Mutual fund performance. *The Journal of business*, *39*(1), pp.119-138.

Shperberg, S.S., Liu, B. and Stone, P., 2022. Learning a shield from catastrophic action effects: never repeat the same mistake. arXiv preprint arXiv:2202.09516.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T. and Lillicrap, T., 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, *362*(6419), pp.1140-1144.

Signal Processing Society., 2017. OpenAI Bot Defeated The World's Best Dota 2 Player. Accessible from: https://signalprocessingsociety.org/newsletter/2017/09/openai-bot-defeated-worlds-best-dota-2-player [Last accessed 3.9.24]

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), pp.1929-1958.

Srinivasan, K., Eysenbach, B., Ha, S., Tan, J. and Finn, C., 2020. Learning to be safe: Deep RL with a safety critic. *arXiv preprint arXiv:2010.14603*.

Stuart, A. and Ord, K., 1994. Kendall's Advanced Theory of Statistics, Distribution Theory (Volume 1).

Sunilkumar, A., Bahrpeyma, F. and Reichelt, D., 2024, August. Positioning Stabilization With Reinforcement Learning for Multi-Step Robot Positioning Tasks in Nvidia Omniverse. In *2024 IEEE 22nd International Conference on Industrial Informatics (INDIN)* (pp. 1-8). IEEE.

Sutton, R.S. and Barto, A.G., 1998. Introduction to reinforcement learning. Vol. 135.

Sutton, R.S., 2018. Reinforcement learning: an introduction. *A Bradford Book*.

Tamar, A., Di Castro, D. and Mannor, S., 2012, June. Policy gradients with variance related risk criteria. In *Proceedings of the twenty-ninth international conference on machine learning* (pp. 387-396).

Thananjeyan, B., Balakrishna, A., Nair, S., Luo, M., Srinivasan, K., Hwang, M., Gonzalez, J.E., Ibarz, J., Finn, C. and Goldberg, K., 2021. Recovery rl: Safe reinforcement learning with learned recovery zones. *IEEE Robotics and Automation Letters*, *6*(3), pp.4915-4922.

Thomas, P., Theocharous, G. and Ghavamzadeh, M., 2015, February. High-confidence off-policy evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 29, No. 1).

Vencl, S., 1979. The Origins of Weapons. On the questions of recognizability of prehistoric arms. Archelogicke.

Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P. and Oh, J., 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *nature*, *575*(7782), pp.350-354.

Wagenmaker, A., Huang, K., Ke, L., Jamieson, K. and Gupta, A., 2024. Overcoming the Sim-to-Real Gap: Leveraging Simulation to Learn to Explore for Real-World RL. *Advances in Neural Information Processing Systems*, *37*, pp.78715-78765.

Watkins, C.J.C.H., 1989. Learning from delayed rewards.

Wirsing, M., 1990. Handbook of theoretical computer science. Temporal and Modal Logic Chapter 16.

White, D.J., 1985. Real applications of Markov decision processes. *Interfaces*, *15*(6), pp.73-83.

Yang, W.C., Marra, G., Rens, G. and De Raedt, L., 2023a. Safe reinforcement learning via probabilistic logic shields. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI-23)*, 19th to 25th August, 2023, Macao, S.A.R.

Yang, Y., Jiang, Y., Liu, Y., Chen, J. and Li, S.E., 2023b. Model-free safe reinforcement learning through neural barrier certificate. *IEEE Robotics and Automation Letters*, *8*(3), pp.1295-1302.

Zhao, W., He, T. and Liu, C., 2021, June. Model-free safe control for zero-violation reinforcement learning. In *5th Annual Conference on Robot Learning*.

Appendix A: Compute Environment

The compute environment used to run the experiments presented in this work is as follows:

Processor: Intel Core i5-8250U CPU - 1.6 GHz to 1.8 GHz

RAM: 8 GB

Hard Drive: 118 GB

Appendix B: Implementation Scripts

Filename	Chapter Referenced
Frozen Lake - DynaQ & Double Q-learning.ipynb	Chapter 2
Frozen Lake - Catastrophic Shield and Intrinsic Punishment.ipynb	Chapter 2
Pacman DQN vs DDQN.ipynb	Chapter 2
Naive Bayes Classifier - Data Testing.ipynb	Chapter 2
Neural Network - Data Testing.ipynb	Chapter 2
Logic Shield.ipynb	Chapter 4
Logic Shield Large Map.ipynb	Chapter 4
Logic Shield Slippery.ipynb	Chapter 4
Pacman - Naive Bayes Shield with DDQN.ipynb	Chapter 4
Pacman - Neural Network Shield.ipynb	Chapter 4
Pacman - Advanced Map with Naive Bayes Shield.ipynb	Chapter 4
Pacman - Advanced Map with Neural Network Shield.ipynb	Chapter 4

Appendix C: Frozenlake Additional Data

Run	Predictive Shield		Catastrophic Shield		Intrinsic Punishment		No Shield	
Unit	SV	AS	SV	AS	SV	AS	SV	AS
1	14	65,545	27	63,753	410	57,602	373	62,185
2	12	65,760	31	67,355	402	57,365	1,362	80,800
3	13	65,080	27	64,614	401	57,598	1,053	74,123
4	14	64,976	33	64,835	307	57,985	2,698	10,557
5	12	64,086	30	63,174	391	57,216	1,020	73,662
6	13	61,533	33	67,354	401	57,575	782	70,156
7	13	62,906	33	64,598	326	57,873	656	67,972
8	11	62,344	31	63,031	354	57,711	1,463	81,592
9	13	63,949	28	62,681	390	57,305	550	66,736
10	14	65,272	33	68,420	397	57,245	594	66,700
Average	**13**	**64,145**	**31**	**64,982**	**378**	**57,548**	**1,055**	**65,448**

Table A.1: Standard 8x8 Frozenlake Results – 4,000 episodes per training run

Run	Predictive Shield		Catastrophic Shield		Intrinsic Punishment		No Shield	
Unit	**SV**	**AS**	**SV**	**AS**	**SV**	**AS**	**SV**	**AS**
1	6,447	598,882	2,937	761,235	4,768	689,422	5,841	637,714
2	5,653	626,915	2,846	771,498	4,756	694,041	5,277	661,089
3	5,835	629,442	2,911	760,304	4,582	700,187	5,548	644,833
4	5,687	612,886	2,873	769,588	4,826	689,368	5,565	644,393
5	5,980	610,967	2,668	776,871	4,671	694,215	4,901	676,678
6	5,585	624,766	2,815	770,604	4,730	692,015	4,962	683,923
7	5,450	639,734	2,984	764,790	4,721	691,578	5,095	680,355
8	5,704	619,564	2,828	771,629	4,713	689,772	5,316	655,194
9	7,564	535,228	2,927	765,631	4,758	692,649	5,570	642,520
10	5,504	634,228	2,937	768,435	4,701	695,338	5,792	658,129
Average	**5,941**	**613,261**	**2,873**	**768,059**	**4,723**	**692,859**	**5,387**	**658,483**

Table A.2: Slippery 8x8 Frozenlake Results – 10,000 episodes per training run

Run	Predictive Shield		Catastrophic Shield		Intrinsic Punishment		No Shield	
Unit	**SV**	**AS**	**SV**	**AS**	**SV**	**AS**	**SV**	**AS**
1	52	532,160	42	479,342	1,900	465,296	3,804	459,409
2	659	475,138	42	493,054	1,905	465,130	2,886	458,705
3	53	531,555	40	480,369	1,523	464,884	7,404	469,373
4	51	531,972	39	474,807	1,678	464,251	7,288	466,095
5	678	482,417	40	477,119	1,689	463,994	2,946	455,567
6	44	531,261	40	481,822	1,613	464,790	7,343	467,823
7	65	533,028	40	480,820	1,599	464,679	8,511	470,323
8	113	542,492	36	476,058	1,725	463,843	3,497	457,306
9	651	475,763	38	480,172	1,630	464,144	3,852	457,374
10	651	474,042	41	482,220	1,640	463,608	3,552	460,136
Average	**302**	**510,983**	**40**	**480,578**	**1,690**	**464,462**	**5,108**	**462,211**

Table A.3: 10x10 Frozenlake Results – 25,000 episodes per training run

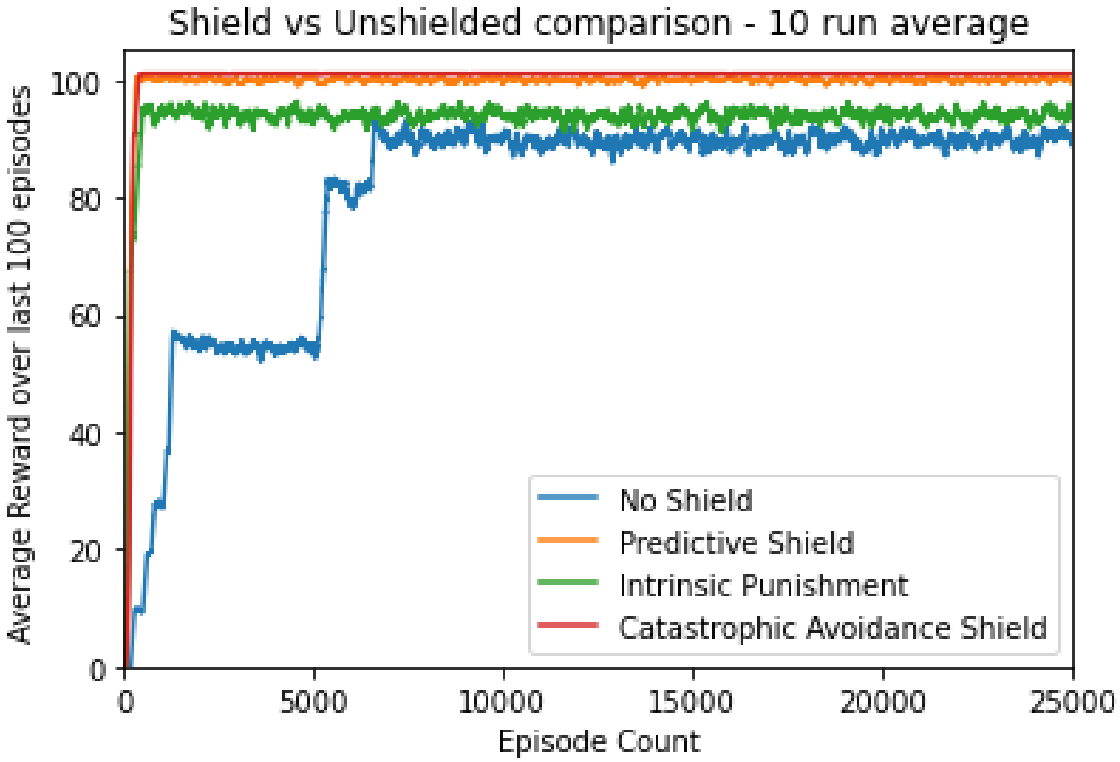


Figure A.1: 10x10 Frozenlake Rewards graph – 25,000 episodes per training run