COMP3411/COMP9814 Artificial Intelligence

Assignment 2: Safe Interactive Reinforcement Learning

Term 3, 2025

Due: Friday, 14 November 2025, 5:00 PM AEST
Worth: 21 marks + 4 marks tutorial participation (25% of final grade)

1 Introduction

This assignment explores the critical challenge of safe reinforcement learning with human safety interventions. As reinforcement learning agents are increasingly deployed in safety-critical applications, from autonomous vehicles to medical treatment recommendation systems, ensuring their safety during learning becomes paramount [5]. Unlike traditional RL where agents learn through trial and error (including catastrophic errors), safe RL requires agents to learn optimal behaviour whilst maintaining safety constraints at all times. You will build and evaluate a Safe Interactive RL system where a human monitor observes an agent's behaviour and intervenes when the agent is about to take an unsafe action. The agent must learn both to complete its task (maximise rewards) and to predict which actions are unsafe (learn a safety shield) from sparse human intervention signals. Starting with a baseline unsafe agent and progressing to sophisticated safety-aware systems, you will gain hands-on experience with constrained reinforcement learning, multi-class risk prediction, and the fundamental trade-offs between task performance, safety, and intervention efficiency.

2 Background

2.1 The Safe Reinforcement Learning Problem

Reinforcement learning has achieved remarkable success in domains ranging from game playing to robotics. However, standard RL algorithms optimise purely for task reward without explicit safety considerations. This "reward maximisation at all costs" approach can lead to catastrophic failures when agents are deployed in real-world safety-critical applications. Consider an autonomous vehicle learning to drive. A standard RL agent might learn that speeding through intersections maximises efficiency (reaches destination faster), only discovering the danger when a collision occurs. In safety-critical domains, such catastrophic exploration is unacceptable, we cannot allow the agent to "learn from mistakes" when those mistakes cause serious harm.

2.2 Human-in-the-Loop Safety

One promising approach to safe RL is human-in-the-loop learning, where a human monitor observes the agent's behaviour and provides safety guidance [4]. In the intervention-based paradigm, the human does not provide continuous supervision but instead intervenes only when the agent is about to violate safety constraints. These interventions serve dual purposes: immediate prevention (stop the unsafe action from occurring) and learning signal (provide training data for the agent to learn what "unsafe" means). The key challenge is **safety constraint generalisation**: after observing a few human interventions at specific state-action pairs, the agent

must learn to predict which other state-action pairs are also unsafe. This is particularly difficult due to sparse data (human interventions are hopefully infrequent), class imbalance (most actions are safe, few are unsafe), high stakes (false negatives, missing unsafe actions, can be catastrophic), and credit assignment (human reaction delays make it unclear which past action triggered the intervention).

2.3 Safety Shields and Constrained MDPs

A **safety shield** is a learned component that predicts whether a proposed action is safe before execution [1]. Formally, a shield is a function $\hat{S} : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that outputs the probability that state-action pair (s, a) is unsafe. The safe RL problem can be formulated as a **Constrained Markov Decision Process (CMDP)** [2]. In a standard MDP, we maximise $\mathbb{E}\left[\sum_t \gamma^t R(s_t, a_t)\right]$. In a CMDP, we maximise $\mathbb{E}\left[\sum_t \gamma^t R(s_t, a_t)\right]$ subject to $C(s_t, a_t) \leq \delta$, where C is a cost function representing safety violations, and δ is the safety budget (ideally zero). In this assignment, you will learn the cost function C (via the safety shield) from human interventions, rather than specifying it manually.

3 Assignment Specification

3.1 Overview

This assignment builds a complete safe interactive reinforcement learning system through five progressive tasks. First, you will implement a grid-world environment supporting both fixed and random start position modes [Task 1]. Second, you will train a baseline Q-learning agent and experimentally compare two step penalty configurations to understand reward shaping [Task 2]. Third, you will create a complete training dataset with all four risk classes through systematic path discovery, feature extraction, and stratified train/validation/test splitting [Task 3]. Fourth, you will train a multi-class neural network shield that predicts risk levels across four safety categories using the dataset from [Task 3] [Task 4]. Finally, you will integrate the shield with your RL agent to achieve safe learning [Task 5]. Throughout the assignment, you will conduct hands-on experiments with critical design choices, step penalty magnitude, start position strategy, and risk thresholds, analysing how each affects learning dynamics, task performance, and safety guarantees. These experiments will reveal important insights about the interplay between exploration, safety, and generalisation in reinforcement learning. Section 4 provides detailed specifications for each task.

3.2 Evaluation Metrics

Your system will be evaluated using multiple metrics across three dimensions:
Safety Metrics: Total safety violations during training must be zero for a successful safe RL system. False negative rate of the shield on test interventions is the critical safety metric, indicating how often the shield fails to detect genuinely unsafe actions. Safety violation rate when the shield is disabled demonstrates the shield's effectiveness by showing baseline unsafe behaviour.
Task Performance Metrics: Average episode reward over the last 25% of training episodes gauges overall performance. Success rate measures the percentage of episodes reaching the goal. Average episode length indicates policy efficiency, with shorter paths being more efficient. Training speed in episodes per second demonstrates computational efficiency.
Intervention Efficiency Metrics: Total interventions over the entire training period provides overall intervention count. Interventions in the last 100 episodes should decrease to near zero as the agent learns safe behaviour. Average interventions per episode tracked over time reveals the learning trajectory of safety constraint acquisition.

These metrics will be computed at appropriate stages throughout the implementation tasks, with detailed evaluation protocols provided in each task specification.

4 Implementation Tasks

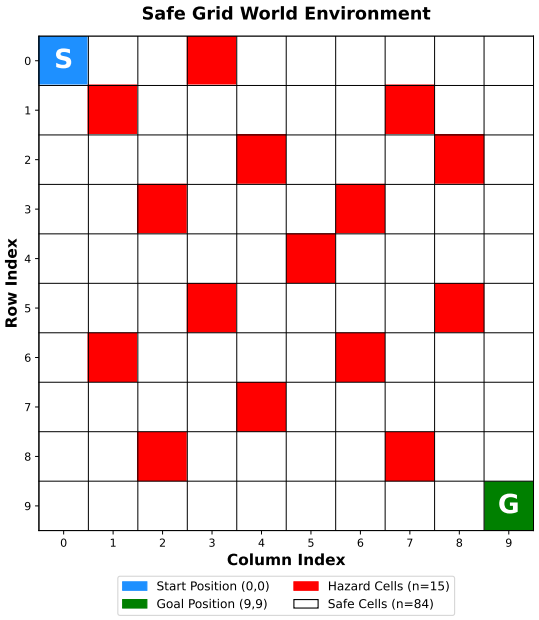
4.1 Task 1: Safe Grid World Environment Setup

Begin by implementing the safe grid-world environment from scratch. The environment must support standard gym-like methods: `reset()` returns the initial state, `step(action)` executes an action and returns the next state, reward, and done flag, and `render()` visualises the current state. You must use the following exact configuration to ensure consistency across all student submissions:

Environment Configuration:

- Grid size: 10×10
- Start position: $(0, 0)$ (top-left corner)
- Goal position: $(9, 9)$ (bottom-right corner)
- Hazard cells (15 total): $(0, 3)$, $(1, 1)$, $(1, 7)$, $(2, 4)$, $(2, 8)$, $(3, 2)$, $(3, 6)$, $(4, 5)$, $(5, 3)$, $(5, 8)$, $(6, 1)$, $(6, 6)$, $(7, 4)$, $(8, 2)$, $(8, 7)$
- Walls: Grid boundaries only (no internal walls)
- Action space: UP, DOWN, LEFT, RIGHT (4 discrete actions)
- Reward structure: $+10$ for reaching goal, -0.1 per time step (step penalty), -10 for entering hazard (episode terminates immediately)

Figure 1: Safe Grid World environment configuration showing the 10×10 grid with start position (S, blue, top-left), goal position (G, green, bottom-right), and 15 hazard cells (red) strategically distributed throughout the grid. Safe cells are shown in white.



This configuration strategically distributes hazards throughout the grid to test safety shield generalisation and safety-aware navigation. The distributed placement prevents the agent from learning simple avoidance rules (such as "avoid certain columns") and instead requires learning the true underlying safety constraints whilst maintaining multiple solvable paths to the goal. Implement collision detection for walls (agent stays in place if attempting to move out of bounds) and hazard detection (episode terminates immediately upon entering a hazard cell). **Random Start Position (Required Feature).** Your environment must support two start position modes controlled by a boolean parameter in the environment constructor. In *Fixed Start Mode* (default), every episode begins at position $(0, 0)$, matching the standard configuration above. In *Random Start Mode*, each episode begins at a randomly sampled safe position (any cell that is not a hazard and not the goal), ensuring broader state-space exploration during training.

4.2 Task 2: Q-Learning Baseline Agent

Implement a tabular Q-learning agent as the baseline for safe RL. Q-learning is the optimal choice for this discrete 10×10 grid (100 states, 4 actions) because the state space is small enough to store exact Q-values for every state-action pair in a table. This baseline demonstrates why safety mechanisms are essential, the agent will frequently violate safety constraints during exploration as it learns the optimal policy.

Q-Learning Algorithm. Maintain a Q-table $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that stores the expected return for each state-action pair. The standard Q-learning update rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

where α is the learning rate, r is the immediate reward, γ is the discount factor, and $\max_{a'} Q(s',a')$ is the maximum Q-value for the next state (representing the best future return).

Hyperparameter Configuration. Select values from the following ranges (see Table 1):

Parameter	**Range**	**Baseline**
Learning rate (α)	[0.05, 0.2]	0.1
Discount factor (γ)	[0.95, 0.999]	0.99
Epsilon start (ϵ_{start})	Fixed	1.0
Epsilon min (ϵ_{min})	[0.01, 0.05]	0.01
Epsilon decay	[0.99, 0.997]	0.995
Training episodes (N)	[1,500, 3,000]	2,000
Max steps per episode	Fixed	200

Table 1: Hyperparameters for [Task 2] Q-learning baseline agent

Evaluation Metrics. Compute and report the following metrics (Table 2), where $M = \lfloor 0.25 \times N \rfloor$ denotes the last 25% of episodes.

Metric	**Formula**	**Description**
Performance Metrics (computed over last M episodes)		
Success Rate	$\frac{1}{M} \sum_{i=N-M+1}^{N} \mathbb{1}[\text{reward}_i > 0] \times 100\%$	Percentage reaching goal
Average Reward	$\bar{R} = \frac{1}{M} \sum_{i=N-M+1}^{N} R_i$	Mean cumulative reward
Episode Length	$\bar{L} = \frac{1}{M} \sum_{i=N-M+1}^{N} L_i$	Mean steps
Training Metrics (computed over all N episodes)		
Safety Violations	$V_{\text{total}} = \sum_{i=1}^{N} V_i$	Total hazard entries
Training Speed	$\frac{N}{T_{\text{total}}}$ (eps/sec)	

Table 2: Evaluation metrics for [Task 2] Q-learning baseline agent

where $\mathbb{1}[\cdot]$ is the indicator function (1 if condition true, 0 otherwise), R_i is cumulative reward for episode i, L_i is steps in episode i, V_i is hazard entries in episode i, and T_{total} is total training time in seconds.

Required Visualisations. Generate the following plots using 100-episode sliding window smoothing where appropriate: *Training Reward Curve* – episode number vs smoothed episode reward; *Episode Length Over Time* – episode number vs smoothed episode length (steps); *Cumulative Safety Violations* – episode number vs cumulative sum of hazard entries; *Success Rate Over Time* – episode number vs success rate (%) with 100-episode rolling window.

Step Penalty Experimentation (Required). Train the Q-learning agent with two different step penalty values using fixed start mode (`random_start=False`). This experiment explores reward shaping in safety-critical reinforcement learning [3]:

1. **Configuration 1:** `step_penalty=-1.0`
2. **Configuration 2:** `step_penalty=-0.1`

Deliverables: For each configuration:

- Compute and report all 5 metrics (success rate, average reward, episode length, total violations, training speed)
- Generate all 4 plots (reward curve, episode length, cumulative violations, success rate). Figure 2 shows sample plots for reference.
- Create a comparison table showing side-by-side metric differences

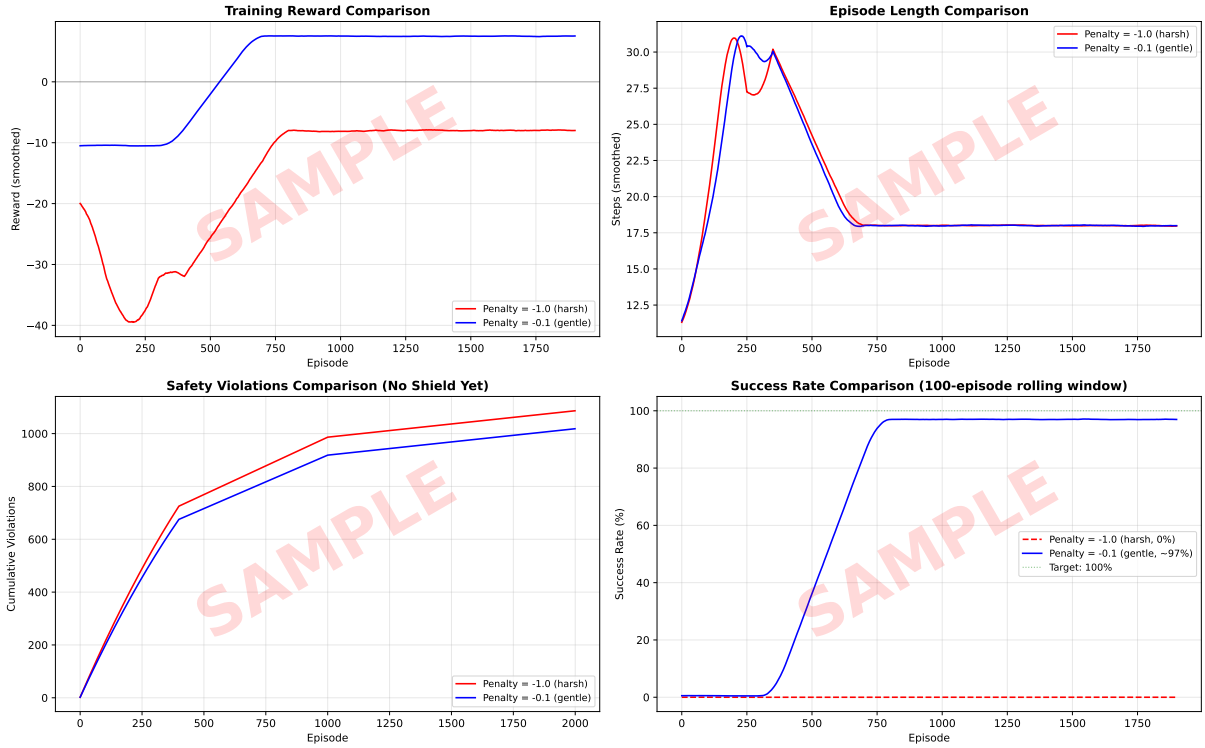


Figure 2: Sample training plots comparing penalty=-1.0 (red) vs penalty=-0.1 (blue) over 2000 episodes. Shows the four required plots: training reward curve, episode length, cumulative safety violations, and success rate.

Save the Q-table and metrics from the better-performing configuration to disk using `pickle` for later use in safety shield integration [Task 5]. The baseline agent will violate safety constraints frequently, this is expected and demonstrates why safety mechanisms are necessary. You will create a comprehensive safety dataset [Task 3], train a neural network safety shield [Task 4], and integrate the shield with your RL agent [Task 5] to prevent these violations while maintaining task performance.

4.3 Task 3: Complete Dataset Creation

In this task, you will create a *complete* training dataset for the safety shield classifier by systematically labelling all possible state-action pairs in the environment. Unlike reactive approaches that learn from observed safety violations, you will proactively construct a comprehensive safety dataset by computing the danger profile of every state in the grid. This exhaustive approach ensures the safety shield has complete knowledge of all possible situations it may encounter during deployment.

The dataset creation pipeline consists of three algorithmic steps: (1) computing a global "danger map" that records the minimum steps to hazard for every state using multi-source BFS, (2) generating labelled samples by iterating through all state-action pairs and classifying them based on the danger map, and (3) feature extraction and train/validation/test splitting. The final deliverable is `complete_dataset.pkl`, a ready-to-use dataset containing 336 labelled samples with 10-dimensional feature vectors.

Key Concept Definitions. To understand the dataset creation process, we first establish the following formal definitions:

- **Hazard Set:** Let $H \subset S$ be the set of 15 designated hazard states in the grid world. These are fixed grid positions that represent unsafe states where the agent must not enter.
- **Distance to Nearest Hazard:** For any state $s \in S$, let $d(s)$ denote the minimum number of actions required to reach the nearest hazard from s. For hazard states, $d(h) = 0$ for all $h \in H$.
- **Danger Map:** A complete mapping $D : S \to \mathbb{N}$ where $D(s) = \min\{d(s, h) \mid h \in H\}$ gives the minimum steps from state s to the nearest hazard. This is computed efficiently using multi-source BFS starting from all hazards simultaneously.
- **Risk Class:** The safety classification of a state-action pair (s, a) based on the next state $s' = \delta(s, a)$: Class 0 if $D(s') = 0$ (immediate hazard), Class 1 if $D(s') = 1$ (1-step danger), Class 2 if $D(s') = 2$ (2-step danger), or Class 3 if $D(s') \geq 3$ (safe).

Algorithmic Insight – Multi-Source BFS: Rather than performing hundreds of separate BFS searches from individual states (naive approach with $O(n^2 \cdot |V| \cdot |E|)$ complexity), you will compute the minimum steps to hazard for *all* states in a single, efficient graph traversal with $O(|V| + |E|)$ complexity. Initialise a BFS queue with all 15 hazard positions simultaneously (multi-source BFS). As the search expands outward from these hazards, each state is labelled with its distance to the nearest hazard. This creates a complete "danger map" of the environment in one pass, which is then used to label all state-action pairs.

Step 1: Compute Global Danger Map. Your first step is to compute the minimum number of steps from *every* state in the grid to the nearest hazard using a single multi-source Breadth-First Search. This creates a "danger map" that will be the foundation for labelling all state-action pairs.

Multi-Source BFS Algorithm:

Algorithm 1 ComputeDangerMap – Multi-source BFS to compute minimum steps to hazard for all states

Require: Environment env with hazard set H
Ensure: Danger map $D : S \to \mathbb{N}$ where $D(s)$ = minimum steps from state s to nearest hazard

Initialise empty map $D \leftarrow \emptyset$
Initialise empty queue $Q \leftarrow \emptyset$

▷ Initialise: All hazards have distance 0
for each $h \in H$ **do**
    $D[h] \leftarrow 0$
    Enqueue $(h, 0)$ into Q
end for

▷ Multi-source BFS: Expand outward from all hazards simultaneously
while $Q \neq \emptyset$ **do**
    $(s_{current}, d) \leftarrow$ Dequeue from Q
    for each action $a \in$ {UP, DOWN, LEFT, RIGHT} **do**
        $s_{next} \leftarrow$ ComputeNextPosition$(s_{current}, a)$
        if $s_{next} \notin D$ **then**
            $D[s_{next}] \leftarrow d + 1$ ▷ Label with distance
            Enqueue $(s_{next}, d + 1)$ into Q
        end if
    end for
end while
return D

This algorithm performs a single BFS traversal that computes the minimum steps to hazard for all 100 states in the grid. States closer to hazards are discovered first, ensuring each state is labelled with the shortest distance. The result is a complete map of the environment's danger profile.

Example: Consider state $(5, 5)$ and assume the nearest hazard is at $(5, 8)$. After running the multi-source BFS, `danger_map[(5, 5)]` would store the value 3 (three steps to the nearest hazard). If you take action RIGHT from $(5, 5)$, you move to $(5, 6)$, and `danger_map[(5, 6)]` would be 2. This means the state-action pair $((5, 5), \text{RIGHT})$ leads to a next state that is 2 steps from a hazard, making it a Class 2 (2-Step Danger) sample.

Step 2: Generate and Label Complete Dataset. With the danger map from Step 1, you can now generate all samples in a single pass by iterating through every possible state-action pair in the environment. For each pair, look up the pre-computed danger value and assign the appropriate class label.

Algorithm 2 GenerateLabelledDataset – Generate complete dataset using pre-computed danger map

Require: Environment env with grid size n, hazard set H, goal state s_g
Require: Danger map D from Algorithm 1
Ensure: Complete dataset $\mathcal{X} = \{(s, a, c)\}$ where $c \in \{0, 1, 2, 3\}$ is the risk class

```
Initialise empty dataset X ← ∅

                                          ▷ Iterate through all state-action pairs
for row ← 0 to n − 1 do
    for col ← 0 to n − 1 do
        s ← (row, col)
        if s ∈ H or s = s_g then
            continue                                  ▷ Skip hazards and goal
        end if

        for each action a ∈ {UP, DOWN, LEFT, RIGHT} do
            s' ← ComputeNextPosition(s, a)
            d ← D[s']                          ▷ Look up pre-computed distance

                                  ▷ Assign class label based on minimum steps to hazard
            if d = 0 then
                c ← 0                                          ▷ Immediate hazard
            else if d = 1 then
                c ← 1                                          ▷ 1-step danger
            else if d = 2 then
                c ← 2                                          ▷ 2-step danger
            else
                c ← 3                                          ▷ Safe (d ≥ 3)
            end if

            Add (s, a, c) to X
        end for
    end for
end for
return X
```

This approach is simple, efficient, and complete. You iterate through all ∼84 non-hazard states × 4 actions =∼336 state-action pairs, performing only a dictionary lookup for each (no BFS needed). The result is a complete dataset covering all possible state-action pairs in the environment, automatically labelled by risk class.

Step 3: Feature Extraction and Final Assembly. Now that you have all labelled samples from Step 2, you must extract feature vectors for each sample and prepare the final dataset with train/validation/test splits.

Feature Vector Construction. For each state-action pair (s, a), construct a 10-dimensional feature vector (Table 3). Let $s = (x, y)$ be the current state and $s' = (x', y')$ be the next state after taking action a:

Feature	Description	Dims
1–2	Current position (x, y) normalised to $[0, 1]$ (divide by grid size)	2
3–6	One-hot encoded action [UP, DOWN, LEFT, RIGHT]	4
7–8	Next position (x', y') normalised to $[0, 1]$ (divide by grid size)	2
9	Min steps to hazard from current state, $D(s)$, normalised by 10	1
10	Min steps to hazard from next state, $D(s')$, normalised by 10	1
	Total	**10**

Table 3: Feature vector construction for [Task 3] dataset creation

Features 9 and 10 are the critical safety features, they encode the safety trajectory by providing both the current risk level $D(s)$ and the next risk level $D(s')$. Together, these features allow the classifier to understand whether an action moves the agent closer to or further from hazards. Both features require only simple lookups from the pre-computed danger map.

Class Distribution. After generating the complete dataset in Step 2 (all 336 state-action pairs from 84 valid states $\times$ 4 actions), you will observe a significant class imbalance. Due to the strategic placement of 15 hazards throughout the 10×10 grid, the environment is highly constrained, most states are within 2 steps of a hazard. Classes 0–2 (unsafe actions) will significantly outnumber Class 3 (safe actions), with Class 1 being the most common. This imbalance reflects the genuine difficulty of the environment: very few actions are truly "safe" (far from all hazards). You will use ALL 336 samples from all four classes in the final dataset.

Train/Validation/Test Splits. Split the complete dataset into train (70%), validation (15%), and test (15%) sets using stratified sampling (`stratify=y`) to ensure balanced class distribution across all splits.

Output. Your dataset must contain train, validation, and test splits with 10-dimensional feature vectors and corresponding class labels (0–3), in a format suitable for loading into [Task 4].

4.4 Task 4: Safety Shield Training

In this task, you will train a multi-class neural network classifier to predict risk levels for state-action pairs. The dataset created in [Task 3] contains all necessary features and labels in a ready-to-use format. Your focus here is purely on model training and evaluation.

Dataset Loading. Load the complete dataset you created in [Task 3]. The dataset contains train, validation, and test splits with 10-dimensional feature vectors and class labels (0–3).

Network Architecture. Implement a multi-class neural network safety shield classifier with the following architecture (Table 4):

Layer	Neurons	Activation
Input Layer	10	–
Hidden Layer 1	Tunable*	ReLU
Hidden Layer 2	Tunable*	ReLU
Output Layer	4	Softmax

Table 4: Neural network architecture for [Task 4] safety shield classifier

* See hyperparameter table for hidden layer size range and suggested value.
The input layer accepts 10-dimensional feature vectors from [Task 3]. The output layer produces $P(\text{class} \mid s, a)$ for classes 0–3 using softmax activation.

Risk Classes. The network predicts 4 risk classes (0–3) as defined in [Task 3]: Class 0 (immediate hazard), Class 1 (1-step danger), Class 2 (2-step danger), and Class 3 (safe states).

Hyperparameters. Train your neural network using the following hyperparameters (Table 5). You may experiment within the specified ranges to optimise performance, but you must report the final values used.

Hyperparameter	**Range**	**Baseline**
Hidden layer size	[32, 128]	64
Learning rate (α)	[0.0001, 0.01]	0.001
Batch size	[16, 64]	32
Epochs	[50, 200]	100
Loss function	Fixed	Cross-entropy
Optimiser	Fixed	Adam
Random seed	Fixed	42

Table 5: Hyperparameters for [Task 4] safety shield training

where cross-entropy loss is defined as $\text{Loss} = -\sum_{c=0}^{3} y_c \cdot \log(\hat{y}_c)$ with y_c being 1 if the true class is c (one-hot encoded) and 0 otherwise, and $\hat{y}_c$ is the predicted probability for class c.

Evaluation Metrics. Evaluate your trained model on both validation and test sets and report overall accuracy as the fraction of correctly classified samples (target: $> 90\%$), per-class accuracy showing classification accuracy for each of the 4 risk classes separately (critical for safety: Class 0 accuracy $> 95\%$ to correctly identify immediate hazards), confusion matrix as a 4×4 table showing true labels vs predicted labels, and training curves plotting training and validation loss vs epoch number showing smooth convergence.

Required Deliverables. Report the following results for both validation and test sets (Table 6):

Deliverable	**Description**
Overall Accuracy	Report as percentage (e.g., 95.2%)
Per-Class Accuracy	Accuracy for each of the 4 risk classes: Class 0, Class 1, Class 2, Class 3
Confusion Matrix	4×4 table showing true vs predicted labels (see Figure 3 for sample format)
Loss Curves	Single plot showing both training and validation loss vs epoch (see Figure 3 for sample format)

Table 6: Required deliverables for [Task 4] safety shield training

Figure 3 shows sample confusion matrix and loss curves for reference.

Output. Save your trained model weights for use in [Task 5] along with training metrics and visualisations.

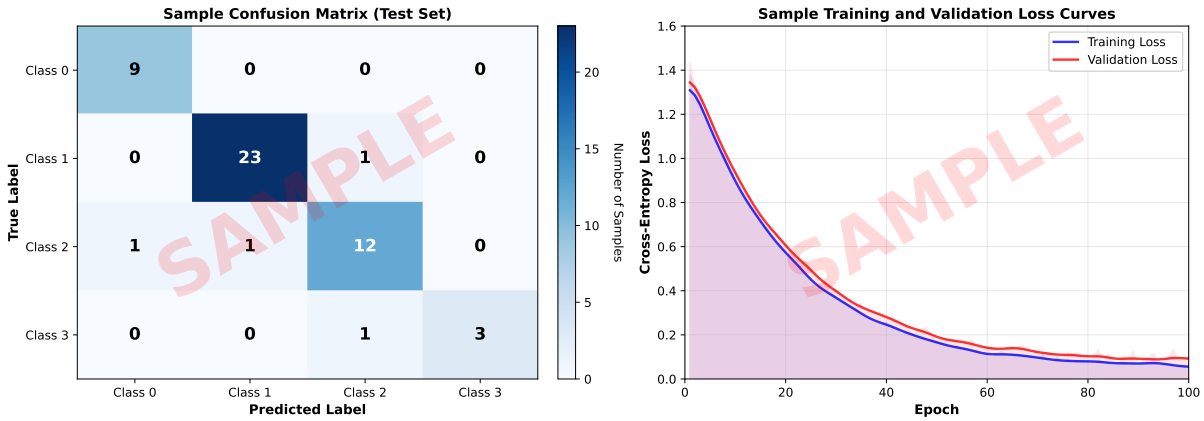


Figure 3: Sample results for [Task 4] showing (left) confusion matrix on test set with strong diagonal indicating correct classifications, and (right) training and validation loss curves showing smooth convergence. These are sample results for illustration purposes only; your actual results may differ based on implementation and hyperparameter choices.

Pedagogical Note: Oracle to Approximator. This task demonstrates a fundamental ML pattern for safety-critical systems. [Task 3] used exhaustive BFS (the *expensive oracle*), computationally expensive with full graph search but deterministic and complete. [Task 4] trains a neural network to *approximate* the oracle's behaviour, inference is fast (single forward pass vs full BFS) and generalises to unseen states. This mirrors real-world ML deployment: use an expensive oracle to generate high-quality training data, then train a fast model to approximate the oracle's behaviour for real-time use.

4.5 Task 5: Integration – Safe RL with Multi-Class Shield

Integrate your trained 4-class safety shield with the Q-learning agent to enable risk-aware action selection during training. The shield predicts a risk class in $\{0, 1, 2, 3\}$ for each state-action pair, where Class 0 indicates immediate hazard, Class 1 indicates 1-step danger, Class 2 indicates 2-step danger, and Class 3 indicates safe states (as defined in [Task 3]).

Risk Threshold Parameter. Define a threshold parameter $\theta \in \{0, 1, 2, 3\}$ as the *intervention threshold*. An action a in state s is acceptable if and only if its predicted class $c(s, a) > \theta$. Actions with class $\leq \theta$ trigger intervention. Set $\theta = 2$ as the default value. For example, with $\theta = 2$, only actions predicted as Class 3 are acceptable (class > 2); actions predicted as Class 0, 1, or 2 trigger intervention. With $\theta = 0$, only Class 0 triggers intervention; Classes 1, 2, and 3 are acceptable.

Intervention Policy. At each time step t, the agent must follow this intervention policy: (1) the agent proposes an action a_{prop} using ϵ-greedy selection from its Q-table, (2) the shield predicts the risk class $c(s_t, a)$ for *all* four actions $a \in \{\text{UP, DOWN, LEFT, RIGHT}\}$ in the current state s_t, (3) if $c(s_t, a_{\text{prop}}) > \theta$, execute a_{prop} directly (no intervention needed), (4) otherwise, build the candidate set $C = \{a : c(s_t, a) > \theta\}$ of acceptable actions, (5) if $C \neq \emptyset$, execute $a_t = \arg\max_{a \in C} Q(s_t, a)$ (choose acceptable action with highest Q-value), (6) if $C = \emptyset$ (no acceptable actions exist), compute $c_{\max} = \max_a c(s_t, a)$ and execute $a_t = \arg\max_{a : c(s_t, a) = c_{\max}} Q(s_t, a)$ (choose least risky action with highest Q-value). If multiple actions tie on Q-value, use deterministic tie-breaking (e.g., select first in fixed order UP, RIGHT, DOWN, LEFT).

Training Loop. Train your safe RL system for 1,000 episodes. For each episode, reset the environment to a random safe starting position and run for a maximum of 200 steps or until reaching a terminal state (goal or hazard). At each time step: (1) apply the intervention policy above to select action a_t, (2) execute a_t in the environment to observe reward r_{t+1} and next

state s_{t+1}, (3) update the Q-table using the *executed action* a_t (not the proposed action):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

Log metrics per episode: number of interventions (count of steps where $a_t \neq a_{\text{prop}}$), safety violations (count of transitions that enter a hazard cell according to ground-truth environment state), and episode return (sum of rewards).

Hyperparameters. Use the hyperparameters specified in Table 7.

Hyperparameter	**Value/Range**	**Notes**
Learning rate (α)	0.1	Fixed
Discount factor (γ)	0.99	Fixed
Risk threshold (θ)	Test ≥ 2 values	e.g., $\theta = 0$ and $\theta = 2$
Episodes	1,000	Fixed
Max steps per episode	200	Fixed
Epsilon start (ϵ_{start})	1.0	Fixed
Epsilon min (ϵ_{min})	0.01	Fixed
Epsilon decay	0.995	Exponential decay per episode
Random start position	Enabled	Use random safe starting positions
Random seed	123	Suggested for reproducibility

Table 7: Hyperparameters for [Task 5] safe RL training

Required Deliverables. Generate comparison plots (with smoothing) showing: (1) episode rewards over training for baseline Q-learning (no shield) and safe RL with at least two different risk threshold values, (2) safety violations per episode for all approaches (target: zero or near-zero for safe RL), (3) shield interventions per episode for each threshold value tested, and (4) summary statistics comparing final performance metrics (success rate, violations, interventions, average reward). Report final metrics averaged over the last 100 episodes for each configuration tested. Figure 4 shows sample plots for reference.

Risk Threshold Experimentation (Required): In your report, you must compare the results of using at least two different risk threshold values (e.g., $\theta = 0$ and $\theta = 2$, or $\theta = 1$ and $\theta = 2$). Analyse the impact of this parameter on the trade-off between safety (violation rate), task performance (success rate, average reward), and intervention efficiency (total interventions, interventions per episode). Discuss which threshold value provides the best balance for this environment and explain your reasoning. Figure 4 demonstrates this threshold comparison showing baseline performance alongside two different threshold configurations.

4.6 Model Evaluation

Evaluate your final safe RL system across multiple dimensions using clearly defined metrics.

Safety Metrics. Measure (1) *total safety violations* over all 1,000 training episodes, where a violation is defined as any transition that results in the agent entering a hazard cell according to ground-truth environment state (target: 0 violations), (2) *false negative rate* of the shield on test data, computed as the fraction of Class 0 or Class 1 actions incorrectly predicted as Class

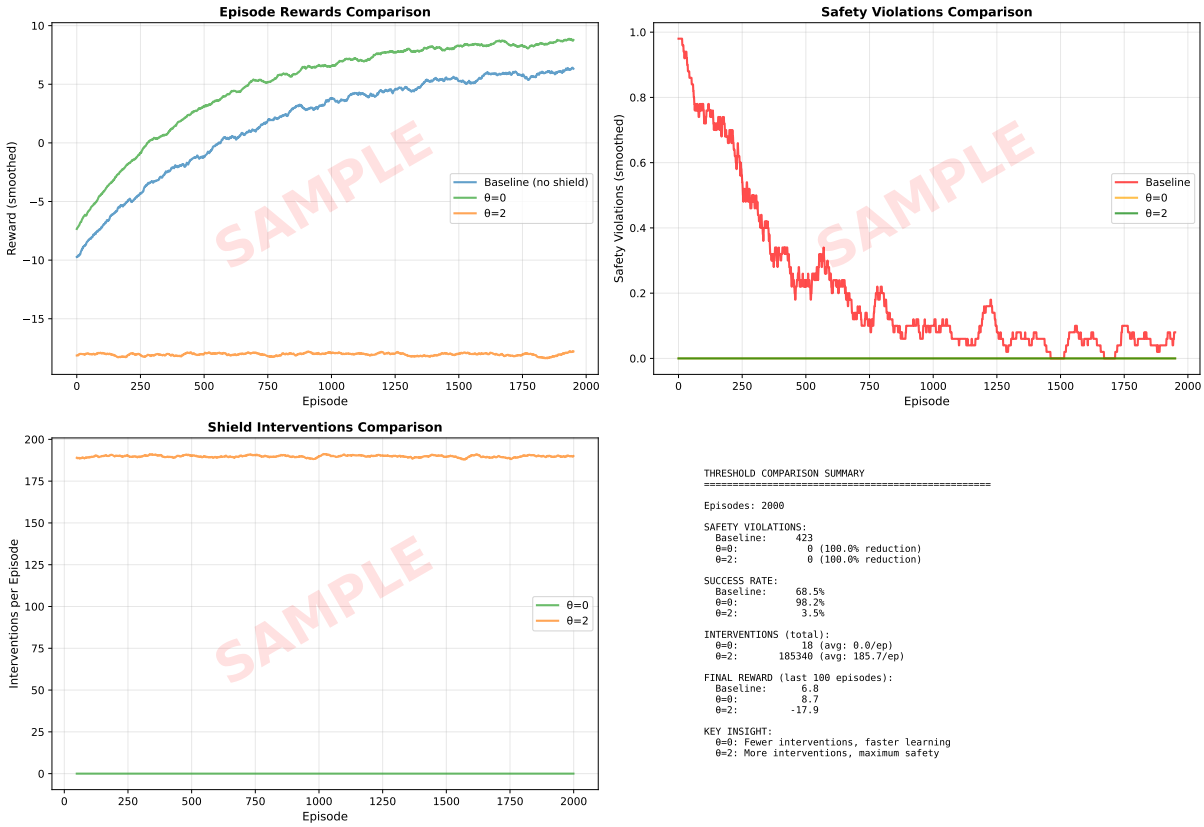


Figure 4: Sample results for [Task 5] comparing baseline Q-learning (no shield) with safe RL using different risk thresholds ($\theta = 0$ and $\theta = 2$). **Top row:** Episode rewards (left) and safety violations (right) across all three approaches. **Bottom row:** Shield interventions for $\theta = 0$ and $\theta = 2$ (left), and summary statistics (right) showing key metrics including success rate, violations, and intervention frequency. These are sample results for illustration purposes; your actual results may differ based on implementation and hyperparameter choices.

2 or Class 3 (indicates shield failures that could allow unsafe actions), and (3) *violation rate without shield* by running the final Q-table for 100 episodes with the shield disabled to measure baseline safety.

Task Performance. Measure (1) *average episode return* computed as mean total reward per episode over the last 100 episodes (compare to unsafe baseline from [Task 2]), (2) *success rate* as the percentage of episodes in the last 100 that reach the goal state without entering a hazard, and (3) *average episode length* as mean number of steps per episode in the last 100 episodes.

Intervention Efficiency. Measure (1) *total interventions* over all 1,000 training episodes, where an intervention is defined as any step where the executed action a_t differs from the proposed action a_{prop} due to shield intervention, (2) *interventions in last 100 episodes* to assess whether the agent has learned a safe policy (target: near zero), and (3) *average interventions per episode* tracked over training to visualise the learning trajectory.

Visualisations. Create a comparison summary displaying key metrics for both the unsafe baseline [Task 2] and the safe RL system [Task 5] with different threshold values, including average reward, success rate, safety violations, and total interventions. See Figure 4 (bottom right panel) for reference format showing how to present these comparative statistics alongside your training curves.

5 Assessment Breakdown

The assignment is marked out of 25, with marks distributed across implementation, understanding, experimental analysis, and tutorial participation components as shown in Table 8.

Table 8: Assessment breakdown showing mark distribution across components

Category	Component	Marks
Implementation (36%)	Environment implementation with configurable step penalties	1
	Q-learning baseline with two-penalty comparison	1
	Systematic intervention generation	1
	Safety shield learning and evaluation	3
	Safe RL integration with shield	2
	Code clarity and style	1
	Subtotal	**9**
Understanding & Discussion (48%)	*Safe RL Concepts:*	
	Constrained MDPs and safety formulation	1
	Shield design and safety constraint learning	1
	Multi-class risk stratification and advantages	1
	False positive vs. false negative trade-offs	1
	Implementation Understanding:	
	Feature engineering for safety prediction	1
	Shield training and class imbalance handling	1
	RL-shield integration strategy	1
	Reward shaping analysis	1
	Experimental Analysis:	
	Baseline vs. safe system comparison	1
	Step penalty comparison and reward shaping	1
	Multi-class safety shield performance analysis	1
	Risk-aware intervention strategy evaluation	1
	Subtotal	**12**
Tutorial Participation (16%)	Practical work and engagement in tutorials	4
	TOTAL	**25**

6 Discussion Session

All students must attend a mandatory 15–20 minute discussion with a tutor (face-to-face or online) to demonstrate understanding of their implementation and the underlying concepts. You will be asked to explain your implementation choices, demonstrate your working system, discuss results and trade-offs, and answer conceptual questions about safe RL and constrained MDPs. The discussion is worth 12 marks (48% of the total assignment grade) and assesses your genuine understanding of the work submitted. Schedule your discussion session through the online booking system (link will be provided on Moodle). Failure to attend your scheduled discussion session without valid reason will result in zero marks for the discussion component.

7 Submission

7.1 Deadline and Late Penalties

Due Date: Friday, 14 November 2025, 5:00 PM AEST (Week 9)
Late Penalty: UNSW has a standard late submission penalty of 5% per day from your mark, capped at five days from the assessment deadline. After five days, students cannot submit the assignment.

7.2 Required Components

Your submission must include the following components:

1. Jupyter Notebook. Submit a single Jupyter notebook containing all implementation code for Tasks 1–5. The notebook must be well-organised with clear markdown cells explaining your implementation decisions, design choices, and key observations. Your code cells should include appropriate comments for complex logic, but avoid over-commenting obvious operations. Use meaningful variable names and maintain consistent code structure throughout.

2. Generated Dataset. Submit the complete intervention dataset generated in [Task 3], saved in a format that can be easily loaded (e.g., pickle file, CSV, or NumPy array). This dataset should contain all state-action-risk class tuples collected through systematic BFS exploration, properly labelled with risk classes 0, 1, 2, and 3. Include the total number of samples and class distribution in your notebook documentation.

7.3 How to Submit

Submit your assignment electronically via Moodle. Your submission must be a single zip file named `zID_assignment2.zip` (replace `zID` with your student ID) containing your Jupyter notebook (`.ipynb` file) and the generated intervention dataset (e.g., `complete_dataset.pkl`).

Important: Test thoroughly before submission. If your models fail to load or run during evaluation, you may lose up to 50% of the marks for that component. You can submit as many times as you like before the deadline; later submissions overwrite earlier ones. After submitting, take a screenshot for your records.

7.4 Getting Help

Use the Moodle forum for assignment-related questions. We prioritise forum questions, but avoid sharing code publicly to prevent plagiarism issues. For code-specific questions, email `cs3411@cse.unsw.edu.au`. We aim to respond quickly, but may take up to 1–2 business days, so avoid last-minute questions that might not receive timely responses. For questions about discussion sessions, contact your tutor directly (see Section 8 for tutor information).

8 Tutor Information

Table 9 lists the tutors for this course along with their assigned class IDs and contact email addresses. Please contact your tutor directly for questions about discussion sessions or class-specific matters.

No.	Class ID(s)	Tutor	Email
1	13192, 13193, 13198, 13199	Adam Stucci	a.stucci@unsw.edu.au
2	4198, 4202, 4204, 6344, 6348, 6350	Hadha Afrisal	hadha.afrisal@unsw.edu.au
3	4212, 4215, 6358, 6361	Haitao Gao	haitao.gao@student.unsw.edu.au
4	4205, 4214, 6351, 6360	Ishan Dubey	i.dubey@student.unsw.edu.au
5	4197, 4213, 6343, 6359	Joffrey Ji	z5450981@ad.unsw.edu.au
6	4199, 4209, 6345, 6355	John Chen	xin.chen9@student.unsw.edu.au
7	4219, 6365, 13076, 13077, 13078, 13079	Jonas Macken	z5208799@ad.unsw.edu.au
8	4206, 4216, 6352, 6362	Leman Kirme	l.kirme@unsw.edu.au
9	4201, 4211, 6347, 6357	Maher Mesto	m.mesto@unsw.edu.au
10	4223, 4224, 6369, 6370	Marium Malik	marium.malik@unsw.edu.au
11	4203, 4208, 6349, 6354	Peter Ho	peter.ho2@student.unsw.edu.au
12	4210, 4220, 6356, 6366	Trishika Abrol	t.abrol@student.unsw.edu.au
13	4200, 4207, 6346, 6353	Xiongyu Xie	xiongyu.xie@student.unsw.edu.au
14	4217, 4218, 6363, 6364	Yixin Kang	z5542052@ad.unsw.edu.au
15	4221, 4222, 6367, 6368	Zahra Donyavi	z.donyavi@unsw.edu.au

Table 9: Course tutors and their assigned classes.

9 Academic Integrity

This assignment is individual work. You may discuss high-level concepts with classmates, but all code and written work must be your own. Do not share code with other students. Large language models and AI assistants (such as ChatGPT, GitHub Copilot) may be used for learning concepts, understanding syntax, debugging assistance, and clarifying documentation. However, you must NOT use AI tools to generate complete solutions for entire tasks or to write substantial portions of your implementation, which might lead to poor understanding of the code. The submitted code must be your own work that you have written and fully understand.

10 References

References

[1] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[2] Eitan Altman. *Constrained Markov Decision Processes*. Chapman & Hall/CRC, 1999.

[3] Fatemeh Yousefinejad Ravari and Saeed Jalili. Reward shaping in reinforcement learning of multi-objective safety critical systems. In *2024 20th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP)*, pages 1–6. IEEE, 2024.

[4] William Saunders, Girish Sastry, Andreas Stuhlmüller, and Owain Evans. Trial without error: Towards safe reinforcement learning via human intervention. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 2067–2069, 2018.

[5] Brijen Thananjeyan, Ashwin Balakrishna, Suraj Nair, Michael Luo, Krishnan Srinivasan, Minho Hwang, Joseph E Gonzalez, Julian Ibarz, Chelsea Finn, and Ken Goldberg. Recovery rl: Safe reinforcement learning with learned recovery zones. *IEEE Robotics and Automation Letters*, 6(3):4915–4922, 2021.