

Towards Human-Level Safe Reinforcement Learning in Atari Library

Afriyadi Afriyadi^{[1]*}, Wiranto Herry Utomo^[2]

Faculty of Computing, President University^{[1],[2]}

Cikarang, Indonesia

afriyadi.it@gmail.com^[1] · wiranto.herry@president.ac.id^[2]

Abstract— Reinforcement learning (RL) is a powerful tool for training agents to perform complex tasks. However, from time-to-time RL agents often learn to behave in unsafe or unintended ways. This is especially true during the exploration phase, when the agent is trying to learn about its environment. This research acquires safe exploration methods from the field of robotics and evaluates their effectiveness compared to other algorithms that are commonly used in complex videogame environments without safe exploration. We also propose a method for hand-crafting catastrophic states, which are states that are known to be unsafe for the agent to visit. Our results show that our method and our hand-crafted safety constraints outperform state-of-the-art algorithms on relatively certain iterations. This means that our method is able to learn to behave safely while still achieving good performance. These results have implications for the future development of human-level safe learning with combination of model-based RL using complex videogame environments. By developing safe exploration methods, we can help to ensure that RL agents can be used in a variety of real-world applications, such as self-driving cars and robotics.

Keywords— reinforcement learning, videogame environment, safety constraint, safe reinforcement learning

I. INTRODUCTION

Reinforcement Learning (RL) is a subfield of machine learning that is widely used in solving decision-making problems [42]. In RL, an agent learns to interact with an environment to achieve a specific goal by taking actions that maximize a reward signal. This technique has been successfully applied in various fields, such as robotics, gaming, and finance [12, 29, 30]. However, unlike humans, ensuring safety while solving a problem is still a big challenge in RL, it is to make the agent learn without causing harm to itself or the environment [27].

Traditional RL algorithms optimize for the maximum reward, which can lead to unsafe behaviors, especially in environments with potentially harmful states. To address this issue, researchers have proposed several approaches [8, 9, 14, 15, 21, 41, 45, 52, 53], all of the research are in the scope of Safe Reinforcement Learning (*Safe-RL*), including adding safety constraints or modifying the reward function to incorporate safety. These approaches aim to maintain a balance

between achieving the maximum reward and ensuring safety.

Despite the advancements in RL algorithms, there is still a gap in developing *Safe-RL* agents that can balance between achieving the maximum reward and ensuring safety. While there have been studies on safe exploration during training and implementation in RL [49], it is mainly focused on simple environments and tasks, such as the *cart-pole* or *mountain-car* tasks. There is a need to evaluate the effectiveness of *Safe-RL* in more complex environments. In addition, there is a need to investigate the results of various training iteration quantities on the reward and safety violations and see the minimum iteration to achieve optimal results for the algorithm.

We have several motivations in this research. First is the fact that in RL research, especially in robotics, researchers are required to perform initialization of the agent and environment state after finishes each iteration or each episode. This initialization is consuming a lot of work for the researcher [55]. Second, the proposed solution to reduce labor by creating simulated environment [55] if effective, but the process of creating simulated environment from scratch itself also required significant amount of labor [10, 54]. In other hand, the videogame today, have a diverse set of almost-realistic environments that can be the learning environment for RL agents. And third, the solution of using simulated environment, and creating simulated environments does not yet include safety concerns.

We propose to develop a framework to perform robotic tasks in videogame environments, instead of developing each environment for each specific experiment. To do this we propose using Atari library as the example videogame environment, specifically with *Super-Mario-Bros* game, which is an easy-to-understand game, the game have complex environments, and require the algorithm to perform near-human like intelligence. And within the game, there are also several states that can end the game, that can be declared as unsafe state. We are expecting this research as a steppingstone to utilizing videogame environments for research purposes and reduce researcher additional work for creating simulated environments for every research. And lastly, we want to explore the impact of using the *Safe-RL* algorithm which usually implemented for

robotics and autonomous driving in videogame environment.

The goal is to explore the potentials from incorporating a safety mechanism in RL algorithms, seeking towards human-like behavior to RL agents when they are introduced with unsafe states and safety concerns, and still maintain optimal rewards during training and deployment. And we want to simulate this safety-constrained behavior in a complex videogame environment, to show the potential of utilizing videogame environments for RL research.

This research proposes a novel approach for Safe-RL by combining modified reward concepts and hand-crafted safety constraints in state-of-the-art RL algorithms, specifically *Double Deep Q-Network (DDQN)* [19], to achieve high rewards while ensuring safe exploration in the presence of unsafe states. Furthermore, this research explores the effectiveness of various training iteration quantities. Monitoring closely the rewards, and safety violations through customized evaluation metrics.

The experiment involves simulating the safety-constrained algorithm in the *Atari Library* and *gym (OpenAI Gym)* [10] environment. The *Atari Library* is a collection of 57 classic *Atari 2600* games that are used as benchmark environments for RL research. *OpenAI Gym* [10] is a toolkit provided by the OpenAI company for developing and comparing reinforcement learning algorithms. It provides several environments that can be used to train RL agents, as well as several tools for evaluating the performance of agents. The usage of *gym* in this research is to investigate whether the agent will act more safely and towards human-like behavior after being introduced with safety constraint and unsafe states compared with the default *gym* settings.

This research also includes developing and evaluating safety mechanisms in agent's exploration method during training and implementation. The evaluation will be based on the performance of the agent in achieving maximum rewards while demonstrating safe exploration in the presence of unsafe states, as well as the number of safety violations.

The research will also include the development of safety violation measurements to be applied to the agents trained using DDQN algorithm. The training iteration quantity will be varied to determine the impact of the number of iterations on the performance of the agent. Additionally, the research will explore the use of modified rewards as a method of enforcing safety constraints during training.

II. PRELIMINARIES

A. Problem Statement

The field of Reinforcement Learning (RL) has gained significant attention in recent years due to its potential for enabling agents to learn from experience and improve decision-making [42]. However, ensuring safety in RL environments remains a crucial challenge. Developing

safety mechanisms is essential to ensure the safety of both the agent and the environment. While researchers in the field of robotics and autonomous driving primarily perform Safe-RL training in simulated environments created specifically for their research, this approach adds extra work for researchers in developing *Safe-RL* agents.

In contrast, the gaming industry has been expanding rapidly and has developed many games that emulate real-world situations. Utilizing video game environments as an alternative to creating new simulation environments from scratch for each research case shows promise. This research aims to investigate the effectiveness of *Safe-RL* algorithms, commonly used in robotics and autonomous driving, in complex video game environments. By incorporating hand-crafted safety constraints and reward modification, the research aims to explore the performance and safety capability of RL agents in these complex video game environments. This initial investigation is expected to provide valuable insights into the potential of using video game environments as simulated environments for the development of human-level RL agents in real-world implementations such as robotics, autonomous driving, and other applicable domains.

The goal of this experiment is to assess whether the implementation of hand-crafted safety constraints to the algorithm will generate safer, human-like behavior while maintaining high rewards [42]. Specifically, the research will use state-of-the-art RL algorithms, such as *Double Deep Q-Network (DDQN)*, as a baseline without safety constraints. The effectiveness of these algorithms in achieving maximum rewards and the frequency of safety violations will be evaluated. The research will then develop and apply hand-crafted safety constraints to the algorithm, incorporating awareness of unsafe states to prevent agents from repeating unsafe actions or revisiting unsafe states during training and exploration. The performance of the agent with safety constraints will be compared to the performance of the baseline algorithms without safety constraints to determine the effectiveness of the safety constraints in reducing safety violations.

To measure the implementation of the hand-crafted safety constraints, a set of unique evaluation metrics will be developed. These metrics will assess the tradeoff between maximum rewards and safety violations for the RL agents in the video game environment. The performance of the safety-constrained algorithms will be compared to that of the non-safety-constrained algorithms, and the results will be presented in an easy-to-understand report.

In summary, this research aims to address the challenge of safety in RL by investigating the effectiveness of Safe-RL algorithms in complex video game environments. By incorporating hand-crafted safety constraints and reward modification, the research aims to explore the performance and safety capability of RL agents. The research objectives include evaluating the effectiveness of safety constraints, developing unique evaluation metrics,

and comparing the performance of safety-constrained and non-safety-constrained algorithms.

B. Literature Review

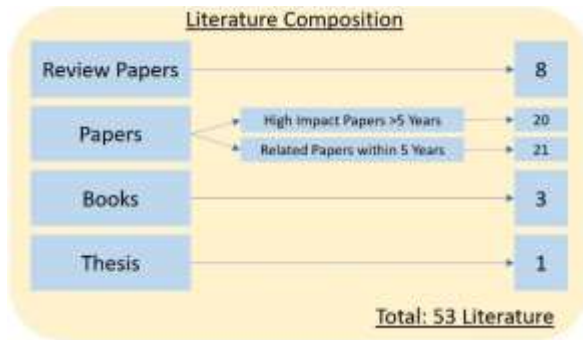


Fig. 1. We use eight recent review papers, 20 high impact papers within more than 5 years range, and relevant papers within 5 years range. We also use three books and one high impact Ph.D. thesis related to Safe-RL development. (Literature Composition)

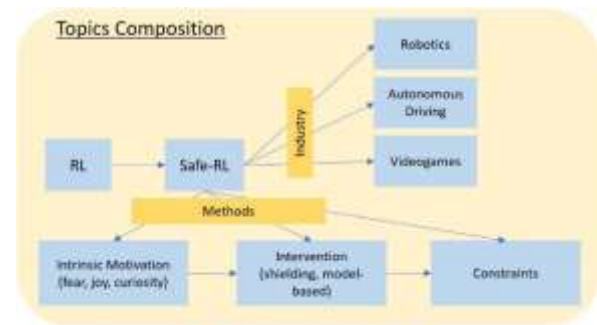


Fig. 2. Start with RL topics, then immediately pivot to Safe-RL. Categorize based on Industry, including Robotics, Autonomous Driving and Videogames. And categorize based on Safe-RL Methods, including Intrinsic Motivations, Intervention and Constraints. (Topics Composition)

We are starting by collecting the latest review papers within the field of RL and Safe-RL. Then continued with collecting information from high-impact papers mentioned in the review papers. And for deeper fundamentals, we also collect and analyze from major RL books and theses. The architecture of the literature is as follows:




We start investigating RL in general, also the challenge and opportunities that still exist in the field. Then we focused on safety concerns in RL, we investigated the safety concerns in the field of robotics, autonomous driving, and videogames. Then we investigate the common methods to implement safety within those fields, including intrinsic motivations, intervention with shielding, and putting safety constraints on the algorithm.

We also investigate the drawbacks of implementing RL in the field of robotics, autonomous driving and videogames, including the labor required to perform RL training in real-world, and how the researchers reduce this labor by performing training in simulated environments.

And how researchers started to use videogame environments to perform RL experiments.

C. Reinforcement Learning (RL)

TABLE I. The difference between Supervised Learning, Unsupervised Learning and Reinforcement Learning.

Classes of Learning Problems		
Supervised Learning	Unsupervised Learning	Reinforcement Learning
Data: (x, y) x is data, y is label	Data: x x is data, no labels	Data: state-action pairs
Goal: Learn function to map $x \rightarrow y$	Goal: Learn underlying structure	Goal: Maximize future rewards over many time steps
Strawberry Example:  This is strawberry	Strawberry Example:  These two items are similar, and should put into same category	Strawberry Example:  Eat this item to stay alive longer

Reinforcement learning (RL) is a popular machine learning approach that allows an agent to learn through interaction with an environment via trial-and-error [42]. The objective of RL is to learn a policy that maximizes the cumulative reward over time, based on state transitions and rewards observed through the agent's actions. The policy is a mapping from states to actions, and the goal is to learn the optimal policy that maximizes the expected cumulative reward. [42, 20]

Compared to supervised and unsupervised learning, RL differs in that RL learns from interaction with the environment rather than a labeled dataset. Supervised learning uses labeled examples from a human expert, while unsupervised learning learns from the underlying structure of data. [42].

RL has gained popularity due to its wide-ranging applications, including robotics, gaming, finance, healthcare, and transportation [46]. In robotics, RL has been used to train robots to perform complex tasks such as grasping and manipulation. In gaming, RL has been used to train agents to play games such as *Chess*, *Go*, and *Poker* at a superhuman level [40]. RL has also been applied in finance to optimize trading strategies and in healthcare to develop personalized treatment plans.

RL is composed of various components, such as the agent, the environment, the reward signal, the policy, and the value functions. The agent is responsible for taking actions in the environment based on its policy. The environment is the external system that the agent interacts with. The reward signal is a numerical feedback signal that the agent receives from the environment, indicating how well it is performing in the task. The policy is the strategy that the agent uses to select its actions in a given state. The

value function is a function that estimates the long-term value of a state or a state-action pair, which the agent uses to make decisions. These components are fundamental to RL and are necessary for the learning process to occur [42]



Fig. 3. RL agent in the state s , and then performing action a based on policy π , resulting in reward r and new state s' . (Components of RL)

D. Markov Decision Process

Markov Decision Processes (MDP) is a widely used framework in RL for modeling decision-making problems with uncertainty [42]. MDP provides a mathematical formulation [16] for solving sequential decision-making problems in stochastic environments.

The Markov Property assumes that the current state s has all the relevant information to predict the future and does not require any information from the past [42, 48]. A MDP is a decision process based on these assumptions. It is represented by a tuple (S, A, P, R, γ) where S denotes the set of states, A denotes the set of actions, P denotes the state transition probability function, R denotes the reward function, and γ denotes the discount factor.

The MDP model assumes that the agent interacts with the environment over a sequence of discrete time steps, $t=1,2,3,\dots,T$. At each time step, the agent observes the current state of the environment and selects an action to perform based on its policy. The environment then transitions to a new state according to the transition probability function, and the agent receives a reward based on the reward function [42, 48]. State transition probability is denoted mathematically as:

$$P_{ss'} = \mathbb{P}[s_{t+1} = s' | s_t = s] \quad (1)$$

s_t denotes the current state of the agent and s_{t+1} denotes the next state. What this equation means is that the transition from state S_t to S_{t+1} is entirely independent of the past.

When working with MDPs, the agent interacts with the environment by selecting actions that lead to a change of state and a corresponding reward. The state refers to the current state of the environment, while the action represents the decision made by the agent. The reward function is responsible for determining the reward the agent receives for taking a specific action in a particular state. Meanwhile,

the transition probability function determines the probability of moving to a new state based on the current state and the action taken by the agent [42].

Returns (G_t) is the total reward expected from the environment, and returns is denoted mathematically as:

$$G_t = r_{t+1} + r_{t+2} + \dots + r_T \quad (2)$$

The discount factor (γ) is a value between 0 and 1 that determines the importance of future rewards in the agent's decision-making process. A discount factor of 0 means that the agent only considers immediate rewards, while a discount factor of 1 means that the agent values all future rewards equally. In most cases, a discount factor between 0.9 and 0.99 is used, as it balances the importance of immediate and future rewards. The discount factor is usually applied to the future rewards in the value update equations. Returns using discount factor Function is denoted mathematically as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots + r_T = \sum_{t=0}^{\infty} \gamma^t r_t \quad (3)$$

Learning rate (α) is a value between 0 and 1 that determines how much the agent updates its value estimates based on new information. A low learning rate means that the agent updates its value estimates slowly and is more resistant to changes in the environment, while a high learning rate means that the agent updates its value estimates quickly and is more responsive to changes in the environment. The learning rate affects how quickly the agent converges to an optimal policy and how much it explores the environment. A learning rate that is too high can cause the agent to overfit to the current environment, while a learning rate that is too low can cause the agent to converge too slowly or get stuck in suboptimal policies.

E. Bellman Equations

The *Bellman equation* is a fundamental aspect of MDP that describes the relationship between the value function of the current state and the value function of the next state [42]. The value function represents the expected cumulative reward obtained by the agent over time. The *Bellman equation* is defined as follows:

Bellman proved that the optimal state value function in a state s is equal to the action a , which gives us the maximum possible expected immediate reward, plus the discounted long-term reward for the next state s' [42], denoted mathematically as:

$$V_*(s) = \max_a \sum_{s'} P_{ss'}^a (r(s, a) + \gamma V_*(s')) \quad (4)$$

Bellman also proved that the optimal state-action value

function in state s and taking action a [42], denoted mathematically as:

$$Q_*(s, a) = \sum_{s'} P_{ss'}^a (r(s, a) + \gamma \max_{a'} Q_*(s', a')) \quad (5)$$

F. Value Functions

Two types of value functions are commonly used in MDP: state-value function ($V(s)$) and action-value function ($Q(s,a)$). On the other hand, the value function or state-value function represents the expected cumulative reward obtained by the agent starting from state s . Value Function is denoted mathematically as:

$$V(s) = \mathbf{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \quad (6)$$

Q-Value (Q-Function) or action-value function represents the expected cumulative reward obtained by the agent starting from state s , taking action a and following the optimal policy thereafter [42]. Q-Function is denoted mathematically as:

$$Q(s, a) = \mathbf{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \quad (7)$$

The *advantage function* quantifies how much a particular action, given a particular situation, is a good or bad decision, or simply calculating what is the advantage of selecting a certain action from a certain state. *Advantage function* is denoted mathematically as:

$$A(s, a) = \mathbf{E}[Q(s, a) - V(s)] \quad (8)$$

G. Policy Iteration and value iterations

Policy iteration and value iteration are two widely used algorithms for solving MDPs [42]. Policy iteration is an iterative algorithm that involves two main steps: policy evaluation and policy improvement. In the policy evaluation step, the value function is computed for a given policy, while in the policy improvement step, the policy is updated to be more greedy with respect to the computed value function. The process of policy evaluation and policy improvement is repeated until the optimal policy is found. On the other hand, value iteration is a one-step lookahead algorithm that updates the value function iteratively until it converges to the optimal value function [42].

The policy is the probability of taking action a given the state s during the timestep t , denoted mathematically as:

$$\pi(a, s) = \Pr(a_t = a \mid s_t = s) \quad (9)$$

Below are two examples for better intuition of policy in RL implementations:

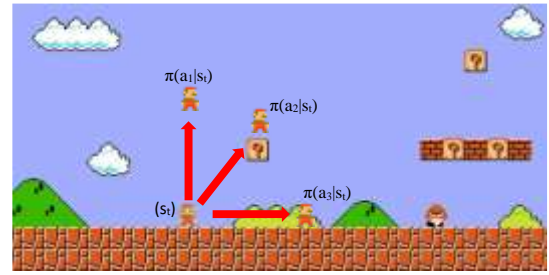


Fig. 4. Mario in the state s_t , and the agent have 3 actions a_1 , a_2 , and a_3 , and policy that calculate the probability of actions given the state s_t and will result Mario in 3 new possible states. (Example of state-action-policy correlation in Super-Mario-Bros)

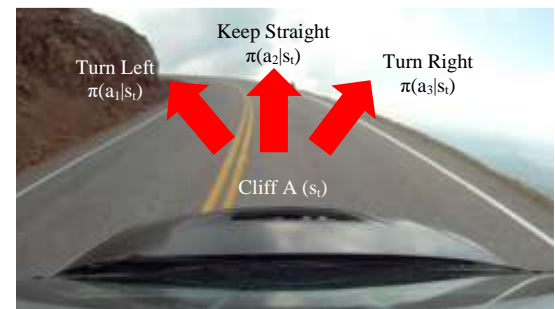


Fig. 5. While the case is same with the Mario example, the safety and cost of training are very different. (state-action-policy correlation in autonomous vehicle)

TABLE II. Examples of RL components in Autonomous Driving and Videogame Environment

Autonomous Driving	Super-Mario-Bros
States - Imagery acquired from camera.	States - Imagery acquired from screen.
Actions: - Brake levels - Acceleration levels - Coupling levels - Gear actions - Steering movement	Action: - No Operation - Left - Right - Up - Down - A - B
Rewards: - Distance achieved	Rewards: - Score Achieved - Completed a level

The inputs in autonomous driving for RL are continuous, while the inputs in *Atari Library* environment are discrete. Here are some points to consider when discussing the inputs for RL in autonomous driving.

Steering movement in Autonomous Driving is not a simple left or right action, it commonly records how many degrees in spins left or right. So, steering itself has so many actions state, considering full degree of steering spin. This

is also true for how deep the brake is pushed, how deep the coupling pushed. So, these actions are considered continuous.

But in *Atari Library*, the action space is considered small and discrete, as it is clear how relatively small options of actions the agent can take.

H. Model-based Reinforcement Learning

Model-Based RL is a popular method in the field of RL that aims to learn the underlying dynamics of a system and use this information to optimize an agent's actions.

The first step in Model-Based RL is to learn the dynamics model of the environment, which involves estimating the transition probabilities and reward function for each state-action pair. One popular approach for learning the dynamics model is through supervised learning, where the model is trained on a dataset of state-action pairs and their corresponding next states and rewards. Another approach is to use unsupervised learning techniques such as autoencoders or generative adversarial networks.

Once the dynamics model is learned, it can be used for planning. One common approach is to use a search algorithm such as Monte Carlo tree search to explore the state space and find the optimal policy. Another approach is to use dynamic programming techniques such as value iteration or policy iteration [42].

One key advantage of Model-Based RL is that it can be more sample-efficient than model-free RL methods [42]. This is because the agent can use the learned dynamics model to simulate the environment and generate training data, rather than relying on trial-and-error exploration. However, Model-Based RL methods require an accurate and reliable model of the environment, which can be challenging to obtain in complex and noisy systems.

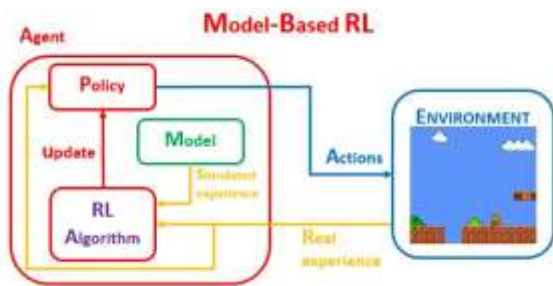


Fig. 6. The agent consists of RL algorithm, policy and internal model from simulated experience. The internal model enables the agent to predict best action in new situations using the internal model (Illustration of Model-Based RL)

I. Model-Free Reinforcement Learning

Model-free RL is a branch of RL that relies on trial and error to learn the optimal action policy without explicitly modeling the underlying dynamics of the environment [42].

There are several popular key methods and algorithms used in model-free RL, such as Monte Carlo (MC) method, Temporal Difference (TD) method, Q-Learning, Deep Q-Learning and SARSA.

Monte Carlo (MC) methods are a type of model-free RL algorithm that estimates the value of a state or state-action pair by sampling returns from the environment [42]. MC methods are simple and effective but suffer from high variance and require episodes to terminate, making them impractical in many scenarios.

Temporal Difference (TD) methods are another type of model-free RL algorithm that use bootstrapping to update value estimates based on the difference between the predicted and actual rewards [42]. TD methods are more efficient than MC methods and can learn online but are sensitive to initial conditions and can be unstable.

Q-learning is a popular TD-based algorithm that learns the optimal Q-values for state-action pairs by iteratively updating a Q-table using the Bellman equation [51]. Q-learning is simple, effective, and can handle large state and action spaces. However, it requires significant exploration and can converge slowly in some environments [29]. the update rule of Q-Learning is denoted mathematically as:

$$Q(s, a) = r(s, a) + \gamma \max_a Q_*(s', a) \tag{10}$$

Deep Q-learning on the other hand, is a recent development in RL, combines Q-learning with deep neural networks to handle high-dimensional state spaces [29, 30]. Deep Q-learning has been successfully applied to complex environments such as *Atari* games and robotics control [25, 30]. However, it is prone to overestimation and instability due to the non-stationarity of the target Q-values [47].

SARSA (state-action-reward-state-action) is another TD-based algorithm that updates Q-values based on the state, action, reward, and next state, action pairs. SARSA is less prone to diverge than Q-learning, but it can be slower to converge [42].

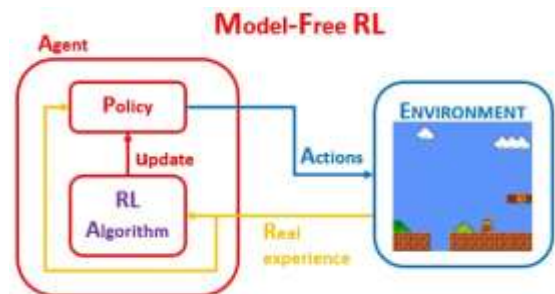


Fig. 7. Unlike Model-Based RL, Model-Free RL doesn't have an internal model, so the agent can only predict best action based on its real experiences. For new situations, it will be explored through trial and error. (Illustration of Model-Free RL)

J. Policy Gradient Methods

Policy gradient methods optimize the policy function directly to maximize the expected return and are a popular class of RL algorithms [44]. The policy function maps states to actions and defines the behavior of the agent in the environment. This literature review provides an overview of the policy gradient methods and their applications in solving RL problems.

Policy gradient methods have gained popularity in recent years due to their ability to learn complex policies for high-dimensional and continuous action spaces. *Actor-critic* methods [22] and *Trust Region Policy Optimization (TRPO)* [37] are some of the most widely used policy gradient algorithms. The future research directions include improving the sample efficiency and convergence rate of the algorithms, developing new algorithms that can handle uncertainty and partial observability, and exploring the applications of policy gradient methods in multi-agent and hierarchical RL problems.

Actor-critic methods combine the policy gradient and value function approximation techniques [44]. The *actor* is responsible for learning the policy function and the *critic* is responsible for estimating the state-value or the action-value function [22]. The *critic* provides feedback to the *actor* by estimating the *advantage function*, which is the difference between the actual return and the estimated value function. The *advantage function* is used to compute the policy gradient, which is then used to update the policy parameters.

TRPO is a policy gradient algorithm that ensures monotonic improvement of the policy by restricting the step size of the policy update [37]. The TRPO algorithm computes the policy update using the conjugate gradient method subject to a constraint on the maximum step size. The constraint ensures that the new policy is close to the old policy and provides stability to the optimization process [37].

K. Exploration and Exploitation

Exploration and exploitation are two critical concepts in RL. Exploration is the process of trying out new actions in order to learn more about the environment and potentially discover more rewarding strategies.

Exploitation is the process of maximizing reward by using actions that are known to be effective. Striking the right balance between exploration and exploitation is key to achieving optimal performance in RL. If an agent explores too little, it may get stuck in a suboptimal solution. If it explores too much, it may waste too much time and resources in trying out different actions, this is also known as *exploration exploitation dilemma* [42].

The epsilon-greedy strategy is a simple exploration strategy in which the agent chooses the action with the highest estimated value with probability $1-\epsilon$, and a

random action with probability ϵ . The epsilon parameter controls the degree of exploration, with higher epsilon values leading to more exploration and lower values leading to more exploitation [42].

L. Safe Reinforcement Learning (Safe-RL)

RL has shown significant promise in solving complex decision-making problems in various domains, ranging from robotics and gaming to healthcare and finance [30, 40]. However, in real-world applications, the trial-and-error process involved in RL can lead to unsafe and undesirable behaviors that could cause harm or damage. Therefore, there is a growing need for safe RL algorithms that can ensure safe exploration and optimize rewards while satisfying constraints [1].

One of the major challenges in safe RL is exploring an environment while ensuring safety. In traditional RL, exploration is often performed through random actions or using heuristics that do not guarantee safety. Safe exploration aims to develop exploration strategies that are safe and optimal [3]. Several approaches have been proposed in the literature, including using safety constraints, learning safety policies, and employing uncertainty-based exploration strategies.

Safe RL faces the critical challenge of guaranteeing the agent's compliance with safety constraints. The safety constraints can be physical, social, and legal. Constraint satisfaction approaches have been proposed to tackle this challenge by integrating the constraints into the RL framework as either soft or hard constraints. Soft constraints allow the agent to violate them to achieve optimal performance, whereas hard constraints must be met at all times. To this end, several methods have been proposed, such as penalizing undesired behavior through modifying the reward function or applying constrained optimization techniques to optimize the policy under the constraints. Studies have shown that these methods have successfully implemented safe RL systems in various domains such as healthcare, robotics, and gaming [3].

Reward shaping is a technique utilized in safe RL to modify the reward function to encourage safe and desirable behavior while discouraging unsafe or undesired actions. The reward function can be tailored to incorporate various safety criteria, such as collision avoidance, social norms, or legal constraints. Expert knowledge-based techniques or inverse RL methods have been proposed in the literature for shaping the reward function. Researchers have proposed using an inverse RL approach for the safe navigation of autonomous vehicles, and a safe RL method using expert knowledge for autonomous UAV navigation.

There have been several successful applications of safe RL in various domains. For instance, safe RL has been applied in robotics for safe control of autonomous robots, in gaming for developing safe and fair game-playing agents, and in healthcare for developing personalized treatment

plans while ensuring patient safety. Case studies in safe RL [13, 17, 23, 24] provide valuable insights into the practical challenges and solutions for safe RL in real-world applications.

III. METHODOLOGY

In this chapter, we will outline the methodologies for this research. First, we will write in detail about the experimental setup, specifications and environment used for this research. This will include the specific RL algorithms and frameworks utilized, as well as the specific tasks and scenarios used to evaluate the effectiveness of safe exploration techniques.

Next, I will describe the metrics and evaluation methods used to assess the performance of the agents. This will include measures of learning efficiency, safety, and overall performance, as well as any relevant benchmarks or baselines used for comparison.

A. Experimental setup

The experiments are simulated in a laptop with Intel Core i7-8650 CPU@1.9GHz (8CPU), 16GB RAM without graphic card.

The experiment is performed in Windows11 22H2 build 22621.1265, programming IDE using Visual Studio Code (VSCode) v1.76.2 with Python extension, Python v3.7.9 [34] for the programming language, and Visual Studio Build Tools 2022 with C++ Desktop Development workload for handling NES builds.

Two virtual environments will be created in VSCode, DDQN environment, and safety-constrained environment. In each environment *stable_baselines3*, *gym_super_mario_bros* and *nes_py*, will be installed as baseline packages.

Within *stable_baselines3*, the main component that I will be using for this research is *gym* (OpenAI Gym [10]), which is a toolkit that provides environments and utilities for the process of developing and measuring RL algorithms performance [40]. It provides a variety of environments (called "gyms") that simulate different kinds of RL problems, such as controlling a robot arm or playing a game. This includes a standardized interface for interacting with the environments, allowing researchers to easily develop and test RL algorithms. It also includes a set of benchmark tasks and metrics for evaluating the performance of RL algorithms. One advantage of *gym* is that it allows researchers to easily compare the performance of different RL algorithms on the same tasks. This can help identify the strengths and weaknesses of different algorithms and can guide future research on RL. Another benefit is that the *gym* offers a wide range of scenarios, such as traditional control issues and challenging, real-world activities like playing games. This enables academics to test RL algorithms on a variety of issues and to investigate new RL applications.

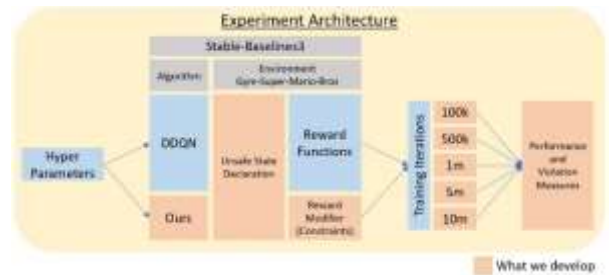


Fig. 8. We develop the experiment above *Stable-Baselines3* library, including declaring unsafe state conditions, modifying reward function, testing in multiple iteration., and developing performance and safety violation measurements (*Experiment Architecture*)

Super-Mario-Bros is a classic video game that is complex enough, so it has been used for multiple testing purposes in the field of RL. The game involves controlling the character *Mario* as he runs and jumps through levels, collecting coins and avoiding obstacles. And algorithms can be trained to play *Super-Mario-Bros* by receiving rewards for completing levels and avoiding obstacles, and adjusting their actions based on these rewards. This allows the algorithms to learn the best strategies for playing the game and achieving a high score. *Super-Mario-Bros* environment is a challenging and complex problem. The game has a large state space, with many possible actions and many different levels and obstacles. This makes it an ideal environment for testing the capabilities of RL algorithms.

In this research, the *gym_super_mario_bros* library will be set using '*SuperMarioBros-1-1-v0*' environment, and the actions will be using '*SIMPLE_MOVEMENT*' that contains 7 actions including *[NoOp]*, *[Right]*, *[Right+A]*, *[Right+B]*, *[Right+A+B]*, *[A]*, *[Left]*.

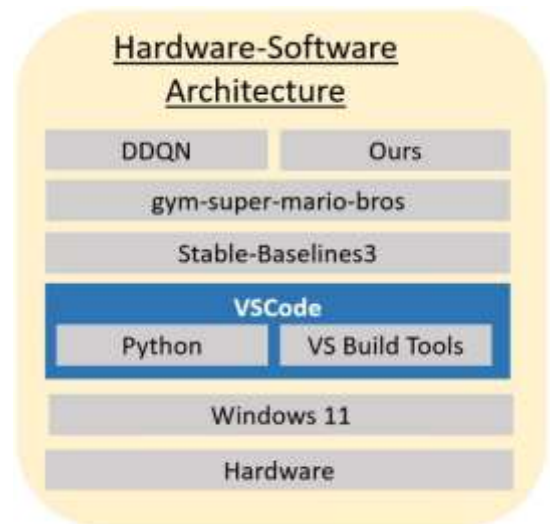


Fig. 9. We perform our experiments on Windows machine, using Python as programming language, and VSCode as code editor. We use *Stable-Baselines3* to provision our base DDQN algorithm, and *gym-super-mario-bros* as the RL environment. (*Hardware-Software Architecture*)

B. RL Algorithms

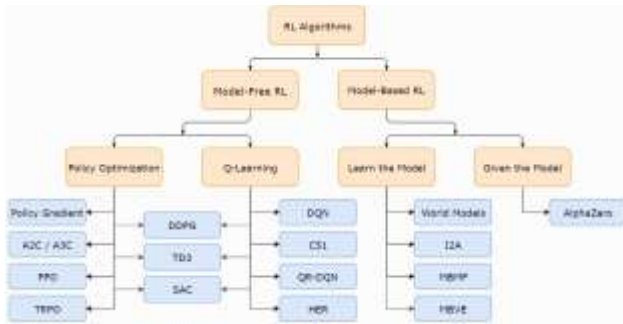


Fig. 10. Current categorization of RL algorithms, source: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html (RL algorithms)

This experiment is using Double Deep Q-Network (DDQN) as the leading algorithms in RL field. Then I will develop safety constraints, to integrate safety into the algorithm.

DDQN is the Q-learning method that is frequently utilized for issues with vast or continuous action spaces [39]. This is because DDQN uses a technique called double Q-learning to reduce the overestimation of action values, which can be a problem in these types of environments. Some specific applications where DDQN has been shown to be effective includes, playing games with high-dimensional or continuous action spaces, such as real-time strategy games or 3D simulations, learning control policies for complex systems with many possible actions, such as power grids or traffic networks, and solving challenging control problems in robotics or other physical systems.

Integrating safety constraints to an algorithm for integrating safety of RL during training and implementation is introduced in [37]. In RL, incorporating safety constraints into the learning process is crucial for training safe and reliable agents. The approach that will be used to ensure safety in this research is through the use of modified rewards. These modified rewards can be designed to discourage the agent from taking unsafe actions or entering unsafe states. For instance, a negative reward can be given for performing an action that leads to death or damage to the player character. This incentivizes the agent to prioritize staying alive and avoiding dangerous situations. By incorporating such safety constraints using modified rewards, RL agents can be trained to achieve optimal rewards while staying within safe and acceptable behavior.

C. Selection of training iteration quantity

The number of iterations required for training an agent depends on several factors such as the complexity of the environment, the size of the action and state spaces, and the desired level of performance. It is generally recommended to start with a smaller number of iterations and gradually increase them to achieve better performance.

In my research, since I am comparing the performance of different algorithms and measuring safety violations, it is important to have a sufficient number of iterations to allow for a fair comparison. A good starting point would be to use several iterations commonly used in the literature [4,5,41] for similar tasks, such as 100k or 500k.

I will also perform a preliminary experiment with a smaller number of iterations to get an idea of the model's performance and safety violations. Based on the results, I will decide if more iterations are necessary to achieve the desired level of performance and safety. However, I always consider that increasing the number of iterations will also increase the computational resources required for training. So, I would also consider the available resources when deciding on the number of iterations.

D. Metrics and Evaluation

In this research, maximum reward and safety violations are considered as the key evaluation metrics. The maximum reward is the primary objective of RL algorithms, as it measures how well the agent can achieve the task or complete the game.

Safety violations, on the other hand, measure the frequency of unsafe actions or states that the agent encounters during training and exploration.

This metric is essential to ensure the safety of the agent and the environment in which it operates. By considering these two-evaluation metrics, we can evaluate the performance of the RL algorithms and make informed decisions on selecting the best algorithm for the task at hand.

IV. EXPERIMENT

We performed some experiments to check if RL agent can show some human-level awareness if the algorithm is incorporated with safety constraints. We also investigated if the performance of RL-agents in obtaining optimal rewards impacted by the safety constraints.

We initially trained RL agents using the leading RL algorithms, which is Double Deep Q-Network (DDQN), created models, and evaluated their performance in terms of reward achievements and safety violations. We also compared these performances when trained for different numbers of iterations, ranging from 100k to 10 million iterations.

To evaluate the impact of safety constraints on the agent's capability in getting optimal rewards, we then trained the agent using the safety-constrained algorithm. We evaluated the performance of the agent using same measurements with DDQN experiments and compared them with the previously trained agents without safety constraints.

We then collected agents' performance data, including the number of times they successfully completed each level, their

average rewards per episode, and their total safety violations. We used these metrics to evaluate the performance of the agents and compare them across different algorithms and training iterations.

These experiments allowed us to understand several implications for implementing safety constraints for RL in the *Super-Mario-Bros* environment.

A. Atari Library

The *gym_super_mario_bros* environment is a collection of *Super-Mario-Bros* games for the Nintendo Entertainment System (NES) platform, which have been adapted as *OpenAI Gym* environments.

The game consists of several components, including *Mario* as the character representation of the RL agent, who is controlled by the agent using actions such as jumping and moving left or right. The enemies, which are the characters that *Mario* must avoid or defeat to progress through the level. The obstacles, that are the objects that *Mario* must navigate around or through to progress through the level, such as pits, walls, and pipes. Power-ups, which are special items that *Mario* can collect to enhance his abilities, include mushrooms, which increase *Mario's* size and strength, and fire flowers, which give *Mario* the ability to shoot fireballs. Last are coins, which are collectible items that provide points and extra lives when enough are collected.

The action space for the *gym_super_mario_bros* environment consists of a set of discrete actions that can be taken by the agent. These actions include moving right, moving left, jumping, jumping while moving right, jumping while moving left, running while moving right, running while moving left, ducking, doing nothing. And the action spaces are grouped into *RIGHT_ONLY*, *SIMPLE_MOVEMENT* and *COMPLEX_MOVEMENT*.

The reward function in the *gym_super_mario_bros* environment is based on the score that the agent receives while playing the game. The score is increased as the agent completes the level, collects coins, and defeats enemies. The score is decreased every second timer passes so the agent gets more rewards to progress through the level quickly and efficiently while avoiding enemies and obstacles. In addition, the agent receives a penalty for losing lives, which encourages the agent to play cautiously and avoid unnecessary risks. While this penalty may represent safety concerns, we want to specifically set that falling into pit is the catastrophic state that should be avoided.

B. Development of Safety Violation Definition and Measures

TABLE I. Pseudocode of our safety violation measurement

Algorithm 1: Safety Violation Measurement	
Input:	y_pos: mario y position, is_dying flag, is_dead flag
Output:	sv: safety violation, nsv: non-safety violation
1:	initialize
2:	while conditions do
3:	if mario is dead with y_pos below ground do
4:	sv+=1
5:	else
6:	nsv+=1
7:	end if
8:	accumulate sv and nsv
9:	end while
10:	output sv and nsv accumulation

During the experiments, it was discovered that the safety violation measurements for the DDQN algorithm were not readily available in the Stable Baselines3 library. As we also decided to represent falling into pit as the catastrophic state, a customized measurement is needed. As a result, the development of these measurements was necessary to evaluate the performance of the agents in terms of safety. To accomplish this, 100k iteration models were analyzed to identify potentially unsafe actions or states. We discovered that falling into pit is detected when *Mario* was either in a “dead” or “dying” status, and the y coordinates is below 250 pixels. Based on these analyses, a set of safety violation measurements was developed as below:

These measurements are applied to the agents trained using DDQN algorithms and will record the balance of safety performance and reward achievements of the agents. And will be used to compare the overall performance of each algorithm in each iteration group.

C. Developing DDQN agents

We developed DDQN models using the stable-baselines3 library. Both models were trained for a range of iterations, from 100k, 500k, 1 million, 5 million and 10 million, using the same hyperparameters and reward function. The models were evaluated based on their average rewards and average safety violations, with safety violation defined as the number of times the agent made an unsafe action, in this case falling into a pit. The performance of the models was compared to determine which algorithm and iteration achieved optimal results in terms of reward and safety. The process of training the algorithm is around 3 weeks, 24 hours per day. This long duration of training mainly caused by the Pytorch that can only utilizes CPU hardware capability to perform matrix multiplication due to our hardware limitations, instead of using GPU that can do matrix multiplication instantly like rendering a game graphics.

Then we finally developed a safety-constrained algorithm for comparison. The generated safety-constrained

models were trained using a similar process as the other models, but with additional safety constraints incorporated into the training process. Specifically, we added constraints to the optimization process to ensure that the agent did not perform unsafe actions during training. This was accomplished by penalizing the agent for taking unsafe actions and adjusting the reward function to prioritize safety over reward maximization. See pseudocode below:

The process of training the constrained algorithm is also around 3 weeks, 24 hours per day but slightly longer due to additional calculation for the modified rewards function. With the same causes due related with hardware capability for matrix multiplication.

TABLE II. Pseudocode of our safety constrained algorithm

Algorithm 2: Safety-Constrained RL Algorithm
Input: RL algorithm M, Reward r
Output: Constrained_Reward
1: initialize M, modified reward function modreward
2: while iteration running do
3: action = predict(observation)
4: new_observation, reward = step(action)
5: reward = modreward(reward)
6: end while

D. Comparison of the performance of the algorithms

After we created DDQN model and safety constrainer DDQN model with same *Super-Mario-Bros* environment, we then evaluated the performance of the created models, both safety-constrained models in terms of their safety violations and reward levels and compared them to the non-constrained DDQN models. We run a 100-episodes evaluation for both DDQN and our model, with the selected iterations, and resulted as below:

```

1 million timestep:
Episode: 1 | dead by falling = +1 : Total: 1 | dead by others = +0 : Total: 0 | total rewards: 1355.0 | total steps: 127
Episode: 2 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 1 | total rewards: 682.0 | total steps: 46
Episode: 3 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 2 | total rewards: 828.0 | total steps: 89
Episode: 4 | dead by falling = +1 : Total: 2 | dead by others = +0 : Total: 2 | total rewards: 1370.0 | total steps: 89
Episode: 5 | dead by falling = +1 : Total: 3 | dead by others = +0 : Total: 2 | total rewards: 1082.0 | total steps: 88
Episode: 6 | dead by falling = +1 : Total: 4 | dead by others = +0 : Total: 2 | total rewards: 1084.0 | total steps: 101
Episode: 7 | dead by falling = +0 : Total: 4 | dead by others = +1 : Total: 3 | total rewards: 637.0 | total steps: 54
Episode: 8 | dead by falling = +0 : Total: 4 | dead by others = +1 : Total: 4 | total rewards: 1458.0 | total steps: 95
Episode: 9 | dead by falling = +0 : Total: 4 | dead by others = +1 : Total: 5 | total rewards: 759.0 | total steps: 121
Episode: 10 | dead by falling = +0 : Total: 4 | dead by others = +1 : Total: 5 | total rewards: 666.0 | total steps: 80
...
...
Episode: 98 | dead by falling = +0 : Total: 27 | dead by others = +1 : Total: 71 | total rewards: 242.0 | total steps: 26
Episode: 99 | dead by falling = +0 : Total: 27 | dead by others = +1 : Total: 72 | total rewards: 656.0 | total steps: 48
Episode: 100 | dead by falling = +0 : Total: 27 | dead by others = +1 : Total: 73 | total rewards: 250.0 | total steps: 21
(700.2, 386.33992804264994, 27, 73)
.....
5 million timestep:
Episode: 1 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 1 | total rewards: 2810.0 | total steps: 1003
Episode: 2 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 2 | total rewards: 2362.0 | total steps: 1003
Episode: 3 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 2 | total rewards: 3105.0 | total steps: 156
Episode: 4 | dead by falling = +1 : Total: 2 | dead by others = +0 : Total: 3 | total rewards: 2870.0 | total steps: 1003
Episode: 5 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 3 | total rewards: 3116.0 | total steps: 689
Episode: 6 | dead by falling = +1 : Total: 1 | dead by others = +0 : Total: 3 | total rewards: 2431.0 | total steps: 119
Episode: 7 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 4 | total rewards: 2719.0 | total steps: 140
Episode: 8 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 5 | total rewards: 2938.0 | total steps: 1003
Episode: 9 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 5 | total rewards: 3109.0 | total steps: 161
Episode: 10 | dead by falling = +1 : Total: 2 | dead by others = +0 : Total: 5 | total rewards: 2431.0 | total steps: 133
...
...
Episode: 98 | dead by falling = +0 : Total: 24 | dead by others = +0 : Total: 27 | total rewards: 3107.0 | total steps: 178
Episode: 99 | dead by falling = +0 : Total: 24 | dead by others = +1 : Total: 28 | total rewards: 2724.0 | total steps: 146
Episode: 100 | dead by falling = +0 : Total: 24 | dead by others = +1 : Total: 29 | total rewards: 2330.0 | total steps: 1003
(2755.51, 443.81472474445906, 24, 29)
.....
10 million timestep:
Episode: 1 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 0 | total rewards: 3098.0 | total steps: 204
Episode: 2 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 1 | total rewards: 2554.0 | total steps: 1003
Episode: 3 | dead by falling = +1 : Total: 1 | dead by others = +0 : Total: 1 | total rewards: 2426.0 | total steps: 428
Episode: 4 | dead by falling = +0 : Total: 1 | dead by others = +0 : Total: 1 | total rewards: 3116.0 | total steps: 147
Episode: 5 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 2 | total rewards: 2138.0 | total steps: 1003
Episode: 6 | dead by falling = +0 : Total: 1 | dead by others = +0 : Total: 2 | total rewards: 3099.0 | total steps: 148
Episode: 7 | dead by falling = +0 : Total: 1 | dead by others = +0 : Total: 2 | total rewards: 3094.0 | total steps: 157
Episode: 8 | dead by falling = +1 : Total: 2 | dead by others = +0 : Total: 2 | total rewards: 2426.0 | total steps: 390
Episode: 9 | dead by falling = +0 : Total: 2 | dead by others = +0 : Total: 2 | total rewards: 3101.0 | total steps: 166
Episode: 10 | dead by falling = +0 : Total: 2 | dead by others = +0 : Total: 2 | total rewards: 3097.0 | total steps: 669
...
...
Episode: 98 | dead by falling = +0 : Total: 17 | dead by others = +0 : Total: 20 | total rewards: 3108.0 | total steps: 152
Episode: 99 | dead by falling = +1 : Total: 18 | dead by others = +0 : Total: 20 | total rewards: 2431.0 | total steps: 126
Episode: 100 | dead by falling = +0 : Total: 18 | dead by others = +0 : Total: 20 | total rewards: 3102.0 | total steps: 156
(2637.49, 761.6797160880681, 18, 20)

```

Fig. 11. The 1m iteration resulted in average reward of 700.2, reward standard deviation of 386.3, failures due to safety violation by 27, and failures by non-safety violation by 73. The 5m iteration resulting in an average reward of 2755.5, reward standard deviation of 443.8, failures due to safety violation by 24, and failures by non-safety violation by 29. The 10m iteration resulted in an average reward of 2637.5, reward standard deviation of 761.7, failures due to safety violation by 18, and failures by non-safety violation by 20. (DDQN upper-bound iteration experiment results)

100k timestep:
 Episode: 1 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 1 | total rewards: 747.0 | total steps: 76
 Episode: 2 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 2 | total rewards: 655.0 | total steps: 57
 Episode: 3 | dead by falling = +1 : Total: 1 | dead by others = +0 : Total: 2 | total rewards: 1097.0 | total steps: 106
 Episode: 4 | dead by falling = +1 : Total: 2 | dead by others = +0 : Total: 2 | total rewards: 1093.0 | total steps: 93
 Episode: 5 | dead by falling = +0 : Total: 2 | dead by others = +1 : Total: 3 | total rewards: 256.0 | total steps: 20
 Episode: 6 | dead by falling = +0 : Total: 2 | dead by others = +1 : Total: 4 | total rewards: 648.0 | total steps: 60
 Episode: 7 | dead by falling = +0 : Total: 2 | dead by others = +1 : Total: 5 | total rewards: 658.0 | total steps: 98
 Episode: 8 | dead by falling = +0 : Total: 2 | dead by others = +1 : Total: 6 | total rewards: 642.0 | total steps: 53
 Episode: 9 | dead by falling = +0 : Total: 2 | dead by others = +1 : Total: 7 | total rewards: 242.0 | total steps: 20
 Episode: 10 | dead by falling = +0 : Total: 2 | dead by others = +1 : Total: 8 | total rewards: 250.0 | total steps: 21
 ...
 Episode: 98 | dead by falling = +1 : Total: 23 | dead by others = +0 : Total: 75 | total rewards: 1088.0 | total steps: 115
 Episode: 99 | dead by falling = +1 : Total: 24 | dead by others = +0 : Total: 75 | total rewards: 1086.0 | total steps: 101
 Episode: 100 | dead by falling = +0 : Total: 24 | dead by others = +1 : Total: 76 | total rewards: 772.0 | total steps: 73
 (678.65, 275.0741490943851, 24, 76)

500k timestep:
 Episode: 1 | dead by falling = +1 : Total: 1 | dead by others = +0 : Total: 0 | total rewards: 1380.0 | total steps: 108
 Episode: 2 | dead by falling = +1 : Total: 2 | dead by others = +0 : Total: 0 | total rewards: 1370.0 | total steps: 74
 Episode: 3 | dead by falling = +0 : Total: 2 | dead by others = +1 : Total: 1 | total rewards: 1468.0 | total steps: 74
 Episode: 4 | dead by falling = +0 : Total: 2 | dead by others = +1 : Total: 2 | total rewards: 759.0 | total steps: 69
 Episode: 5 | dead by falling = +1 : Total: 3 | dead by others = +0 : Total: 2 | total rewards: 1089.0 | total steps: 76
 Episode: 6 | dead by falling = +0 : Total: 3 | dead by others = +1 : Total: 3 | total rewards: 789.0 | total steps: 54
 Episode: 7 | dead by falling = +0 : Total: 3 | dead by others = +1 : Total: 4 | total rewards: 1613.0 | total steps: 86
 Episode: 8 | dead by falling = +0 : Total: 3 | dead by others = +1 : Total: 5 | total rewards: 622.0 | total steps: 45
 Episode: 9 | dead by falling = +0 : Total: 3 | dead by others = +1 : Total: 6 | total rewards: 1476.0 | total steps: 88
 Episode: 10 | dead by falling = +1 : Total: 4 | dead by others = +0 : Total: 6 | total rewards: 1088.0 | total steps: 79
 ...
 Episode: 98 | dead by falling = +0 : Total: 22 | dead by others = +1 : Total: 76 | total rewards: 629.0 | total steps: 29
 Episode: 99 | dead by falling = +1 : Total: 23 | dead by others = +0 : Total: 76 | total rewards: 1362.0 | total steps: 75
 Episode: 100 | dead by falling = +0 : Total: 23 | dead by others = +1 : Total: 77 | total rewards: 1735.0 | total steps: 95
 (1151.48, 434.55769421332303, 23, 77)

Fig. 12. The 100k iteration experiment resulting in average reward of 678.7, reward standard deviation of 275.1, failures due to safety violation by 24, and failures by non-safety violation by 76. The 500k iteration resulted in an average reward of 1151.5, reward standard deviation of 434.6, failures due to safety violation by 23, and failures by non-safety violation by 77. (*DDQN lower-bound iterations experiment results*)

And for ours:

1 million timestep:
 Episode: 1 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 1 | total rewards: 637.0 | total steps: 48
 Episode: 2 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 2 | total rewards: 627.0 | total steps: 30
 Episode: 3 | dead by falling = +1 : Total: 1 | dead by others = +0 : Total: 2 | total rewards: 1077.0 | total steps: 141
 Episode: 4 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 3 | total rewards: 1193.0 | total steps: 65
 Episode: 5 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 4 | total rewards: 858.0 | total steps: 73
 Episode: 6 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 5 | total rewards: 771.0 | total steps: 57
 Episode: 7 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 6 | total rewards: 626.0 | total steps: 33
 Episode: 8 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 7 | total rewards: 618.0 | total steps: 29
 Episode: 9 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 8 | total rewards: 653.0 | total steps: 43
 Episode: 10 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 9 | total rewards: 638.0 | total steps: 40
 ...
 Episode: 98 | dead by falling = +0 : Total: 24 | dead by others = +1 : Total: 74 | total rewards: 629.0 | total steps: 36
 Episode: 99 | dead by falling = +0 : Total: 24 | dead by others = +1 : Total: 75 | total rewards: 616.0 | total steps: 29
 Episode: 100 | dead by falling = +0 : Total: 24 | dead by others = +1 : Total: 76 | total rewards: 779.0 | total steps: 61
 (921.54, 380.73600880400056, 24, 76)

5 million timestep:
 Episode: 1 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 1 | total rewards: 1183.0 | total steps: 60
 Episode: 2 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 2 | total rewards: 2938.0 | total steps: 1003
 Episode: 3 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 2 | total rewards: 3117.0 | total steps: 417
 Episode: 4 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 3 | total rewards: 2708.0 | total steps: 142
 Episode: 5 | dead by falling = +1 : Total: 1 | dead by others = +0 : Total: 3 | total rewards: 2429.0 | total steps: 148
 Episode: 6 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 4 | total rewards: 2554.0 | total steps: 1003
 Episode: 7 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 5 | total rewards: 784.0 | total steps: 57
 Episode: 8 | dead by falling = +1 : Total: 2 | dead by others = +0 : Total: 5 | total rewards: 2427.0 | total steps: 853
 Episode: 9 | dead by falling = +0 : Total: 2 | dead by others = +0 : Total: 6 | total rewards: 2314.0 | total steps: 1003
 Episode: 10 | dead by falling = +0 : Total: 2 | dead by others = +1 : Total: 7 | total rewards: 2186.0 | total steps: 1003
 ...
 Episode: 98 | dead by falling = +1 : Total: 18 | dead by others = +0 : Total: 62 | total rewards: 2430.0 | total steps: 952
 Episode: 99 | dead by falling = +0 : Total: 18 | dead by others = +1 : Total: 63 | total rewards: 2330.0 | total steps: 1003
 Episode: 100 | dead by falling = +0 : Total: 18 | dead by others = +1 : Total: 64 | total rewards: 2030.0 | total steps: 1003
 (2331.58, 529.9094296198173, 18, 64)

10 million timestep:
 Episode: 1 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 1 | total rewards: 2707.0 | total steps: 143
 Episode: 2 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 1 | total rewards: 3096.0 | total steps: 178
 Episode: 3 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 1 | total rewards: 3103.0 | total steps: 160
 Episode: 4 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 1 | total rewards: 3107.0 | total steps: 203
 Episode: 5 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 2 | total rewards: 250.0 | total steps: 17
 Episode: 6 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 2 | total rewards: 3117.0 | total steps: 156
 Episode: 7 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 2 | total rewards: 3116.0 | total steps: 159
 Episode: 8 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 2 | total rewards: 3099.0 | total steps: 162
 Episode: 9 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 2 | total rewards: 3108.0 | total steps: 145
 Episode: 10 | dead by falling = +0 : Total: 0 | dead by others = +0 : Total: 2 | total rewards: 3116.0 | total steps: 152
 ...
 Episode: 98 | dead by falling = +0 : Total: 2 | dead by others = +0 : Total: 27 | total rewards: 3110.0 | total steps: 408
 Episode: 99 | dead by falling = +0 : Total: 2 | dead by others = +0 : Total: 27 | total rewards: 3116.0 | total steps: 153
 Episode: 100 | dead by falling = +0 : Total: 2 | dead by others = +0 : Total: 27 | total rewards: 3104.0 | total steps: 157
 (2703.98, 843.7955792726103, 2, 27)

Fig. 13. The 1m iteration resulting in average reward of 921.5, reward standard deviation of 380.7, failures due to safety violation by 24, and failures by non-safety violation by 76. The 5m iteration resulting in an average reward of 2331.6, reward standard deviation of 529.9, failures due to safety violation by 18, and failures by non-safety violation by 64. The 10m iteration resulted in an average reward of 2704, reward standard deviation of 843.8, failures due to safety violation by 2, and failures by non-safety violation by 27. (*Ours upper-bound iterations experiment results*)


```

100k timestep:
Episode: 1 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 1 | total rewards: 649.0 | total steps: 46
Episode: 2 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 2 | total rewards: 250.0 | total steps: 15
Episode: 3 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 3 | total rewards: 647.0 | total steps: 49
Episode: 4 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 4 | total rewards: 250.0 | total steps: 15
Episode: 5 | dead by falling = +1 : Total: 1 | dead by others = +0 : Total: 4 | total rewards: 1088.0 | total steps: 78
Episode: 6 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 5 | total rewards: 653.0 | total steps: 45
Episode: 7 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 6 | total rewards: 636.0 | total steps: 52
Episode: 8 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 7 | total rewards: 250.0 | total steps: 15
Episode: 9 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 8 | total rewards: 651.0 | total steps: 48
Episode: 10 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 9 | total rewards: 626.0 | total steps: 52
...
...
Episode: 98 | dead by falling = +0 : Total: 20 | dead by others = +1 : Total: 78 | total rewards: 269.0 | total steps: 16
Episode: 99 | dead by falling = +0 : Total: 20 | dead by others = +1 : Total: 79 | total rewards: 250.0 | total steps: 15
Episode: 100 | dead by falling = +0 : Total: 20 | dead by others = +1 : Total: 80 | total rewards: 682.0 | total steps: 47
(629.28, 277.4209465775791, 20, 80)
.....
500k timestep:
Episode: 1 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 1 | total rewards: 649.0 | total steps: 40
Episode: 2 | dead by falling = +0 : Total: 0 | dead by others = +1 : Total: 2 | total rewards: 644.0 | total steps: 51
Episode: 3 | dead by falling = +1 : Total: 1 | dead by others = +0 : Total: 2 | total rewards: 1376.0 | total steps: 92
Episode: 4 | dead by falling = +0 : Total: 1 | dead by others = +1 : Total: 3 | total rewards: 654.0 | total steps: 46
Episode: 5 | dead by falling = +1 : Total: 2 | dead by others = +0 : Total: 3 | total rewards: 1387.0 | total steps: 80
Episode: 6 | dead by falling = +0 : Total: 2 | dead by others = +1 : Total: 4 | total rewards: 638.0 | total steps: 43
Episode: 7 | dead by falling = +0 : Total: 2 | dead by others = +1 : Total: 5 | total rewards: 1744.0 | total steps: 102
Episode: 8 | dead by falling = +1 : Total: 3 | dead by others = +0 : Total: 5 | total rewards: 1371.0 | total steps: 79
Episode: 9 | dead by falling = +0 : Total: 3 | dead by others = +1 : Total: 6 | total rewards: 682.0 | total steps: 75
Episode: 10 | dead by falling = +0 : Total: 3 | dead by others = +1 : Total: 7 | total rewards: 652.0 | total steps: 40
...
...
Episode: 98 | dead by falling = +0 : Total: 18 | dead by others = +1 : Total: 80 | total rewards: 633.0 | total steps: 34
Episode: 99 | dead by falling = +1 : Total: 19 | dead by others = +0 : Total: 80 | total rewards: 1377.0 | total steps: 81
Episode: 100 | dead by falling = +0 : Total: 19 | dead by others = +1 : Total: 81 | total rewards: 648.0 | total steps: 38
(1001.78, 413.43607921902515, 19, 81)

```

Fig. 14. The 100k iteration resulting in average reward of 629.3, reward standard deviation of 277.4, failures due to safety violation by 20, and failures by non-safety violation by 80. The 500k iteration resulting in an average reward of 1001.8, reward standard deviation of 413.4, failures due to safety violation by 19, and failures by non-safety violation by 81. (Ours lower-bound iterations experiment results)

From the experiments, the data is compiled to get the total rewards, total violation, and total episode completion for each iteration, both for DDQN and our model. As we declared that falling into a pit as a safety violation, we will accumulate “dead by falling” into Violation variable, mathematically expressed as below:

$$Violation = \frac{\sum_1^{episodes} dead_by_falling}{episodes} \quad (11)$$

Then we accumulate the “total reward” variable into Reward, and get the average total rewards from all episodes, mathematically expressed as below:

$$Reward = \frac{\sum_1^{episodes} total_rewards}{episodes} \quad (12)$$

Lastly, we extract the information about the completion of each episode, this is where the agent didn’t die of falling into pit or any other causes, and the “done” flag is achieved within the episode, mathematically expressed as:

$$Completion = \frac{\sum_1^{episodes} 1 - (dead_by_falling + dead_by_others)}{episodes} \times 100\% \quad (13)$$

We calculate using above three formulas for each iteration, from 100k, 500k, 1 million, 5 million, and 10 million for DDQN and our model, and the results are summarized as follows:

The results indicate that a sufficiently trained DDQN agent with 10 million iterations, can complete the stage with 62 times out of 100. And 18 safety violations out of 100. However, when agents were trained with lower iteration numbers, for example, the 100k iterations have 0 completion and 24 safety violations, and 500k iterations also have 0 completion and 23 safety violations.

For our proposed safety-constrained algorithm, it is shown that in lower iterations, the safety-constrained algorithm has slightly lower safety violations compared to DDQN. With safety constraints during training, the agents were able to achieve lower safety violations while maintaining relatively equal rewards. However, it should be noted that incorporating safety constraints comes at the cost of sacrificing computing resources due to additional calculation for safety-measurement process.

The results of our experiments demonstrate the importance of having a lot of iterations for training RL agents, and the impact of the choice of algorithm on reward achievements and safety violations. Additionally, the findings highlight the significance of incorporating safety constraints during training, especially for short iteration agents to develop robust and Safe-RL agents.

TABLE III. Comparison of performance of DDQN and our safety-constrained algorithm.

Iterations	Parameters	DDQN ^[19,47]	Ours
100k	Reward	678.65	629.28
	Violation	24	20
	Completion	0%	0%
500k	Reward	1151.48	1001.78
	Violation	23	19
	Completion	0%	0%
1m	Reward	700.2	921.54
	Violation	27	24
	Completion	0%	0%
5m	Reward	2755.51	2331
	Violation	24	18
	Completion	47%	18%
10m	Reward	2637.49	2703.98
	Violation	18	2
	Completion	62%	71%

E. The Implications of the findings

The results that are shown have some implications for using videogame environment for the development of Safe-RL. First, the results show the need for enough training iterations to let the algorithm show significant performance while maintaining low safety violations. This highlights the importance of having large and diverse interactions with the environment, and to consider computing capability of the

hardware to train RL agents effectively.

Second, the results of DDQN algorithms models show that different types of algorithms can significantly impact safety violations.

The balance between safety, performance and computing resources should be carefully considered when developing safe and robust RL agents, particularly in environments where trial-and-error learning can cause harm to humans.

F. Limitation of the studies

The study on Safe-RL in the *Super-Mario-Bros* environment using the safety-constrained algorithm has some limitations that should be considered. Firstly, the study only focused on two particular algorithms and one game environment, which may limit the generalizability of the results to other RL algorithms and game environments.

Secondly, the study did not explore the impact of different reward functions on the performance of the safety-constrained algorithm. While the author argues that their reward function is effective in promoting safe behavior, it is possible that different reward functions could lead to different results.

Thirdly, the study did not investigate the impact of other safety constraints, such as constraints on exploration, on the performance of the safety-constrained algorithm. Incorporating additional safety constraints may affect the balance between safety and performance, and further research is needed to investigate the trade-off between these factors.

Finally, the study did not consider the impact of human players on the performance of the safety-constrained algorithm. While the authors argue that the *Super-Mario-Bros* environment is a suitable proxy for real-world scenarios, the presence of human players may introduce additional complexity and safety considerations that are not present in the game environment.

G. Future directions for research

Further research is required to achieve human-level intelligence [43], which involves developing memory representation of a state combined with image dimensionality reduction. Humans have a perfect image of what they see in the first few seconds and then reduce it over time, and we believe that this can act as a crucial step towards efficient world modeling [18]. Given the advanced gaming field, we can develop world models using videogame simulations when combined with Safe-RL. Therefore, incorporating Safe-RL in videogame environments can be a significant step towards achieving robust and safe AI systems.

V. CONCLUSION

The research presented in this paper investigated the use of safe reinforcement learning (RL) in a complex videogame environment like *Super-Mario-Bros*. The results showed that the proposed safety-constrained algorithm was able to learn to play the game effectively while also complying with safety constraints and avoiding undesirable behaviors.

The results of this study have implications for the future development of RL algorithms. The ability to perform safe RL in complex videogame environments is a step towards using more advanced and realistic videogame environments as simulation platforms for RL training. This could lead to the development of RL agents that are capable of performing complex tasks in real-world environments.

The research also highlighted the need for further exploration in model-based RL. Model-based RL methods can be more effective than model-free methods at achieving safety in RL tasks. In addition, using or adding human knowledge and thought process into the RL agent to develop human-like model-based RL. This could be a promising direction for ensuring that the agent's behavior aligns with human values and ethical principles.

Finally, the development of more sophisticated evaluation metrics and benchmarks for RL, especially in videogame environments, can help to provide a standardized and objective measure of the effectiveness of different RL algorithms. This is important for comparing the performance of different RL algorithms and for identifying the best algorithms for specific tasks.

REFERENCES

- [1]. Achiam, J., Held, D., Tamar, A. and Abbeel, P., 2017, July. Constrained policy optimization. In International conference on machine learning (pp. 22-31). PMLR.
- [2]. Altman, E., 1999. Constrained Markov decision processes (Vol. 7). CRC press.
- [3]. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J. and Mané, D., 2016. Concrete problems in AI safety. arXiv preprint arXiv:1606.06565.
- [4]. Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O. and Zaremba, W., 2017. Hindsight experience replay. Advances in neural information processing systems, 30.
- [5]. Badia, A.P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z.D. and Blundell, C., 2020, November. Agent57: Outperforming the atari human benchmark. In International conference on machine learning (pp. 507-517). PMLR.
- [6]. Badia, A.P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A. and Blundell, C., 2020. Never give up: Learning directed exploration strategies. arXiv preprint arXiv:2002.06038.
- [7]. Bakker, B., 2001. Reinforcement learning with long short-term memory. Advances in neural information processing systems, 14.
- [8]. Berkenkamp, F., 2019. Safe exploration in reinforcement learning: Theory and applications in robotics (Doctoral dissertation, ETH Zurich).

- [9]. Berkenkamp, F., Turchetta, M., Schoellig, A. and Krause, A., 2017. Safe model-based reinforcement learning with stability guarantees. *Advances in neural information processing systems*, 30.
- [10]. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W., 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.
- [11]. Broekens, J., Jacobs, E. and Jonker, C.M., 2015. A reinforcement learning model of joy, distress, hope and fear. *Connection Science*, 27(3), pp.215-233.
- [12]. 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, pp.411-444.
- [13]. Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T. and Efron, A.A., 2018. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*.
- [14]. Chow, Y., Nachum, O., Faust, A., Duenez-Guzman, E. and Ghavamzadeh, M., 2019. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*.
- [15]. Dalal, G., Dvijotham, K., Vecerik, M., Hester, T., Paduraru, C. and Tassa, Y., 2018. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*.
- [16]. Deisenroth, M.P., Faisal, A.A. and Ong, C.S., 2020. *Mathematics for machine learning*. Cambridge University Press.
- [17]. Gu, S., Yang, L., Du, Y., Chen, G., Walter, F., Wang, J., Yang, Y. and Knoll, A., 2022. A review of safe reinforcement learning: Methods, theory and applications. *arXiv preprint arXiv:2205.10330*.
- [18]. Hafner, D., Pasukonis, J., Ba, J. and Lillicrap, T., 2023. Mastering Diverse Domains through World Models. *arXiv preprint arXiv:2301.04104*.
- [19]. Hasselt, H., 2010. Double Q-learning. *Advances in neural information processing systems*, 23.
- [20]. Hellaby, W.C.J.C., 1989. Learning from delayed rewards.
- [21]. Jayant, A.K. and Bhatnagar, S., 2022. Model-based Safe Deep Reinforcement Learning via a Constrained Proximal Policy Optimization Algorithm. *Advances in Neural Information Processing Systems*, 35, pp.24432-24445.
- [22]. Konda, V. and Tsitsiklis, J., 1999. Actor-critic algorithms. *Advances in neural information processing systems*, 12.
- [23]. Ladosz, P., Weng, L., Kim, M. and Oh, H., 2022. Exploration in deep reinforcement learning: A survey. *Information Fusion*.
- [24]. Li, Y., 2022. Deep reinforcement learning: Opportunities and challenges. *arXiv preprint arXiv:2202.11296*.
- [25]. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., 2020. Continuous control with deep reinforcement learning. *US Patent*, 15(217,758).
- [26]. Lin, L.J., 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8, pp.293-321.
- [27]. Lipton, Z.C., Azizzadenesheli, K., Kumar, A., Li, L., Gao, J. and Deng, L., 2016. Combating reinforcement learning's sisyphian curse with intrinsic fear. *arXiv preprint arXiv:1611.01211*.
- [28]. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K., 2016, June. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928-1937). PMLR.
- [29]. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- [30]. 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.
- [31]. Nichol, A., Pfau, V., Hesse, C., Klimov, O. and Schulman, J., 2018. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*.
- [32]. Pathak, D., Agrawal, P., Efron, A.A. and Darrell, T., 2017, July. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning* (pp. 2778-2787). PMLR.
- [33]. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M. and Dormann, N., 2021. Stable-baselines3: Reliable reinforcement learning implementations. *The Journal of Machine Learning Research*, 22(1), pp.12348-12355.
- [34]. Raschka, S. and Mirjalili, V., 2019. *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd.
- [35]. Ray, A., Achiam, J. and Amodei, D., 2019. Benchmarking safe exploration in deep reinforcement learning. *arXiv. arXiv preprint arXiv:1910.01708*, 7.
- [36]. Schmeckpeper, K., Rybkin, O., Daniilidis, K., Levine, S. and Finn, C., 2020. Reinforcement learning with videos: Combining offline observations with interaction. *arXiv preprint arXiv:2011.06507*.
- [37]. Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P., 2015, June. Trust region policy optimization. In *International conference on machine learning* (pp. 1889-1897). PMLR.
- [38]. Schulman, J., Moritz, P., Levine, S., Jordan, M. and Abbeel, P., 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.
- [39]. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- [40]. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587), pp.484-489.
- [41]. 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*.
- [42]. Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. MIT press.
- [43]. Sutton, R.S., Bowling, M.H. and Pilarski, P.M., 2022. The Alberta Plan for AI Research. *arXiv preprint arXiv:2208.11173*.
- [44]. Sutton, R.S., McAllester, D., Singh, S. and Mansour, Y., 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.
- [45]. Thomas, G., Luo, Y. and Ma, T., 2021. Safe reinforcement learning by imagining the near future. *Advances in Neural Information Processing Systems*, 34, pp.13859-13869.
- [46]. Thumm, J. and Althoff, M., 2022, May. Provably safe deep reinforcement learning for robotic manipulation in human environments. In *2022 International Conference on Robotics and Automation (ICRA)* (pp. 6344-6350). IEEE.
- [47]. 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).
- [48]. Van Otterlo, M. and Wiering, M., 2012. Reinforcement learning and markov decision processes. *Reinforcement learning: State-of-the-art*, pp.3-42.
- [49]. Wagener, N.C., Boots, B. and Cheng, C.A., 2021, July. Safe reinforcement learning using advantage-based intervention. In *International Conference on Machine Learning* (pp. 10630-10640). PMLR.
- [50]. Wan, T. and Xu, N., 2018. Advances in experience replay. *arXiv preprint arXiv:1805.05536*.
- [51]. Watkins, C.J. and Dayan, P., 1992. Q-learning. *Machine learning*, 8, pp.279-292.
- [52]. Zhang, L., Shen, L., Yang, L., Chen, S., Yuan, B., Wang, X. and Tao, D., 2022. Penalized proximal policy optimization for safe reinforcement learning. *arXiv preprint arXiv:2205.11814*.
- [53]. Zhang, Y., Vuong, Q. and Ross, K., 2020. First order constrained optimization in policy space. *Advances in Neural Information Processing Systems*, 33, pp.15338-15349.
- [54]. Todorov, E., Erez, T. and Tassa, Y., 2012, October. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ*

- international conference on intelligent robots and systems (pp. 5026-5033). IEEE.
- [55]. Zhao, W., Queralta, J.P. and Westerlund, T., 2020, December. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In 2020 IEEE symposium series on computational intelligence (SSCI) (pp. 737-744). IEEE.