Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Artificial Intelligence (38^{th}cycle)

Enhancing Safety in Reinforcement Learning Training through Transformers Filtering

By

Mario Fiorino
******

Supervisor(s):
Prof. Antonio Coronato, Supervisor
Prof. Mario Ciampi, Co-Supervisor

Politecnico di Torino
2026

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Mario Fiorino
2026

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Abstract

Artificial Intelligence (AI) refers to the field of study and the collection of techniques aimed at developing machine systems capable of reproducing advanced cognitive processes such as learning, reasoning, and decision-making. Over recent decades, AI has evolved rapidly, emerging as a transformative discipline with profound impacts on science, industry, and society. Among its paradigms, Reinforcement Learning (RL) has played a fundamental role in advancing AI capabilities, enabling systems to learn optimal behaviors through interactions with their environments. RL has been successfully applied across diverse domains, including robotics, healthcare, resource management, and large language model tuning, where it has, in several instances, demonstrated superhuman performance. More specifically, RL focuses on designing algorithms that enable an agent (an autonomous software entity) to make sequential decisions aimed at achieving optimal performance. Through iterative interactions with its environment, the agent seeks to maximize a feedback signal, commonly referred to as the reward, which evaluates the effectiveness of its actions.

The interaction between the agent and the environment—particularly in model-free approaches, where systems lack an explicit prior model of the environment and acquire knowledge solely through trial-and-error exploration—has highlighted both the strengths and limitations of the RL paradigm. Strengths include its flexibility in learning complex behavior without requiring explicit models. Significant limitations, including sample inefficiency and safety risks, arise from the uncontrolled exploration inherent to model-free learning. In particular, in safety-critical domains, such exploration can lead to severe or irreversible consequences due to the absence of predictive safeguards. For instance, trial-and-error exploration in an autonomous driving system might entail executing maneuvers that risk collisions and serious accidents. Similarly, in an automated medical diagnostic system, such exploration could involve suggesting treatments that potentially result in adverse patient outcomes. Traditional

model-free RL algorithms are not suited to handle such high-stakes scenarios, as they lack mechanisms to guarantee safe behavior during learning.

In response to these challenges, the field of Safe Reinforcement Learning (Safe RL) has emerged, aiming to incorporate explicit safety considerations into the learning process, seeking to optimize performance rewards while ensuring that the agent operates within predefined safety constraints. Since its inception, several approaches have been proposed, notably Constraint-Based Methods and Shielding Methods. Although these methodologies have shown promising results, they exhibit several limitations. Both rely heavily on the explicit formulation of safety constraints, which are frequently partially unknown, highly complex, or impractical to define, interpret, and implement. Moreover, verifying the correctness and completeness of these constraints often poses additional challenges. Providing additional insight, in the context of model-free RL, Constraint-Based Methods face a fundamental limitation: during training, the agent inevitably violates constraints due to the inherent trial-and-error mechanism, thereby restricting their applicability primarily to simulated environments. Shielding Methods, while offering mechanisms to prevent unsafe actions, require an a priori model of the environment to be effective, rendering them unsuitable for model-free scenarios.

This dissertation addresses these limitations by introducing a novel framework, Safe Reinforcement Learning via Transformer Filtering, which leverages the representational power of Transformer neural networks to implicitly learn safety constraints directly from data. Once trained on sequences of expert safe trajectories, the Safety Transformer acts as a dynamic filter over the agent's action space, constraining exploration to safety-consistent behaviors. Fundamentally, this framework enables safe online learning in model-free RL settings without requiring the explicit modeling of constraints.

Empirical validation was conducted across two distinct domains: a deterministic navigation task and a stochastic biomedical control problem. The results demonstrate that the proposed approach effectively eliminates unsafe interactions during training in deterministic settings and significantly mitigates them in stochastic environments, without the filtering mechanism adversely affecting reward accumulation. In summary, the experimental findings indicate that the proposed framework maintains safety without compromising the agent's learning performance, permitting convergence toward optimal or near-optimal policies.

Keywords: Safe Reinforcement Learning, Model-Free Reinforcement Learning, Safety Transformers Filtering.

Thesis Overview

This dissertation is organized into six chapters, which progressively present the background, motivations, development, and evaluation of the proposed Safe Reinforcement Learning framework via Transformer Filtering.

Chapter 1: Introduction to Reinforcement Learning. The first chapter provides the theoretical foundations of Reinforcement Learning. It introduces the Markov Decision Process formalism; delineates the principal algorithmic paradigms: Monte Carlo, Temporal-Difference, and gradient-based methods; and highlights the core open issues that motivate ongoing research.

Chapter 2: Safe Reinforcement Learning Approaches. The second chapter reviews the principal methodologies developed to ensure safety in Reinforcement Learning. It focuses on Constraint-Based and Shielding Methods, detailing their theoretical formulation and practical implementations. The discussion critically examines their limitations, such as the need for explicit constraint modeling and the infeasibility of achieving zero constraint violations in model-free contexts.

Chapter 3: Safe Reinforcement Learning via Transformer Filtering. This chapter introduces the central contribution of this thesis: the Transformer Filtering framework. It begins by outlining the problem statement and the research contributions, followed by an overview of Transformer neural network architectures, their mechanisms, and their applications in Reinforcement Learning. The proposed framework is then presented in detail, illustrating how safety can be learned implicitly from data and integrated into the training loop of Reinforcement Learning algorithm.

Chapter 4: Experimental Evaluation in Deterministic Environment - Navigation Task. This chapter describes the implementation and evaluation of the proposed framework in a fully deterministic grid-based navigation environment. It discusses the architecture and training of the Safety Transformer, the integration with a basic Policy Gradient algorithm, and provides empirical results demonstrating that the framework achieves complete safety during training without hindering policy optimality.

Chapter 5: Experimental Evaluation in Stochastic Environment - Type 1 Diabetes Management. This chapter extends the evaluation to a stochastic biomedical control domain using the UVA/Padova Type 1 Diabetes Simulator. It details the problem formulation and environment configuration, the training of a domain-specific Safety Transformer, and its integration into the training loop of a Proximal Policy Optimization algorithm. The results confirm the framework's robustness, showing significant reductions in unsafe states and high performance, even under high uncertainty.

Chapter 6: Conclusions and Future Directions. The final chapter summarizes the main findings and contributions of the research, discussing their implications. It also critically examines extant limitations, and outlines promising avenues for future research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Reinforcement Learning

Originating in China over 3000 years ago, Go is a board game formally classified in the scientific literature as a perfect-information game with fixed termination. It is much more complex than chess, and its mastery has long been regarded as one of the greatest challenges in the field of Artificial Intelligence (AI); primarily due to its huge search space and the inherent difficulty of evaluating board positions and moves. The number of legal board positions on a square board with length 19 is around: $2.08 \cdot 10^{170}$ [87], making it far larger than any other known perfect-information game[1].

The year 2016 is considered a milestone in the history of AI research, as *AlphaGo* [80], an AI agent developed by DeepMind, defeated professional Go player Lee Sedol in a five-game match held in Seoul, South Korea, from March 9th to March 15th, with a final score of four games to one. Notably, with 18 World Titles, Lee Sedol is regarded as one of the strongest players in the history of the game.

In the press conference following the first game, Sedol stated: *"I was very surprised, because I didn't think I would lose the game. [. . .] I didn't think that AlphaGo would play the game in a such perfect manner"*. After the second game, he commented on the defeat: *"From the beginning of the match, there was not a moment in time that I*

[1] To give you an idea of how big this number is, consider that the number of atoms in the observable universe is estimated to be around 10^{80}.

felt I was leading the game". After losing the third game as well, he simply said: *"I want to apologize for being so powerless"*.

In the second game, AlphaGo made a move that would later acquire legendary status: the *Move 37*. Lee Sedol was stunned by the move and later reflected on the moment, saying: *"This move was really creative and beautiful [. . .]. This move made me think about Go in a new light"* [2].

One of the keystones of AlphaGo's success was the use of deep neural networks trained through a form of AI known as **Reinforcement Learning (RL)**. Through this technique, AlphaGo was able to develop strategies that exceeded human intuition, allowing an AI system to achieve *superhuman performance* in a domain previously considered inaccessible.

Reinforcement Learning is an interdisciplinary domain integrating principles from Machine Learning and Optimal Control, concerned with developing algorithms that train software, also referred to as *agent*, to make sequential decisions in order to achieve optimal performance.

It originates from an intuitive principle: the process of learning through interaction with the environment. In everyday life, humans continuously engage in such adaptive behavior: whether attempting to throw a ball into a basket or preparing a meal; by observing how the environment responds to their actions and subsequently adjusting their behavior to influence outcomes. Within the computational domain, RL can be regarded as the translation of these learning mechanisms into algorithmic and mathematical frameworks.

The first scientific investigations and the formalization of RL can be traced back to the period between 1950 and 1980, which was primarily characterized by the development of Dynamic Programming and the formalization of sequential decision-making through Markov Decision Processes. The subsequent period, from 1980s to 2000, was characterized by the rise of model-free methods, which enabled agents to learn optimal behaviors directly from experience without requiring explicit models of the environment. A key milestone of this phase was the introduction of Temporal Difference Learning, which combined Dynamic Programming and Monte Carlo methods to learn value functions more effectively. From 2000 to 2020, the field progressed significantly with the integration of deep learning techniques, giving

[2] All statements reported refer to documentary [1].

rise to the era of Deep Reinforcement Learning. This modern phase has produced algorithms that leverage deep neural networks to approximate complex behavior, enabling state-of-the-art performance in a wide range of applications [3].

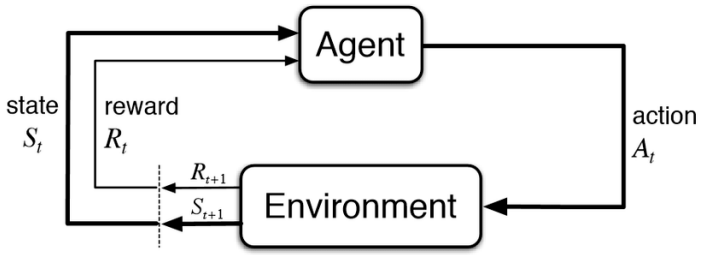

Fig. 1.1 The Agent-Environment Interaction Loop [85]

The diagram above illustrates the fundamental components of RL and their interrelationships. The *environment* represents the external system with which the *agent* interacts. At each time step, the agent perceives an observation of the environment's current state and, based on this, selects an *action*, i.e., a move or decision. Simultaneously, the agent receives a *reward signal* from the environment that quantifies the desirability of the current state or the effectiveness of the selected action. The environment then transitions to a new state, influenced by the agent's action and by its own dynamics. The primary objective of the agent is to learn a behavior that maximizes its long-term cumulative reward. The diverse methodologies within RL consist of algorithms designed to enable an agent to learn this behavior.

To date, RL has demonstrated superhuman performance in domains such as complex board games [80] [81] [19] and video games [68] [15] [94]. Beyond games, RL has shown applicability across a wide range of fields[4]: autonomous driving [64], economics and finance [43] [25], energy systems management [23], healthcare [101] [49], intelligent transportation systems [104] [26], recommender systems [79] [103], resource allocation [99] [61] [44], robotics [54] [5] [41] [2], and, more recently, tuning of large language models [105] [45] [102].

[3] Detailed information regarding the historical evolution of the RL is provided in the paper [6].
[4] The following list is presented in alphabetical order.

1.1 Mathematical Formalism

Currently, the standard mathematical formalism for RL is the **Markov Decision Process (MDP)**, which is defined as a tuple:

$$\mathscr{M} := \langle S, A, P, R, \rho_0, \gamma \rangle, \tag{1.1}$$

where:

- S is the state space, i.e., the set of all possible states of the environment;
- A is the action space;
- $P : S \times A \to \Delta(S)$ is the transition probability function[5], with

  $$P(s' \mid s, a) = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a),$$

  denoting the probability of transitioning to state s' from state s after taking action a;
- $R : S \times A \times S \to \mathbb{R}$ is the reward function, where

  $$r_t = R(s_t, a_t, s_{t+1}),$$

  defines the scalar feedback obtained when the agent moves from s_t to s_{t+1} via action a_t;
- ρ_0 is the initial state distribution, specifying the probability of the starting state s_0.
- $\gamma \in [0, 1]$ is the reward discount-rate parameter, also known simply as discount factor, a constant that determines the importance of future rewards relative to immediate rewards.

The defining feature of an MDP is the *Markov property*, which asserts that the system dynamics are memoryless. Formally, this means that the distribution of the next state

[5]Note, the symbol $\Delta(S)$ represents the all possible probability distributions over the state space S

depends only on the current state and action, and not on the history of past states or actions:

$$\Pr(s_{t+1} \mid s_0, a_0, \ldots, s_t, a_t) = \Pr(s_{t+1} \mid s_t, a_t). \tag{1.2}$$

Beyond these fundamentals, a number of additional concepts are essential for understanding and operating in the domain of RL. The principal ones are outlined below:

Policy. The policy, typically denoted by π, defines the decision-making rule followed by the agent. Formally, a policy specifies how the agent selects actions given its current state. It can be categorized as either:

- Deterministic policies, where the action is a deterministic mapping from the state:

$$a_t = \pi(s_t), \tag{1.3}$$

- *Stochastic policies*, where the action is drawn from a probability distribution over actions conditioned on the current state:

$$a_t \sim \pi(\cdot \mid s_t). \tag{1.4}$$

In simple scenarios, the policy can be implemented as a lookup table. In more complex cases, policies are typically represented by a parameterized function, such as a neural network. The parameters can be adjusted during training to improve the agent's performance.

Trajectories. A trajectory represents a sequence of states and actions experienced by the agent during interaction with the environment:

$$\xi = (s_0, a_0, s_1, a_1, \ldots) \tag{1.5}$$

Returns. The objective of an RL agent is to maximize some form of cumulative reward, also referred to as the *return*, collected along a trajectory. Typically, the return is defined as the discounted sum of rewards received over time:

$$R(\xi) = \sum_{t=1}^{T} r_0 + \gamma^t \, r_t \tag{1.6}$$

Where r_0 denote the immediate reward obtained after executing action a_0 in state s_0, and γ, the *discount factor*, determines the relative importance of future rewards compared to immediate ones. The parameter T represents the time horizon of the task. In the simplest case, T is finite, and the agent-environment interaction naturally reaches some *terminal state* at time step T. This formulation is known as the *episodic* setting. In contrast, in *continuing* tasks, the interaction does not terminate but continues indefinitely, as in continuous control or process-regulation problems. In such cases, the horizon T tends toward infinity (in the sum T = ∞), and the return is defined as *infinite-horizon discounted return*, wherein the discount factor is constrained to the interval $0 \leq \gamma < 1$. This restriction ensures the convergence of the sum.

Value Functions To evaluate and compare policies, it is useful to define value functions that quantify the expected return starting from a given state or state-action pair.

- The *state-value function*:

$$V^{\pi}(s) = \mathbb{E}_{\xi \sim \pi}\big[R(\xi) \mid s_0 = s\big], \tag{1.7}$$

  which gives the expected return starting in state s and following policy π.

- The *action-value function*:

$$Q^{\pi}(s,a) = \mathbb{E}_{\xi \sim \pi}\big[R(\xi) \mid s_0 = s, a_0 = a\big], \tag{1.8}$$

  which gives the expected return starting in state s, taking action a, and then following π.

The relationship between these two functions is given by:

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(\cdot|s)}\big[Q^{\pi}(s,a)\big]. \tag{1.9}$$

In many cases, it is often more useful to evaluate the relative quality of an action rather than its absolute value. This concept is formalized by the *Advantage function*:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s). \tag{1.10}$$

Intuitively, $A^{\pi}(s,a)$ quantifies how much better (or worse) it is to take action a in state s, compared to the average return of all possible actions in a given state s, under a policy π (i.e., the expected return of following π from that state). Advantage functions play a central role in modern Deep RL algorithms.

The Reinforcement Learning Problem Given a stochastic environment[6] and a policy π, the probability of generating a certain trajectory ξ is:

$$P(\xi \mid \pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1} \mid s_t, a_t)\, \pi(a_t \mid s_t), \tag{1.11}$$

where ρ_0 is the initial state distribution. The *objective function*, usually denoted by $J(\pi)$, which quantifies the expected return under policy π, is defined as:

$$J(\pi) = \int_{\xi} P(\xi \mid \pi) R(\xi)\, d\xi = \mathbb{E}_{\xi \sim \pi}[R(\xi)]. \tag{1.12}$$

The central optimization problem in RL is to find the **optimal policy**:

$$\pi^{\star} = \arg\max_{\pi} J(\pi). \tag{1.13}$$

The pursuit of $\pi^{\star}$ drives the development of diverse RL algorithms, each offering an approach to approximating or finding this optimal behavior.

[6] In a deterministic environment, the formulation is essentially the same but simplified, as the transition function is no longer probabilistic. Instead of mapping to a distribution over states, it maps to a single, specific next state.

1.2 Key Algorithms

This section presents a selection of key algorithms to clarify the foundational principles of RL and its recent developments.

RL algorithms can be categorized according to several dimensions that define their learning paradigms and modes of interaction with the environment. The following classification is widely used in the literature:

Model-Based / Model-Free: *Model-free* algorithms proceed without explicit knowledge of the environment's transition dynamics. They possess neither the state-transition probability distribution $P(s_{t+1}|s_t, a_t)$ nor the reward function $R(s_t, a_t, s_{t+1})$, and therefore cannot engage in forward planning or trajectory simulation. Learning is accomplished exclusively through trial-and-error interaction. Conversely, *Model-based* algorithms employ an internal model of the environment's dynamics, which allows the agent to plan by simulating future trajectories and estimating the expected outcomes of actions before execution, streamlining the learning process by focusing on the policy itself.

Value-Based / Policy-Based Methods: *Value-based* methods focus on learning an estimate of a value function, typically the state-value function $V(s)$ or the action-value function $Q(s,a)$. An optimal policy is then derived implicitly by selecting actions that maximize the learned value function. *Policy-based* methods directly learn a (parameterized) policy function, commonly by employing gradient ascent optimization to maximize the expected cumulative reward. This approach bypasses the necessity for estimation of the value function.

Offline / Online RL: In *online* RL, learning occurs through direct and continuous interaction with the environment, allowing the agent to update its policy in real-time as new experiences are gathered. *Offline* RL relies on a fixed dataset of pre-collected interactions and learns a policy without any further access to the environment.

1.2.1 Monte Carlo Methods

Monte Carlo methods are a class of statistical techniques that approximate solutions via random sampling. Their theoretical base relies on the law of large numbers: by averaging outcomes over a sufficiently large set of independent trials, the estimates converge to their expected values.

Within RL, these methods employ repeated sampling of trajectories (i.e., sequences of state, action, and reward transitions) obtained through direct interactions with the environment, in order to obtain estimates of value functions or, parameters of the optimal policy. To illustrate this idea, a simple version of the *Every-Visit Monte Carlo* algorithm [85] is presented below:

Algorithm 1 Every-Visit Monte Carlo Control

Initialize $\pi(s)$, $Q(s,a)$, and $Returns(s,a) \leftarrow \emptyset$ for all (s,a)
for each episode **do**
  Choose (s_0,a_0) randomly such that all pairs have probability > 0
  Generate an episode $(s_0,a_0,r_1,s_1,a_1,r_2,\ldots,s_T)$ following π
  $G \leftarrow 0$
  for time step $t = T-1$ down to 0 **do**
    $G \leftarrow \gamma G + r_t$
    Append G to $Returns(s_t,a_t)$
    $Q(s_t,a_t) \leftarrow \text{average}(Returns(s_t,a_t))$
    $\pi(s_t) \leftarrow \arg\max_a Q(s_t,a)$
  end for
end for

This algorithm[7] iteratively improves the policy using Monte Carlo estimates of the action-value function $Q(s,a)$ based on sampled returns. As the number of visits to each state–action pair increases, $Q(s,a)$ converges to its optimal action-value function $Q^*(s,a)$, allowing the policy to converge to the optimal one, in formula:

$$\pi^*(s_t) = \arg\max_a Q^*(s_t,a). \tag{1.14}$$

[7]The notation *Every-Visit* means that this approach updates $Q(s,a)$ using all occurrences of (s,a) within an episode; as opposed to the *First-Visit* variant, which considers only the first occurrence per episode.

1.2.2 Temporal-Difference Methods

Monte Carlo methods estimate a value functions by averaging sampled returns obtained from complete episodes. In practice, one must wait until the end of an episode to observe the total return, which is then used to update value estimates. This approach can be inefficient in episodes extremely long (waiting until the end of an episode would be prohibitive), or in some environments in which no terminal state exists.

Temporal-Difference (TD) methods update value estimates at each time step without waiting for an episode to finish. This idea builds upon Bellman's concepts of Dynamic Programming [14], leveraging *bootstrapping*, i.e., updating estimates based on other learned estimates. The key insight is that a value functions can be defined recursively in terms of subsequent estimates. Formally, consider the action-value function, $Q^{\pi}(s,a)$ under a given policy π, this satisfies the following Bellman equation:

$$Q^{\pi}(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \Big[r + \gamma \mathbb{E}_{a' \sim \pi(\cdot|s')} \big[Q^{\pi}(s',a') \big] \Big]. \tag{1.15}$$

Where:

- $s' \sim P(\cdot \mid s,a)$ denotes that the next state s' is sampled from the environment's transition dynamics conditioned on the current state–action pair (s,a);
- r is the immediate reward received when transitioning from s to s' after taking action a;
- $a' \sim \pi(\cdot \mid s')$ indicates that the subsequent action is drawn from the policy π given state s'.

This formulation enables TD methods to iteratively refine value estimates by combining immediate rewards with discounted future estimates.

A common example of a TD method is *Q-learning* [91] algorithm:

Algorithm 2 Q-Learning

Require: Learning rate $\alpha \in (0,1]$, exploration parameter ε

1: Initialize $Q(s,a)$ arbitrarily for all (s,a)
2: **for** each episode **do**
3: Initialize state s
4: **for** each step of episode **do**
5: Select action a from s using a policy derived from Q (e.g., ε-greedy)
6: Execute action a, then observe reward r and next state s'
7: Update: $Q(s,a) \leftarrow Q(s,a) + \alpha \Big[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \Big]$
8: $s \leftarrow s'$
9: **end for**
10: **end for**

In Line 5 of the algorithm, the action selection mechanism is based on a behavior policy[8], which in simpler contexts, is implemented using the ε-greedy strategy. Formally, the *ε-greedy policy* selects an action a in a given state s according to the following probability distribution:

$$\pi(a \mid s) = \begin{cases} 1-\varepsilon, & \text{if } a = \arg\max\limits_{a'} Q(s,a') \\ \frac{\varepsilon}{|A(s)|-1}, & \text{otherwise} \end{cases} \tag{1.16}$$

Where, $A(s)$ indicates all the actions available in s; while $\varepsilon \in (0,1)$ is a scalar that governs the degree of exploration. Thus, with probability $1-\varepsilon$, the agent selects the greedy action that maximizes the current estimate of the action-value function. Conversely, with probability $\frac{\varepsilon}{|A(s)|-1}$, one of the remaining (non-greedy) actions is selected at random. This strategy provides a balance between exploitation of known information, crucial for the algorithm's convergence; and exploration of the environment, crucial for generating new data and discovering optimal solutions.

[8]The behavior policy is the one used by the agent to interact with the environment and generate data. In contrast, the target policy is the one that the algorithm is trying to learn. These two policies do not necessarily coincide, as seen in Q-learning algorithm where they differ.

In line 7, the algorithm applies the Bellman update rule, which is the central mechanism of TD learning. The estimate of the action-value function $Q(s,a)$ is updated according to:

- the term $r+\gamma \max_{a'} Q(s',a')$, which is called *TD target*, represents the one-step bootstrapped target, combining the observed reward with the best predicted future return from the next state.
- the difference $r+\gamma \max_{a'} Q(s',a') - Q(s,a)$, known as *TD error*, quantifies the discrepancy between the current estimate and the target.

By iteratively reducing this error through incremental updates, the algorithm converges towards the optimal action-value function $Q^*(s,a)$. As in Monte Carlo methods, once convergence is achieved, the optimal policy is obtained from:

$$\pi^*(s_t) = \arg\max_a Q^*(s_t,a). \tag{1.17}$$

Under mild assumptions (for more details, refer to the textbook by Sutton and Barto [85]), the Q-learning algorithm converges to the optimal action-value function as the number of visits to each state-action pair grows to infinity.

In general, TD methods often converge faster than Monte Carlo methods and are computationally more efficient in practice. TD learning generally exhibits lower variance than Monte Carlo methods because it depends on single-step bootstrapped targets, while Monte Carlo estimates rely on complete returns aggregated across many steps. However, TD methods may introduce approximation bias, since updates are based on estimated rather than actual returns.

1.2.3 Gradient-based Methods

Deep Reinforcement Learning (Deep RL) refers to the integration of deep neural networks within RL frameworks, leveraging their capacity for complex nonlinear function approximation. Compared to tabular structures, neural networks provide a solution for environments with an intractably large state-action space[9]. From a practical perspective, in RL a deep neural networks serve as function approximators to represent policy, and/or a value function, in a parameterized form, where the parameters correspond to network weights and biases.

Among the most studied Deep RL approaches are *Policy Gradient Methods*, which parameterize the policy function $\pi_\theta(a|s)$ and optimize its parameters θ to maximize expected return via gradient-based techniques. These algorithms do not require the explicit estimation of a state–action value function; they learn a mapping from states to actions (or distributions over actions) directly. The key idea is to increase the likelihood of actions that yield higher returns and reduce the likelihood of actions associated with lower returns.

Formally, the optimization objective can be defined as:

$$J(\theta) = \mathbb{E}_{\xi\sim\pi_\theta}\big[R(\xi)\big], \tag{1.18}$$

where ξ denotes a trajectory generated under policy π_θ, and $R(\xi)$ is the return of that trajectory. The goal is to identify the optimal parameters θ^* that maximize $J(\theta)$:

$$\theta^* = \arg\max_\theta J(\theta). \tag{1.19}$$

Policy gradient algorithms achieve this via *gradient ascent* on policy performance, i.e., iteratively updating the parameters in the direction of the gradient of the objective:

$$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta). \tag{1.20}$$

This iterative process is repeated until the policy parameters converge, or until a predefined stopping criterion.

[9] A look-up table would necessitate prohibitive computational resources in terms of both memory storage and retrieval time.

Note, a central challenge arises in computing the gradient $\nabla_\theta J(\theta)$, since in model-free settings the environment's dynamics and reward distribution are unknown and no closed-form expression exists. This difficulty is addressed by the *Policy Gradient Theorem*[10], which provides the following formulation:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\xi \sim \pi_\theta} \left[\sum_{t=0}^{T} R_t(\xi) \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right], \tag{1.21}$$

where $R_t(\xi) = \sum_{t'=t}^{T} r_{t'}$, commonly referred to as *Return-to-go*.

Policy gradient algorithms exhibit good efficiency in high-dimensional spaces. A key advantage is their simplified implementation for learning complex policies, such as stochastic policies. A disadvantage is their high variance, which can lead to slow convergence. The following example presents a simplified version of the *Actor–Critic* algorithm. This approach incorporates a baseline function to reduce the high variance of policy gradient estimates. More specifically, the *Actor* corresponds to the policy $\pi_\theta(a \mid s)$, which selects actions, while the *Critic* estimates a value function parameterized by w, providing a learned baseline for the actor's updates.

Algorithm 3 Actor–Critic Algorithm

1: **Require:** Learning rates $\alpha > 0$ (actor), $\beta > 0$ (critic)
2: Initialize policy parameters θ and value function parameters w
3: **for** each episode **do**
4: Initialize state s
5: **for** each step of the episode **do**
6: Sample action $a \sim \pi_\theta(\cdot \mid s)$
7: Execute action a, observe reward r and next state s'
8: $\delta \leftarrow r + \gamma V_w(s') - V_w(s)$
9: $w \leftarrow w + \beta\, \delta\, \nabla_w V_w(s)$
10: $\theta \leftarrow \theta + \alpha\, \delta\, \nabla_\theta \log \pi_\theta(a \mid s)$
11: $s \leftarrow s'$
12: **end for**
13: **end for**

Note, in line 9, the value function V_w is updated via Semi-gradient method [85].

[10] For a mathematical proof of the theorem, refer to the book [85].

1.3 Open Issues

There are several open challenges in RL that are worth mentioning; these serve both as research frontiers and as criteria against which to evaluate new frameworks.

Sample Efficiency. Sample efficiency refers to the ability of an RL algorithm to achieve acceptable performance with a minimal number of interactions with the environment. Although many RL algorithms are theoretically guaranteed to converge, in high-dimensional state and action spaces, achieving this convergence in practice requires millions of interactions with the environment. Such requirements impose substantial costs, or may prove downright impractical. To delve deeper into this topic, a central aspect lies in the exploration–exploitation trade-off. Excessive exploration wastes resources on inefficient actions, whereas excessive exploitation risks premature convergence to suboptimal policies. Despite the centrality of this trade-off, there is currently no theoretical framework that prescribes an optimal balance. Instead, the choice of exploration strategy is typically determined by heuristic approaches.

Reward Design. The specification of reward signals constitutes a crucial point of RL, as it directly defines the objective that the agent is expected to maximize. Designing reward functions that accurately reflect the intended goals is rarely straightforward in practice. This process requires a thorough understanding of the task domain and its objectives, as well as the appropriate functional structure to shape the reward. For instance, poorly designed rewards can induce an agent to abuse some features of the reward function, without achieving the desired outcome. This phenomenon, commonly referred to as reward hacking [11], occurs when the agent discovers strategies that maximize the specified reward while failing to accomplish the true task. Further difficulties arise in scenarios involving sparse, delayed, or multi-objective rewards, all of which are frequent in complex tasks. Sparse or delayed rewards slow down or impede learning by providing limited feedback; while multi-objective reward formulations require balancing competing criteria, adding complexity to the learning process.

Safety and System Constraints. Standard model-free RL algorithms depend on trial-and-error exploration, which is often incompatible with safety-critical environments such as autonomous driving, healthcare systems, and industrial automation. In such domains, there exist system constraints that must never, or rarely, be violated.

Random exploratory behavior, which is typical of RL algorithms, cannot be tolerated if it leads to unsafe or catastrophic outcomes. Incorporating safety constraints into the learning process is a critical open problem that directly impacts the feasibility of RL in practice.

Chapter 2

Safe Reinforcement Learning Approaches

Before delving into the technical discussion, it is essential to clarify the notion of safety. According to the Cambridge Dictionary, safety is defined as *"a state in which, or a place where, one is protected and not exposed to danger or risk"* [22]. While this definition is quite broad, it highlights the inherent complexity of the concept, which extends beyond purely practical considerations to encompass philosophical and ethical dimensions. Indeed, assessing safety often requires a careful understanding of the operational environment, the objectives pursued, and the moral principles or societal norms established by the system's designer or community. In the context of this dissertation, the following definition is adopted:

Safety Definition: *Safety is the ability of agents to operate autonomously without causing harm to themselves or to the surrounding environment during real-world interactions.*

The precise nature and severity of what constitutes *"harm"* are inherently context-dependent and may vary significantly depending on the domain of application.

Standard model-free RL paradigms are unable to offer safety guarantees during the exploration phases of learning. This is a direct consequence of their trial-and-error nature and the absence of any prior knowledge regarding the environment's dynamics. While this has no impact when training occurs entirely within a virtual environment,

it becomes a critical concern in real-world settings, where exploratory actions can lead to unsafe outcomes. Domains particularly affected by this limitations include[1]: autonomous driving [46], energy resource management [21], financial applications [12], healthcare systems [92], human interaction with large language model[2] [50], robotics [20].

Safe Reinforcement Learning (Safe RL) is a specialized subfield of RL concerned with the development of learning policies that not only maximize the expected return but also adhere to safety constraints during both the training and deployment phases. In contrast to traditional RL, which is exclusively focused on reward maximization, Safe RL explicitly incorporates mechanisms to prevent agents from undertaking actions that could result in failures, violations of operational constraints, or other unsafe outcomes in real-world applications. The objective of Safe RL is therefore twofold: to ensure that agents achieve long-term reward optimization, as guided by a reward signal, while simultaneously respecting safety requirements that guarantee acceptable system performance and mitigate risk. This section introduces the primary approaches currently utilized in Safe RL: Constraint-Based Methods and Shielding Methods.

[1] The list is presented in alphabetical order.

[2] I.e., generating harmful, biased, or inappropriate content.

2.1 Constraint-Based Methods

Constraint-Based Methods are typically formulated within the framework of **Constrained Markov Decision Processes (CMDP)** [8], in which the objective is to maximize the expected return while ensuring that additional cost measures remain bound to prescribed thresholds. This approach can be considered a natural extension of the MDP framework, as it preserves the core optimization principle of maximizing expected rewards while augmenting it with constraints. Formally, a CMDP is defined as the tuple:

$$\langle S, A, P, R, \rho_0, \gamma \rangle \cup C, \tag{2.1}$$

where $\langle S, A, P, R, \rho_0, \gamma \rangle$ corresponds to a standard MDP (Section 1.1), and C which encodes the constraints applied to the policy. In essence, $C = \{C_1, C_2, \ldots, C_k\}$ is a set of cost functions, C_i, that maps a state-action pair to a real number:

$$C_i : S \times A \to \mathbb{R} \tag{2.2}$$

This cost functions are subject to certain constraints, the formulation of which can vary depending on the application domain. These constraints restrict the overall policy space Π, defining a subset of admissible policies that satisfy the constraint conditions, denoted as Π_S. In mathematical terms, this relationship is:

$$\Pi_S \subseteq \Pi, \tag{2.3}$$

The goal of a CMDP is to optimize reward performance while ensuring compliance with the cost constraints. Formally, the optimization problem is:

$$\max_{\pi \in \Pi_S} J(\pi) \tag{2.4}$$

where $J(\pi)$ is the expected return under policy π, defined in Eq. (1.12).

This Section presents three of the most common formulations for safety constraints, beginning with the **Expected Cumulative Constraint**. This formulation is currently the most studied and underlies the majority of CMDP-based methods [40].

Analogous to the reward-based value functions, are defined cost state-value functions $V_{C_i}^{\pi}$ and cost action-value functions $Q_{C_i}^{\pi}$. For instance, a straightforward version of the cumulative cost state-value function can be expressed as:

$$V_{C_i}^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} C_i(s_t, a_t) \,\middle|\, s_0 = s\right], \tag{2.5}$$

where $C_i(s_t, a_t)$ denotes the *immediate constraint cost* incurred at time step t when the agent is in state s_t and takes action a_t.

Let $b_i \in \mathbb{R}$ is the threshold that restricts the expected cumulative values of cost function C_i, and k denotes the number of cost function considered; then the set of admissible policies that satisfy the constraint conditions is formally defined as:

$$\Pi_S = \left\{\pi \in \Pi \;\middle|\; V_{C_i}^{\pi}(s) \leq b_i\right\} \quad \forall i \in \{1, \ldots, k\}. \tag{2.6}$$

Expected Instantaneous Constraints. Alternatively to cumulative cost formulations, several studies (e.g., [17] [9]) focus on constraining the instantaneous cost at each time step. The objective here is to enforce safety requirements locally. Formally, the constraint is defined as

$$\mathbb{E}_{\pi}\big[C_i(s_t, a_t)\big] \leq \nu_t^i, \quad \forall t, \tag{2.7}$$

where $\nu_t^i \in \mathbb{R}$ is the per-step threshold. So, unlike the cumulative constraint, which bounds the expected cumulative cost across an entire trajectory, this formulation enforces compliance at every step.

Probability Constraints. Formulations for safety constraints can also be based on probabilistic criteria, such as bounding the probability that cumulative costs exceed a specified threshold. In this setting, the probabilistic constraint is expressed as

$$\mathbb{P}\left(\sum_{t=0}^{T} C_i(s_t, a_t) \geq \eta_i\right) \leq \varepsilon_i, \tag{2.8}$$

where $\eta_i \in \mathbb{R}$ denotes the cumulative cost threshold for each trajectory, and $\varepsilon_i \in (0,1)$ specifies the acceptable probability bound. Intuitively, these constraints enforce that the likelihood of the cumulative cost surpassing the threshold η_i must remain below ε_i. This logic can also be extended to the case of instantaneous constraints.

Note, at the application level, the cost constraint formulations presented can have diverse interpretations, such as limits on risk exposure or resource consumption bounds; depending on the characteristics of the domain.

Several methods have been proposed to solve the CMDP framework. The most prevalent ones are outlined below.

2.1.1 Primal-Dual Optimization

In mathematical optimization, *Lagrangian relaxation* refers to the process by which complex optimization problems are simplified through the relaxation of certain constraints via the introduction of Lagrangian multipliers [16]. To illustrate the concept, consider the following programming problem:

$$\begin{aligned} \min_{x \in \mathbb{Z}_+^n} \quad & z_{\mathscr{P}}(x) = c^\top x \\ \text{s.t.} \quad & A_1 x \geq b_1, \\ & A_2 x \geq b_2, \end{aligned} \tag{2.9}$$

where $c \in \mathbb{R}^n$, A_1, A_2 are constraint matrices, and b_1, b_2 are vectors of appropriate dimension. $\mathbb{Z}_+^n$ denotes set of all n-dimensional vectors whose components are non-negative integers.

A Lagrangian relaxation of problem (2.9) is obtained by relaxing the constraints $A_1 x \geq b_1$ and incorporating them into the objective function using a vector of Lagrange multipliers $\lambda \geq 0$. This leads to the relaxed problem:

$$\begin{aligned} \min_{x \in \mathbb{Z}_+^n} \quad & z_{\mathscr{L}}(x, \lambda) = c^\top x + \lambda^\top (b_1 - A_1 x) \\ \text{s.t.} \quad & A_2 x \geq b_2. \end{aligned} \tag{2.10}$$

In this formulation, violations of the relaxed constraints $A_1 x \geq b_1$ are penalized directly in the objective function[3] by the multipliers λ. For each fixed choice of λ, this yields a relaxed problem that is generally easier to solve than the original one. The optimal objective value of the relaxed problem for a given λ provides a valid lower bound (in minimization problems) or upper bound (in maximization problems) on the optimal value of the original problem. By optimizing over all possible multipliers $\lambda \geq 0$, one can obtain the "tightest" possible bound.

This method holds significant utility for solving the CMDP framework [8] [29] [33] [82], and the associated procedures are commonly referred to as *Primal-Dual Optimization* (PDO) methods. To better illustrate this, consider the following generic optimal control problem:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} R(s_t, a_t, s_{t+1}) \;\middle|\; \pi_\theta\right] \tag{2.11}$$

$$\begin{aligned} \max_{\theta} \quad & J(\theta) \\ \text{s.t.} \quad & U_i(\theta) \geq b_i, \quad i = 1, \ldots, k. \end{aligned} \tag{2.12}$$

where $J(\theta)$ is the expected return under policy π_θ, and the functions $U_i(\theta)$ represent constraint values with thresholds b_i. Note that the direct solution of this type of constrained problem is typically a non-convex problem (non-convex optimization is at least NP-hard). The PDO method, proposes an alternative solution: solving its relaxed form. The corresponding Lagrangian objective is defined as:

$$\mathscr{L}(\theta, \lambda) = J(\theta) - \sum_{i=1}^{k} \lambda_i \big(U_i(\theta) - b_i\big), \tag{2.13}$$

where $\lambda \in \mathbb{R}_+^k$ are the Lagrange multipliers associated with the constraints. And, the optimization problem becomes:

$$(\theta^*, \lambda^*) = \arg\min_{\lambda} \max_{\theta} \mathscr{L}(\theta, \lambda) \tag{2.14}$$

[3]Note, $z_{\mathscr{L}}(x, \lambda)$, is also called *Lagrangian objective function*.

The linear simplification transforms the optimization problem into a convex one, which can be solved using gradient-based techniques. In practice, this results in a mini-max problem: the policy parameters are updated to maximize the objective via gradient ascent, while the multipliers are updated to minimize it via gradient descent. This iterative procedure ideally converges to a saddle point (θ^*, λ^*), corresponding to a constraint-satisfying policy.

Note, PDO methods are used for cumulative, instantaneous and probabilistic constraint formulation [62].

Although proven to be effective, this approach has several drawbacks that warrant consideration. The methodology exhibits optimization challenges due to its sensitivity to the initialization of the Lagrange multipliers. Furthermore, the two-time-scale nature of the optimization, where the policy parameters are updated on a different timescale than the multipliers[4], poses significant practical difficulties. A more critical drawback from a safety standpoint, is the lack of guarantees regarding the safety of policies generated during the training process. In essence, given the trial-and-error nature of exploration and the lack of prior knowledge of the environment, the constraints may not be satisfied, particularly in the early stages of training.

2.1.2 Constrained Policy Optimization

Constrained Policy Optimization (CPO) methods [4] extend the *Trust Region Policy Optimization* (TRPO) approach [76] to solve CMDP problems involving cumulative constraints.

In TRPO, the goal at each iteration is to update the policy in a way that maximizes the expected return while ensuring that the new policy remains within a predefined trust region around the current policy. This trust region is defined by bounding the distance between the updated policy and the current policy, with the Kullback–Leibler divergence, used as the distance metric. A more detailed explanation follows: let π denote a policy, and let π' denote its next iteration updated version. The difference in their objectives can be expressed using the *relative policy performance identity* [51]:

[4]Multipliers are solved on a slower timescale.

$$J(\pi') - J(\pi) = \mathbb{E}_{\xi\sim\pi'}\left[\sum_{t=0}^{T}\gamma^t A^{\pi}(s_t, a_t)\right], \tag{2.15}$$

where γ represents the discount factor, $A^{\pi}(s,a)$ denotes the advantage function under the older policy π, and the expectation is taken over trajectories $\xi = (s_0, a_0, s_1, s_2 \dots)$ generated by the new policy π'. To mathematically derive this identity, refer to the book [39].

The identity (2.15), provides a way to measure the improvement of a new policy π' over a reference policy π. If the difference is strictly positive, then π' achieves higher expected return than π; so π' is an improved policy. During policy training, ensuring at least this type of relationship: $J(\pi') - J(\pi) \geq 0$, would guarantee monotonic performance improvement[5]. This would be highly desirable when optimizing policies parameterized by neural networks, which are susceptible to significant performance degradation (collapse) following bad updates [34].

Select π' such that maximizes the performance difference $J(\pi') - J(\pi)$, is equivalent to maximizing $J(\pi')$. Formally:

$$\max_{\pi'}\Big(J(\pi') - J(\pi)\Big) \iff \max_{\pi'} J(\pi'), \tag{2.16}$$

from which derive:

$$\max_{\pi'} J(\pi') = \max_{\pi'}\ \mathbb{E}_{\xi\sim\pi'}\left[\sum_{t=0}^{T}\gamma^t A^{\pi}(s_t, a_t)\right]. \tag{2.17}$$

This formulation seems to suggest an interesting optimization update rule, as it quantifies the performance of the new policy π' by leveraging the Advantage function derived from the existing policy π. However, for a policy update, the expectation in (2.17) requires trajectories to be generated by π'. But the new policy π' is not available until after the update has been performed. This is a paradox, consequently computing this expectation is not possible during training. To make the update tractable, one typically resorts to approximations, for example by re-weighting trajectories from π using importance sampling ratios:

[5]Note, in the worst case, when the parameters of the two policies are equal, no improvement is got.

$$J(\pi') - J(\pi) \approx \mathbb{E}_{\xi\sim\pi}\left[\sum_{t=0}^{T} A^{\pi}(s_t, a_t)\frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)}\right]. \tag{2.18}$$

The resulting expression, known as the *Surrogate Advantage*, is something that can be optimized using trajectories sampled from the old policy π. It is important to acknowledge that this formulation is an approximation, and its use in the update introduces a degree of error, which cannot guarantee monotonic performance improvement at every policy iteration. However, the error can be formally bounded in terms of the Kullback-Leibler divergence between the two policies. When the policies are close in terms of their Kullback-Leibler divergence, the approximation is highly accurate.

Finally, let π_θ denote a parameterized policy with parameters θ, and, θ_{old} represents the parameter of the policy before the optimization step; the TRPO formulation can be expressed as:

$$\max_{\theta} J(\theta) \quad \text{s.t.} \quad \bar{D}_{\mathrm{KL}}(\theta \,\|\, \theta_{\mathrm{old}}) \leq \delta. \tag{2.19}$$

Where, $J(\theta)$ is the Surrogate Advantage objective, expressed in right side of (2.18). Let Kullback-Leibler divergence denoted as D_{KL}, the average Kullback-Leibler divergence can be expressed as:

$$\bar{D}_{\mathrm{KL}}(\theta \,\|\, \theta_{\mathrm{old}}) = \mathbb{E}_{s\sim\pi_{\theta_k}}\Big[D_{\mathrm{KL}}\big(\pi_\theta(\cdot|s) \,\|\, \pi_{\theta_{\mathrm{old}}}(\cdot|s)\big)\Big]. \tag{2.20}$$

and, δ is a hyperparameter that needs to be tune[6].

Since the theoretical TRPO update is not directly tractable, particularly for large neural network policies, approximations are employed during implementation. Specifically: a linear approximation for the Surrogate Advantage objective via a first-order Taylor expansion (i.e., linearization around the parameters of politics π_{old}), and a second-order Taylor expansion for the approximation of the KL-divergence constraint.

$$J(\theta) \approx g^{\top}(\theta - \theta_{old}) \tag{2.21}$$

[6]Note, Kullback-Leibler is always non-negative.

where g represents the gradient vector of the Surrogate Advantage objective function at the point θ_{old}.

$$\bar{D}_{\mathrm{KL}}(\theta \,\|\, \theta_{\mathrm{old}}) \approx \frac{1}{2}(\theta - \theta_{old})^{\top} H (\theta - \theta_{old}) \tag{2.22}$$

where H is the Hessian matrix of the Kullback-Leibler divergence function at the point θ_{old}.

Finally, the following optimization rule can be obtained:

$$\begin{aligned} \theta_{k+1} \;=\; \arg\max_{\theta} \quad & g^{\top}(\theta - \theta_k) \\ \text{s.t.} \quad & \tfrac{1}{2}(\theta - \theta_k)^{\top} H (\theta - \theta_k) \;\leq\; \delta, \end{aligned} \tag{2.23}$$

Note, both g and H are defined with respect to θ_k. This optimization problem can be analytically resolved using Lagrangian duality, wherein the Lagrange multiplier λ is typically reduced to an algorithm parameter. Formally:

$$\theta_{k+1} \;=\; \theta_k \;-\; \lambda\, H^{-1} g. \tag{2.24}$$

The CPO formulation is based on the idea of incorporate constraints directly into the trust-region optimization framework. Formally, utilizing the approximations outlined above, and, let

$$\hat{c}_i \;=\; V_{C_i}^{\theta_k} \;-\; b_i, \tag{2.25}$$

where: $V_{C_i}^{\theta_k}$ denotes the expected cumulative cost for i-th constraint under policy π_{θ_k}, and, b_i is the cost bound for the i-th constraint.

The optimization problem can be defined as follows:

$$\begin{aligned} \theta_{k+1} \;=\; \arg\max_{\theta} \quad & g^{\top}(\theta - \theta_k) \\ \text{s.t.} \quad & \hat{c}_i + q_i^{\top}(\theta - \theta_k) \;\leq\; 0, \\ & \tfrac{1}{2}(\theta - \theta_k)^{\top} H (\theta - \theta_k) \;\leq\; \delta, \end{aligned} \tag{2.26}$$

where: q_i is the gradient of i-th cost function in θ_k. Regarding g and H, as indicated in the previous formulas, denote respectively the gradient of the Surrogate Advantage

objective and Hessian of the Kullback-Leibler divergence. This optimization problem can be solved efficiently using Lagrange duality. Therefore, consider:

- $Q = [q_1, \ldots, q_n]$
- $\hat{c} = [\hat{c}_1, \ldots, \hat{c}_n]^\top$
- $\eta = g^\top H^{-1} Q$
- $Z = Q^\top H^{-1} Q$

The dual problem associated with the formulation (2.26) can then be expressed as:

$$\min_{\lambda \geq 0,\, \iota \succeq 0} \left(\frac{\lambda}{2} \delta - \iota^\top \hat{c} + \frac{1}{2\lambda} \Big(g^\top H^{-1} g - 2\eta^\top \iota + \iota^\top Z \iota \Big) \right), \tag{2.27}$$

If (λ^*, ι^*), the value of Lagrange multipliers, constitute an optimal solution to the problem (2.27), then the corresponding primal solution is:

$$\theta^* = \theta_k + \frac{1}{\lambda^*} H^{-1} \Big(g - Q\, \iota^* \Big). \tag{2.28}$$

The algorithm thus proceeds by solving the dual optimization problem (2.27) for the optimal multipliers (λ^*, ι^*), and then substituting them into (2.28) in order to obtain the policy update direction.

Although this class of algorithms guarantees a monotonic improvement of the policies during the training process and produces stable performance; however, it introduces several sources of error and practical challenges. First, approximation used can produce error may cause divergence from the true optimization objective, and currently, there is no rigorous theoretical analysis quantifying this gap. In application, this can degrade policy performance, and expose to frequent constraint violations during training. Second, the practical implementation of this method necessitates the computation and storage of the inverse of the matrix H, which may introduce additional high computational costs at each update step, in an algorithm that is already intrinsically expensive.

2.1.3 Projection-based Constrained Policy Optimization

Projection-Based Constrained Policy Optimization (PCPO) [98], leveraging the principles of CPO framework (Section 2.1.2), is a methodology that optimizes the policies subject to cumulative constraints, by decoupling policy improvement and constraint satisfaction. Its basic procedure is composed of two steps for each update:

1) Reward improvement step: in the first phase, the algorithm maximizes the expected cumulative reward by employing the TRPO approach [76]. This step generates an intermediate policy that does not guarantee compliance with the safety constraints. Let the intermediate policy obtained in this step be denoted as $\pi_{k+\frac{1}{2}}$, with its parameters represented by $\theta_{k+\frac{1}{2}}$; formally this step can be expressed as:

$$\begin{aligned} \theta_{k+\frac{1}{2}} \;&=\; \arg\max_{\theta} \; \mathbb{E}_{\substack{s\sim d^{\pi_{\theta_k}} \\ a\sim\pi_\theta}} \left[A_R^{\pi_{\theta_k}}(s,a) \right] \\ &\text{s.t.} \quad \bar{D}_{\mathrm{KL}}(\theta \,\|\, \theta_{\mathrm{k}}) \;\leq\; \delta. \end{aligned} \tag{2.29}$$

where:

- $d^{\pi_{\theta_k}}$ is the discounted state visitation distribution induced by the policy π_k, formally: $d^{\pi_k}(s) \;=\; (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \, P(s_t = s \mid \pi_k)$
- $A_R^{\pi_{\theta_k}}$ is the reward Advantage function under π_k
- $\bar{D}_{\mathrm{KL}}$ represents the average Kullback-Leibler divergence, see (2.20) for details.
- δ is the maximum allowable KL divergence

In simple term, the intermediate policy is constrained to lie within a δ-neighbourhood of the policy before the update, π_k.

2) Reward projection step: subsequently, the second phase consists in projecting the intermediate policy onto the constraint (safe) set. Formally:

$$\begin{aligned} \theta_{k+1} \;&=\; \arg\min_{\theta} \; D\left(\pi_\theta, \pi_{\theta_{k+\frac{1}{2}}} \right) \\ &\text{s.t.} \quad J_C(\theta_k) + \mathbb{E}_{\substack{s\sim d^{\pi_{\theta_k}} \\ a\sim\pi_\theta}} \left[A_C^{\pi_{\theta_k}}(s,a) \right] \;\leq\; b. \end{aligned} \tag{2.30}$$

where:

- D is a metric that quantifies the distance between policies, and, must be minimized. Note, the authors of the paper [98] that introduced this method, consider two distance measures D: the ℓ_2 norm and the Kullback–Leibler divergence.
- $J_C(\theta_k)$ is the cumulative cost constraint under the current policy π_k
- $A_C^{\pi_{\theta_k}}$ is the cost constraint Advantage function under π_k
- b is the constraint threshold.

This step ensures that the constraint-satisfying policy resulting π_{k+1}, is as close as possible to the intermediate policy $\pi_{k+\frac{1}{2}}$, obtained in previous step.

Analogous to the CPO framework, the implementation of this method also necessitates the employment of practical approximations. The overall PCPO update rule, derived from these approximations, is as follows:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^\top H^{-1} g}} H^{-1} g - \max\left(0, \frac{\sqrt{\frac{2\delta}{g^\top H^{-1} g}}\, a^\top H^{-1} g + h}{a^\top L^{-1} a}\right) L^{-1} a \quad (2.31)$$

where:

- g is the gradient of the reward Advantage function,
- a is the gradient of the cost constraint Advantage function,
- $h = J_C(\theta_k) - b$, is the constraint violation of the policy π_k,
- H is the Hessian of the KL divergence constraint
- L is the metric, matrix for the projection distance.

Note that the first part of the subtraction concerns to step 1, while the second part relates to step 2. To understand the derivation, refer to the original paper [98].

PCPO demonstrates superior sample efficiency compared to CPO (as the latter's update rule often tend to a slow progress in learning constraint-satisfying policies). Additionally, PCPO exhibits reduced constraint violations while achieving comparable reward improvements during training. However, deriving from CPO, PCPO inherits its main drawbacks, including computational expense (due to its reliance on TRPO update), and, potential safety risks associated with unsafe actions during training; for which theoretical guarantees regarding their control and measurement continue to be lacking.

2.1.4 Interior-point Policy Optimization

Interior-Point Policy Optimization (IPO) [60], drawing inspiration from Interior-point optimization methods [18], augments the policy optimization objective with logarithmic barrier functions to enforce cumulative constraints. IPO builds upon *Proximal Policy Optimization* (PPO) algorithm [78], which is a variant of TRPO (see Section 2.1.2), which maintains the core principles of trust region optimization while offering a simplified implementation; in other words, it use a few tricks to keep the new policies sufficiently close to the preceding one. Notably, PPO is characterized by increased computational efficiency relative to TRPO; nonetheless, empirical findings indicate that PPO's performance is at least on par with that of TRPO.

PPO defines the clipped objective:

$$\mathscr{L}_{\text{CLIP}}(\theta) \;=\; \mathbb{E}_{\xi\sim\pi_\theta}\left[\sum_{t=0}^{T}\min\left(\rho_t(\theta)A_t^{\pi_\theta},\; \text{clip}(\rho_t(\theta),\; 1-\varepsilon,\; 1+\varepsilon)A_t^{\pi_\theta}\right)\right] \quad (2.32)$$

In formula:

- Expectation is taken over a trajectory ξ sampled from the policy π_θ parameterized by θ.
- The importance ratio, $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. Where $\pi_{\theta_{\text{old}}}$ represents previous policy parameterized by θ_{old}.
- $A_t^{\pi_\theta}$ denotes the Advantage estimate at time step t under policy π_θ.

- The operator clip$(\cdot)$ constrains the importance ratio, preventing $\rho_t(\theta)$ from moving outside the interval $[1-\varepsilon, 1+\varepsilon]$.
- The scalar $\varepsilon > 0$ is a clipping hyperparameter that controls the trust region.

IPO extends this formulation by incorporating cumulative constraints into the optimization problem:

$$\begin{aligned} \max_{\theta} \quad & \mathscr{L}_{\text{CLIP}}(\theta) \\ \text{s.t.} \quad & V_{C_i}^{\theta} \leq b_i, \quad i = 1, \ldots, k \end{aligned} \tag{2.33}$$

where $V_{C_i}^{\theta}$ denotes the expected cumulative cost for constraint i under policy π_θ. To simplify (2.33), IPO replaces the constraints with differentiable logarithmic barrier functions. Specifically, for each constraint, let's define the indicator function $I(V_{C_i}^{\theta})$ as follows:

$$I(V_{C_i}^{\theta}) = \begin{cases} 0, & V_{C_i}^{\theta} \leq b_i, \\ -\infty, & V_{C_i}^{\theta} > b_i. \end{cases} \tag{2.34}$$

The logarithmic barrier function is as a smooth differentiable approximation to the indicator function, in this case, defined as:

$$\phi(V_{C_i}^{\theta}) = \begin{cases} \frac{\log\left(-V_{C_i}^{\theta} + b_i\right)}{\varsigma_i}, & \text{if } V_{C_i}^{\theta} < b_i, \\ -\infty, & \text{if } V_{C_i}^{\theta} \geq b_i. \end{cases} \tag{2.35}$$

where $\varsigma_i > 0$ indicate a hyperparameter; if its value increases, the approximation to the indicator function improves. The overall objective becomes:

$$\mathscr{L}_{\text{IPO}}(\theta) = \mathscr{L}_{\text{CLIP}}(\theta) + \sum_{i=1}^{k} \phi(V_{C_i}^{\theta}). \tag{2.36}$$

In this form, first-order optimization methods can be employed to update the parametric policy.

IPO builds upon PPO algorithm, inheriting the trust-region property, thereby facilitating monotonic policy improvement and ensuring stable learning performance.

In more, it is relatively straightforward to implement compared to other constrained methods, especially when dealing with multiple constraints. However, a limitation is that IPO assumes the training policies are feasible (safe) upon initialization; approaches to address this issue are discussed in [59] [61]. Furthermore, a theoretical analysis to guarantee performance has yet to be provided. Finally, similar to other methods, there can be frequent violations of constraints during training.

2.1.5 Lyapunov-based Methods.

Lyapunov-based methods [72][30] address expected cumulative constraints by constructing Lyapunov functions, a class of scalar potential functions traditionally used to analyze system stability [52]. More precisely: a Lyapunov function is as a non-negative scalar function of a system's state, such that its value non increase (or monotonically decreases) along the system's trajectories (i.e. when the states change).

To formalize these methods, consider, for each state s in space state S, the generic Bellman operator with respect to a certain policy π and a generic cost function h, defined as follows:

$$T_h^{\pi}[V](s) \;=\; \sum_{a\in A} \pi(a \mid s) \left[h(s,a) + \gamma \sum_{s'\in S} P(s' \mid s,a) V(s') \right]. \tag{2.37}$$

In this section, also consider the function $d(s)$, immediate constraint cost (state-dependent); whose values are bounded within the interval $[0, D_{max}]$. Additionally, the corresponding expected cumulative constraint cost function is introduced:

$$D_{\pi}(s_0) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t d(s_t) \,\middle|\, \pi, s_0 \right] \tag{2.38}$$

where s_0 is the initial state and γ it's the usual discount factor. The safety constraints are defined as:

$$D_{\pi}(s_0) \leq d_0. \tag{2.39}$$

where $d_0 \geq 0$ is constraint threshold.

To simplify the exposition, this section focuses on solving the CMDP problem under a single immediate constraint cost[7].

The set of valid Lyapunov candidate function with respect to an initial state s_0 and constraint threshold d_0, is defined as:

$$\mathscr{L}^{\pi_B}(s_0, d_0) = \left\{ L \;\middle|\; T_d^{\pi_B}[L](s) \leq L(s),\ \forall s \in S;\quad L(s_0) \leq d_0 \right\} \tag{2.40}$$

where:

- L is a Lyapunov function $L : S \to \mathbb{R}_{\geq 0}$
- π_B is a feasible policy. I.e., it satisfies the safety constraints, formally: $D_{\pi_B}(s_0) \leq d_0$
- $T_d^{\pi_B}[L](s)$ denotes the operator defined in Equation (2.37), applied to the function L, incorporating the constraint cost d, under policy π_B.
- The conditions inside the set specify:

$$\begin{aligned} &T_d^{\pi_B}[L](s) \leq L(s), && \text{(monotonicity constraint)} \\ &L(s_0) \leq d_0, && \text{(initial constraint).} \end{aligned}$$

Crucially, it must be noted that: this formulation assume to have access at least one policy π_B.

For any Lyapunov function $L \in \mathscr{L}^{\pi_B}(s_0, d_0)$, let define the set of L-induced policies as:

$$\mathscr{F}_L = \{\pi \mid T_d^{\pi}[L](s) \leq L(s),\ \ \forall s \in S\}. \tag{2.41}$$

Due to the contraction property of the operator T_d^{π}, together with the condition $L(s_0) \leq d_0$, any policy $\pi \in \mathscr{F}_L$ is guaranteed to be feasible [30]. However, $\mathscr{F}_L$ does not necessarily contain an optimal solution. The central point in the Lyapunov approach is to design a Lyapunov function $L \in \mathscr{L}^{\pi_B}(s_0, d_0)$ such that $\mathscr{F}_L$ contains

[7]The extensions are conceptually straightforward but may be computationally demanding.

an optimal policy π^*. Formally:

$$L(s) \;\geq\; T_d^{\pi^*}[L](s),\; \forall s \in S; \quad L(s_0) \;\leq\; d_0 \tag{2.42}$$

Chow et al. in paper [30], demonstrate that a Lyapunov function satisfying the above condition can be delineated as:

$$L^{\varepsilon}(s) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \big(d(s_t) + \varepsilon(s_t)\big) \;\Big|\; \pi_B,\; s_0 = s\right] \tag{2.43}$$

In practice, this formulation introduces an auxiliary term $\varepsilon(s) \geq 0$, which can be interpreted as a compensatory, state-dependent immediate constraint cost. This term serves to quantify the maximum constraint budget that may be leveraged for policy improvement (from π_B to π^*). Formally, $\varepsilon(s)$ is uniformly bounded[8] by:

$$\varepsilon^*(s) \approx D_{TV}(\pi^* \| \pi_B)(s) \;\cdot\; D_{\max} \tag{2.44}$$

where:

$$D_{\mathrm{TV}}(\pi^* \| \pi_B)(s) = \frac{1}{2} \sum_{a \in A} \big|\pi_B(a \mid s) - \pi^*(a \mid s)\big| \tag{2.45}$$

This measure quantifies the difference in action distributions between the two policies at that state. Note, $D_{\mathrm{TV}}(\pi^* \| \pi_B)(s)$ requires knowledge of the optimal policy π^*, which is typically unknown, or difficult to approximate. In order to resolve this problem, the authors of the paper, replace $\varepsilon^*(s)$ with a computable surrogate auxiliary cost $\tilde{\varepsilon}(s)$. Explicitly, $\tilde{\varepsilon}(s)$ is computed by solving the following Linear Programming problem:

$$\tilde{\varepsilon}(s) \in \arg\max_{\varepsilon} \left\{ \sum_{S} \varepsilon(s) :\; d_0 - D_{\pi_B}(s_0) \geq \mathbf{1}(s_0)^{\top} \left(I - \gamma\{P(s'|s, \pi_B(s)\}\right)^{-1} \varepsilon \right\} \tag{2.46}$$

where $\mathbf{1}(s_0)$ denotes a one-hot vector with its non-zero element corresponding to state $s = s_0$. In essence, the condition: $d_0 - D_{\pi_B}(s_0) \geq \mathbf{1}(s_0)^{\top} (I - \gamma\{P(s'|s, \pi_B(s)\})^{-1} \varepsilon$

[8] I.e., $\varepsilon(s) \in [-\varepsilon^*(s), \varepsilon^*(s)]$

ensures that ε not exceed the available safety-constraint budget, which is quantified by the difference between d_0 and D_{π_B}.

Leveraging the Lyapunov function $L^{\tilde{\varepsilon}}$ constructed on $\tilde{\varepsilon}$, Chow et al. introduced an algorithm wherein at each iteration k, the function $L^{\tilde{\varepsilon}_k}$ is recomputed according to Formula (2.46), with respect to the current baseline policy $\pi_B = \pi_k$. This method (named Safe Policy Iteration) guarantees monotonic policy improvement and asymptotic convergence.

Nevertheless, especially during the initial stages of training, transient constraint violations may occur. In more, a technical limitation of these approaches is the requirement to solve a Linear Programming problem at every iteration, which can involve considerable computational resources. Furthermore, the reliance on a feasible baseline policy for initialization[9] may be restrictive in practical domains where such a policy is difficult to obtain.

2.1.6 Gaussian Processes-based Methods

A *Gaussian Processe* (GP) can be thought of as a generalization of the Gaussian probability distribution, which characterizes random variables with scalars or vectors (in the case of multivariate distributions), whereas, a GP defines a probability distribution over functions. Formally, let mean function $m(x)$ and a covariance function $k(x,x')$ of a process $f(x)$ be defined as:

$$m(x) = \mathbb{E}[f(x)], \qquad k(x,x') = \mathbb{E}\big[(f(x)-m(x))(f(x')-m(x'))\big], \tag{2.47}$$

the Gaussian Process is denoted as

$$f(x) \sim GP\big(m(x), k(x,x')\big), \tag{2.48}$$

completely specified by its mean function and covariance function.

GPs models are often used in Machine Learning for regression and classification problem [74]. Specifically, in RL, it is used to model the dynamics of the system and/or the value functions [56] [71].

[9] I.e., it is assumed that the initial policy π_0 is a feasible policy.

In Safe RL, GPs have found interesting applications in the estimation of instantaneous constraint cost functions [88] [83] [90]. In classical constrained RL algorithms, the safety function is typically assumed to be known a priori. The main distinction of GP-based approaches is that the agent must explore the state space to learn a safety function, that is a priori unknown, while guaranteeing satisfaction of the safety constraints during exploration. Clearly, directly solving this problem is intractable without any presupposition. Indeed, without even an initial knowledge of which state-action pairs are safe, an agent cannot take any safe action from the very beginning. To address these difficulties, two key assumptions are commonly adopted:

- Safe initialization: the agent begins in an initial set of states $S_0 \subseteq S$ which is known in advance to be safe.
- Regularity of the safety instantaneous function: basically, the idea is that similar states tend to have similar safety-cost values (or similar level of safety). Formally, it is assumed that the state space S is endowed with a positive definite kernel function $k_g(\cdot,\cdot)$ and the safety function $g : S \to \mathbb{R}$ is assumed to have a bounded norm in the associated Reproducing Kernel Hilbert Space (RKHS)[10] [75]. In simple terms, the kernel function k_g is used to capture the similarity between states. These assumptions allow to model g as a GP:

$$g(s) \sim GP\big(m_g(s), k_g(s,s')\big), \tag{2.49}$$

  Furthermore, g is assumed to be L-Lipschitz continuous with respect to some distance metric $d(\cdot,\cdot)$ defined on S.

By modeling unknown safety costs via GPs, an agent can divide the state space into safe, uncertain, and unsafe areas; and try to maximize the cumulative reward safely.

Although GP methods have proven effective in terms of safety and data efficiency. However, relying on probabilistic safety estimates implies that safety constraints may be violated, particularly in the presence of noise and highly complex environments. Finally, this approach is not suitable for domains where the two assumptions above do not hold.

[10] Note, the norm induced by the inner product of the RKHS, measures how smooth functions are with respect to the kernel.

2.2 Shielding Method

Shielding [55] is a runtime control methodology that synthesizes a reactive component, referred to as *shield*, to monitor and correct a system's outputs to ensure adherence to predefined safety specifications. This methodology has recently gained attention for its effectiveness in enforcing safety within RL frameworks [7] [48] [67] [32][24]. In this regard, two primary configurations have been proposed for integrating the shield into the RL workflow:

Post-shielding: In this configuration the shield acts as a runtime controller, placed after the RL agent. In essence, it continuously monitors the state of the environment and the actions chosen by the agent. Unsafe actions are intercepted and replaced with safe alternatives (if exist), thereby preventing violations of safety constraints during execution.

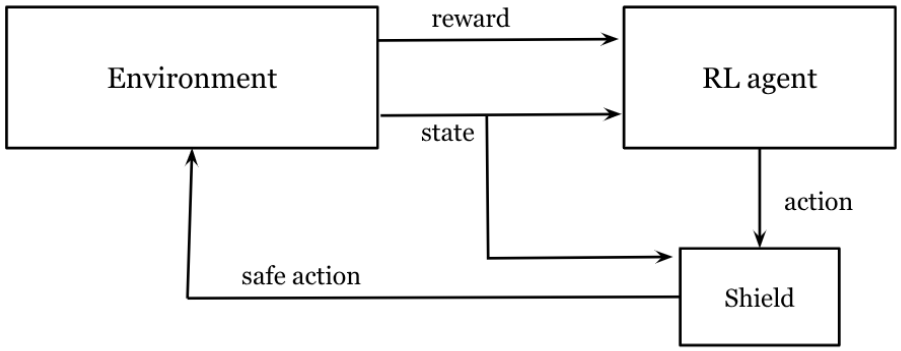

Fig. 2.1 Post-shielding intervention mechanism

The shield intercepts and blocks potentially unsafe actions before execution, ensuring system compliance with predefined safety constraints.

Pre-shielding: Instead of monitoring the agent's behavior, in this configuration the shield provides a filtered set of permissible actions, from which the agent can select. This approach ensures that all actions considered by the agent are safe by construction.

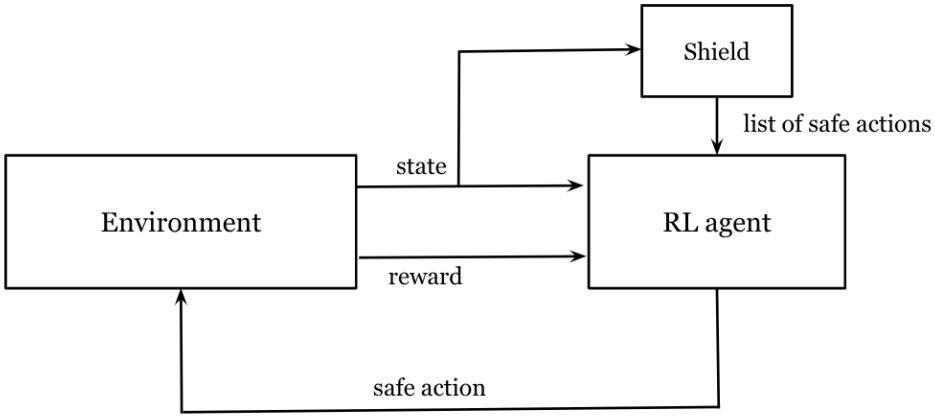

Fig. 2.2 Pre-shielding action space restriction.

The shielding mechanism evaluates and limits the agent's available actions before decision-making, ensuring only safe alternatives remain accessible.

Shielding provides two main types of safety guarantees: *Absolute* or *Probabilistic*. When employing deterministic models, the resulting shields manage safety as a qualitative and absolute measure. Specifically, any action that fails to satisfy the designated safety criteria is classified as unsafe; otherwise, it is classified as safe. This establishing a clear dichotomy between permissible and non-permissible actions. However, in real-world applications characterized by inevitable uncertainty and noise, Absolute guarantees frequently prove to be ineffective. Moreover, even when these guarantees are effective, they are often excessively rigid and restrictive. Probabilistic shields relax this assumption by assigning a degree of safety (typically a value ranging from 0 to 1) to each state-action pair. Actions, that in certain state, maintain the probability of safety above a specified threshold are deemed permissible. This threshold can be tuned, making the shield either more conservative or permissive depending on the application.

Shields are derived from specifications of safety properties combined with abstract models of the environment dynamics. Note, these specifications are typically independent of the agent's reward objective.

Construction techniques for shield specifications typically employ diverse approaches, depending on the domain and the type of safety guarantees require. These notably include: Formal Methods (in particular frameworks based on Temporal Logic), Probabilistic and Statistical models, and Control-based techniques.

The following example from paper [7], which can easily abstracted into a finite-state MDP, illustrates how Temporal Logic specifications can be effectively utilized in shield construction. The system consists of a water tank with limited capacity of 100 liters, equipped with a heater, that kept warm the water, and a controllable inflow valve. The water level in the tank evolves dynamically depending on the inflow and outflow rates:

- The outflow is bounded between 0 and 1 liters per second.
- The inflow, when the valve is open, ranges between 1 and 2 liters per second, and is 0 when the valve is closed.

The heater maintains the water at a desired temperature, and its energy consumption depends on the current filling level of the tank, although the precise functional dependence is unknown to the agent. The learning goal is to minimize energy consumption while respecting all operational constraints.

To prevent physical damage or unsafe behavior, two key safety constraints must always be satisfied:

- The tank must never overflow or run dry.
- To reduce valve wear, the valve position (open or closed) must remain unchanged for at least 3 seconds once switched.

These constraints can be formalized using Linear Temporal Logic (LTL). Let l_t denote the tank water level at time t, and $v_t \in \{\text{open}, \text{closed}\}$ denote the valve state. The safety property is encoded as an LTL formula ensuring bounded water level and restricted switching frequency:

$$\varphi_{\text{safety}} = G\big(0 < l_t < 100\big) \wedge G\Big(\big(v_t \neq v_{t-1}\big) \rightarrow XXX\,\big(v_t = v_{t-1}\big)\Big), \tag{2.50}$$

where:

- G is the globally temporal operator, enforcing safety at all times;
- X denotes the next temporal operator, enforcing that the valve must remain stable for 3 consecutive time steps after each switch.

Based on this LTL safety specification, a shield can be constructed to guarantee that any RL agent controlling the valve respects the safety property at all times. The shield monitors or filters the agent's proposed actions: if an action would lead to a safety violation (e.g., overflow or rapid valve toggling), the shield overrides it with a safe alternative. Otherwise, the action is executed unchanged.

To provide a more comprehensive overview, another clear example concerning the formulation of shield specifications, drawing inspiration from probabilistic theories, is presented from the referenced paper [35]. The setup is as follows:

- A mobile agent navigates within a discrete grid-world environment (size 10×6).
- Each of the four actions (up, down, left, right) is selected uniformly at random.
- Two distinct states are labeled as failure (terminal) states.
- The objective is for the agent to learn to navigate across the grid from one side to the other, by avoiding failure states or, alternatively, by selecting the path with the lowest probability of encountering such states, while minimizing the number of time steps required.

In this case, shield specifications are formulated based on a probabilistic measure, termed *Hazard Indicator*, *H(s)*, by the paper's authors. This indicator, in the current setup, quantifies, for each initial state s, the total probability of reaching either failure terminal within three steps under uniform random action selection. The corresponding heatmap of $H(s)$ is illustrated in Figure 2.3.

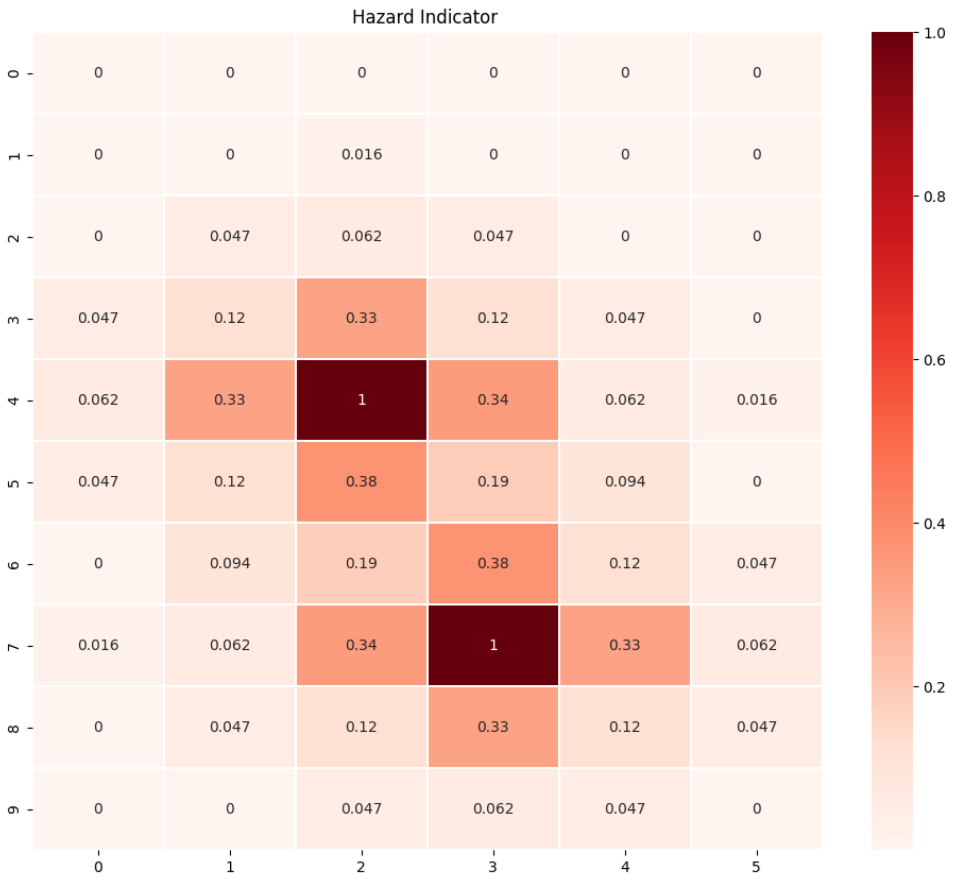

Fig. 2.3 Probabilistic Safety Map via Hazard Indicators [35].

Visualization of the Hazard Indicators across the state space. The index values, ranging from 0 (minimal risk) to 1 (the designated failure state), reflect the degree of risk associated with each state. Note, the distribution reveals a bell-shaped contour centered on the failure state.

During the RL training phase, the sampling of the actions was modulated according to the values of the Hazard Indicator. Specifically, in states where the indicator value exceeded a predefined safety threshold, the sampling strategy transitioned from unconstrained (or free) exploration to a guided-exploration mode. This guided mode actively steered the agent toward subsequent states exhibiting lower Hazard Indicator values. Note, the sensitivity of the shield (whether more restrictive or permissive) can be directly controlled by adjusting the value of this safety threshold.

Shielding offers several advantages over traditional constraint-based methods. It provides strong safety assurances, with empirical results showing zero (or near-zero) violation rates, and it is relatively straightforward to implement. Nevertheless, shielding has limitations:

Dependence on the model. The shield's correctness relies on the accuracy of the abstract model. If the model is incomplete or incorrect, safety may not be guaranteed.

Possible decrease in learning effectiveness. By filtering out unsafe actions, shields may reduce the agent's ability to explore the environment, potentially slowing down learning or leading to suboptimal policies.

Safety Specification Design. Defining appropriate safety specifications is inherently non-trivial, particularly for complex dynamical systems where safety is not an instantaneous property but depends on temporal evolution and sequential interactions. This challenge requires both domain-specific expertise to formalize relevant safety conditions and the ability to translate them into computationally tractable representations. Moreover, overly complex formulations may lead to excessive computational overhead during shield construction or execution.

Chapter 3

Safe Reinforcement Learning via Transformers Filtering

3.1 Problem Statement and Contributions

In the preceding section, we reviewed several dominant approaches to Safe Reinforcement Learning.

Constraint-based methods, require explicit specification of safety constraints, which are often inherently incomplete and, when formalization is possible, mathematically challenging to express. Moreover, these methods involve optimization procedures (e.g., min-max) that can introduce significant issues during training. Ultimately, a practical limitation of such approaches is that they cannot guarantee safe exploration in model-free setting. During training, while the agent optimizes expected return under constraints on some cumulative cost-risk function, it may visit unsafe states as part of its exploratory process. This is particularly problematic in domains where even a few dangerous actions can lead to catastrophic outcomes. While building high-fidelity simulators can help mitigate the risks of unsafe exploration, this approach merely shifts the problem: designing accurate simulators for complex, real-world environments is itself a huge challenge (often requiring extensive domain expertise and significant computational resources). In many cases, such as modeling all possible responses in a human interaction, it is impractical.

Consequently, constraint-based methods remain too risky for applications demanding direct deployment in the real world, where unsafe actions are unacceptable.

Currently, the most common Shielding-based approaches aim to prevent the agent from executing unsafe actions by either filtering (Preemptive Shielding configuration) or correcting (Post-posed Shielding configuration) its decisions based on the formulation of predefined safety specifications. Their implementation requires access to a well-defined abstraction of the environment dynamics, as well as a predefined set of safety conditions and logical rules that enable the identification of unsafe actions. This may require the identification and knowledge of numerous parameters and their interdependencies. Such assumptions limit the applicability of Shielding-based methods in scenarios where such information is unavailable, partially unknown, difficult to obtain in practice, complex to comprehend and formalize, or difficult to ascertain in terms of correctness.

These limitations of existing paradigms in Safe Reinforcement Learning motivate the development of alternative frameworks. In response, there is growing interest in approaches that can implicitly extract knowledge from large-scale datasets, a trend facilitated by the unprecedented volume of digital data generated in the current digital era, driven by the proliferation of sensors, Internet of Things devices, digital platforms, etc.

This dissertation introduces a novel framework within this research perspective: **Safe Reinforcement Learning via Transformer Filtering**. The central idea is to leverage the generalization capabilities of Transformer neural networks, trained exclusively on demonstrations of safe trajectories, to evaluate the safety of possible actions, and enable filtering during the training phase of RL agents. Unlike prior approaches that encode safety through manually specified constraints or rules/logic, the Safety Transformer learns a data-driven representation of safety.

Once trained, the Transformer model is integrated into the action-selection loop of any RL training algorithm[1]. By permitting only actions predicted to be safe, it constrains the agent's exploration to safe regions of the state space, thereby guiding the learning process without (or significantly reducing) exposing the system to risky behaviors. This structure permits the agent to have a direct (and safe) interaction with the environment during the reward-maximization phase of RL training, offering practical utility in real-world applications where simulators are unavailable.

[1]The framework is algorithm-agnostic, compatible with any RL algorithm

Ultimately, the structural decoupling of safety and performance optimization within the framework ensures that the Safety Transformer processes only safety-related information, remaining entirely independent from the optimization of reward signals. Experimental results presented in this dissertation confirm this separation, demonstrating that safety filtering does not degrade final performance.

At its core, this research is motivated by several fundamental challenges that persist in the current state of the art of Safe RL:

1. **Efficiency of constraint decoding and implementation.** How can safety knowledge be represented, decoded, and implemented within the learning architecture of an RL system? This question is particularly non-trivial in high-dimensional or complex domains, where safety cannot be easily formalized through explicit constraints or analytical models.

2. **Minimization of constraint violations in model-free learning.** How can a model-free RL process be structured to minimize, or ideally eliminate, constraint violations during policy training? As shown in dominant approaches presented in the previous section, maintaining adherence to safety constraints throughout training continues to be an open problem, and achieving *zero constraint violation* remains an elusive objective in Safe RL.

3. **Preserving performance optimality.** Does the incorporation of safety knowledge impose restrictions that hinder the RL algorithm's primary objective of maximizing cumulative reward? The imposed safety constraints restrict the agent's exploration of the environment's state-action space and may unintentionally exclude regions containing safe, yet undiscovered, optimal states. This limitation can consequently hinder the agent's ability to locate the globally optimal policy or synthesize novel, high-performing strategies. This point introduces the discussion of a crucial aspect: given RL's ability to generate solutions beyond human intuition, we must assess the degree to which human-defined safety constraints impose a ceiling on the agent's performance.

3.2 Transformers Overview

3.2.1 Sequence Modeling

Sequence Models are a class of Machine Learning models designed to process and analyze sequential data, such as texts, time series, audio signals, etc. Unlike traditional models that assume data samples are independently and identically distributed (i.i.d.), sequence models are built to capture and exploit the ordered dependencies that exist within sequential data. The motivation behind Sequence Models is the recognition that in many real-world applications, the current data point is influenced by previous data points. They have demonstrated success across a wide range of applications, including but not limited to: Natural Language Processing[2], Image Captioning, Audio and Speech Recognition, Time Series Forecasting, DNA Sequence Analysis. The field of sequence modeling has evolved over the past few decades, progressing through several key architectures:

- Recurrent Neural Networks (RNNs): Introduced in the 80s, RNNs represent one of the earliest neural architectures designed for sequence processing. Unlike standard Feed-Forward neural networks, RNNs feature a recurrent structure with a hidden state that acts as an internal memory to accumulate and utilize information from previous inputs of the sequence.
- Long Short-Term Memory (LSTM): Proposed at the end of the 90s LSTM networks address the vanishing gradient problem inherent in traditional RNNs, enabling the learning of long-range dependencies..
- Transformers: Introduced by Vaswani et al. in 2017 [89], the Transformer architecture revolutionized sequence modeling by entirely discarding recurrence in favor of self-attention mechanisms.

The transition from RNNs to LSTMs, and ultimately to Transformers, marks a significant advancement in the capability of models to handle increasingly complex sequence tasks. Today, Transformer-based models (and their variants), represent the state of the art in numerous sequence modeling domains.

[2]Specifically: Language Translation, Text Classification and Segmentation, Text Summarization, Text Generation, Automated Symbolic Reasoning.

3.2.2 Structure of the Transformer Models

The **Transformer** architecture is based on the *Encoder–Decoder* framework[3]. In this architecture, the Encoder maps an input sequence of symbol representations $(x_1, \ldots, x_n)$, also referred to as a sequence of *tokens*[4], into a sequence of continuous latent representations $(z_1, \ldots, z_n)$. Given this latent representation, the Decoder subsequently generates an output sequence $(y_1, \ldots, y_m)$ of symbols, one element at a time. Note, the model generates one token using previously generated tokens as additional input to predict the next one.

The remarkable success of the Transformer architecture is primarily attributed to the introduction of a *attention mechanism*, exactly the *self-attention*, which operates within both the Encoder and Decoder module. In simple terms, the attention mechanism enables each element within an input sequence to attend to all other elements, thereby allowing the model to capture global dependencies across the entire sequence. More formally, the attention mechanism operates by generating contextually informed representations for each input element of a sequence.

Looking at it more closely, in the original Transformer architecture [89], the attention mechanism is implemented via the *Scaled Dot-Product Attention* formulation, built upon three trainable models[5]: Query (Q), Key (K), and Value (V). More precisely, initially, three distinct vectors: the query vector, the key vector, and the value vector, are derived from each embedding vector[6] of the input sequence. Note that the key, query, and value vectors are obtained by passing each vector embedding through the three aforementioned models, which, in contemporary practice, are typically trained using Self-Supervised Learning [42] on a large dataset. Subsequently, the following formula is applied:

$$\text{Attention}(Q,K,V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{3.1}$$

where d_k denotes the dimension of the query and key vectors. The scaling factor $\sqrt{d_k}$ facilitates the stabilization of gradients during the training process.

[3] For a thorough discussion of the Encoder-Decoder architecture, refer to text [28].

[4] A token is the basic unit of data that an AI system processes.

[5] In essence, the operation performed is a linear transformation applied to the input data: $y = xW^\top + b$, where x is the input, W is the weight matrix, b is the bias vector, and y is the output.

[6] Vector embedding is a numerical representation of data that maps it into a space where elements with similar meanings or contexts are positioned closer.

A single attention process may be insufficient to capture the diverse relationships and dependencies present in complex data. To address this limitation, the Transformer employs *Multi-Head Attention* mechanism, which performs Scaled Dot-Product Attention multiple times, each with its own learned parameters. The various attention processes (also known as *heads*) operate independently (in parallel), and only their outputs are concatenated and linearly transformed to produce the final representation.

The Figure 3.1 presents a schematic overview of the original Transformer architecture. It is possible to observe two principal components: the Encoder and the Decoder. Both components are organized as stacks of identical layers, each comprising Multi-Head Attention mechanisms and Position-wise Feed-Forward neural networks, connected through residual connections followed by layer normalization.

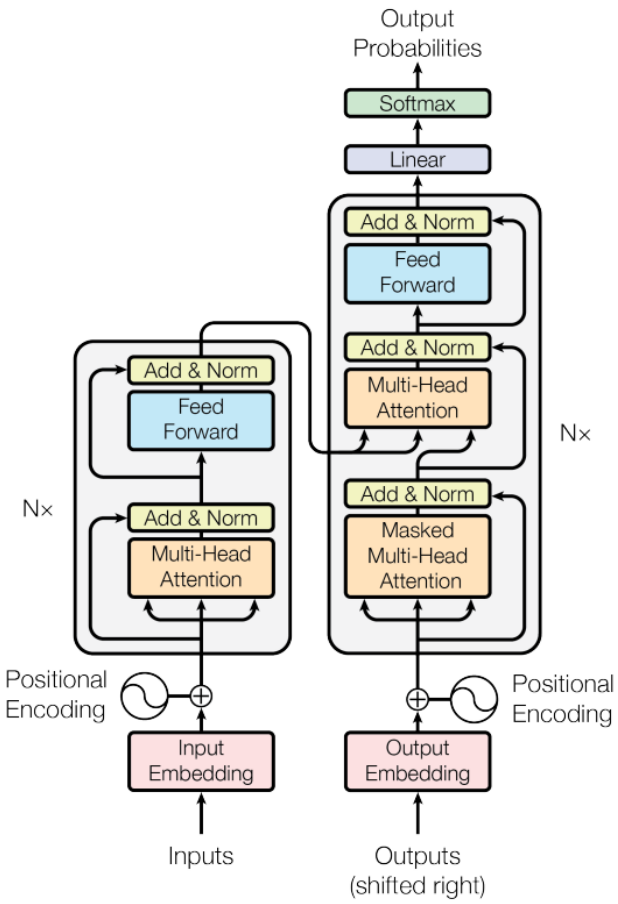

Fig. 3.1 Structural Components of the Transformer Model [89]

More specifically, the left portion of Figure 3.1 depicts the Encoder structure. The input tokens are first transformed into dense vector representations through an *Input Embedding* layer. A *Positional Encoding* is then added to these embeddings to inject information about the sequential order of the tokens, since the self-attention mechanism itself is permutation-invariant. Each encoder layer subsequently applies two primary sub-layers: a *Multi-Head Attention* mechanism (which enables each token to attend to all other tokens in the sequence) and a fully connected *Feed Forward network* (that lead to better Transformer performance). Residual connections and normalization follow each sub-layer to facilitate gradient propagation and stable optimization. The outputs from all encoder layers are passed to the decoder stack.

The right portion of the figure illustrates the Decoder module. It shares a similar structure to the encoder but includes an additional *Masked Multi-head Attention* sub-layer. This uses Scaled Dot-Product Attention and a mask for future positions, ensuring that the predictions for the token at position n depend exclusively upon the previously generated token outputs (preceding at n. I.e., depend only on the known outputs). Note, the decoder receives two inputs: the sequence of tokens produced in preceding steps (processed through output embeddings and positional encodings) and the encoded representations from the Encoder stack.

The final output is then passed through a trainable linear transformation and a Softmax layer to produce a probability distribution over the model's token vocabulary.

Since the introduction of the original Transformer architecture, numerous variants and extensions have been developed. The following survey [10] cataloged over seventy distinct Transformer-based models, highlighting the rapid growth of this research area. This development also underscores the versatility of Transformers, which have been successfully applied across diverse fields such as Natural Language Processing [69], Computer Vision [53], and Multimodal Learning [96].

3.2.3 Limitations of Transformer Models

Although the Transformer has established state-of-the-art performance, several open research challenges persist, both in terms of theoretical understanding and practical deployment of models. The primary issues are summarized below:

Computational Complexity. The Transformer architecture is characterized by the quadratic time and memory complexity of its self-attention mechanism: $O(n^2)$, where n represents the sequence length [89]. This entails high computational costs, and poses challenges for deployment on resource-constrained devices.

Training Instability. Transformers are sensitive to hyperparameters, particularly when dealing with long input sequences. Achieving stable convergence during training often requires significant effort in designing specialized optimizers and learning rate schedulers [58].

Limited Interpretability. Although extensive research has been conducted on attention visualization [100] and attribution techniques [3], the internal representations learned by Transformers remain only partially understood. Without clear insight, Transformer-based systems can behave as black boxes, hindering their adoption in sensitive or regulated applications.

Data Requirements. Transformer-based models typically demand very large datasets to achieve generalization. In settings where only limited data is available, these models may struggle to converge or may exhibit suboptimal performance.

Hallucination and Output Inconsistency. Transformer architectures may occasionally generate outputs that are inconsistent with the underlying training data, a phenomenon commonly referred to as *hallucination* [70]. Such behavior is particularly evident in large generative models, which may produce apparently credible, but factually incorrect or logically inconsistent outputs.

3.2.4 Transformers in Reinforcement Learning

Since a trajectory represents a temporal sequence of state–action transitions, RL can also be viewed as a sequence modeling problem, where the goal is to produce a sequence of actions that, when executed in an environment, yields a sequence of high rewards. From this perspective, it becomes conceptually appealing to investigate whether high-capacity sequence models, such as Transformers, can be successfully adapted to model RL trajectories.

Cutting-edge research [27] [47] led by the Berkeley Artificial Intelligence Research group has pioneered this line of inquiry. Their work treats trajectories as sequences of discretized tokens, consisting of states, actions, and rewards; and trains Transformer models autoregressively on these sequences to generate optimal future actions. In essence, the method involves tokenizing entire trajectories and employing Transformers to learn policies from datasets of pre-collected (offline) trajectories. De facto, this sequential modeling paradigm can be interpreted as a particular instantiation of Offline RL.

In the Offline RL setting [57], the agent does not actively interact with the environment to collect new experience. Instead, it learns exclusively from a fixed dataset of pre-collected trajectories, typically generated by arbitrary or suboptimal behavior policies. This approach is particularly promising in domains where online exploration is costly, risky, or impractical. However, Offline RL must address a significant challenge known as distributional shift: during deployment (i.e. when it will actually be used in the environment), the learned policy may encounter state–action pairs that lie outside the distribution represented in the dataset (i.e., not contained in the dataset). Formally, while the function approximator (i.e., policy or value function) is trained under one distribution, it is evaluated under another, due both to the change in the state visitation distribution of the new policy and to the optimization process itself [57]. Such mismatch leads to instability and degraded performance, if not properly addressed.

A notable example in this research direction is the Decision Transformer (DT) [27]. In this formulation, trajectories are expressed as:

$$\xi = (\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, \ldots, \hat{R}_T, s_T, a_T), \tag{3.2}$$

where $\hat{R}_t$ denotes the Return-to-go (the return accumulated from timestep t until the end of the trajectory). The tokens representing the states, actions, and returns-to-go are input into the Transformer (a GPT architecture [73]), which autoregressively predicts the optimal actions by employing a causal self-attention mask. The Figure 3.2 illustrates the schematic representation of the structure and flow of DT.

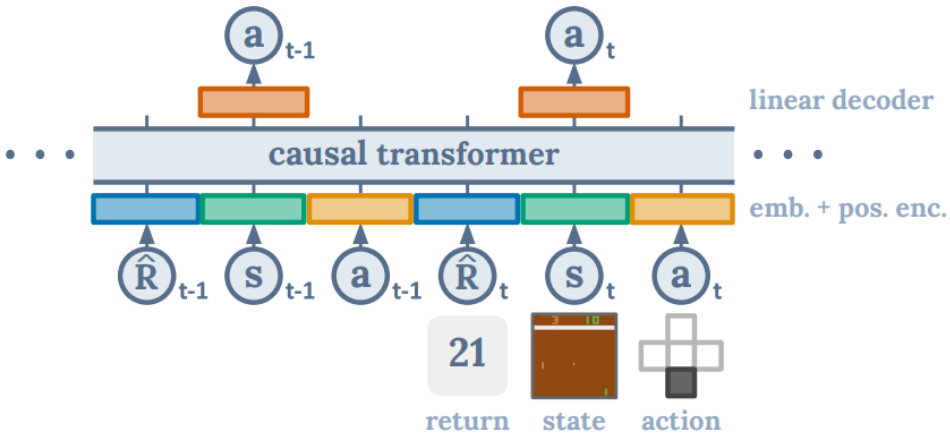

Fig. 3.2 Visualizing the Decision Transformer Architecture [73]

Note, DT can generate appropriate actions without requiring traditional Dynamic Programming mechanisms. In essence, this approach replaces key components of the classical RL pipeline (such as design the reward, the value function estimation, the exploration heuristics, etc) with sequence modeling.

Empirical results demonstrate that, the DT achieves performance that matches or exceeds state-of-the-art Offline RL baselines across various benchmark environments [27]. However, this approach inherits the typical limitations of Transformer models; additionally, predictions with Transformers are generally slower and more resource-intensive compared to conventional single-step policy networks commonly employed in RL. Consequently, the action selection process may be affected by latency periods. Such delays currently preclude the use of DT for real-time control in most dynamic systems.

Transformer-based architectures have also begun to be explored within the field of Safe RL. A interesting contribution in this direction is the Constrained Decision

Transformer (CDT), proposed in the work [63], which extends the original DT framework to address Constrained Markov Decision Processes in the offline setting. Specifically, CDT models trajectories while conditioning on a sequence of tokens representing the state, action, reward, and constraint cost. Its objective is to generate actions conditioned jointly on the reward return and the constraint cost return, thereby enabling the learning of policies that maximize cumulative rewards while ensuring that the expected constraint cost remains below a specified safety threshold. Unlike the original DT, which predicts deterministic action sequences, CDT adopts a stochastic Gaussian policy representation for its output.

While offering an intriguing intuition, the CDT framework inherits the typical limitations of Constraint-Based methods (e.g., how to encode the constraint cost token), Transformer-based architectures and Offline RL paradigms. Ultimately, as acknowledged by the authors, achieving zero-constraint violations remains an unresolved challenge.

3.3 Transformer Filtering Framework for Safe Reinforcement Learning

In this work, the term **Safety Transformer** will indicate a neural network architecture, derived from the Transformer model, designed to assess the safety of actions based on a given state-trajectory history. Its primary function is to output a probability distribution over the safety of available actions in the last state of a trajectory, thereby guiding an AI agent towards safe exploration. Note, modeling sequences of states as tokens for input into a Transformer, means assuming that the state space is either discrete, or has been discretized into a finite set of tokens.

The Safety Transformer model is trained exclusively with safe trajectories, defined as sequences composed entirely of safe states (or state-action pairs), generated by human expert demonstrations, rule-based criteria, or Artificial Intelligence techniques such as Generative Adversarial Networks [38]. It is important to note that these trajectories are required to be safe but not necessarily optimal with respect to performance maximization. In other words, their primary objective is to capture safety-relevant patterns rather than optimal behavior. The safety quality of these trajectories is critical to ensuring the reliable performance of the Transformer model.

Upon completion of the training phase, the model is used in inference mode for downstream integration within RL applications. In particular, the framework proposed in this study integrates the Safety Transformer model into the training pipeline of an arbitrary RL algorithm. More specifically:

1. **Input to Safety Transformer:** the Safety Transformer receives as input a trajectory history $H_t = (s_0, s_1, \ldots, s_t)$, representing the sequence of states observed up to time step t.

2. **Safety Prediction:** the Safety Transformer processes this history H_t and produces a probability distribution over the safety of all available actions that can be performed in last state of the trajectory (also referred to as the current state s_t). Formally:

$$P_{\text{safety}}(a \mid H_t) \quad \text{for all } a \in A_t \tag{3.3}$$

where A_t denotes all available actions at step t in s_t.

3. **Action Selector Module:** in this phase, a criterion is applied to filter the viable actions. For instance, only actions for which the predicted safety probability $P_{\text{safety}}(a \mid H_t)$ exceeds a predefined threshold τ are considered viable candidates for execution. Actions whose predicted safety probability falls below this threshold are discarded, as they are evaluated as unsafe by the Safety Transformer. Note, the value of this threshold is task-dependent.

4. **Reinforcement Learning Algorithm input:** once a set of safe candidate actions, denoted as a_{safe}, has been identified through the filtering process; it is provided as input to the RL algorithm's action selection phase. Note, this framework is designed to be agnostic to the specific RL algorithm employed and independent of the internal action selection criterion within that algorithm. Practically, the RL agent can safely sample any action from this viable candidate set. Once an action has been selected, it is executed within the environment, contributing to the agent's learning process as it seeks to optimize its policy for solving the given task.

5. **Trajectory Update:** after the chosen action is executed and a state transition occurs, the newly observed state s_{t+1} is appended to the trajectory history: $(s_0, s_1, \ldots, s_t, s_{t+1})$. This updated history is then fed back into the Safety Transformer for the subsequent step, creating a loop.

This process aims to ensure that the RL agent's training exploration occurs within safe operational boundaries, preventing the execution of unsafe actions. Simultaneously, it enables the RL agent to interact directly with the environment, thereby providing the opportunity to explore the state-action space and improve its policy toward optimality. The diagram below, Fig.3.3, provides a visual representation of the described process.

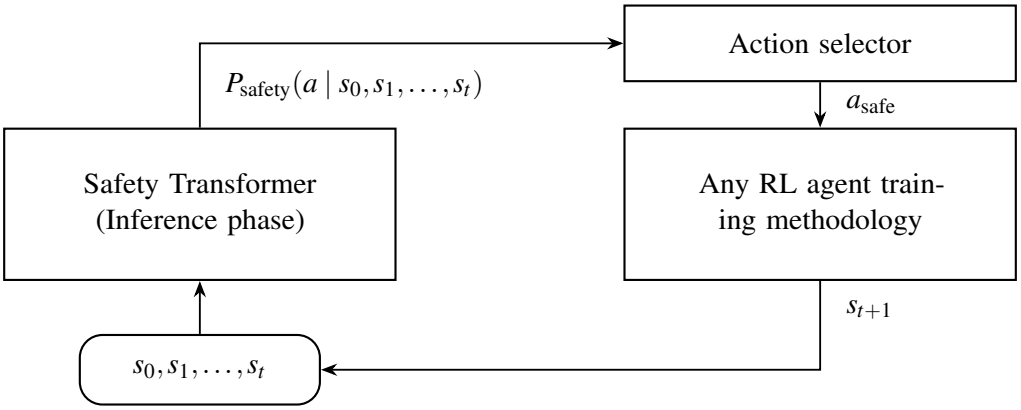

Fig. 3.3 Integration of the Safety Transformer in a Reinforcement Learning Training Loop.

To conclude this discussion, it can be useful to highlight some key conceptual and methodological distinctions between the proposed Transformer Filtering Framework and the paradigms of the Decision Transformer (DT) and Constrained Decision Transformer (CDT) introduced in Section 3.2.4.

First, the proposed Safety Transformer model performs tokenization exclusively over safe state representations or, in more intricate domains, over safe state-action pairs. In contrast to DT and CDT, this methodology does not utilize tokens representing generic states or actions, nor does it incorporate tokens corresponding to rewards and constraints[7]. Furthermore, the Safety Transformer focuses on learning the latent structure of safe behavioral patterns, without requiring any explicit definition, engineering design, formalization, or decoding of constraint functions (i.e., the constraint costs tokens).

Second, the reward optimization process remains entrusted to a conventional RL algorithm that interacts directly and online with the environment, rather than relying on pre-collected offline datasets. In doing so, it avoids the distributional shift and static-data limitations inherent to offline RL approaches, such as DT and CDT.

[7]This design choice reflects a deliberate decoupling between safety representation and reward optimization

Chapter 4

Experimental evaluation in deterministic environment: Navigation Task

To conduct a first assessment of the effectiveness of the proposed framework, a fully deterministic grid-based navigation task is introduced, featuring both dynamic and static failure modes. This environment is designed to highlight the capability of the Transformer-based filtering mechanism to identify safe behavioral patterns, allowing the agent to avoid failure states during RL training, while also demonstrating that it does not impede the agent's ability to discover and converge to the optimal policy.

The explicitly structured nature of the environment provides a controlled experimental setting, aiding in both rigorous evaluation and clear dissemination of the framework's principles.

4.1 Description of the Navigation Task Environment

The experiment evaluates the performance of a mobile agent on a navigation task within a two-dimensional, discrete 5x5 grid, where the task is complicated by the presence of various failure states. The agent starts from a fixed initial position and aims to reach a designated goal located on the opposite side of the grid in the minimum number of steps, while completely avoiding failure states, referred to as traps, throughout its trajectory. This defines the optimal policy sought.

The state space S is composed of all integer-valued coordinate pairs (x, y) within the grid boundaries. To introduce risk into the task, the environment contains multiple trap, which terminate the episode if entered. These are of two types:

- Fixed traps: Two static cells act as persistent hazards.
- Mobile trap: A single mobile trap cycles deterministically through a predefined sequence of positions: $[(0,2),(1,2),(2,2)]$. The trap transitions to the next position every 3 time steps. After reaching $(2,2)$, it restarts its position to $(0,2)$, repeating the cycle indefinitely. This pattern introduces temporal variation in failure states.

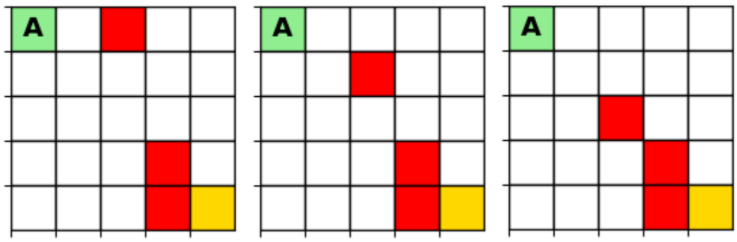

Fig. 4.1 Visualization of the environment dynamics in the 5×5 grid

Each subfigure represents a different timestep ($t = 0,3,6,9,12,...$) in the environment, illustrating the cyclic movement of the mobile trap. The green cell indicates the agent's fixed starting position at the top-left corner. Red cells represent trap states, which include both fixed and mobile traps. The gold cell denotes the goal state, located in the bottom-right corner. The mobile trap cycles through a predefined set of positions, changing location every 3 timesteps, and completing a full cycle before returning to its initial position.

The agent's action space A comprises 5 discrete actions: `North`: move one cell upward; `East`: move one cell to the right; `South` move one cell downward; `West`: move one cell to the left; `Pause`: remain in the current cell.

All state transitions are deterministic. The optimal policy implemented by the agent within this environment is also deterministic. In this environment, an episode terminates under one of the following conditions: the agent reaches the goal state, enters either a fixed or mobile trap, or or reach the maximum allowed trajectory length, set to 24 steps, a value deemed sufficient for the scope of this investigation.

It is important to note that the agent has no prior knowledge of the locations of trap states, nor any understanding of the logic governing the movement of mobile traps. The experiment will demonstrate the potentiality of the Safety Transformer, once adequately trained, to infer these patterns, enabling the agent to navigate the environment safely and effectively.

4.2 Safety Transformer for Navigation Task

The primary objective of the Safety Transformer is to produce (during its inference phase) a probability distribution over the safety of available actions, conditioned on a given input trajectory, i.e., a sequence of previously visited states. For example, consider an agent that has followed the trajectory

$$[(0,0),\ (0,0),\ (1,0),\ (2,0),\ (2,0),\ (3,0),\ (3,1),\ (3,1),\ (2,1),\ (1,1),\ (1,2)]$$

up to time step $t = 11$. When this sequence is provided as input to the Safety Transformer, the model is expected to output a safety distribution over the available actions at step $t+1$, such as:

$$\text{N} : 0.009, \quad \text{E} : 0.248, \quad \text{S} : 0.247, \quad \text{W} : 0.248, \quad \text{P} : 0.248.$$

In this illustrative case, the `North` action, which would produce a collision with the mobile trap at step 12, is assigned a significantly lower safety probability relative to the other actions, which are safe at all. This prediction reflects the model's capacity to infer safety-relevant patterns from trajectory histories and guide the agent toward safe decisions in the navigation task.

How does the model achieve this? During the training phase, the Safety Transformer is trained exclusively on sequences of safe trajectories, which are obtained

from expert demonstrations. Further details regarding the training data and methodology will be provided in the following section.

4.2.1 Architecture and Training

The model processes as input a variable-length sequence of tokens that represent the states trajectory, and produces in output a probability distribution over the safety of a fixed set of candidate actions.

From a technical perspective, the Safety Transformer model employs a standard Transformer Encoder architecture with 8 attention heads, and 4 layers. Input sequences are embedded through a learnable token embedding layer and augmented with learned positional encodings. A sequential module of a two layer linear network with ReLU activation (and dropout) projects the encode representations to a probability distribution over 5 actions via Softmax.

Training optimizes the Kullback-Leibler divergence between predicted and target safety distributions using AdamW ($\text{lr} = 10^{-4}$, weight decay 10^{-5}) with learning rate scheduling and gradient clipping.

4.2.2 Training Dataset Analysis

The Safety Transformer model is trained exclusively on a dataset composed of safe trajectories. In this context, a safe trajectory is defined as a sequence of states that originates from the initial starting position and evolves over a finite number of steps (25), without ever encountering a trap state, neither static nor dynamic. These trajectories are not required to reach the designated goal state; indeed, only 4 out of the 325 terminate at the goal, and none of them follow an optimal path in terms of task efficiency. The primary design criterion for these trajectories is safety, not optimality.

Model performance is highly dependent on both the quantity and quality of the training data. In this study, a training dataset of 325 safe trajectories was constructed, each with a length of 25 steps. Notably, all trajectories are distinct from one another. During dataset construction, the *expert* objective was to explore and include as many different safe states and transitions as possible to ensure maximal coverage of the safe state-action space. This condition is critical not only for ensuring safety but

also for enabling an effective subsequent RL training process; specifically, to ensure that the Safety Transformer model does not hinder the RL agent's ability to explore and converge toward the optimal policy. For instance, if a dataset systematically excludes state $(2,2)$ at every time step (not only during those in which it is unsafe, due to the presence of a mobile trap), the Safety Transformer model will consistently classify this state as unsafe all the time. As a result, it will be avoided throughout the RL training process, consequently impeding the exploration necessary for optimal policy convergence.

From a statistical perspective, the size of dataset remains extremely small relative to the combinatorial space of possible safe trajectories. To illustrate, assuming the environment with 22 accessible safe states, and an average of 4 safe actions per state, the number of potential trajectories of length 25 grows exponentially.

Offline RL Baseline. To gain an interesting insight about the training dataset, it is conducted a control experiment using a basic Offline RL setup [57]. The same Policy Gradient algorithm[1] used in performance testing in the Section 4.3, was applied solely to the data available in the dataset, without any further interaction with the environment. The conducted experiment yielded a suboptimal policy which was able to solve the environment in 17 steps. This outcome suggests that the dataset lacks sufficient information for the agent to learn the optimal policy, which, as showed in Figure 4.2, requires only 8 steps. Upon inspection, this limitation becomes evident. The optimal policy requires the agent to execute specific actions in specific state at precise time steps; for instance, achieving the optimal performance is necessary selecting the action `East`, in state $(2,3)$ at time step $t = 5$. If the dataset only includes the action `Pause` for that state and time, the agent is unable to infer an alternative. Moreover, in some instances, the dataset provides no information at all for certain states at particular time steps: meaning the agent is presented with an empty list of possible actions.

In conclusion, the training dataset's construction does not assume that *experts* possess knowledge of the optimal policy for environment resolution. Instead, it only requires them to demonstrate experience operating under safety conditions.

[1]Note: all hyperparameters, such as learning rate, number of training epochs, discount factor, etc.., were kept consistent with that configuration

4.2.3 Assessing the Robustness of the Safety Transformer

To assess the predictive effectiveness and generalization capability of the Safety Transformer model, a *stress test* was conducted. An evaluation script was developed to generate randomized safe trajectories under Transformer-guided. Trajectory lengths were uniformly sampled, ranging from 3 to 24 steps. The testing procedure involved generating trajectories by selecting the next action based on the Safety Transformer's predicted safety distribution. More specifically, given an input sequence of states, the model outputs a safety distribution across all available actions. Actions exceeding a predefined safety threshold were considered viable, and one of these viable actions was then uniformly randomly sampled for execution.

In all experiments conducted on this navigation task, the safety threshold τ was set to $\tau = 0.15$

For instance, consider the initial state with a trajectory consisting of only the starting position:

$$[(0,0)]$$

Remind that no traps are present in the immediate vicinity. In this case, the Transformer model may predict the following safety distribution:

$$\text{N} : 0.200, \quad \text{E} : 0.200, \quad \text{S} : 0.200, \quad \text{W} : 0.200, \quad \text{P} : 0.200$$

Since all predicted probabilities exceed the threshold τ, all five actions are considered viable, one is sampled uniformly from this set, and finally, performed. In contrast, consider a more complex trajectory approaching a hazardous region:

$$[(0,0),\ (1,0),\ (2,0),\ (2,1),\ (2,1),\ (2,1),\ (2,1),\ (3,1),\\ (2,1),\ (2,1),\ (2,2),\ (2,3),\ (2,3),\ (2,3),\ (2,3)].$$

In this scenario, a well-trained model is expected to produce a safety distribution similar to:

$$\text{N} : 0.329, \quad \text{E} : 0.325, \quad \text{S} : 0.009, \quad \text{W} : 0.010, \quad \text{P} : 0.328$$

thereby limiting viable actions to `North`, `East`, and `Pause`. Unsafe actions such as `South` and `West`, which may lead to a trap, are rejected.

Threshold Selection. The threshold value $\tau = 0.15$ was selected based on both theoretical considerations and empirical limitations. In an ideal setting, where the dataset is well-distributed and sufficiently comprehensive, one would expect safe actions to consistently be assigned probabilities of at least 0.2. This arises from the fact that for each state there are 5 possible actions. In the most favorable scenario, where all 5 actions are safe, they are each assigned an equal probability. This principle is mathematically represented as:

$$\tau = \frac{1}{|A_S|}, \qquad \text{where } A_S \text{ denotes the set of all possible safe actions} \tag{4.1}$$

However, given the practical limitations associated with finite and potentially imbalanced datasets, it is prudent to adopt a more conservative threshold to mitigate the risk of discarding valid safe actions. Conversely, setting the threshold too low (e.g., $\tau = 0.05$) increases the likelihood of admitting unsafe actions, thereby compromising safety. As evidenced by the experimental results, this choice proved to be both reasonable and effective in practice.

Evaluation Results. The Transformer model was evaluated using the setting describe above: the model generates a safety distribution over all available actions, among those actions that exceed the threshold value τ, one was uniformly randomly sampled for execution. A summary of the empirical results is presented below:

Table 4.1 Stress Test Results for Navigation Task

Metric	**Value**
Number of sampled trajectories	50,000
Total number of steps	798,051
Average trajectory length (steps)	15.96
Total number of trap encounters	**0**

The model avoided all fixed and dynamic traps throughout the stress test. These results indicate that the Safety Transformer successfully generalized the safety constraints encoded in the training data, maintaining robust predictive accuracy even under previously unseen trajectory patterns.

4.3 REINFORCE algorithm with Safety Transformer Integration

To evaluate the effectiveness of the Safety Transformer within a learning based decision-making process, it is implemented a RL agent trained using a basic version of the Policy Gradient methods[2].

The objective is to train the agent to learn an optimal policy for navigating the environment while adhering to safety constraints inferred from the Safety Transformer.

4.3.1 RL agent Architecture

The agent's policy is parameterized by a Feed-Forward neural network, which maps an input state vector to a set of 5 output logits, each corresponding to a possible action. The network architecture is defined as follows: the input state vector is composed of the agent's spatial coordinates and the current time step: (x, y, t). This is then processed by a fully connected hidden layer containing 128 units, followed by a ReLU activation function. The final output layer is also fully connected, producing 5 action logits (without Softmax activation). Essentially, this model serves as the policy π_θ.

[2]For more details on this methods, refer to Section 1.2.3.

4.3.2 Training Algorithm

The following pseudocode describes the training procedure.

Algorithm 4 REINFORCE Training with Safety Transformer Filtering

Require: Safety Transformer ST, Safety threshold τ, Policy π_θ, Maximum steps per episode $T_{\max}$, Learning rate α.

Initialize policy parameters θ
for episode $= 1$ to N **do**
  Initialize environment state
  Initialize episode buffer $\mathscr{D} \leftarrow \emptyset$
  for time step $t = 0$ to $T_{\max}$ **do**
    Query ST: $P_{\text{safety}}(a \mid H_t)$
    Filter actions: $A_t^{\text{safe}} \leftarrow \{a \in A \mid P_{\text{safety}}(a \mid H_t) > \tau\}$
    if $A_t^{\text{safe}} = \emptyset$ **then**
      break {No safe actions available}
    end if
    Sample action $a_t \sim \pi_\theta(A_t^{\text{safe}} \mid s_t)$
    Execute action a_t, observe $(s_{t+1}, r_t, \text{done})$
    Store transition tuple: $\mathscr{D} \leftarrow \mathscr{D} \cup \{(s_t, \log \pi_\theta(a_t \mid s_t), r_t)\}$
    Update trajectory history: $H_{t+1} \leftarrow H_t \cup \{s_{t+1}\}$
    if done **then**
      break
    end if
  end for
  For each step t in buffer $\mathscr{D}$, compute the Rewards-to-go: $R_t = \sum_{t'=t} r_{t'}$
  Compute policy gradient loss: $\mathscr{L}(\theta) = \sum_t \log \pi_\theta(a_t \mid s_t) \cdot R_t$
  Update policy parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta \mathscr{L}(\theta)$
end for

The provided algorithm illustrates the integration of the Safety Transformer into a Policy Gradient procedure. At each time step, prior to action selection by the RL agent, the Transformer is queried with the current historical state sequence H_t to predict a safety probability distribution over all available actions:

$$P_{\text{safety}}(a \mid H_t), \quad \forall a \in A.$$

This distribution enables a filtering process in which only actions whose predicted safety probability exceeds a predefined threshold, $\tau = 0.15$, are retained as viable candidates for the training. From this filtered subset, the RL agent samples one action according to its policy distribution, and performs it. At the end of each episode, the policy is updated using the REINFORCE algorithm [93].

General notes on implementation:

The RL training procedure is configured with a maximum episode duration of 24 time steps. While a lower value could have been permissible, a higher value would have invalidated the Safety Transformer's outputs, which was trained on sequences of up to 25 steps.

Regarding the condition : $A_t^{\text{safe}} = \emptyset$, specified in line 8. It should be noted that, in all experiments presented in this dissertation, no instances were observed in which the set of viable actions was empty[3]. However, in more complex environments, such edge cases may arise. In such situations, additional safeguards, potentially including human intervention, may be necessary to prevent the agent from entering hazardous states, rather than simply terminating the episode.

[3]This could be attributed also to the intrinsic dynamics of the environment

4.4 Results of Navigation Task Experiment

The training procedure successfully enabled the RL agent to converge to the **optimal policy**, as evidenced by the trajectory analysis presented in Figure 4.2. Throughout the training process, a total of 35,215 steps were executed across 2,000 episodes. Significantly, **no trap encounters**, either fixed or mobile, were recorded during training. It is pertinent to recall that a standard model-free RL algorithm, in the absence of any action-filtering mechanism, would have certainly encountered hazardous states repeatedly (potentially thousands of times) during its exploratory phase before learning to avoid them and converge to an optimal policy.

This result provides empirical evidence that the Safety Transformer was effective in enforcing safety conditions without compromising the agent's ability to explore the environment or converge on an optimal policy. In other words, safety was ensured while preserving performance.

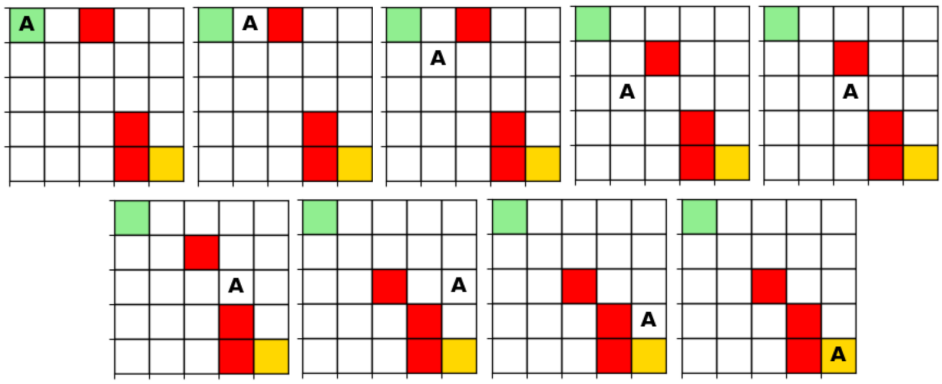

Fig. 4.2 Illustration of the Optimal Policy for solving the navigation task

The figure illustrates the agent's optimal policy for solving the grid-environment navigation task. Each subfigure displays the grid state at a given time-step: the green cell denotes the fixed starting position, the gold cell represents the goal state, and red cells indicate trap locations (both fixed and moving mobile trap). The character 'A' marks the agent's current position within each step. This visualization illustrates the agent's successful navigation from the starting point to the goal state, achieving the minimum possible number of steps while avoiding all hazardous states.

Overall, in the context of fully deterministic environments, the proposed framework demonstrates a high degree of reliability: both in terms of its protective filtering mechanism and its compatibility with the objective of performance maximization in RL algorithms.

Chapter 5

Experimental evaluation in stochastic environment: T1 Diabetes Management

In this experiment, the effectiveness of the proposed framework is evaluated in a highly stochastic setting: the UVA/PADOVA Type 1 Diabetes Simulator [65]. This environment was chosen to assess the capability of the Transformer filtering mechanism to identify safety-relevant behavioral patterns within a non-deterministic system. In this stochastic context, the objective is not the complete avoidance of failure states, which is inherently unattainable, but rather their significant reduction during RL training. Simultaneously, the experiment seeks to demonstrate that the framework does not hinder the agent's ability to discover and converge toward an optimal policy.

5.1 Description of the Type 1 Diabetes Simulator

Type 1 Diabetes (T1D) is a chronic autoimmune disease in which the pancreatic islets produce little or no insulin, an anabolic polypeptide hormone that plays a central role in regulating carbohydrate metabolism and maintaining blood glucose concentrations within physiologically safe levels. In the absence of sufficient endogenous insulin secretion, patients with T1D must rely on exogenous insulin administration, typically through multiple daily injections or continuous subcutaneous insulin infusion, to prevent metabolic complications such as hypoglycemia (low blood glucose levels). Determining the appropriate insulin dose is traditionally performed by physicians, who tailor therapy to the patient's individual characteristics and clinical history. This process, however, is challenging due to the nonlinear and stochastic dynamics of the glucose–insulin regulatory system.

In response to these challenges, research has led to the development of a closed-loop system for automatically managing blood glucose levels in people, known as the Artificial Pancreas System (APS), which integrates continuous glucose monitoring (CGM) sensors, subcutaneous insulin pumps, and a control algorithm to mimic the function of a healthy pancreas. The APS aims to approximate physiological glucose regulation by dynamically adjusting insulin delivery in response to sensor readings, thereby reducing the risk of hypo- and hyperglycemic excursions. A schematic overview of the APS is presented in Figure 5.1, illustrating its core components.

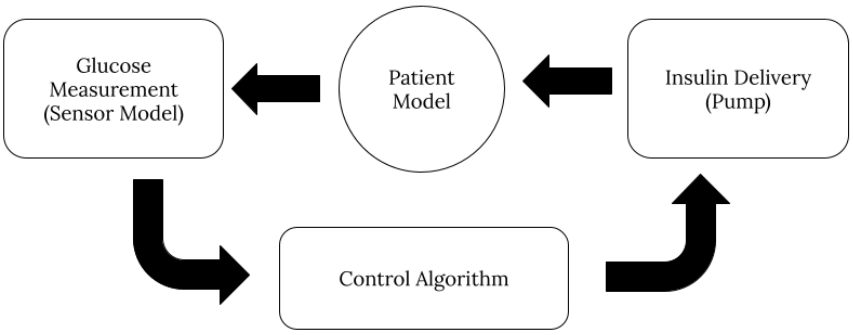

Fig. 5.1 Artificial Pancreas System: a closed-loop control scheme

In recent years, in silico testing environments have become indispensable tools for the development and evaluation of closed-loop insulin delivery strategies. Among these, the UVA/PADOVA Type 1 Diabetes Simulator [65], developed collaboratively

by the University of Virginia (UVA) and the University of Padova, has emerged as the de facto standard for preclinical validation. Importantly, this simulator has been accepted by the United States Food and Drug Administration, as a substitute to animal trials for the pre-clinical testing of insulin treatment strategies [31]. According to the literature [86], it remains the most widely adopted simulation platform in the field.

The simulator consists of four main components:

- Patient: a mathematical model of the glucose–insulin dynamics in T1D individuals.
- Sensor: a model of CGM devices, including physiological and technical measurement noise.
- Controller: a module that computes insulin dosages based on sensor readings and control algorithms.
- Pump: a model of discrete insulin delivery.

In this experiment, the open-source Python implementation of the UVA/PADOVA simulator was employed: `simglucose` [95], which has been successfully utilized in several prior investigations [86, 36]. The simulator is compatible with the OpenAI Gym interface, enabling seamless integration with RL algorithms. It supports various configurations, including 30 virtual patients: 10 adolescents, 10 adults, 10 children.

5.1.1 Environment Configuration

For the purposes of this study, it was employed a customized configuration of the `simglucose` environment. The main characteristics of the environment are summarized below.

Patient model. The selected virtual patient is `adolescent#002`, which corresponds to an in silico subject belonging to the adolescent group. The choice of `adolescent#002` was motivated by its characterization as the most behaviorally unstable profile within the simulator, exhibiting the highest degree of stochastic perturbations [95]. This variability renders glycemic control more challenging when compared to other models. Precisely for this reason, `adolescent#002` constitutes a

demanding benchmark for assessing the effectiveness of the proposed Transformer-based safety filtering framework, thereby providing a compelling evidence for demonstrating both the robustness and the generalizability of the methodology.

For each episode, the maximum simulation horizon was set to 361 steps, corresponding to approximately 18 hours of real-world glucose dynamics. Note that an episode may terminate prematurely if the value of observation, CGM, remains below 39 mg/dL for an extended sequence of steps, typically between 20 and 35 consecutive steps, corresponding to approximately more then one hour of real time. In more, a deterministic meal intake schedule is implemented to emulate realistic daily carbohydrate disturbances. The scenario consists of two meals administered at fixed times:$(7\,,\,0.2\,),(13\,,\,0.5\,)$ where each tuple denotes the meal time (hours) and corresponding carbohydrate amount. This predefined structure provides a controlled source of stochasticity in the glucose dynamics.

Observation space. At each time step, the agent receives as input the continuous glucose monitoring (CGM) value, expressed in mg/dL. For reasons related to computational lightening, the environment wrapper explicitly casts this measurement into integer form. In the present experimental setup, the observation space is bounded within the interval $[39, 360]$ mg/dL, reflecting the physiologically relevant range considered by the simulator. It should be noted that the initial CGM value for each episode are subject to stochastic oscillations.

Action space. The action space corresponds to discrete insulin infusion rates, ranging from a minimum basal rate of 0.000 U/min to a maximum of 0.070 U/min. The space is discretized into 15 equidistant levels: $\{0.000, 0.005, 0.010, \ldots, 0.070\}$, resulting in an action space cardinality of $|A| = 15$.

Safety constraints. Glycemic safety is defined with respect to CGM values. Specifically, the safe operational range is set to:

$$70 \text{ mg/dL } \leq \text{CGM} \leq \text{ } 180 \text{ mg/dL}.$$

Any state outside this interval is labeled unsafe, as values below 70 mg/dL correspond to hypoglycemia and values above 180 mg/dL indicate hyperglycemia, both clinically hazardous conditions [13].

The medical literature introduces the metric **Time in Range (TIR)**, which is defined as the percentage of time during which glucose values remain within the safe threshold of 70–180 mg/dL.

Goal. The objective is to train a RL agent capable of learning an optimal insulin dosing policy that maximizes the proportion of time spent within the clinically optimal glycemic range of 90–150 mg/dL [66]. In this regard, the metric **Time in Optimal Range (TIOR)** has been introduced in this work.

Reward Function. The reward function was designed in accordance with the objectives described above. At each time step, the agent receives a scalar reward r_t based on the observed CGM value:

$$r_t = \begin{cases} 0.8, & \text{if } 90 \leq \text{CGM}_t \leq 150 \\ 0.5, & \text{if } 70 \leq \text{CGM}_t < 90 \text{ or } 150 < \text{CGM}_t \leq 180 \\ -2.0 - 0.1 \cdot (70 - \text{CGM}_t), & \text{if } \text{CGM}_t < 70 \\ -1.0 - 0.05 \cdot (\text{CGM}_t - 180), & \text{if } \text{CGM}_t > 180 \end{cases}$$

Note, hypoglycemia (CGM $<$ 70 mg/dL) is penalized more severely than hyperglycemia. This is due to the greater risks associated with hypoglycemia in the chosen patient model. Furthermore, penalties are scaled progressively with the deviation from the safety thresholds, ensuring that larger departures from the safe range incur proportionally greater costs.

Overview. To provide an overview of the glycemic dynamics within the simulation environment, Figure 5.2 illustrates CGM trajectories over time under different insulin delivery strategies. The resulting glucose profiles show substantial variability, arising from both the underlying physiological model and stochastic measurement noise. This variability introduces significant challenges for robust policy learning.

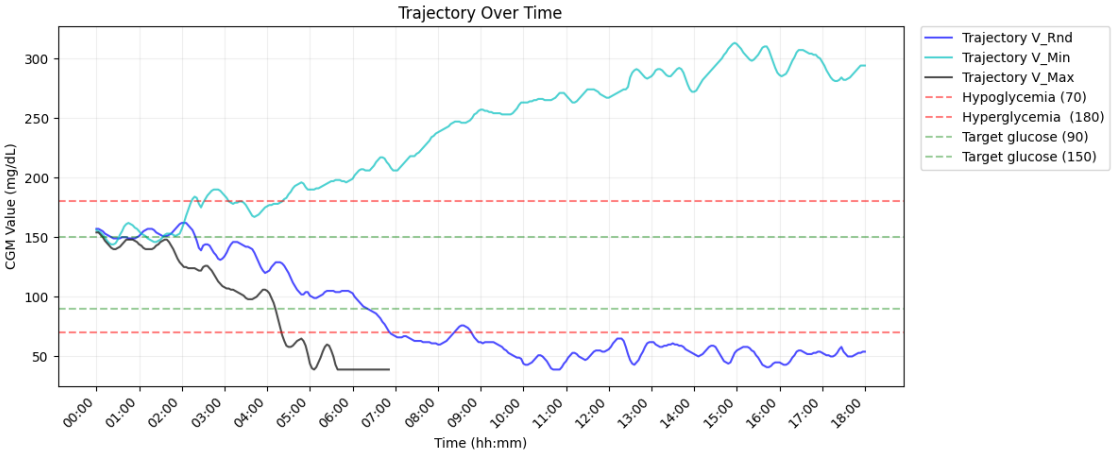

Fig. 5.2 Glucose trajectories under non-optimal insulin policies in the simglucose

The figure illustrates three CGM trajectories for the `adolescent#2` model, under different insulin control strategies: [blue] random insulin values sampled from the action space; [cyan] a constant insulin rate of 0.0; [black] a constant insulin rate at the maximum value of 0.070. Notably, in this last case, the episode was terminated prematurely because the CGM value remained below 39 mg/dL for more than 20 consecutive time steps. Dashed red lines denote hypoglycemia (< 70 mg/dL) and hyperglycemia (> 180 mg/dL) thresholds; dashed green lines indicate the target glucose range (90 - 150 mg/dL). All strategies, in the absence of optimal control, result in CGM values exceeding one emergency threshold, demonstrating the critical need for adaptive insulin management.

5.2 Safety Transformer for Blood Glucose Control

This Safety Transformer model is designed to operate as a safety filter for the available discrete insulin delivery actions, during its inference phase. Given a patient's historical trajectory: $(s_0, a_0, s_1, a_1, s_2, a_2, \ldots, a_{t-1}, s_t)$ at a given time step t, the model outputs a probability distribution over the 15 available insulin dosing actions. The distribution is structured such that higher probabilities are assigned to actions deemed safe, while actions predicted to lead to unsafe glycemic states are assigned probabilities close to zero. So, the ultimate objective of this Safety Transformer is to learn to guide action selection so that the glucose level remains within the safety range: (70 - 180 mg/dL).

For illustrative purposes, consider two scenarios: if all 15 actions are predicted to be safe, the model should produce a uniform distribution (approximately 6.67% per action). If only 4 out of 15 actions are classified as safe, the unsafe actions will receive probabilities close to zero, while the 4 viable actions will each be assigned probabilities close to 25%.

Note: unlike the deterministic navigation task, the stochastic nature of this environment necessitated training the model on sequences explicitly composed of state–action pairs, i.e., $(s_0, a_0, s_1, a_1, \ldots)$.

5.2.1 Architecture and Training

This model employs a Transformer Encoder with 8 attention heads and 6 layers. To enhance data representation, it processes state-action sequences through separate embedding pathways: states are embedded via a linear projection while actions utilize a learned embedding layer. Both are combined at the end and then augmented by learned positional encodings. The architecture uses GELU activation fuction. Finally, the encoder output is processed through a sequential module consisting of a two layer linear network with GELU activation function (and a dropout mechanism), yielding the action logits. These logits are converted into a probability distribution via the Softmax function.

Training employs AdamW optimization ($\text{lr} = 3 \times 10^{-4}$, weight decay 10^{-4}) with cross-entropy loss, cosine annealing learning rate scheduling, and gradient clipping.

Weights are initialized using Xavier normalization (the method is described in details [37]) for linear layers and small Gaussian noise for embedding layers.

5.2.2 Training Dataset Analysis

Training of the Safety Transformer model was conducted using a dataset comprised solely of safe trajectories. Each trajectory is represented as a sequence of alternating states and actions: $(s_0, a_0, s_1, a_1, s_2, a_2, \dots)$, where s_i denotes the observed state (in this case, the CGM measurement) and a_i denotes the corresponding insulin dosing action. A trajectory is classified as *safe* if and only if every state along the trajectory satisfies the condition : $70 \leq \text{CGM}_t \leq 180 \quad \forall t$.

The dataset contains 900 safe trajectories, each of fixed length 362 time steps. Each trajectory maintain safety for the entire horizon. During data generation, particular care is taken to ensure coverage and diversity across the safe region of the state space, by exploring a wide range of admissible CGM values. All trajectories contain in the dataset are mutually distinct.

Offline RL Baseline. As an additional experimentation step, analogous to the Navigation Task described in the previous section, a control experiment was conducted to gain further insight into the training datasets used, adopting a basic Offline RL setup. Specifically, the same PPO algorithm employed in the subsequent performance evaluation was trained exclusively on trajectories contained in the dataset, without any further interaction with the environment. The outcomes of this experiment are illustrated in Figure 5.3. It is observable that the Offline-PPO policy failed to achieve an optimal glycemic regulation. However, the resulting trajectories are better than those obtained by a completely random strategy (Figure 5.2). Quantitatively, the resulting glucose dynamics yielded 88.3% Time in Range (70–180 mg/dL) and 42.9% Time in Optimal Range (90–150 mg/dL).

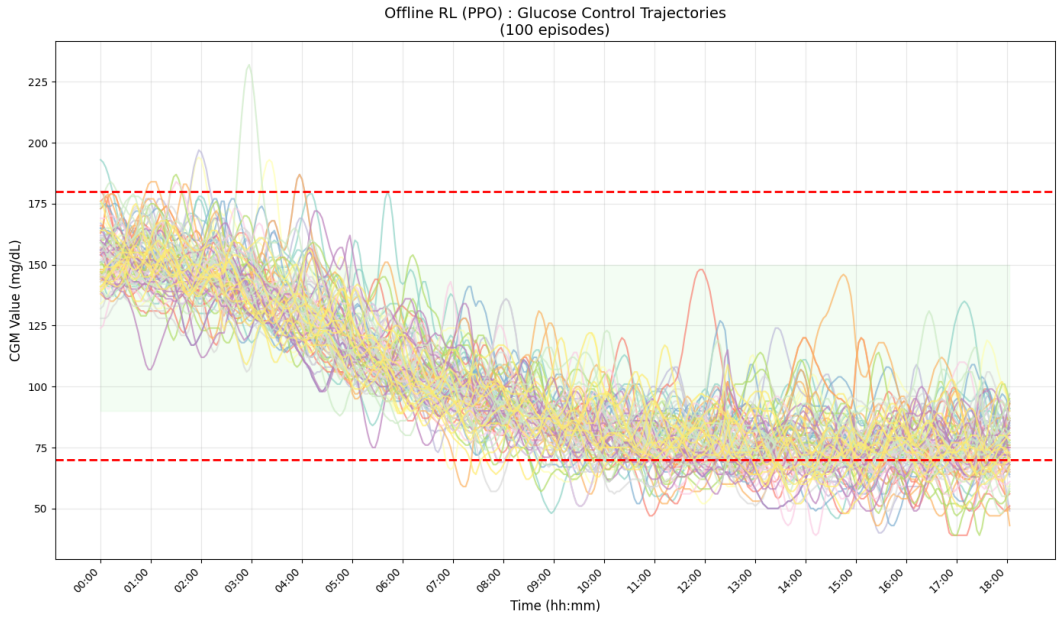

Fig. 5.3 Performance of the policy trained exclusively on the dataset via the Offline PPO algorithm

In Figure, 100 glucose trajectories generated by a PPO agent trained using Offline RL approach. Red dashed lines mark the hypoglycemia (70 mg/dL) and hyperglycemia (180 mg/dL) thresholds, while the shaded green region corresponds to the clinical target range (90–150 mg/dL).

In short, the training dataset's, taken in its naive form, does not contain sufficient information to derive an optimal policy.

5.2.3 Assessing the Robustness of the Safety Transformer

As in the previous experimentation, the predictive effectiveness and generalization capability of the Safety Transformer model were evaluated through a dedicated *stress test*. In this procedure, trajectories of fixed length 361 steps were generated by iteratively selecting actions according to the model's predicted safety distribution. At each time step, the Safety Transformer produced a probability distribution over all available actions. Only actions whose predicted safety probability exceeded a predefined threshold were deemed viable. From this set of viable actions, the subsequent action was then sampled uniformly randomly and executed, thereby constructing the trajectory under the Transformer's guidance.

Threshold Selection. The same considerations discussed in Section 4.2.3 motivated the selection of a safety threshold $\tau = 0.05$. In an ideal scenario, with a well-distributed and sufficiently comprehensive dataset, one would expect safe actions to consistently receive probability values of at least 0.066 (rounding down). However, given the practical constraints associated with finite and potentially imbalanced datasets, it is advisable to adopt a more conservative threshold.

Evaluation Results. Empirical results follows in table.

Table 5.1 Stress Test Results for Type 1 Diabetes Management

Metric	**Value**
Number of sampled trajectories	10,000
Total number of steps	3,610,000
Total number of unsafe steps	28,501
Time in Range (TIR)	**99.2%**

Despite the stochasticity of the simulator, the Safety Transformer model consistently guided the agent toward maintaining glucose levels within the defined safe range, as reflected in the high TIR metric. This provides empirical evidence of the model's effectiveness in filtering unsafe actions.

5.3 PPO algorithm with Safety Transformer Integration

To assess the effectiveness of the proposed framework in obtaining an optimal (or near-optimal) policy, ensuring safety during the training phase, the trained Safety Transformer model was integrated into the action-selection training loop of a RL algorithm. Especially, Proximal Policy Optimization (PPO) algorithm [78] was deemed particularly well-suited for the context.

5.3.1 RL agent Architecture

PPO agent, is parameterized as a neural network with distinct Actor and Critic heads. Its architecture is composed of three modules:

- Feature Extractor. A shared feedforward network projects the input state representation into a latent feature space. This extractor consists of two fully connected layers of dimension 128, each followed by a GELU activation and layer normalization.
- Actor Network (Policy Head). The actor maps the latent features into a vector of dimension 15, corresponding to the discrete insulin dosing actions. The output logits are normalized via a Softmax operation to form a categorical distribution over actions, from which the agent samples during interaction with the environment.
- Critic Network (Value Head). In parallel, the critic maps the latent features into a scalar value, estimating the expected return $V(s)$ of the current state, which is subsequently used for policy updates within the PPO algorithm.

Formally, given an input state s, the forward pass yields both the action logits and the value estimate:

$$f_\theta(s) \mapsto \big(\text{logits}(a|s),\, V(s)\big).$$

5.3.2 Training Algorithm

The training procedure is delineated in the pseudocode provided below.

Algorithm 5 PPO Training with Safety Transformer Filtering

Require: PPO Policy π_θ, Value function V_θ, Safety Transformer ST, Safety threshold τ, Maximum steps per episode $T_{\max}$, Discount factor γ, GAE parameter λ, Clipping parameter ε, Learning rate α.

Initialize policy parameters θ
for episode $= 1$ to N **do**
  Reset environment, obtain initial state s_0
  Initialize trajectory buffer $\mathscr{D} \leftarrow \emptyset$
  for time step $t = 0$ to $T_{\max}$ **do**
    Query ST : $P_{\text{safety}}(a \mid H_t)$
    Filter actions: $A_t^{\text{safe}} \leftarrow \{a \in A \mid P_{\text{safety}}(a \mid H_t) > \tau\}$
    if $A_t^{\text{safe}} = \emptyset$ **then**
      break {No safe actions available}
    end if
    Sample action $a_t \sim \pi_\theta(A_t^{\text{safe}} \mid s_t)$
    Execute action a_t, observe $(s_{t+1}, r_t, \text{done})$
    Store transition: $\mathscr{D} \leftarrow \mathscr{D} \cup \{(s_t, a_t, \log \pi_\theta(a_t \mid s_t), r_t, V_\theta(s_t)\}$
    Update history: $H_{t+1} \leftarrow H_t \cup \{s_{t+1}\}$
    if done **then**
      break
    end if
  end for
  Using the stored buffer $\mathscr{D}$, compute for each step t:
    Rewards-to-go: $R_t = \sum_{t'=t} r_{t'}$
    Advantage function via GAE : $A_t^{\pi_\theta} = \delta_t + \gamma\lambda A_{t+1}^{\pi_\theta}$
    where: $\delta_t = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$
  Actor loss: $\mathscr{L}_{actor} = \sum_t \min\left(\rho_t A_t^{\pi_\theta},\ \text{clip}(\rho_t, 1-\varepsilon, 1+\varepsilon) A_t^{\pi_\theta}\right)$
   where ρ_t is the importance ratio.
  Critic loss: $\mathscr{L}_{critic} = \sum_t (V_\theta(s_t) - R_t)^2$
  Compute PPO loss: $\mathscr{L}(\theta) = \mathscr{L}_{actor} - \mathscr{L}_{critic}$
  Update policy parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta \mathscr{L}(\theta)$
end for

The overall training pipeline remains conceptually identical to that presented in the preceding Section 4.3.2. The key modification lies in the adoption of the PPO algorithm.

To estimate the advantage function, the Generalized Advantage Estimation (GAE) [77] is applied. GAE has become the de facto standard in current policy gradient algorithms because it reduces variance in gradient estimates while maintaining a controlled level of bias. Note, the parameter λ provides a bias–variance trade-off.

The PPO objective function is composed of two terms :

- Actor loss (policy term), which constrains the policy update via a clipping operator, preventing large updates that could destabilize training.
  Note, the importance ratio, ρ_t, is based on the action log probabilities obtained from the current and its preceding actor networks. This is implemented by taking the exponent of the difference between these two log probabilities. In the case that no changes occur between the two networks, it follows that $\rho_t = 1$.
- Critic loss (value function term), which minimize the distance between the predicted state value and the empirical return.

5.4 Results of T1 Diabetes Management Experiment

Over the course of 1,083,000 training steps, the agent encountered a total of 10,152 unsafe glycemic states. To assess the consistency and added value of the Safety Transformer in enforcing safe behavior, a comparative experiment was conducted using the same PPO algorithm but without the Transformer filter. The comparison between these two methods is summarized in the table below.

Table 5.2 Unsafe Glycemic States during training with and without the Safety Transformer Filter.

Configuration	**Total Training Steps**	**Unsafe Glycemic States**
PPO + Safety Transformer	1,083,000	10,152
PPO (No Safety Transformer)	1,083,000	245,859

The results demonstrate the effectiveness of the Safety Transformer in significantly reducing exposure to unsafe glycemic states during training. While the number of training steps is equal in both configurations, incorporating the Safety Transformer **reduces the occurrence of unsafe states by nearly 96%**. These findings indicate that the Safety Transformer plays a crucial role in reducing the frequency of unsafe states during training.

Regarding the performance assessment of PPO model obtained from training[1], as presented in Figure 5.4. Despite some outliers, the PPO agent demonstrated strong overall performance. Quantitatively, on 100 trajectories, the model achieved a **TIR of 99.5%** for time steps fell within the safe range and a **TIOR of 84.9%** within the optimal clinical range. The mean CGM value across all trajectories was 134.3 mg/dL with a standard deviation of 15.4 mg/dL.

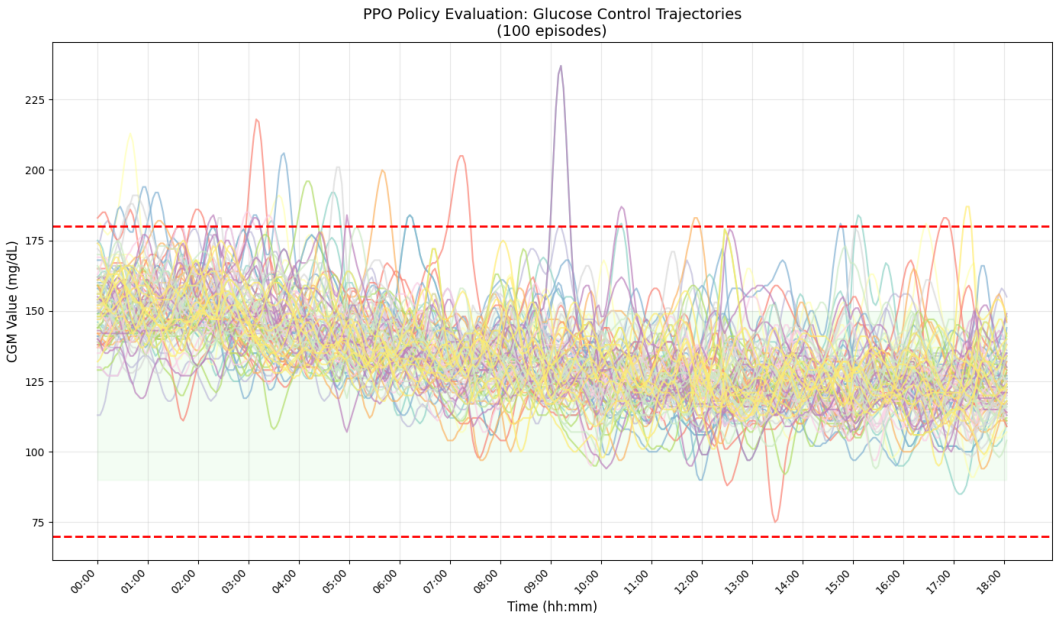

Fig. 5.4 Optimal PPO Policy Evaluation

Evaluation of a trained Proximal Policy Optimization, on 100 simulated episodes. Each trajectory represents a sequence of CGM generated by the PPO agent for the `adolescent#2` patient model. Red dashed lines denote the the safety range (70-180 mg/dL) thresholds; while the green shaded region indicates the clinically optimal glucose range (90–150 mg/dL).

[1]Note: the performance evaluation was conducted exclusively on the PPO model, without any action filtering by the Safety Transformer.

In conclusion, these results indicate that the proposed framework not only enabled the convergence of the PPO algorithm for performance optimization but also, within a highly stochastic environment, demonstrably reduced[2] the incidence of safety boundary violations during training.

[2]Note, within highly stochastic environments, the attainment of absolute safety guarantees is inherently impractical.

Chapter 6

Conclusions and Future Directions

6.1 Conclusions

This dissertation introduced and evaluated a novel framework for Safe RL based on the use of Transformer-based filtering. Unlike traditional approaches that rely on mathematical formulations of constraints or explicit safety specifications, the proposed method learns a data-driven representation of safety directly from demonstrations of safe trajectories. The Safety Transformer model, once trained, acts as a filter over the agent's action space, effectively guiding exploration during training.

This approach was empirically validated in two domains of increasing complexity: a deterministic navigation task, and the stochastic environment of glucose regulation in Type 1 Diabetes, modeled through the UVA/PADOVA simulator. The major findings of this work can be summarized as follows:

Safe exploration in deterministic settings. In the navigation task, the Transformer filtering mechanism completely eliminated unsafe state visits during training. This result highlights the reliability of the framework in structured environments.

Safe exploration in stochastic settings. In the T1 Diabetes management task, the integration of the Safety Transformer with the PPO algorithm led to a sharp reduction in unsafe glycemic states during training. These outcomes demonstrate the framework's effectiveness in mitigating risk in environments with inherent stochasticity and complexity.

Direct interaction with the environment. The safety-related results obtained in both deterministic and stochastic settings indicate that the proposed framework holds strong potential for direct application in real-world contexts, as it effectively prevents or, in stochastic scenarios, substantially mitigates the occurrence of unsafe events.

Safety without performance degradation. In both deterministic and stochastic environments, the Safety Transformer successfully prevented or substantially reduced unsafe actions during training, while preserving the RL agent's ability to converge to optimal policies. Notably, this implies that expert demonstrations of optimal or suboptimal policies are not required (it is sufficient to have a dataset of safe trajectories), since the task of deriving optimal policies is entirely delegated to the RL agent.

Generalizability of the approach. The Safety Transformer framework is algorithm-agnostic and can be integrated with any RL paradigm, demonstrating substantial methodological flexibility.

Overall, the results demonstrate that Transformer-based filtering represents a promising trend for advancing Safe RL methodologies. By inferring implicit safety constraints directly from data, it proves particularly suitable and effective in environments characterized by the abundance of digital data, a condition that has become increasingly common in contemporary settings. It also offers a viable alternative in domains where explicit formalization of safety constraints is infeasible or prohibitively complex. Moreover, the framework's capacity for direct interaction with the real environment mitigates the need for constructing complex and often unattainable high-fidelity simulators.

6.2 Limitations

Although the proposed framework in the validation experiments has shown promise, several limitations (mainly attributable to the inherent nature of Transformer models and their current lack of robust theoretical foundations[1]) deserve critical attention:

Dependence on datasets. The quality and diversity of the dataset of safe trajectories play a crucial role in the effectiveness of the Safety Transformer. Poor coverage of the state-action space or hidden biases in the dataset may lead to overly conservative or unsafe filtering.

Hallucinatory outputs in Transformer Although this issue was not observed in the present experiments, it has been reported in more sophisticated models with a very large number of parameters. Such inconsistencies, even if they happen rarely, can be problematic in safety critical environments, where erroneous predictions (i.e., erroneously classifying an unsafe action as safe) could dramatically misguide the filtering process. This issue is further compounded by the limited interpretability of Transformer models: adequate understanding of the prediction mechanism is essential for ensuring reliability in safety-critical domains. It is pertinent to conclude that ongoing research aimed at improving model interpretability and addressing such inconsistent outputs is highly active [84].

Computational cost. Transformer models are associated with significant computational demands, particularly during training. Although ongoing progress in hardware acceleration and more efficient architectures will mitigate this issue, computational overhead currently remains a practical challenge for deploying such models, especially when limited resources are available.

[1] Refer to Section 3.2.3.

6.3 Future Directions

The results of this study open several directions for future investigation:

Evaluation in high-dimensional safety-critical domains. Thus far, validation has been carried out in relatively low- to medium-dimensional environments. A crucial next step involves extending this evaluation to high-dimensional state-action spaces. Such testing will require substantial computational resources; however, it is essential to assess the scalability and practical viability of the framework when applied to real-world tasks with complex safety constraints.

Integration with multimodal models. The integration of Safety Transformers with multimodal models represents another promising research avenue. Multimodal learning [97], which simultaneously processes heterogeneous data streams such as text, images, audio, and other signals, could greatly enhance the capacity of the framework to capture safety-relevant features that are not accessible from a single modality. Such multimodal integration has the potential to further improve both the generalization and adaptability of Safe RL agents across domains.

Theoretical guarantees. While the empirical results demonstrate strong safety performance, the lack of formal guarantees remains a limitation. Future research should aim to establish rigorous theoretical foundations for Transformer-based safety filtering. For example, in this dissertation the selection of safety thresholds (Section 4.2.3 and Section 5.2.3) was performed by bridging theoretical and heuristic methods. Developing a mathematical–statistical theory for threshold calibration could significantly improve robustness, especially in highly stochastic environments. Another promising avenue is to study bounds[2] on the probability of entering unsafe states under Transformer filtering, thereby linking experimental observations with formal metrics.

[2]Upper or lower limit.

References

[1] Alphago. Documentary film, 2017. Directed by Greg Kohs. Available at https://www.alphagomovie.com/.

[2] Saminda Wishwajith Abeyruwan, Laura Graesser, David B. D'Ambrosio, Avi Singh, Anish Shankar, Alex Bewley, Deepali Jain, Krzysztof Marcin Choromanski, and Pannag R. Sanketi. i-Sim2Real: Reinforcement Learning of Robotic Policies in Tight Human-Robot Interaction Loops. In *Proceedings of The 6th Conference on Robot Learning (CoRL)*, volume 205 of *Proceedings of Machine Learning Research*, pages 212–224, 2023.

[3] Samira Abnar and Willem Zuidema. Quantifying Attention Flow in Transformers. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

[4] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained Policy Optimization. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 22–31. PMLR, 2017.

[5] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving Rubik's Cube with a Robot Hand. *arXiv preprint arXiv:1910.07113*, 2019.

[6] Miquel Noguer I Alonso. Reinforcement Learning: A Historical and Mathematical Overview (1950-2024). *SSRN Electronic Journal*, 2024.

[7] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe Reinforcement Learning via Shielding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018.

[8] Eitan Altman. *Constrained Markov Decision Processes*. CRC Press, 1999.

[9] Sanae Amani, Christos Thrampoulidis, and Lin Yang. Safe Reinforcement Learning with Linear Function Approximation. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2021.

[10] Xavier Amatriain, Ananth Sankar, Jie Bing, Praveen Kumar Bodigutla, Timothy J. Hazen, and Michaeel Kazi. Transformer models: an introduction and catalog. *arXiv preprint arXiv:2302.07730*, 2023.

[11] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete Problems in AI Safety. *arXiv preprint arXiv:1606.06565*, 2016.

[12] Yahui Bai, Yuhe Gao, Runzhe Wan, Sheng Zhang, and Rui Song. A Review of Reinforcement Learning in Financial Applications. *Annual Review of Statistics and Its Application*, 12:209–232, 2025.

[13] Tadej Battelino, Thomas Danne, Richard M. Bergenstal, Stephanie A. Amiel, Roy Beck, Torben Biester, Emanuele Bosi, Bruce A. Buckingham, William T. Cefalu, Kelly L. Close, Claudio Cobelli, Eyal Dassau, J. Hans DeVries, Kim C. Donaghue, Klemen Dovc, Francis J. Doyle III, Satish Garg, George Grunberger, Simon Heller, Lutz Heinemann, Irl B. Hirsch, Roman Hovorka, Weiping Jia, Olga Kordonouri, Boris Kovatchev, Aaron Kowalski, Lori Laffel, Brian Levine, Alexander Mayorov, Chantal Mathieu, Helen R. Murphy, Revital Nimri, Kirsten Nørgaard, Christopher G. Parkin, Eric Renard, David Rodbard, Banshi Saboo, Desmond Schatz, Keaton Stoner, Tatsuiko Urakami, Stuart A. Weinzimer, and Moshe Phillip. Clinical Targets for Continuous Glucose Monitoring Data Interpretation: Recommendations from the International Consensus on Time in Range. *Diabetes Care*, 42(8):1593–1603, 2019.

[14] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716–719, 1952.

[15] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d.O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv preprint arXiv:1912.06680*, 2019.

[16] Dimitri P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic press, 2014.

[17] Abhinav Bhatia, Pradeep Varakantham, and Akshat Kumar. Resource Constrained Deep Reinforcement Learning. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling*, volume 29, pages 610–620, 2019.

[18] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[19] Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2017.

[20] Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig. Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5, 2022.

[21] Van-Hai Bui, Srijita Das, Akhtar Hussain, Guilherme Vieira Hollweg, and Wencong Su. A Critical Review of Safe Reinforcement Learning Techniques in Smart Grid Applications. *arXiv preprint arXiv:2409.16256*, 2024.

[22] Cambridge University Press. Cambridge Dictionary Online. https://dictionary.cambridge.org/. Accessed 2025.

[23] Di Cao, Weihao Hu, Junbo Zhao, Guozhou Zhang, Bin Zhang, Zhou Liu, Zhe Chen, and Frede Blaabjerg. Reinforcement Learning and Its Applications in Modern Power and Energy Systems: A Review. *Journal of Modern Power Systems and Clean Energy*, 8(6), 2020.

[24] Steven Carr, Nils Jansen, Sebastian Junges, and Ufuk Topcu. Safe Reinforcement Learning via Shielding under Partial Observability. In *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 37, No. 12: AAAI-23*, 2023.

[25] Arthur Charpentier, Romuald Elie, and Carl Remlinger. Reinforcement Learning in Economics and Finance. *Computational Economics*, 62:425–462, 2023.

[26] Dong Chen, Mohammad Hajidavalloo, Zhaojian Li, Kaian Chen, Yongqiang Wang, Longsheng Jiang, and Yue Wang. Deep Multi-agent Reinforcement Learning for Highway On-Ramp Merging in Mixed Traffic. *IEEE Transactions on Intelligent Transportation Systems*, 24(11):11623–11638, 2023.

[27] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision Transformer: Reinforcement Learning via Sequence Modeling. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*, 2021.

[28] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[29] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-Constrained Reinforcement Learning with Percentile Risk Criteria. *Journal of Machine Learning Research*, 18:1–51, 2018.

[30] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A Lyapunov-based Approach to Safe Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 8092–8101, 2018.

[31] Claudio Cobelli and Boris Kovatchev. Developing the UVA/Padova Type 1 Diabetes Simulator: Modeling, Validation, Refinements, and Utility. *Journal of Diabetes Science and Technology*, 17(6), 2023.

[32] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe Exploration in Continuous Action Spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[33] Dongsheng Ding, Kaiqing Zhang, Tamer Başar, and Mihailo R. Jovanović. Natural Policy Gradient Primal-Dual Method for Constrained Markov Decision Processes. In *Advances in Neural Information Processing Systems*, volume 33, pages 8378–8390, 2020.

[34] Yan Duan, Xi Chen, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, volume 48, 2016.

[35] Mario Fiorino, Muddasar Naeem, Mario Ciampi, and Antonio Coronato. Defining a Metric-Driven Approach for Learning Hazardous Situations. *Technologies*, 2024.

[36] Ian Fox, Joyce Lee, Rodica Pop-Busui, and Jenna Wiens. Deep Reinforcement Learning for Closed-Loop Blood Glucose Control. In *Proceedings of the Machine Learning for Healthcare Conference*, volume 126 of *Proceedings of Machine Learning Research*, pages 1–28, 2020.

[37] Xavier Glorot and Yoshua Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, volume 9, pages 249–256, 2010.

[38] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27, 2014.

[39] Laura Graesser and Wah Loon Keng. *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. Addison-Wesley Professional, 2019.

[40] Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, and Alois Knoll. A Review of Safe Reinforcement Learning: Methods, Theories, and Applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):11216–11235, 2024.

[41] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, 2017.

[42] Jie Gui, Tuo Chen, Jing Zhang, Qiong Cao, Zhenan Sun, Hao Luo, and Dacheng Tao. A Survey on Self-Supervised Learning: Algorithms, Applications, and Future Trends. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9052–9071, 2024.

[43] Ben Hambly, Renyuan Xu, and Huining Yang. Recent Advances in Reinforcement Learning in Finance. *Mathematical Finance*, 2023.

[44] Tehreem Hasan, Farwa Batool, Mario Fiorino, Giancarlo Tretola, and Musarat Abbas. Efficient Maritime Healthcare Resource Allocation Using Reinforcement Learning. In *19th Conference on Computer Science and Intelligence Systems (FedCSIS)*, pages 615–620. IEEE, 2024.

[45] Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching Large Language Models to Reason with Reinforcement Learning. *arXiv preprint arXiv:2403.04642*, 2024.

[46] Rohan Inamdar, S. Kavin Sundarr, Deepen Khandelwal, Varun Dev Sahu, and Nitish Katal. A Comprehensive Review on Safe Reinforcement Learning for Autonomous Vehicle Control in Dynamic Environments. *Advances in Electrical Engineering, Electronics and Energy*, 10:100810, 2024.

[47] Michael Janner, Qiyang Li, and Sergey Levine. Offline Reinforcement Learning as One Big Sequence Modeling Problem. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*, 2021.

[48] Nils Jansen, Bettina Könighofer, Sebastian Junges, Alex Serban, and Roderick Bloem. Safe Reinforcement Learning Using Probabilistic Shields. In *Proceedings of the International Conference on Concurrency Theory (CONCUR)*, volume 171 of *LIPIcs*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[49] Pushkala Jayaraman, Jacob Desman, Moein Sabounchi, Girish N. Nadkarni, and Ankit Sakhuja. A Primer on Reinforcement Learning in Medicine for Clinicians. *npj Digital Medicine*, 7(337), 2024.

[50] Dai Josef, Pan Xuehai, Sun Ruiyang, Ji Jiaming, Xu Xinbo, Liu Mickel, Wang Yizhou, and Yang Yaodong. Safe RLHF: Safe Reinforcement Learning from Human Feedback. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

[51] Sham Kakade and John Langford. Approximately Optimal Approximate Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning*, pages 267–274, 2002.

[52] Hassan K. Khalil. *Nonlinear Systems*. Prentice-Hall, New Jersey, 1996.

[53] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Khan, and Mubarak Shah. Transformers in Vision: A Survey. *ACM Computing Surveys*, 54(10s):1–41, 2022.

[54] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement Learning in Robotics: A Survey. *International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[55] Bettina Könighofer, Mohammed Alshiekh, Roderick Bloem, Laura Humphrey, Robert Könighofer, Ufuk Topcu, and Chao Wang. Shield Synthesis. *Formal Methods in System Design*, 51:332–361, 2017.

[56] Malte Kuss and Carl E. Rasmussen. Gaussian Processes in Reinforcement Learning. In *Advances in Neural Information Processing Systems 16 (NIPS)*, 2003.

[57] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *arXiv:2005.01643*, 2020.

[58] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the Difficulty of Training Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5747–5763. Association for Computational Linguistics, 2020.

[59] Yongshuai Liu, Jiaxin Ding, and Xin Liu. A Constrained Reinforcement Learning Based Approach for Network Slicing. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–6. IEEE, 2020.

[60] Yongshuai Liu, Jiaxin Ding, and Xin Liu. IPO: Interior-Point Policy Optimization under Constraints. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)*, volume 34, pages 4940–4947, 2020.

[61] Yongshuai Liu, Jiaxin Ding, and Xin Liu. Resource Allocation Method for Network Slicing Using Constrained Reinforcement Learning. In *In The International Federation for Information Processing (IFIP) Networking*, 2021.

[62] Yongshuai Liu, Avishai Halev, and Xin Liu. Policy Learning with Constraints in Model-free Reinforcement Learning: A Survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4508–4515. IJCAI, 2021.

[63] Zuxin Liu, Zijian Guo, Yihang Yao, Zhepeng Cen, Wenhao Yu, Tingnan Zhang, and Ding Zhao. Constrained Decision Transformer for Offline Safe Reinforcement Learning. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 21611–21630. PMLR, 2023.

[64] Xiaobai Ma, Jiachen Li, Mykel J. Kochenderfer, David Isele, and Kikuo Fujimura. Reinforcement Learning for Autonomous Driving with Latent State Inference and Spatial-Temporal Relationships. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6064–6071, 2021.

[65] Chiara Dalla Man, Francesco Micheletto, Dayu Lv, Marc Breton, Boris Kovatchev, and Claudio Cobelli. The UVA/PADOVA Type 1 Diabetes Simulator: New Features. *Journal of Diabetes Science and Technology*, 8(1):26–34, 2014.

[66] MedlinePlus. Recommended Blood Sugar Targets. https://medlineplus.gov/ency/patientinstructions/000086.htm, 2024. U.S. National Library of Medicine.

[67] Daniel Melcer, Christopher Amato, and Stavros Tripakis. Shield Decentralization for Safe Multiagent Reinforcement Learning. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, 2022.

[68] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[69] Narendra Patwardhan, Stefano Marrone, and Carlo Sansone. Transformers in the Real World: A Survey on NLP Applications. *Information (MDPI)*, 14(4), 2023.

[70] Binghui Peng, Srini Narayanan, and Christos Papadimitriou. On Limitations of the Transformer Architecture. *arXiv preprint arXiv:2402.08164*, 2024.

[71] Shenglin Peng and Qianmei (May) Fen. Reinforcement learning with Gaussian processes for condition-based maintenance. *Computers & Industrial Engineering*, 158:107321, 2021.

[72] Theodore J. Perkins and Andrew G. Barto. Lyapunov Design for Safe Reinforcement Learning. *Journal of Machine Learning Research*, 3:803–832, 2002.

[73] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training, 2018. OpenAI technical report; available online.

[74] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[75] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[76] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[77] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[78] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347*, 2017.

[79] Guy Shani, David Heckerman, and Ronen I. Brafman. An MDP-Based Recommender System. *Journal of Machine Learning Research*, 6:1265–1295, 2005.

[80] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529:484–489, 2016.

[81] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[82] Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive Safety in Reinforcement Learning by PID Lagrangian Methods. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020.

[83] Yanan Sui, Alkis Gotovos, Joel Burdick, and Andreas Krause. Safe Exploration for Optimization with Gaussian Processes. In *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, volume 37, pages 997–1005. PMLR, 2015.

[84] Praneet Suresh, Jack Stanley, Sonia Joseph, Luca Scimeca, and Danilo Bzdok. From Noise to Narrative: Tracing the Origins of Hallucinations in Transformers. In *Advances in Neural Information Processing Systems (NeurIPS 2025)*, 2025.

[85] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018.

[86] Miguel Tejedor, Ashenafi Zebene Woldaregay, and Fred Godtliebsen. Reinforcement Learning Application in Diabetes Blood Glucose Control: A Systematic Review. *Artificial Intelligence in Medicine*, 104, 2020.

[87] John Tromp and Gunnar Farnebäck. Combinatorics of Go. In H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, editors, *Computers and Games. CG 2006*, volume 4630 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2007.

[88] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe Exploration in Finite Markov Decision Processes with Gaussian Processes. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NeurIPS 2016)*, pages 4312–4320, 2016.

[89] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[90] Akifumi Wachi, Yanan Sui, Yisong Yue, and Masahiro Ono. Safe Exploration and Optimization of Constrained MDPs Using Gaussian Processes. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018.

[91] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[92] Jenna Wiens, Suchi Saria, Mark Sendak, Marzyeh Ghassemi, Vincent X. Liu, Finale Doshi-Velez, Kenneth Jung, Katherine Heller, David Kale, Mohammed Saeed, Pilar N. Ossorio, Sonoo Thadaney-Israni, and Anna Goldenberg. Do no harm: a roadmap for responsible machine learning for health care. *Nature Medicine*, 25(9):1337–1340, 2019.

[93] Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992.

[94] Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory Douglas, Dion Whitehead, Peter Dürr, and Hiroaki Kitano. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602:223–228, 2022.

[95] Jinyu Xie. Simglucose v0.2.1. [Online], 2018. Available at https://github.com/jxx123/simglucose.

[96] Peng Xu, Xiatian Zhu, and David A. Clifton. Multimodal Learning with Transformers: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(10):12113–12132, 2023.

[97] Peng Xu, Xiatian Zhu, and David A. Clifton. Multimodal Learning With Transformers: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(10), 2023.

[98] Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J. Ramadge. Projection-Based Constrained Policy Optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[99] Hao Ye, Geoffrey Ye Li, and Biing-Hwang Fred Juang. Deep Reinforcement Learning Based Resource Allocation for V2V Communications. *IEEE Transactions on Vehicular Technology*, 68(4):3163–3173, 2019.

[100] Catherine Yeh, Yida Chen, Aoyu Wu, Cynthia Chen, Fernanda Viégas, and Martin Wattenberg. AttentionViz: A Global View of Transformer Attention. *IEEE Transactions on Visualization and Computer Graphics*, 30(1), 2024.

[101] Chao Yu, Jiming Liu, and Shamim Nemati. Reinforcement Learning in Healthcare: A Survey. *ACM Computing Surveys*, 55(1):1–36, 2023.

[102] Yuexiang Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Shengbang Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, and Sergey Levine. Fine-Tuning Large Vision-Language Models as Decision-Making Agents via Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS 2024) Main Conference Track*, 2024.

[103] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiwang Yang, Xiaobing Liu, Hui Liu, and Jiliang Tang. DEAR: Deep Reinforcement Learning for Online Advertising Impression in Recommender Systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 750–758, 2021.

[104] Ruijie Zhu, Lulu Li, Shuning Wu, Pei Lv, Yafei Li, and Mingliang Xu. Multi-agent broad reinforcement learning for intelligent traffic light control. *Information Sciences*, 619:509–525, 2023.

[105] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-Tuning Language Models from Human Preferences. *arXiv preprint arXiv:1909.08593*, 2020.

Appendix A

Publications

This appendix lists the publications produced during the author's three years of doctoral research.

- Mario Fiorino, Muddasar Naeem, Antonio Coronato. Reinforcement Learning in Healthcare. In *Handbook on Smart Health*. Studies in Health Technology and Informatics Book Series, ISBN 978-1-64368-607-3 (online), Sage/IOS Press, 2025. [*The book is expected to be available by December 2025*] https://www.iospress.com/catalog/books/handbook-on-smart-health

- Umamah Bint Khalid, Mario Fiorino, Madiha Haider S., Musarat Abbas. Evaluating Depression and Stress Among Young Adults Using DASS-21: Towards Personalized Intervention Strategies. *Position Papers of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS)*, ACSIS, Vol. 44, pages 49–54, 2025. http://dx.doi.org/10.15439/2025F7347

- Fiorino, M.; Naeem, M.; Ciampi, M.; Coronato, A. Defining a Metric-Driven Approach for Learning Hazardous Situations. *Technologies* 2024, 12, 103. https://doi.org/10.3390/technologies12070103

- Muddasar Naeem, Mario Fiorino, Pia Addabbo, Antonio Coronato. Integrating Artificial Intelligence Techniques in Cell Mechanics. *ACSIS*, Vol. 41, pages 111–116 (2024). DOI: http://dx.doi.org/10.15439/2024F4351

- Tehreem Hasan, Farwa Batool, Mario Fiorino, Giancarlo Tretola, Musarat Abbas. Efficient Maritime Healthcare Resource Allocation Using Reinforcement

Learning. *ACSIS*, Vol. 39, pages 615–620 (2024). DOI: http://dx.doi.org/10.15439/2024F8855

- Zahra Abbas, Mario Fiorino, Syed Muhammad Naqi, Musarat Abbas. COVID-19 Prediction Infrastructure Using Deep Learning. *Volume 32: Workshop Proceedings of the 19th International Conference on Intelligent Environments (IE2023)*, pages 125–134. https://ebooks.iospress.nl/doi/10.3233/AISE230020

- Ahsan Ismail, Mario Fiorino, Mussarat Abbas, Madiha Haider Syed, Zaib Ullah. Cloud-Based Monitoring System for Personalized Home Medication. *Volume 32: Workshop Proceedings of the 19th International Conference on Intelligent Environments (IE2023)*, pages 113–124. https://ebooks.iospress.nl/doi/10.3233/AISE230019