A Statistical Framework for Counterexample Driven Refinement of Reinforcement Learning Controllers

Manas Sashank Juvvi



Master of Science Thesis

TU Delft

Delft Center for Systems and Control

A Statistical Framework for Counterexample Driven Refinement of Reinforcement Learning Controllers

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Manas Sashank Juvvi

August 10, 2026

Faculty of Mechanical Engineering (ME) · Delft University of Technology





Copyright © Delft Center for Systems and Control (DCSC)
All rights reserved.

Delft University of Technology
Department of
Delft Center for Systems and Control (DCSC)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical Engineering (ME) for acceptance a thesis entitled

A Statistical Framework for Counterexample Driven Refinement of Reinforcement Learning Controllers

by

Manas Sashank Juvvi

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: August 10, 2026

Supervisor:

Dr. M. (Manuel) Mazo Espinosa

Committee:

Dr. L. (Luca) Laurenti

Dr. C. (Chris) Pek

Abstract

Reinforcement learning has demonstrated remarkable performance in complex robotic tasks by learning optimal policies through interaction with the environment. However, its deployment in safety-critical robotics applications remains limited due to the lack of safety guarantees, particularly when safety requirements are specified using formal specification languages such as Signal Temporal Logic (STL). Traditional formal methods can provide rigorous safety certificates, but their computational complexity often renders them impractical for high-dimensional systems or systems with unknown dynamics. To bridge this gap, statistical safety certificates have emerged as computationally tractable alternatives that provide Probably Approximately Correct (PAC) guarantees from finite data alone, without requiring a model of the system.

This thesis presents an iterative framework that synthesizes, certifies, and refines reinforcement learning controllers for robots subject to bounded-horizon STL specifications with black-box dynamics. The framework follows a train-certify-refine loop. A policy is first trained to maximize STL satisfaction. It is then certified using a finite number of rollouts drawn from the true, unknown distribution of starting configurations, yielding a distribution-free PAC guarantee on the specification violation rate. When certification fails, the rollout data is used to identify a region of the initial condition space where the policy performs worst. The policy is then retrained with emphasis on this failure region while preserving previously certified behavior through continual learning, and re-certified. This loop repeats until the PAC certificate is achieved or a maximum number of iterations is reached.

The framework is evaluated on a differential-drive robot navigating obstacle-filled arenas under multi-objective STL specifications that combine reach-avoid, stay, and safety requirements.

Table of Contents

Preface vii

Acknowledgements ix

1 Introduction 1
1.1 Related Work . . . 2
1.1.1 Model-Based Control Synthesis . . . 2
1.1.2 Reinforcement Learning for STL Tasks . . . 3
1.1.3 Formal Verification of Neural Network Controllers . . . 4
1.2 Approach and Contributions . . . 6

2 Background 9
2.1 Signal Temporal Logic . . . 9
2.1.1 Syntax . . . 9
2.1.2 Derived Operators . . . 10
2.1.3 Quantitative robustness semantics . . . 10
2.2 Reinforcement Learning and Policy Optimization . . . 11
2.2.1 Markov Decision Processes and Reinforcement Learning . . . 11
2.2.2 Neural Networks . . . 11
2.2.3 Proximal Policy Optimization . . . 12
2.3 PAC Guarantees and Binomial Confidence Bounds . . . 12
2.3.1 Probabilistic setup . . . 13
2.3.2 Application to policy certification . . . 13
2.3.3 Assumptions and limitations . . . 14
2.4 Continual Learning . . . 14
2.4.1 Continual Learning: Elastic Weight Consolidation . . . 14
2.4.2 Continual Learning: Progressive Neural Networks . . . 15

3 Methodology **17**
3.1 Problem Formulation . . . 17
3.1.1 System and Specification . . . 17
3.1.2 Policy and Closed-Loop Trajectory . . . 18
3.1.3 Formal Problem Statement . . . 19
3.2 Markov Decision Process Setup . . . 20
3.2.1 State Space and Observation Augmentation . . . 20
3.2.2 Action Space . . . 26
3.2.3 Transition Probability . . . 26
3.2.4 Reward Function . . . 26
3.2.5 Initial Augmented State, Horizon, and Discount Factor . . . 27
3.3 Policy Optimization . . . 28
3.3.1 Training Objective . . . 28
3.3.2 Advantage Estimation . . . 29
3.3.3 Network Architecture . . . 29
3.4 Statistical Validation and Counterexample Extraction . . . 30
3.4.1 PAC Certification . . . 31
3.4.2 Robustness Surrogate Landscape and Candidate Region Family . . . 31
3.4.3 Threshold Selection via a Failure-Concentration Score . . . 32
3.4.4 Counterexample Region Extraction . . . 33
3.4.5 Biased Resampling . . . 33
3.5 Continual Learning for Iterative Refinement . . . 34
3.5.1 Elastic Weight Consolidation . . . 34
3.5.2 Progressive Neural Networks . . . 34
3.5.3 Comparison and Trade-offs . . . 35
3.6 Complete Algorithm . . . 35
3.6.1 Algorithm Description . . . 35
3.6.2 Complexity Analysis . . . 36
3.6.3 Convergence and Failure Modes . . . 36

4 Implementation **39**
4.1 Simulation Platform . . . 39
4.2 Realising the MDP . . . 39
4.2.1 State Space Construction . . . 40
4.2.2 Reward and Robustness . . . 40
4.2.3 Initial State, Horizon, and Discount . . . 41
4.3 Policy Optimisation with rsl-rl . . . 41
4.3.1 Training Loop . . . 41
4.3.2 Practical Implementation Choices . . . 41
4.4 Certification and Counterexample-Extraction Pipeline . . . 42

4.4.1 Validation and Certification . . . 42
4.4.2 Surrogate Fitting and Threshold Selection . . . 43
4.4.3 Region Extraction and Spawn Injection . . . 43
4.5 Continual Learning . . . 44
4.5.1 Task Boundary Protocol . . . 44
4.5.2 Elastic Weight Consolidation . . . 45
4.5.3 Progressive Neural Networks . . . 45
4.6 Summary . . . 45

5 Experiments and Results **47**
5.1 Robot: TurtleBot3 Burger . . . 47
5.2 Setup 1: Reach-and-Stay with Obstacle Avoidance . . . 48
5.2.1 STL Specification and Environment . . . 48
5.2.2 State, Reward, and Termination Design . . . 49
5.2.3 Certification and Refinement . . . 49
5.3 Setup 2: Multi-Goal Navigation with Obstacle Avoidance . . . 54
5.3.1 STL Specification and Environment . . . 54
5.3.2 Certification and Refinement with EWC . . . 55
5.3.3 Certification and Refinement with Progressive Neural Networks . . . 56
5.3.4 Comparison of EWC and PNN on Setup 2 . . . 59
5.4 Discussion . . . 59

6 Conclusions and Future Work **63**
6.1 Concluding Remarks . . . 63
6.2 Future Work . . . 65

A Setup 3: Random-Goal Navigation **67**
A.1 Environment and Specification . . . 67
A.2 Results . . . 67

B Experimental Configuration **69**
B.1 Setup 1 . . . 69
B.2 Setup 2 . . . 71

Glossary **81**

Preface

Dear reader,

Before you lies my thesis, which concludes my journey as a master's student at TU Delft. Although it marks the end of my master's, this work is not the product of the past few months alone. Looking back, it feels like a collection of experiences, conversations, and moments of curiosity that gradually led me here.

My interest in robotics led me to pursue a bachelor's degree in Mechatronics, work on exciting projects, and later take up a research internship at IISc on advanced control techniques, which resulted in two conference publications.

Joining the master's programme in Systems and Control was a big step for me. It gave me the chance to explore control and the math behind it, in much greater depth. This Master's took incredible effort, and while there were frustrating moments along the way, those challenges made the experience all the more meaningful. It has been incredibly rewarding, and I hope the foundation built here continues to prove valuable in the years ahead.

This thesis builds on interests carried over from my earlier work and countless conversations with my supervisor, colleagues, friends and researchers at conferences. Endless hours of debugging, fine-tuning models, and working through mathematical details shaped this work. These moments were challenging, but they are what ultimately defined the journey.

Behind every chapter lies a journey of curiosity, perseverance, and growth that extends beyond the research itself. I hope you enjoy reading it, and that this work stirs a few questions in your mind and perhaps intrigues you to work on them.

Acknowledgements

First and foremost, my deepest gratitude goes to my parents and my brother for their unwavering love and support throughout this long journey. Your constant encouragement and reassuring presence kept me going through every stage of this Master's degree.

I am deeply grateful to my supervisor, Dr. Manuel Mazo, for his guidance and support throughout this work. Working under his mentorship taught me to think critically and approach problems with rigor. He gave me the space to explore new directions while providing the guidance and support I needed to stay on track. I also want to thank the SYNC lab members and everyone in the statistical techniques reading group for all the insightful discussions. I would also like to thank Dr. Luca Laurenti and Dr. Chris Pek for serving on my thesis committee and for taking the time to read and evaluate this work.

I'm grateful for my time with my former crew at the FOCAS Lab and for Dr. Pushpak's guidance during my time there, before beginning my master's. Working with that group deepened my interest in the field and laid the foundation for everything I've done since.

To the amazing friends I made along the way, thank you for keeping life fun and balanced. To keep this page shorter than the thesis itself, I'm referencing abbreviated group chat names, mostly to save my friends from the embarrassment of our inside jokes. Huge thanks to all my friends in DM, BATP, SG, KNSA, and FBG groups for the endless banter, late night gaming, and moral support.

A special mention goes to my pets, who did the job of keeping my parents entertained, and happy while I was away focusing on this Master's. Finally, a huge thank you to Gary.

Delft, University of Technology — Manas Sashank Juvvi
August 10, 2026

Chapter 1

Introduction

Autonomous robots are increasingly being deployed in environments where they are expected to complete complex tasks without human supervision. They are used to explore remote locations, assist in medical procedures, transport goods in warehouses, inspect infrastructure, and operate in environments that may be difficult or unsafe for people. In these applications, a robot is expected to do more than simply reach a destination or complete a task. It must do so safely, within operational constraints, and in a manner that can be guaranteed before deployment.

Consider a planetary rover collecting a rock sample. Reaching the sampling location is only one part of the task. The rover must arrive within the available operating window, remain in contact with the rock long enough for scientific measurements to be completed, avoid collisions with surrounding terrain during the maneuver, and safely continue with the rest of its mission. Since communication with Earth is intermittent and delayed, the rover must execute these decisions autonomously once the task commences. Any mistake may compromise valuable scientific data or reduce the lifetime of the robot.

A similar pattern appears in many terrestrial robots. A surgical robot must reach the correct position, maintain stable contact with tissue while a procedure is performed, and withdraw safely without damaging surrounding tissues. An autonomous mobile robot in a warehouse must collect and deliver items in the correct order, complete the task within operational deadlines, and always give way to nearby workers. In each case, task completion depends not only on what the robot does, but also on when it does it, how long it takes to do them, whether it remains safe throughout the entire task and other deadline requirements.

These requirements cannot be expressed as simple instantaneous constraints. Reaching a goal at a single moment does not guarantee that the robot arrived on time, stayed there for the required duration, or remained safe before and after reaching the goal. Instead, the desired behaviour is defined by the complete trajectory of the robot over the course of a task. The robot may need to visit several locations in a prescribed order, satisfy timing constraints, remain within safe operating regions, and avoid unsafe states throughout execution. Failure to satisfy any one of these conditions means the overall task has failed.

Designing controllers that satisfy such trajectory-level requirements remains challenging. Many robots operate in environments whose dynamics are difficult to model accurately because of unmodelled disturbances, or complex interactions with the environment. Even when the robot's own dynamics are well characterised, the environment introduces additional uncertainty through contact forces, terrain variation, or the presence of other agents, making the full system dynamics effectively unknown in practice. Furthermore, even when a dynamics model is available, directly optimising for trajectory-level specifications is computationally difficult as the resulting optimisation problem is non-convex and high-dimensional. Reinforcement learning controllers sidestep both difficulties by learning behaviours directly from interaction, without requiring an accurate mathematical model and without solving a high-dimensional optimisation problem explicitly. However, good performance during training does not by itself provide confidence that a learned policy will consistently satisfy safety requirements once deployed.

This thesis develops a framework for learning controllers that satisfy complex behavioural specifications in systems with unknown dynamics. The framework is guided by four main requirements. It should work without requiring an accurate model of the system dynamics. It should enforce requirements over the entire trajectory rather than at isolated time instants. It should provide a safety certificate before deployment so that the expected rate of specification violations is bounded. Finally, it should remain practical for deep neural network policies and robotic tasks with long planning horizons.

The following sections review existing approaches for learning, verification, and formal specification of robotic behaviour, and identify the gaps that motivate the framework developed in this thesis.

1.1 Related Work

The four requirements stated above structure the review that follows. Each subsection examines a class of methods for synthesizing controllers under temporal logic specifications, identifies which of the four requirements it satisfies, and characterizes the limitations that prevent it from addressing the remaining ones. The review covers model-based synthesis, reinforcement learning with Signal Temporal Logic (STL) and formal verification of neural network controllers. A comparison table states the research gap that motivates the framework developed in Chapter 3.

1.1.1 Model-Based Control Synthesis

In model-based STL (defined formally in Section 2.1) control synthesis, the specification is encoded as mixed-integer constraints and solved within a model predictive control (MPC) framework. Raman et al. [1] formulated STL-constrained MPC for discrete-time systems, encoding robustness bounds as mixed-integer linear programs (MILPs). Sadraddini and Belta [2] extended this to robust MPC for linear systems with additive bounded disturbances, providing worst-case guarantees. Raman et al. [3] embedded the MILP encoding in a counterexample-guided reactive synthesis loop, alternating between control synthesis and adversarial environment falsification within a receding-horizon scheme to enforce STL properties under nondeterministic disturbances.

Abstraction-based synthesis [4, 5] takes a different route by constructing a finite-state transition system from the continuous dynamics via simulation or bisimulation relations, so that synthesis reduces to a game on the abstraction. Tools such as TuLiP [6] provide automated pipelines from temporal logic specifications to controllers. The approach is sound, guarantees on the abstraction transfer to the original system, but the state-space discretization scales exponentially with the continuous state dimension.

These methods provide exact specification enforcement, but they share two critical limitations for the problem class considered in this thesis. First, the number of decision variables scales exponentially with the specification's nesting depth and horizon length, making solve times impractical for specifications with more than a few nested temporal operators. Second, the encoding requires the dynamics to be known in closed form.

1.1.2 Reinforcement Learning for STL Tasks

Since reinforcement learning does not require explicit dynamics models, a growing body of work uses it to synthesize controllers that satisfy STL specifications.

STL robustness as a reward signal. The most direct approach uses the quantitative robustness of the STL formula (defined formally in Section 2.1) as the RL reward. Aksaray et al. [7] introduced this direction by shaping Q-learning rewards with STL robustness for discrete-state systems, demonstrating that robustness-guided exploration outperforms sparse task-completion rewards. Varnai and Dimarogonas [8] systematically compared several STL robustness metrics and identified properties desirable for reward generation, including monotonicity and sensitivity to specification structure. These works demonstrate that STL robustness is an effective optimization surrogate, but the limitation is that the robustness value depends on the full trajectory prefix through the nested min and max operations of the temporal operators.

Funnel-based reward shaping. A related line of work replaces the raw robustness signal with a prescribed time-varying envelope. Saxena et al. [9] construct a funnel, a continuous lower bound on the robustness that is required to hold at every time instant, and design it so that a trajectory whose robustness remains inside the funnel satisfies the specification by construction. The learning problem is then reduced to keeping the robustness signal within the funnel, which yields a dense reward at each step. The construction is restricted to a fragment of STL for which such an envelope can be defined, and the satisfaction argument is conditional on the funnel remaining feasible for the unknown dynamics, a condition that is assumed rather than certified.

Constrained policy optimization. Rather than shaping the reward, an alternative treats STL satisfaction as a constraint. Ikemoto and Ushio [10] formulate deep RL under STL constraints using Lagrangian relaxation, converting the constrained problem into a min-max optimization over a dual variable. But like all Lagrangian approaches, it satisfies the constraint only asymptotically. No finite-sample or distribution-level guarantee on the resulting policy is provided.

Hierarchical decomposition and temporal grounding. A third direction decomposes the specification before learning. TGPO [11] flattens a nested STL formula into timed reachability subgoals and invariance constraints, grounds the free time variables by sampling, and trains a time-conditioned policy on the resulting dense, stage-wise reward. We adopt TGPO* as the primary learning-based baseline in Section 5.2.3. Its correctness claim, is empirical: the authors state that the method provides no formal guarantee, and satisfaction is reported as a success rate averaged over random seeds. In prior work by the author of this thesis, Juvvi et al. [12] take a complementary co-design route, training a task-independent library of RL policies for motion primitives, fitting reachability time estimators to their spatiotemporal behaviour, and composing them with a sampling-based planner. Feasibility follows from the primitive library rather than from a dynamics model, but satisfaction is argued constructively only for the returned trajectory, which is what motivates the policy-level certification developed here.

Shielding and barrier methods. Policy shielding [13] imposes a runtime safety filter that replaces unsafe actions with a pre-verified fallback. The shield provides deterministic safety during execution but requires either a pre-computed safe policy or a verified dynamics model. Control barrier functions (CBFs) [14] enforce forward invariance of a safe set via a pointwise constraint on the control input, making them well-suited as real-time safety filters for reinforcement learning policies. However, in their vanilla formulation, CBFs guarantee instantaneous constraint satisfaction but require a known (or assumed-bounded) dynamics model and cannot natively express temporal ordering, deadlines, or duration requirements.

1.1.3 Formal Verification of Neural Network Controllers

When a controller is learned, safety certification becomes a separate verification problem. Formal verification tools for cyber-physical systems include SMT solvers such as Z3 [15] for constraint checking. These tools were designed for low-dimensional systems with known, polynomial dynamics. For deep neural network controllers with unknown dynamics, they become computationally intractable. For feedforward ReLU networks, this is NP-hard because the verifier must enumerate a combinatorial number of activation patterns [16]. For non-ReLU activations, convex relaxation methods provide certified safe regions, but these regions shrink as the network deepens, and true safe regions can be under-estimated.

Counterexample-Guided Inductive Synthesis (CEGIS) [17] offers a structured paradigm for iterative refinement in which a candidate controller is verified against the full specification. If a counterexample is found, it is added to the constraint set and synthesis is repeated. CEGIS has been applied to neural Lyapunov functions [18], and safety shields for RL policies [19]. FOSSIL [18] integrates a neural Lyapunov learner with an SMT verifier, proposing to sample additional training points near counterexamples for fine-tuning; however, the approach requires ReLU activations and small architectures. Wang et al. [20] embed branch-and-bound verification directly into the training loop for neural control barrier functions, but report that verification time increases for systems beyond six state variables. Zhu et al. [19] extract a deterministic program from an RL policy and verify it via CEGIS to serve as a safety shield, but the extraction step discards representational capacity.

Counterexample-Guided Abstraction Refinement (CEGAR) [21] extends the counterexample paradigm by refining the model rather than the candidate solution, where verification is performed on a coarse abstraction and spurious counterexamples trigger refinement of the abstraction. Coppola et al. [22] use a data-driven verification framework based on SAlCA abstractions with distribution-free probabilistic guarantees for RL-controlled battery systems, though the abstraction's state space scales exponentially with discretization resolution and memory length.

The common bottleneck across all CEGIS-based approaches is the verification step. SMT and MILP solvers must reason over high-dimensional, piecewise-nonlinear functions composed with nonlinear dynamics over multi-step horizons. Published results are restricted to networks with at most a few thousands of parameters and short horizons, far below the scale of RL policies for robotics applications. Inspired specifically by the synthesize-then-verify loop of CEGIS, our approach adapts this iterative counterexample generation process while replacing full formal verification with statistical guarantees.

Statistical verification of learned policies. A line of work in literature replaces exact verification with statistical guarantees. Lindemann et al. [23] used conformal prediction for STL runtime verification, constructing prediction regions around future system states with distribution-free coverage guarantees and using these regions to bound the probability of specification violation during operation. Lindemann et al. [24] also presented a survey of formal verification and control with conformal prediction, covering safe planning and control synthesis under distribution shift. The remaining conformal prediction literature in control uses CP to quantify uncertainty within optimization-based control frameworks. Prediction regions obtained from data are incorporated into planning or model predictive control to account for uncertain disturbances, model errors, or the behavior of other agents. Recent work has also combined conformal prediction with Signal Temporal Logic by constructing prediction regions for uncontrollable agents and synthesizing controllers that satisfy STL specifications for all trajectories contained within these regions, thereby providing high-probability satisfaction guarantees. In these approaches, conformal prediction characterizes uncertainty while the controller is synthesized by solving an optimization problem subject to the resulting uncertainty sets. These works therefore differ from our approach, where statistical guarantees are placed directly on the learned policy through a PAC certification framework rather than through uncertainty-aware online planning.

Positioning and Research Gap. Table 1.1 summarizes the methods reviewed in Sections 1.1.1–1.1.3. A pattern is visible: methods that provide formal guarantees, whether deterministic (model-based synthesis, CEGIS) or pointwise (safety filters), all require a dynamics model or pre-computed abstraction, whereas the RL methods that operate without one offer at best asymptotic or expected-value claims that do not yield a deployable confidence bound on the trained policy's violation rate. Safety filters, whether logic-based shields [13] or CBF-based [14], enforce safety at each control step but do not certify trajectory-level STL satisfaction and the filter itself requires a model or safety abstraction. CP-STL runtime verification [23] provides distribution-free coverage for STL without a dynamics model, but it monitors an executing trajectory rather than certifying a policy before deployment. The gap is therefore at the intersection of the four columns: no existing method provides a finite-sample, distribution-free bound on the violation rate of a deep neural network policy trained

Table 1.1: Comparison of related approaches. Column 1: whether a dynamics model is required. Column 2: whether the method accepts STL specifications (✓), a related but distinct language such as LTL or pointwise invariance (∼), or no temporal logic (✗). Column 3: ✓ marks a guarantee that holds by construction or with quantified confidence, ∼ a claim that holds only asymptotically, or subject to approximation, and ✗ a purely empirical success rate. Column 4: whether the method has been demonstrated on deep, high-dimensional policies.

Method	*Unknown dynamics*	*STL specification*	*Formal / statistical guarantee*	*Scales to deep policies*
Model-based synthesis				
MILP-STL synthesis [1]	✗	✓	✓	✗
Abstraction-based [4]	✗	✓	✓	✗
RL with STL specifications				
Ikemoto and Ushio [10]	✓	✓	∼	✓
Funnel-based STL RL [9]	✓	✓	∼	✓
F-MDP [7]	✓	✓	∼	✗
TGPO / TGPO* [11]	✓	✓	∼	✓
Primitive co-design [12]	✓	∼	∼	∼
Shielding and barrier methods				
Safety filters [13, 14]	∼	∼	✓	✓
Verification of learned controllers				
CEGIS + SMT [18]	✗	∼	✓	✗
CP-STL runtime verification [23]	∼	✓	✓	✓
This work	✓	✓	✓	✓

under STL specifications without the dynamics. This work fills that gap through a PAC certification framework applied to the trained policy, combined with an iterative refinement loop that reduces the violation rate when the certificate is not met.

1.2 Approach and Contributions

This thesis proposes a three-stage iterative framework: *synthesize, validate, refine*. The synthesis stage trains a policy via Proximal Policy Optimization against a reward derived from the quantitative robustness of the specification, using timer signals to resolve the non-Markovian structure without recurrent architectures. The validation stage certifies the trained policy with a distribution-free bound on its violation rate, computed from independent rollouts alone, and extracts geometric counterexample regions from the detected failures. The refinement stage retrains the policy on those regions under continual-learning constraints that prevent regression on previously certified behavior, after which the loop returns to validation. The framework draws on formal methods, reinforcement learning, statistics, and continual learning. The key insight connecting these fields is that what formal methods call a counterexample is the same object that statistical testing produces as a failure sample, and this

object can drive targeted policy refinement provided forgetting is controlled.

The specific contributions are as follows.

1. **A state/observation-augmented Markov decision process formulation that converts finite-horizon behavioral specifications into a standard Markov setup, enabling feedforward policies to condition on temporal clause progress without recurrent architectures.** The augmentation extends the underlying state with always timers and eventually timers for each temporal sub-clause, resolving the non-Markovian problem identified at a constant per-step computational cost. The formulation is developed in Section 3.2.

2. **A statistical validation and counterexample extraction pipeline that certifies the trained policy with a distribution-free PAC bound on the violation rate and identifies the initial-condition regions responsible for failure.** Certification requires only independent rollouts from the deployment distribution and no access to the dynamics model or the network internals, overcoming the scalability bottleneck of exact verification methods reviewed in Section 1.1.3. The geometric counterexample regions produced by the pipeline serve directly as retraining targets for the refinement stage. The pipeline is developed in Section 3.4.

3. **An iterative counterexample-guided refinement loop that reduces the certified violation rate on identified failure regions without degrading performance on previously certified regions.** Two forgetting-prevention approaches are evaluated. Elastic Weight Consolidation, which penalizes parameter drift from previously learned behavior, and Progressive Neural Networks, which prevents forgetting by freezing prior network columns. The refinement loop is developed in Section 3.5 and its properties are validated in Chapter 5.

Chapter 2 introduces the mathematical prerequisites, namely Signal Temporal Logic and its robustness semantics, Proximal Policy Optimization, distribution-free PAC bounds, and continual-learning techniques, each motivated by the role it plays in later chapters. Chapter 3, the core of the thesis, formalizes the problem, develops the timer-augmented training scheme, describes the statistical validation and counterexample extraction pipeline, and presents the continual-learning refinement loop, closing with the complete algorithm and its complexity analysis. Chapter 4 presents the implementation details of the algorithm introduced in Chapter 3, motivating the key engineering decisions and discussing their impact on computational performance. Chapter 5 reports confidence intervals, robustness plots, and failure-case analysis.

Chapter 2

Background

This chapter introduces the mathematical tools that Chapter 3 requires: specification languages, reinforcement learning, neural networks, statistical verification, and continual learning. Each section opens with an explanation of why the tool is needed, then gives the formal definition, and a note on how the tool is used in the methodology.

2.1 Signal Temporal Logic

The reward signal, the timer-based observation augmentation, and the validation criterion in Chapter 3 all require a specification language that is interpreted directly over real-valued discrete-time signals, can express time bounds, and provides a continuous quantitative measure of specification satisfaction.. Signal Temporal Logic (STL) [25] provides all three, and is therefore the specification language used throughout this thesis. This thesis is restricted to the *bounded* fragment of STL, where every temporal operator has a finite upper bound $t_2 < \infty$. This restriction guarantees that every formula has a computable finite horizon for the episodic validation procedure in Section 3.4.

2.1.1 Syntax

Let $\xi : \mathbb{N} \to \mathcal{X} \subseteq \mathbb{R}^n$ be a discrete-time signal, and write $x_t \coloneqq \xi(t) \in \mathcal{X}$ for its value at time step t. A signal observed over a finite horizon H is written $\xi = x_{0:H} = (x_0, x_1, \ldots, x_H)$, and this trajectory notation is used for the closed-loop rollouts. Let ϕ be a predicate defined by a function $h : \mathcal{X} \to \mathbb{R}$ such that ϕ holds at time t if $h(x_t) \geq 0$ and does not hold otherwise. The bounded STL syntax is

$$\varphi ::= \text{true} \mid \phi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_{[t_1,\, t_2]}\varphi \mid \Box_{[t_1,\, t_2]}\varphi, \tag{2.1}$$

where $[t_1, t_2] \subseteq \mathbb{N}$ with $0 \leq t_1 \leq t_2 < \infty$. The symbols $\neg$, $\vee$, and $\wedge$ denote Boolean negation, disjunction, and conjunction. The temporal operator $\Diamond_{[t_1,t_2]}\varphi$ (*eventually*: φ holds at some time in $[t_1, t_2]$) and $\Box_{[t_1,t_2]}\varphi$ (*always*: φ holds at every time in $[t_1, t_2]$) both require $t_2 < \infty$, distinguishing bounded STL from the full STL grammar, which permits $t_2 = \infty$.

2.1.2 Derived Operators

The grammar (2.1) is sufficient to express a wide range of properties. Several operators can be expressed syntactically using the primitives above.

Implication. Logical implication $\varphi_1 \Rightarrow \varphi_2$ is a Boolean connective, and can be expressed as $\neg\varphi_1 \vee \varphi_2$.

Next. In discrete time, the next operator $\bigcirc\varphi$ asserts that φ holds one step into the future. It is expressible within the bounded grammar as a point-interval eventually operator

$$\bigcirc\varphi \;\equiv\; \Diamond_{[1,1]}\varphi, \tag{2.2}$$

since $\Diamond_{[1,1]}\varphi$ is satisfied at time t if and only if φ holds at time $t+1$. Iterating, the k-step-ahead next operator satisfies $\bigcirc^k\varphi \equiv \Diamond_{[k,k]}\varphi$.

Remark (Until operator). Although the bounded until operator $\mathbf{U}_{[t_1,t_2]}$ is omitted from the grammar, it can be incorporated into the proposed framework in a straightforward manner. Supporting $\mathbf{U}$ requires introducing an additional timer signal to encode its temporal progress, analogous to the timer construction described in Section 3.2.1. Since the specifications considered in this thesis do not require the until operator, it is omitted for simplicity.

2.1.3 Quantitative robustness semantics

Let $\xi = x_{0:H}$ be a signal covering the full specification horizon H. The satisfaction of an STL formula φ by ξ is quantified by the robustness metric $\rho_\varphi(\xi, t) \in \mathbb{R}$ [26, 8]. The signal ξ satisfies φ, denoted $\xi \models \varphi$, if and only if $\rho_\varphi(\xi, 0) \geq 0$. Furthermore, the magnitude indicates the margin: $\rho_\varphi(\xi, t) > 0$ provides a margin to violation, while $\rho_\varphi(\xi, t) < 0$ quantifies the degree of violation. The robustness is defined recursively:

$$\rho_\phi(\xi, t) = h(x_t), \tag{2.3}$$
$$\rho_{\neg\varphi}(\xi, t) = -\rho_\varphi(\xi, t), \tag{2.4}$$
$$\rho_{\varphi_1\wedge\varphi_2}(\xi, t) = \min(\rho_{\varphi_1}(\xi, t),\ \rho_{\varphi_2}(\xi, t))\,, \tag{2.5}$$
$$\rho_{\varphi_1\vee\varphi_2}(\xi, t) = \max(\rho_{\varphi_1}(\xi, t),\ \rho_{\varphi_2}(\xi, t))\,, \tag{2.6}$$
$$\rho_{\Box_{[t_1,t_2]}\varphi}(\xi, t) = \min_{t'\in t+[t_1,t_2]} \rho_\varphi(\xi, t'), \tag{2.7}$$
$$\rho_{\Diamond_{[t_1,t_2]}\varphi}(\xi, t) = \max_{t'\in t+[t_1,t_2]} \rho_\varphi(\xi, t'). \tag{2.8}$$

Negation flips the sign, conjunction takes the worst-case margin across conjuncts, disjunction takes the best-case, "always" takes the worst margin across the time window, and "eventually" takes the best. This quantitative semantics provides a continuous surrogate objective that can be maximized for policy optimization and thresholded at zero for certification.

Because every formula in this thesis is evaluated from the start of the episode, the abbreviation $\rho_\varphi(\xi) \equiv \rho_\varphi(\xi, 0)$ is used throughout Chapters 3 to 5, and the second argument is written explicitly only when robustness is evaluated at an intermediate time step.

2.2 Reinforcement Learning and Policy Optimization

2.2.1 Markov Decision Processes and Reinforcement Learning

The policy synthesis stage in Chapter 3 requires a framework for sequential decision-making under unknown dynamics. We model the agent-environment interaction as a Markov Decision Process (MDP) [27], defined by the tuple

$$\mathcal{M} = (\mathcal{S},\, \mathcal{A},\, \mathcal{P},\, r,\, \gamma), \tag{2.9}$$

where $\mathcal{S} \subset \mathbb{R}^n$ is the state space, $\mathcal{A} \subset \mathbb{R}^{n_u}$ is the action space, $\mathcal{P}(s' \mid s, a)$ is the transition probability from state s to s' under action a, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor controlling the relative weight of future rewards.

At each discrete time step t, the agent observes $s_t \in \mathcal{S}$, selects an action $a_t \sim \pi(\cdot \mid s_t)$ from a stochastic policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, and receives reward $r(s_t, a_t)$. The symbol a denotes the action, or equivalently the control input, throughout this thesis. The objective of reinforcement learning (RL) is to find an optimal policy $\pi^\star$ maximising the expected discounted return:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \, r(s_t, a_t) \right]. \tag{2.10}$$

Because the transition model $\mathcal{P}$ is unknown in our setting, $\pi^\star$ cannot be computed analytically and must instead be learned from environment interaction. Policy gradient methods compute this by directly optimising $J(\pi)$ with respect to the policy parameters θ, estimating gradients from sampled trajectories. The specific algorithm used in this work is described in Section 2.2.3.

Remark (State and augmented state). Throughout this thesis, the environment is assumed to be fully observable. Consequently, the terms *state* and *observation* are used interchangeably, since the observation contains the complete physical state of the system.

Assumption (Markov property). The transition probability $\mathcal{P}(s' \mid s, a)$ depends only on the current state-action pair (s_t, a_t), not on the history $s_{0:t-1}$.

2.2.2 Neural Networks

The policies trained in Chapter 3 are parameterized by neural networks. A feedforward neural network with L layers defines a function $g_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ by the composition

$$g_\theta(x) = W_L \, \sigma(W_{L-1} \, \sigma(\cdots \sigma(W_1 x + b_1) \cdots) + b_{L-1}) + b_L, \tag{2.11}$$

where $W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $b_\ell \in \mathbb{R}^{n_\ell}$ are the weight matrix and bias vector of layer ℓ, σ is a nonlinear activation function applied element-wise, and $\theta = \{W_1, b_1, \ldots, W_L, b_L\}$ collects all learnable parameters. Common choices for σ include the Rectified Linear Unit $\mathrm{ReLU}(z) = \max(0, z)$, the hyperbolic tangent $\tanh(z)$, and the sigmoid $\sigma(z) = 1/(1 + e^{-z})$.

For stochastic policies, the network outputs the parameters of a probability distribution over actions. A Gaussian policy, for example, outputs a mean vector $\nu_\theta(s)$ and a (possibly state-independent) standard deviation σ_θ, and actions are sampled as $a \sim \mathcal{N}(\nu_\theta(s), \mathrm{diag}(\sigma_\theta^2))$.

2.2.3 Proximal Policy Optimization

Among policy gradient methods, Proximal Policy Optimization (PPO) [28] is widely adopted for continuous control, where the action space $\mathcal{A} \subseteq \mathbb{R}^{n_u}$, due to its balance of sample efficiency and training stability. PPO addresses a fundamental challenge in policy gradient methods. If the policy changes too much in a single update, the advantage estimates become inaccurate and training can diverge. PPO constrains each update by maximizing a clipped surrogate objective:

$$\mathcal{L}^{\mathrm{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\Big(w_t(\theta)\,\hat{A}_t,\ \mathrm{clip}(w_t(\theta),\,1{-}c_{\mathrm{clip}},\,1{+}c_{\mathrm{clip}})\,\hat{A}_t\Big)\Big], \tag{2.12}$$

where $w_t(\theta) = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)$ is the importance weight between the current and previous policy, $\hat{A}_t$ is the estimated advantage (typically computed using Generalized Advantage Estimation), and c_{clip} is the clipping parameter. Once w_t leaves $[1 - c_{\mathrm{clip}}, 1 + c_{\mathrm{clip}}]$ in the direction favoured by the advantage, the gradient is clipped to zero, which prevents the update from overshooting into a region where the collected data is no longer representative. PPO also uses a value function loss $\mathcal{L}^{\mathrm{VF}}(\theta) = (V_\theta(s_t) - V_t^{\mathrm{target}})^2$ to train the critic, and an entropy bonus $\mathcal{L}^{\mathrm{ENT}}(\theta) = \mathcal{H}[\pi_\theta(\cdot \mid s_t)]$ to encourage exploration. The combined objective is

$$L(\theta) = \mathcal{L}^{\mathrm{CLIP}}(\theta) - c_1\mathcal{L}^{\mathrm{VF}}(\theta) + c_2\mathcal{L}^{\mathrm{ENT}}(\theta), \tag{2.13}$$

where c_1 and c_2 are coefficients balancing the three terms. Equation (2.13) is written as an objective to be *maximized*; the corresponding minimization loss used in Section 2.4.1 is $\mathcal{L}_{\mathrm{PPO}}(\theta) = -L(\theta)$.

Algorithm selection rationale: DQN. Deep Q-Networks (DQN) [29] extend tabular Q-learning to continuous state spaces using neural function approximation but assume a discrete action space. Although the action space could be discretized, doing so would unnecessarily restrict the controller, so continuous-control algorithms are considered instead.

Algorithm selection rationale: PPO over SAC. Soft Actor-Critic (SAC) [30] is the primary off-policy alternative for continuous control and a genuinely competitive baseline, offering higher sample efficiency through experience replay and principled maximum-entropy exploration. It has been used successfully for STL-constrained synthesis [9, 7] and remains preferable when environment interactions are expensive, as in direct real-robot training. Running a massive number of environments in parallel during training produces on-policy data at high throughput, reducing the practical advantage of SAC's sample efficiency. PPO is therefore used throughout this thesis, while SAC remains a preferred alternative when sample efficiency is more important.

2.3 PAC Guarantees and Binomial Confidence Bounds

The validation stage in Chapter 3 must certify that the trained policy's violation rate is bounded, using only finite rollout data and without access to the network's internal structure or the dynamics model. Probably Approximately Correct (PAC) guarantees from statistical learning theory provide exactly the required finite-sample, distribution-free certificate.

2.3.1 Probabilistic setup

Let $(\mathcal{Z}, \mathcal{G}, P)$ be a probability space where $\mathcal{Z}$, $\mathcal{G}$, and P denote a sample set, a σ-algebra, and a probability measure respectively. We consider $N_{cal}+1$ i.i.d. random variables $Z_1, \ldots, Z_N, Z_{N_{cal}+1}$ with values in $\mathcal{Z}$. The set

$$\mathcal{D}_{\text{cal}} \;=\; (Z_1, \ldots, Z_N), \qquad N_{cal} \;=\; \big|\mathcal{D}_{\text{cal}}\big|, \tag{2.14}$$

is the *calibration set*, reserved exclusively for uncertainty quantification, and N_{cal} denotes its size. The variable $Z_{N_{cal}+1}$ represents a new test sample drawn from the same distribution [31].

PAC guarantees are parameterized by a user-specified target pair $(E, \delta) \in [0,1]^2$, where E is the maximum acceptable error probability on future samples and δ is the maximum acceptable probability that the guarantee itself fails.

Definition 2.3.1 ((E,δ)-PAC Guarantee)**.** Let Γ denote a decision rule constructed from a training set and held fixed after calibration on $\mathcal{D}_{\text{cal}}$, and let $\mathcal{E}(Z) \in \mathcal{G}$ denote the measurable *error event* associated with Γ on a sample Z. The *inner guarantee* bounds the error probability on a new test point for the fixed calibrated rule,

$$P(\mathcal{E}(Z_{N_{cal}+1}) \mid \mathcal{D}_{\text{cal}}) \;\le\; E, \tag{2.15}$$

while the *outer guarantee* quantifies how often this holds over random draws of the calibration set,

$$P^N(P(\mathcal{E}(Z_{N_{cal}+1}) \mid \mathcal{D}_{\text{cal}}) \le E) \;\ge\; 1-\delta, \tag{2.16}$$

where P^N denotes probability over the i.i.d. draw of $Z_1, \ldots, Z_N$ [31]. A rule satisfying (2.16) is called (E,δ)-PAC certified.

Equation (2.16) is the main property of a training-conditional PAC guarantee: with confidence at least $1-\delta$ over the finite calibration sample, the conditional error rate on future operation does not exceed E.

2.3.2 Application to policy certification

In our setting, $Z = x_0 \sim \mu_0$, the decision rule Γ is the fixed policy under test, and the error event is $\mathcal{E}(x_0) = \{\rho_\varphi < 0\}$, that is, trajectory violating the specification.

Definition 2.3.2 (Violation Rate)**.** The *violation rate* of a fixed policy π is

$$\mathcal{V}(\pi) = \mathbb{P}_{x_0 \sim \mu_0}\left[\rho_\varphi(\xi_{x_0}^\pi) < 0\right], \tag{2.17}$$

where $\xi_{x_0}^\pi$ is the closed-loop trajectory from x_0 under policy π. The policy is (E,δ)-PAC certified if, with confidence at least $1-\delta$ over the calibration draw, $\mathcal{V}(\pi) \le E$.

The certificate is computed from observed rollouts as follows. Let n_{viol} denote the number of trajectories among the N_{cal} i.i.d. rollouts of $\mathcal{D}_{\text{cal}}$ on which $\rho_\varphi < 0$. The Clopper-Pearson interval provides an exact one-sided upper bound on $\mathcal{V}(\pi)$ at confidence $1-\delta$,

$$\begin{aligned} \epsilon_{\text{CP}}(n_{\text{viol}}, N_{cal}, \delta) \;&=\; \inf\left\{ \epsilon \in [0,1] \;\middle|\; \sum_{j=0}^{n_{\text{viol}}} \binom{N_{cal}}{j} \epsilon^j (1-\epsilon)^{N_{cal}-j} \le \delta \right\} \\ &=\; \text{Beta}^{-1}(1-\delta;\; n_{\text{viol}}+1,\; N_{cal}-n_{\text{viol}}), \end{aligned} \tag{2.18}$$

where $\mathrm{Beta}^{-1}(p_{\mathrm{beta}};\, u, v)$ is the p_{beta}-level quantile of the $\mathrm{Beta}(u, v)$ distribution. The policy is declared (E, δ)-PAC certified whenever

$$\epsilon_{\mathrm{CP}}(n_{\mathrm{viol}}, N_{cal}, \delta) \;\leq\; E, \tag{2.19}$$

since this implies $\mathcal{V}(\pi) \leq E$ at confidence $1-\delta$.

2.3.3 Assumptions and limitations

The bound requires only that the N_{cal} initial conditions in $\mathcal{D}_{\mathrm{cal}}$ are drawn independently from μ_0. No assumptions on the functional form of μ_0, the dynamics f, or the policy are needed. The policy must be *fixed* during validation, a condition preserved in the proposed framework because synthesis and validation occur in strictly alternating phases. The certificate characterizes the *global* violation rate over μ_0 and provides no spatial information about where failures concentrate. Extracting that structure requires post-processing the failure set, as described in Section 3.4.

2.4 Continual Learning

The refinement stage in Chapter 3 retrains the policy on counterexample regions where validation detected failures. Sequential optimization of the form can overwrite previously acquired useful behaviors when refining a new counterexample region. To mitigate this form of catastrophic forgetting [32, 33], this thesis considers two continual-learning approaches, Elastic Weight Consolidation (EWC) and Progressive Neural Networks (PNN), which are introduced in the following sections.

2.4.1 Continual Learning: Elastic Weight Consolidation

Elastic Weight Consolidation (EWC) addresses catastrophic forgetting by selectively protecting the network weights that matter most for past performance, rather than allowing unconstrained updates. The Fisher information identifies those weights: it measures how sensitive the log-likelihood of the policy's actions is to a perturbation of each weight. The diagonal Fisher entry for weight θ_j is

$$F_j = \mathbb{E}_{s\sim d^{\pi_\theta}}\left[\left(\frac{\partial}{\partial\theta_j}\log\pi_\theta(a \mid s)\right)^2\right], \tag{2.20}$$

where d^{π_θ} is the state distribution induced by the current policy. A large F_j means that θ_j is a weight that encodes behaviour critical to current performance; a small F_j means it can be updated with little risk. EWC uses these values as per-weight stiffness coefficients in a quadratic penalty. After completing refinement round k with weights $\theta^\star_k$, the loss for round $k{+}1$ becomes

$$\mathcal{L}_{k+1}(\theta) = \mathcal{L}_{\mathrm{PPO}}(\theta) + \frac{\lambda_{\mathrm{EWC}}}{2}\sum_j F_j^{(k)}\left(\theta_j - \theta^\star_{j,k}\right)^2, \tag{2.21}$$

where $\mathcal{L}_{\text{PPO}}(\theta) = -L(\theta)$ is the minimization form of the PPO objective (2.13) and λ_{EWC} controls the regularization strength, so both terms are minimized jointly. Weights with a large Fisher coefficient incur a steep penalty for deviating from their round-k values and are effectively anchored, while weights with a small coefficient remain free to adapt to new counterexample data.

Assumptions. EWC relies on a diagonal approximation of the Fisher matrix, which ignores correlations between weights and can therefore under-regularize a pair of weights that are jointly important but individually of moderate importance. The approximation is standard [32, 33] because the full $|\theta| \times |\theta|$ matrix is intractable to store. Benchmarks on continual robotic RL [34] show that EWC substantially reduces forgetting relative to naïve fine-tuning, but suppresses network plasticity when λ_{EWC} is too high.

2.4.2 Continual Learning: Progressive Neural Networks

An alternative to regularization-based forgetting prevention is structural isolation by ensuring that no gradient from the current refinement round can reach the weights that encode previously certified behaviour. Progressive Neural Networks (PNN) [35] achieve this by adding a new network column for each round while freezing all prior columns.

The key property of this architecture is that the new column does not specialise from scratch. Lateral connections supply activations from the frozen source column directly into each layer of the new column, making the already-learned feature representations of the source available to the new column during training. The new column can therefore build on existing low-level and high-level features rather than relearning them, which both accelerates specialisation on the new region and ensures that the source column's behaviour is preserved by construction.

In the general formulation, a column draws lateral input from every preceding column. This thesis uses the restricted variant in which lateral input is drawn from the initial column alone, so the forward pass at layer ℓ is

$$h_\ell^{(0)} = f\left(W_\ell^{(0)} h_{\ell-1}^{(0)}\right), \qquad h_\ell^{(m)} = f\left(W_\ell^{(m)} h_{\ell-1}^{(m)} + C_\ell^{(m,0)} h_{\ell-1}^{(0)}\right) \quad \text{for } m \geq 1, \tag{2.22}$$

where $W_\ell^{(m)}$ are the new column's weights, $C_\ell^{(m,0)}$ is the lateral connection weight matrix initialised to a small value, and only $\{W_\ell^{(m)}, C_\ell^{(m,0)}\}$ receive gradients. Every column $m \geq 1$ receives the same lateral input from column 0, so no column accumulates more frozen input than another.

PNN guarantees zero forgetting by construction. The cost is linear growth in the total number of weights, since each round adds one full column. For the number of refinement rounds considered in Chapter 5, at most ten, this growth remains manageable. For longer sequences, pruning or knowledge distillation would be necessary.

Having introduced STL for specification encoding, neural networks and PPO for model-free policy optimisation, binomial PAC bounds, EWC and PNN for forgetting prevention during sequential refinement, the next chapter develops the proposed framework. Chapter 3 opens with the formal problem statement, then develops the framework in Sections 3.2, 3.3, 3.4, and 3.5, using the tools introduced above.

Chapter 3

Methodology

This chapter develops the proposed framework for synthesizing and certifying reinforcement learning controllers to satisfy Signal Temporal Logic specifications. Section 3.1 formalizes the problem. Section 3.2 constructs the decision process on which learning takes place, Section 3.3 specifies how Proximal Policy Optimization exploits trains on this MDP, Section 3.4 certifies the resulting policy and extracts counterexample regions, and Section 3.5 uses them to run the refinement step. Section 3.6 presents the complete algorithm by combining the preceding components, together with a complexity analysis and discussion of failure modes.

3.1 Problem Formulation

3.1.1 System and Specification

Consider a discrete-time dynamical system with state $x_t \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ and action $a_t \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$, governed by

$$x_{t+1} = f(x_t, a_t), \tag{3.1}$$

where $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the (possibly unknown) transition map. The state x_t contains only the quantities that f propagates, namely the pose and the body velocities of the robot.

Following Section 2.2.1, a_t denotes the action, or equivalently the control input applied to the plant. Time is discretised with a fixed step Δt, and the index $t \in \{0, 1, 2, \ldots\}$ counts simulation steps, so that step t corresponds to elapsed time $t\,\Delta t$. The transition map f is accessed only through a simulator, which returns x_{t+1} for any pair (x_t, a_t) without a closed-form expression. This black-box assumption is standard in reinforcement learning and reflects the use of physics engines or hardware-in-the-loop simulators for training.

The robot does not start every episode/rollout from the same configuration. The starting configuration of episode i is a random variable

$$x_0^{(i)} \sim \mu_0, \tag{3.2}$$

where μ_0 is a probability distribution over starting configurations (for example, a distribution over the position and orientation of the robot in the arena at the start of a run). Its support,

$$\mathcal{X}_0 \;=\; \mathrm{supp}(\mu_0) \;=\; \{\, x \in \mathcal{X} \;\mid\; \mu_0(x) > 0 \,\} \;\subseteq\; \mathcal{X}, \tag{3.3}$$

is the set of all starting configurations that can actually be drawn. Following the scope stated in Section 3.1.3, μ_0 places mass only on the zero-velocity slice of $\mathcal{X}$, so $\mathcal{X}_0$ varies over starting position and heading alone. The distribution μ_0 is unknown, starting an episode provides one sample from it, but the probability distribution itself is not known. The certification is therefore distribution-free, relying on nothing about μ_0 beyond the ability to draw independent samples.

A behavioral requirement over the system is expressed as a Signal Temporal Logic formula φ, constructed from the grammar introduced in Section 2.1, whose predicates are written ϕ. The *specification horizon* $T_\varphi > 0$ is the length of the full specification: the largest time bound appearing in φ, expressed in seconds. It induces the episode length in simulation steps, $H = \lceil T_\varphi / \Delta t \rceil$, so that a trajectory covering the full specification horizon consists of $H + 1$ states. The quantitative robustness semantics (equations (2.3) through (2.8)) assign a real-valued score $\rho_\varphi(\xi) \in \mathbb{R}$ to every finite trajectory

$$\xi \;=\; x_{0:H} \;=\; (x_0, x_1, \ldots, x_H), \tag{3.4}$$

with the property

$$\rho_\varphi(\xi) \;\geq\; 0 \;\iff\; \xi \;\models\; \varphi. \tag{3.5}$$

The magnitude $|\rho_\varphi(\xi)|$ measures the margin of satisfaction or violation. Throughout, $\rho_\varphi(\xi)$ denotes the robustness of a single trajectory ξ evaluated from step 0, and is always a scalar.

3.1.2 Policy and Closed-Loop Trajectory

With the system and specification defined, the policy needs to be computed to satisfy the specification. The policy closes the loop around (3.1), we use a parametric stochastic policy

$$\pi_\theta, : \mathcal{S} \to \Delta(\mathcal{U}), \tag{3.6}$$

parameterized by a neural network with weights $\theta \in \Theta$, maps states to distributions over actions. The domain $\mathcal{S}$ is the augmented state space constructed in Section 3.2.1; it consists of the observation together with the timer signals. Writing $s_t \in \mathcal{S}$ for the augmented state at step t, a fixed policy π_θ and an initial condition x_0 produce the closed-loop trajectory

$$\xi_{x_0}^{\pi_\theta} = (x_0, x_1, \ldots, x_H), \qquad x_{t+1} = f\big(x_t, \pi_\theta(s_t)\big), \tag{3.7}$$

where, by a slight abuse of notation, $\pi_\theta(s_t)$ denotes a sample drawn from $\pi_\theta(\cdot \mid s_t)$. The trajectory ξ is recorded as a sequence of states.

Training and certification act on two policies derived from the same parameters, and the distinction matters because the certificate is a statement about the deployed controller. Training uses the stochastic π_θ of (3.6), whose exploration noise is essential to policy gradient estimation and certification, deployment use its deterministic *mean policy*:

$$\bar{\pi}_\theta(s_t) \;=\; \mathbb{E}\big[a \;\big|\; a \sim \pi_\theta(\cdot \mid s_t)\big] \;=\; \nu_\theta(s_t), \tag{3.8}$$

where $\nu_\theta(s_t)$ is the mean of the Gaussian output. Under $\bar{\pi}_\theta$ the trajectory $\xi_{x_0}^{\bar{\pi}_\theta}$ is a deterministic function of x_0, so the only randomness entering the certificate is the draw of the initial condition. Every quantity in Section 3.4 is therefore defined with respect to $\bar{\pi}_\theta$.

The violation rate of the deployed policy $\bar{\pi}_\theta$ under the true initial condition distribution μ_0 is defined by Definition 2.3.2 as

$$\mathcal{V}(\bar{\pi}_\theta) \;=\; \mathbb{P}_{x_0\sim\mu_0}\big[\rho_\varphi\big(\xi_{x_0}^{\bar{\pi}_\theta}\big) < 0\big], \tag{3.9}$$

which is the probability that a trajectory, starting from a random initial condition drawn from μ_0, violates the specification.

3.1.3 Formal Problem Statement

The system (3.1), the specification φ, the policy class (3.6), and the violation rate (3.9) are the ingredients needed to state the problem formally.

Problem Statement 3.1.1. Given a discrete-time system (3.1) with black-box dynamics f, a true initial condition distribution μ_0 with support $\mathcal{X}_0$, an STL specification φ with horizon T_φ, a violation tolerance $E \in (0,1)$, and a confidence level $\delta \in (0,1)$:

1. **Synthesis:** Find a policy in the class $\Pi = \{\pi_\theta : \theta \in \Theta\}$ that maximizes expected robustness,
$$\max_{\theta\in\Theta} \;\; \mathbb{E}_{x_0\sim\mu_0}\Big[\rho_\varphi\big(\xi_{x_0}^{\pi_\theta}\big)\Big]. \tag{3.10}$$

2. **Certification:** Certify, from finite data, that the deployed mean policy $\bar{\pi}_\theta$ of (3.8) achieves a PAC guarantee on the violation rate,
$$\mathbb{P}\big[\mathcal{V}(\bar{\pi}_\theta) \le E\big] \;\ge\; 1-\delta, \tag{3.11}$$
where the probability is taken over a random draw of the calibration set used to compute the certificate.

3. **Refinement:** If certification fails, improve π_θ on identified failure regions, then re-certify.

These requirements give rise to three challenges. Requirement 1 requires learning a policy in a high-dimensional, non-convex parameter space from trajectory-level rewards. Requirement 2 requires certifying performance from finite samples without knowledge of the true initial condition distribution μ_0. Requirement 3 requires refining the policy on newly identified counterexamples while avoiding catastrophic forgetting.

Scope. The framework is restricted to deterministic dynamics with stochastic initial position and heading only. Stochastic transitions, process and measurement noise, nonzero initial velocity, partial observability, and online adaptation during deployment are not addressed in this thesis and possible extensions along each of these directions are discussed as research directions in Section 6.2.

The proposed framework addresses Problem 3.1.1 through an iterative pipeline, illustrated in Figure 3.1. The unit of iteration is a *round*, indexed by k: train the policy, draw a fresh calibration set $\mathcal{D}_{\text{cal}}^{(k)}$ of N_{cal} initial conditions from μ_0 and roll out from each to test the certificate, and, if it fails, extract the region of starting configurations responsible. Round 0 trains the policy on initial conditions drawn from μ_0 and each subsequent round retrains on a distribution biased toward the failure region identified in the previous one. Validation draws from μ_0 in every round, ensuring that the certificate remains a statement about the original initial-state distribution rather than the biased training distribution. The three stages of this loop are developed in throughout the remainder of the chapter, starting with the decision process on which training operates.

3.2 Markov Decision Process Setup

The synthesis objective (3.10) is an optimization problem over policies acting within a sequential decision process. A standard Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r_{\text{tot}}(\cdot), \gamma)$, as introduced in Section 2.2.1, defines the state space, action space, transition probability, reward function, and discount factor. Applying this formulation directly to an STL specification makes the decision process non-Markovian because temporal operators aggregate state information over time windows. This section defines the MDP that resolves this challenge and explain each of the components of the tuple.

Definition 3.2.1 (MDP)**.** The *MDP* is the tuple

$$\mathcal{M}_\varphi \;=\; (\mathcal{S},\; \mathcal{A},\; \mathcal{P},\; r_{\text{tot}}(\cdot),\; T_\varphi,\; \gamma), \tag{3.12}$$

where $\mathcal{S}$ is the augmented state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition probability, $r_{\text{tot}}(\cdot)$ is the reward function, T_φ is the episode horizon in seconds, corresponding to $H = \lceil T_\varphi/\Delta t \rceil$ simulation steps, and γ is the discount factor. An episode begins from the augmented initial state $s_0 = [x_0^{\text{obs}} \mid s_{\varphi,0}]$, where $x_0 \sim \mu_0$ and $s_{\varphi,0}$ is the all-zero timer vector defined in Section 3.2.5.

3.2.1 State Space and Observation Augmentation

An MDP requires a state on which the transition and the reward depend only through its current value for the Markov property to hold. The physical state x_t of (3.1) violates this property in two ways when STL specifications are involved.

The first violation is the environment. The robots action should depend on the current state only, which depends on where the goal and the obstacles are, and x_t records neither. The MDP state therefore includes the task-relative quantities

$$x_t^{\text{obs}} = f_{\text{obs}}(x_t) \in \mathcal{X}^{\text{obs}} \subseteq \mathbb{R}^{n_{\text{obs}}}, \tag{3.13}$$

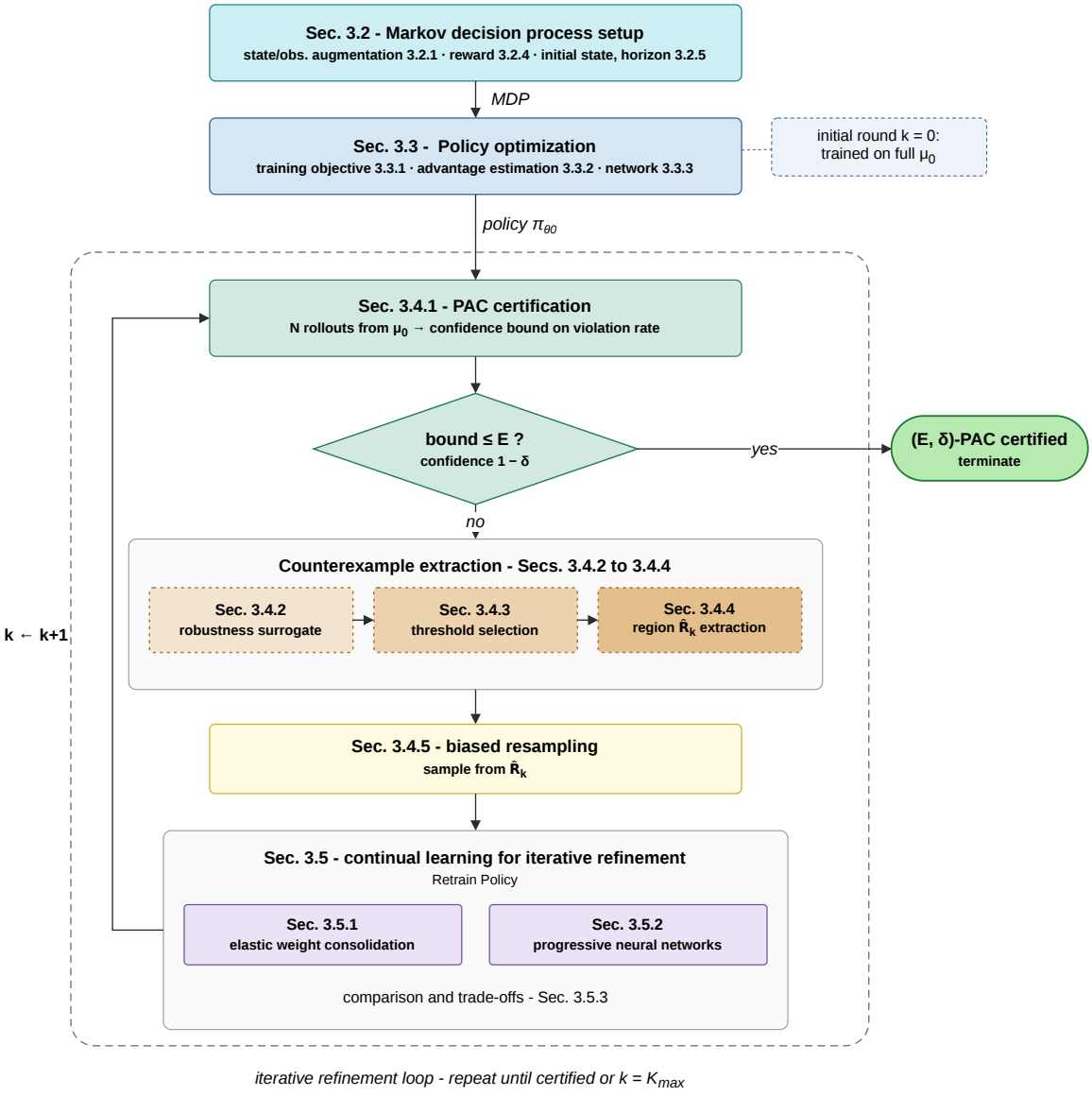


Figure 3.1: Overview of the proposed framework. The workflow consists of policy training, validation, failure-region identification, and iterative refinement to improve policy safety and performance while providing statistical certification guarantees.

where f_{obs} appends to x_t the goal position in the body frame, the heading error, and the position of the nearest obstacle for instance. Because the environment is static, f_{obs} is a fixed function evaluated at each step.

The second violation is the specification. The robustness $\rho_\varphi(\xi)$ depends on the *entire* trajectory through the $\square$ and $\lozenge$ operators (equations (2.7) and (2.8)), so a policy acting on x_t^{obs} alone cannot determine whether a stay requirement is partially satisfied or how much time remains before a deadline expires. The decision process is non-Markovian in x_t^{obs}.

The standard approaches are to augment the state with explicit history or to use a recurrent architecture, such as a Long Short-Term Memory (LSTM) network or a Recurrent Neural Network (RNN), to retain information from previous observations. The τ-MDP formulation [7] encodes the required history of physical states directly in the augmented state space, which is sufficient to evaluate any STL subformula whose time window fits within the history length. The cost is that the state space grows exponentially with the length of history retained, rendering learning intractable for the long horizons, the approach can solve short-horizon tasks with 2-layer STL specifications. The F-MDP formulation [36] addresses this by replacing the raw history with a set of discrete binary flags, one per STL subformula, that track whether each subformula has been satisfied so far. This reduces the augmentation to a size that grows only with the number of subformulas, but the binary flags discard quantitative progress information: a flag records whether a subformula is complete or not only.

We propose a extension of the F-MDP flag idea. Rather than discrete completion indicators, we augment x_t^{obs} with *timer signals* encoding progress within the time window of each temporal operator as a real value in $[0, 1]$: an *always fraction* per $\square$ subformula and an *eventually fraction* per $\lozenge$ subformula. These capture the temporal context the policy requires to act Markovianly without maintaining explicit state history or tracking robustness margins in the state.

Timer Signals

The temporal operators $\square$ and $\lozenge$ in φ aggregate predicate values over time windows. A policy acting on the current physical state x_t alone cannot determine how far through a required interval the episode has progressed, whether a predicate has been violated at an earlier step, or how much of a deadline has expired. The decision process is therefore non-Markovian in x_t, and a policy that must reason about temporal progress cannot act correctly without access to this history.

Timer signals resolve this by summarising the relevant temporal context in a compact, per-operator scalar. Every bounded temporal operator in φ contributes one timer to the augmented state, taking values in $[0, 1]$ where 0 indicates no progress and 1 indicates the operator places no further demand on the policy. The two operator types, $\lozenge$ and $\square$, produce timers of different kinds, defined below and illustrated in Figure 3.2.

Eventually fraction (timer for $\lozenge$ operators). A $\lozenge_{[T_{s,i}, T_{d,i}]}$ operator requires its argument to be satisfied at some point within the interval $[T_{s,i}, T_{d,i}]$. The timer encodes the fraction of the

interval that has elapsed,

$$\tau_{i,t}^{\text{eventually}} \;=\; \text{clip}\left(\frac{t\,\Delta t - T_{s,i}}{T_{d,i} - T_{s,i}},\; 0,\; 1\right). \tag{3.14}$$

This fraction starts at zero when $t\,\Delta t = T_{s,i}$, increases linearly, and reaches 1 at the deadline $T_{d,i}$, giving the policy a direct urgency signal (Figure 3.2a). One important note, the eventually fraction is a clock. It tracks elapsed time, not whether the argument of the $\Diamond$ operator was ever satisfied. A policy observing $\tau_{i,t}^{\text{eventually}}$ alone cannot determine whether the obligation has already been met.

Always fraction (timer for $\Box$ operators). A $\Box$ operator requires a predicate ϕ_i to be satisfied over a specified time interval. The always fraction records both how far through the required interval the episode has progressed and whether the predicate has remained satisfied throughout.

Case 1: full-horizon safety ($\Box_{[0,T_\varphi]}\phi$). When the interval spans the entire episode, a single violation falsifies the specification and terminates the episode. There is no partial progress to encode, and no timer is introduced.

Case 2: fixed-window invariance ($\Box_{[T_{lo,i},\,T_{hi,i}]}\phi_i,\ [T_{lo,i}, T_{hi,i}] \neq [0, T_\varphi]$). When the interval is a proper sub-window, a single violation renders the requirement unsatisfiable for the rest of the episode. A binary latch $z_{i,t} \in \{0, 1\}$, initialised to one, is set irreversibly to zero the first time ϕ_i is violated inside the window,

$$z_{i,t} \;=\; z_{i,t-1} \cdot \mathbb{1}\big[\, t\,\Delta t \notin [T_{lo,i}, T_{hi,i}] \;\vee\; \rho_{\phi_i}(x_t) \geq 0 \,\big], \qquad z_{i,0} = 1, \tag{3.15}$$

where ρ_{ϕ_i} is the predicate robustness from (2.3). The always fraction combines this latch with a normalised clock over the window,

$$\tau_{i,t}^{\text{always}} \;=\; z_{i,t} \cdot \text{clip}\left(\frac{t\,\Delta t - T_{lo,i}}{T_{hi,i} - T_{lo,i}},\; 0,\; 1\right). \tag{3.16}$$

When $z_{i,t} = 0$ the timer collapses to zero irrespective of the clock value, signalling that the operator can no longer be satisfied. Figure 3.2b illustrates both the satisfied and violated trajectories.

Nested $\Diamond\Box$ operators. The eventually fraction provides urgency but carries no memory of whether the argument was ever satisfied. The always fraction has memory: it records unbroken satisfaction and latches once the obligation is met. Pairing the two timers therefore resolves the limitation of the standalone $\Diamond$. The always fraction records *whether* the argument is being satisfied, and the eventually fraction records *how much time remains* to achieve it.

This pairing arises naturally in the combination $\Diamond_{[T_{s,i},T_{d,i}]}\Box_{[0,T_{a,i}]}\phi_i$, which requires ϕ_i to be continuously satisfied for $T_{a,i}$ seconds in the $\Diamond$ time window. This particular nesting covers a wide range of practical tasks in robotics such as reaching a goal within a time window and staying there.

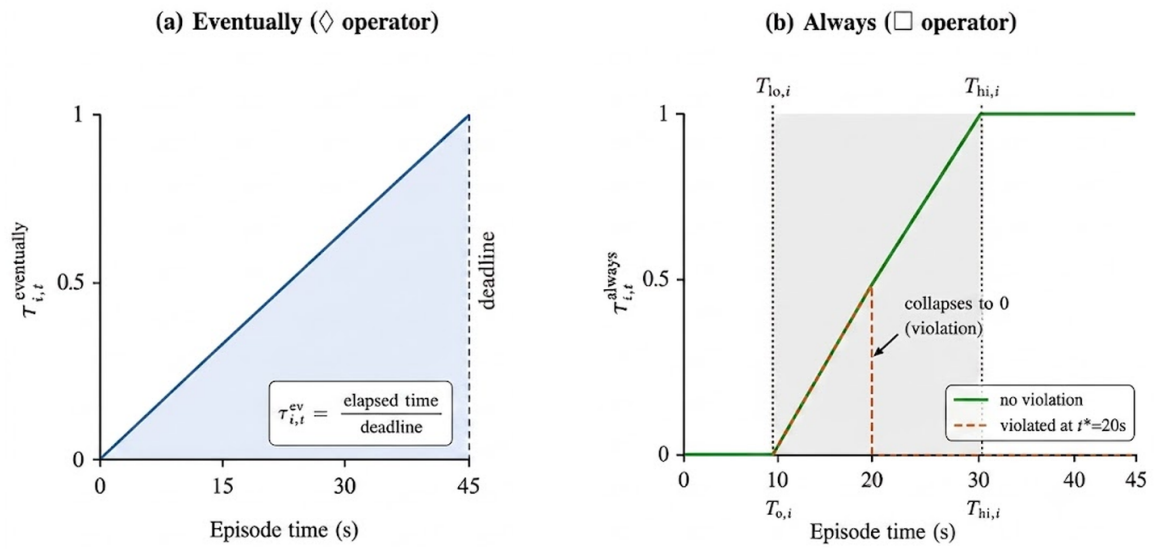


Figure 3.2: The two timer types. Panel (a) encodes urgency for ◊ operators. Panel (b) encodes both progress and integrity for □ operators: the green trajectory satisfies the window requirement, while the orange trajectory is permanently reset to zero at the moment of violation.

The outer $\Diamond_{[T_{s,i},T_{d,i}]}$ contributes $\tau^{\text{eventually}}_{i,t}$ of (3.14), unchanged from the standalone case. The inner $\Box_{[0,T_{a,i}]}$ requires an always fraction, but the form differs from Case 2 because ϕ_i must be satisfied for $T_{a,i}$ consecutive seconds starting at any point before the deadline, so a fixed-window clock is not appropriate. Instead, a counter $z^{\text{stay}}_{i,t}$ accumulates unbroken satisfaction time,

$$z^{\text{stay}}_{i,t} = \begin{cases} T_{a,i}, & \text{if } z^{\text{stay}}_{i,t-1} \geq T_{a,i}, \\ (z^{\text{stay}}_{i,t-1} + \Delta t)\,\mathbb{1}[\rho_{\phi_i}(x_t) \geq 0], & \text{otherwise}, \\ 0, & \text{initially } (t = 0). \end{cases} \tag{3.17}$$

Any interruption in satisfaction resets the counter to zero. The first branch freezes it at $T_{a,i}$ once the obligation is met, making satisfaction a permanent outcome that cannot be reversed. This freeze is the property that Proposition 3.2.1 below relies on. The always fraction is the normalised counter,

$$\tau^{\text{always}}_{i,t} = \text{clip}\left(\frac{z^{\text{stay}}_{i,t}}{T_{a,i}},\ 0,\ 1\right). \tag{3.18}$$

Together, $(\tau^{\text{eventually}}_{i,t}, \tau^{\text{always}}_{i,t})$ gives the policy both quantities it needs, how much of the deadline has elapsed, and what fraction of the required hold duration $T_{a,i}$ has been accumulated. When $\tau^{\text{always}}_{i,t} = 1$ the inner □ obligation is fully met and the ◊ operator is satisfied. A nonzero value $\tau^{\text{always}}_{i,t} \in (0, 1)$ indicates that accumulation is in progress but not yet complete. Until satisfaction is reached, $\tau^{\text{eventually}}_{i,t}$ conveys the urgency.

Remark (Deeper nesting). The construction can be extended to operator nesting deeper than two levels with minimal changes. Each additional operator contributes one timer of the matching type, and in most practical specifications, no modifications to the definitions above

are required.

Remark (Standalone $\Diamond$ subformulas). Proposition 3.2.1 below assumes every $\Diamond$ subformula appears nested as $\Diamond_{[T_{s,i},T_{d,i}]}\Box_{[0,T_{a,i}]}\phi_i$. When a $\Diamond$ appears without an inner $\Box$, whether its argument was satisfied at some earlier step is not recorded by $\tau_{i,t}^{\text{eventually}}$ alone, because the eventually fraction is only a clock. Adding one binary satisfaction flag per such subformula to (3.20) restores the sufficiency conclusion, at a cost of one additional state dimension per such operator.

The Augmented State Space

The timers defined above are assembled into the state as follows. Let $N_\Box$ denote the number of $\Box$ subformulas in φ that carry a timer, including the fixed-window and nested cases but excluding the full-horizon safety case, and let $N_\Diamond$ denote the number of $\Diamond$ subformulas. The timers are collected into a single vector,

$$s_\varphi(t) \;=\; [\tau_{1,t}^{\text{always}},\, \ldots,\, \tau_{N_\Box,t}^{\text{always}},\, \tau_{1,t}^{\text{eventually}},\, \ldots,\, \tau_{N_\Diamond,t}^{\text{eventually}}] \;\in\; \mathbb{R}^{N_\Box+N_\Diamond}, \tag{3.19}$$

and concatenated with the observation to form the augmented state,

$$s_t \;=\; [x_t^{\text{obs}} \mid s_\varphi(t)] \;\in\; \mathbb{R}^{n_{\text{obs}}+N_\Box+N_\Diamond}, \tag{3.20}$$

where $x_t^{\text{obs}} \in \mathcal{X}^{\text{obs}}$ is the observation of (3.13), containing position, orientation, velocities, and task-relative coordinates, and $\mid$ denotes concatenation. The augmented state space is therefore

$$\mathcal{S} \;=\; \mathcal{X}^{\text{obs}} \times [0,1]^{N_\Box+N_\Diamond} \;\subseteq\; \mathbb{R}^{n_{\text{obs}}+N_\Box+N_\Diamond}, \tag{3.21}$$

which is the domain of the policy declared in (3.6). The timer components make $\pi_\theta(a_t \mid s_t)$ depend on time making the policy time-varying.

Each timer requires only a single arithmetic update per step, and the augmentation adds $N_\Box + N_\Diamond$ dimensions to the state vector, independent of the specification horizon. The following proposition formalises the key property of this construction. The timers provide a sufficient statistic for the satisfaction status of every temporal subformula. A feedforward policy acting on s_t has access to everything it needs to reason about the specification.

Proposition 3.2.1 (Sufficiency of the timer augmentation)**.** *Suppose every $\Diamond$ subformula of φ appears nested as $\Diamond_{[T_{s,i},T_{d,i}]}\Box_{[0,T_{a,i}]}\phi_i$. Then for every temporal subformula of φ, whether it is satisfied over steps $0,\ldots,t$ is a deterministic function of the augmented state s_t of* (3.20)*.*

Proof. Every temporal subformula of φ falls into one of three cases, corresponding to the three cases of Section 3.2.1.

1. *Full-horizon safety* ($\Box_{[0,T_\varphi]}\phi$). Any violation terminates the episode, so at every step the episode reaches the operator is satisfied. Its status is constant along any continuing trajectory and is therefore trivially a function of s_t.

2. *Fixed-window invariance* ($\Box_{[T_{lo,i},\,T_{hi,i}]}\phi_i$). The operator has been violated by step t if and only if $z_{i,t} = 0$. For $t\,\Delta t > T_{lo,i}$, the clock factor in (3.16) is strictly positive, so $z_{i,t} = 0$ if and only if $\tau_{i,t}^{\text{always}} = 0$. For $t\,\Delta t \le T_{lo,i}$, the window has not yet opened and no violation can have occurred. In both cases the satisfaction status is determined by $\tau_{i,t}^{\text{always}}$ and the elapsed time $t\,\Delta t$, both of which are available in s_t.

3. *Nested* $\Diamond\Box$ *operator* ($\Diamond_{[T_{s,i},T_{d,i}]}\Box_{[0,T_{a,i}]}\phi_i$). The combined operator is satisfied by step t if and only if the inner $\Box$ has been satisfied, that is $z_{i,t}^{\text{stay}} \geq T_{a,i}$. By the freeze in (3.17), this holds if and only if $\tau_{i,t}^{\text{always}} = 1$, and the freeze preserves this value for all subsequent steps. The outer $\Diamond$ is satisfied whenever the inner $\Box$ is; $\tau_{i,t}^{\text{eventually}}$ additionally indicates whether the deadline $T_{d,i}$ is still open. Both operators are therefore determined by timers present in s_t.

In each case the satisfaction status is a deterministic function of s_t. □

Remark (Satisfaction status versus quantitative margin). Proposition 3.2.1 establishes that the timers encode whether each subformula is satisfied. They do not recover the exact robustness value $\rho_\varphi(\xi)$, which depends on the full trajectory. However, the timers do carry an indirect sense of margin. For example, satisfying a $\Diamond\Box$ operator while $\tau_{i,t}^{\text{eventually}}$ is still well below 1 implies the deadline was met with time to spare, which corresponds to a positive robustness contribution from that operator.

3.2.2 Action Space

The augmented state (3.20) fixes the input of the policy, its output is the action. The action space $\mathcal{A} = \mathcal{U} \subseteq \mathbb{R}^{n_u}$ is inherited directly from the physical actuator limits of the controlled system (3.1), and its elements are the actions a_t of (3.1). The specific bounds are system-dependent and documented in Chapter 5.

3.2.3 Transition Probability

Given a state and an action, the transition probability $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ governs the evolution of the augmented state. Since the dynamics (3.1) are deterministic, $\mathcal{P}$ assigns unit mass to a single successor. The physical component updates via the simulator, $x_{t+1} = f(x_t, a_t)$, the observation x_{t+1}^{obs} is computed from it, and the timer component follows by re-evaluating the always and eventually-fraction definitions in section 3.2.1 at x_{t+1}, so no separate map needs to be defined. The successor augmented state is

$$s_{t+1} \;=\; \big[\, x_{t+1}^{\text{obs}} \;\big|\; s_\varphi(t+1) \,\big], \tag{3.22}$$

with $s_\varphi(t+1)$ obtained by re-evaluating (3.18), (3.16) and (3.14) at step $t+1$.

3.2.4 Reward Function

The remaining component to specify is the signal that scores these transitions. Using the trajectory-level robustness ρ_φ directly as a terminal reward provides no per-step gradient signal, leading to sparse learning. The reward is therefore decomposed into two components,

$$r_{\text{tot}}(\xi) \;=\; \alpha\, \rho_\varphi(\xi) \;+\; \sum_{t=0}^{H} r_{\text{aux}}(s_t, a_t), \tag{3.23}$$

where ξ is the full closed-loop trajectory (3.7), $\rho_\varphi(\xi)$ is the trajectory-level STL robustness, $\alpha > 0$ is a scaling coefficient, and $r_{\text{aux}}(s_t, a_t)$ is a step-level auxiliary reward that provides dense shaping (proximity bonuses, constraint penalties). The specific form and weights of the auxiliary terms, and the value of α, are documented in Appendix B.

Robustness Value Computation

The terminal reward requires $\rho_\varphi(\xi)$, but the full trajectory is only available once the episode ends. To avoid storing the trajectory, each temporal operator maintains a running accumulator that is updated at every step from the predicate robustness $\rho_{\phi_i}(x_t)$ of (2.3). A $\square$ operator accumulates by running minimum over its window and a $\lozenge$ operator by running maximum, matching the semantics of (2.7) and (2.8) exactly. Each accumulator is active while $t\,\Delta t$ lies inside the operator's time window and frozen once the window closes. At termination, the frozen per-operator values are combined according to the Boolean structure of φ via (2.4)–(2.6), giving $\rho_\varphi(\xi)$ exactly. This is the quantity used as the terminal reward component $\alpha\,\rho_\varphi(\xi)$ in (3.23) and during certification (Section 3.4).

3.2.5 Initial Augmented State, Horizon, and Discount Factor

What remains of the tuple (3.12) are the quantities that define an episode: where it starts, how long it runs, and how future reward is weighted within it.

Initial augmented state. An episode begins by drawing a starting configuration $x_0 \sim \mu_0$ and setting every timer to zero, so the initial augmented state is

$$s_0 \;=\; [x_0^{\text{obs}} \mid s_{\varphi,0}], \qquad s_{\varphi,0} \;=\; [\underbrace{0,\ldots,0}_{N_\square},\ \underbrace{0,\ldots,0}_{N_\lozenge}]. \tag{3.24}$$

Every always fraction is zero because no satisfaction time has been accumulated and no fixed window has opened, and every eventually fraction is zero because no simulation time has elapsed, so the only source of randomness in s_0 is the starting configuration x_0, distributed according to μ_0. Every violation latch is initialised to $z_{i,0} = 1$, which is consistent with $\tau_{i,0}^{\text{always}} = 0$ through (3.16).

Horizon T_φ. The episode length equals the specification horizon T_φ, the length of the full STL specification, corresponding to $H = \lceil T_\varphi/\Delta t \rceil$ simulation steps.

Remark (Sound early termination). An episode may be cut short before step H only when the outcome of ρ_φ is already determined. A full-horizon safety operator captures the minimum predicate value observed over the episode, and the top-level conjunction takes the minimum across operators, so once a safety predicate such as avoid hitting obstacles is violated, no continuation of the trajectory can restore $\rho_\varphi(\xi) \geq 0$. Terminating on violation therefore preserves the sign of the robustness, which is the quantity certification depends on, and it is the only truncation used.

Discount factor γ. The discount factor is set close to 1 (e.g., $\gamma = 0.99$) to keep returns near-undiscounted over the temporal windows of φ. The motivation is the robustness semantics, the min and max aggregations weight all time steps within their interval equally. So a discount significantly below 1 would make PPO undervalue predicate satisfaction near the end of a temporal window, misaligning the RL and STL objectives. Over a finite horizon, returns remain bounded regardless.

The decision process $\mathcal{M}_\varphi$ is now fully specified, a policy acting on $\mathcal{S}$ has the temporal context required by φ, and the reward (3.23) scores its behaviour. What Requirement 1 of Problem 3.1.1 still needs is the optimization procedure that finds such a policy.

3.3 Policy Optimization

This section addresses policy optimization using PPO on the MDP $\mathcal{M}_\varphi$. The augmented state space and the rewards resolve the structural obstacles identified in Section 3.2. The per-step reward decomposition for PPO, the advantage estimation, and the network architecture are explained in this section.

3.3.1 Training Objective

The policy π_θ is trained by maximizing the PPO clipped surrogate objective (2.12) over trajectories collected by rolling out π_θ in $\mathcal{M}_\varphi$. Each episode starts from an initial condition drawn from the current spawn distribution and runs for the fixed horizon T_φ. Because the STL robustness $\rho_\varphi(\xi)$ is defined only over a complete trajectory, it cannot serve as a per-step training signal. The reward (3.23) is therefore distributed over the episode as a dense auxiliary component emitted at every step and the terminal robustness $\rho_\varphi(\xi)$ appended at episode end,

$$r_t \;=\; r_{\text{aux}}(s_t, a_t) \quad \text{for } t < H, \qquad r_H \;=\; r_{\text{aux}}(s_H, a_H) \;+\; \alpha\, \rho_\varphi(\xi), \tag{3.25}$$

where $\alpha > 0$ is the scaling coefficient of (3.23). Summing (3.25) over the episode recovers (3.23) exactly. The auxiliary reward r_{aux} encodes the sub-goals implicit in φ, rewarding progress toward the goal, penalizing proximity to obstacles and arena boundaries, and rewarding a bonus on entering the goal for the first time. These terms do not directly measure ρ_φ, but they guide the policy toward trajectories for which $\rho_\varphi > 0$ is likely.

Maximizing the expected cumulative return

$$J(\theta) \;=\; \mathbb{E}_{\xi \sim \pi_\theta}\left[\sum_{t=0}^{H} r_t\right] \tag{3.26}$$

is a proxy for maximizing $\mathbb{E}[\rho_\varphi(\xi)]$. The dense terms supply a credit-assignment signal across the long horizon T_φ, while the terminal $\alpha\, \rho_\varphi(\xi)$ ties the objective to the true specification measure at the end of the episode, subject to the discounting set out in Remark 3.3.1.

Remark (Terminal robustness reward). The robustness term $\alpha\,\rho_\varphi(\xi)$ is added only at the terminal step of the trajectory. Since PPO optimizes the discounted return, this terminal reward is discounted when computing returns and advantage estimates for earlier states. Therefore, its contribution decreases for actions that occur further from the end of the episode.

3.3.2 Advantage Estimation

Optimizing (3.26) requires an estimate of how much each action contributes to the return. PPO uses an actor-critic architecture consisting of a policy network (actor) $\pi_\theta(a_t|s_t)$ and a value network (critic) $V_\psi(s_t)$. The actor defines the current control policy and selects actions according to the state s_t, while the critic estimates the expected discounted return from that state. During training, the actor interacts with the environment to collect trajectories, and the resulting rewards are used to update both networks. The critic provides a baseline for the expected performance of a state, allowing the actor update to focus on whether the selected action produced a better or worse outcome than expected.

The advantage function measures the relative quality of an action with respect to the value predicted by the critic. A direct estimate based on complete trajectory returns can have high variance, since it depends on all future rewards after the selected action. PPO therefore uses Generalized Advantage Estimation (GAE) [37], which combines information from multiple future steps to obtain a more stable estimate:

$$\hat{A}_t = \sum_{l=0}^{H-t-1} (\gamma\lambda_{\mathrm{GAE}})^l e_{t+l}^{\mathrm{TD}}, \tag{3.27}$$

where

$$e_t^{\mathrm{TD}} = r_t + \gamma V_\psi(s_{t+1}) - V_\psi(s_t) \tag{3.28}$$

is the temporal-difference error, representing the difference between the observed one-step reward and the return predicted by the critic. The discount factor γ determines the contribution of future rewards, while λ_{GAE} controls the balance between relying on the critic prediction and using longer-horizon return information.

The estimated advantages are then used in the PPO clipped objective to update the actor parameters θ, encouraging actions with positive advantage values and reducing the probability of actions with negative advantages. In parallel, the critic parameters ψ are updated by minimizing the prediction error between the estimated returns and the value function output. This alternating optimization allows the actor to improve the policy while the critic continuously adapts its estimate of future performance. The values of γ and λ_{GAE} used in this thesis are reported in Chapter 4.

3.3.3 Network Architecture

Both actor and critic are function approximators, and their parameterization completes the specification of the training procedure. The policy π_θ and value function V_ψ are parameterized as separate multi-layer perceptrons (MLP) with no shared parameters. Both networks receive the full augmented state s_t as input. The policy network outputs the parameters of a Gaussian distribution over actions, whose mean $\nu_\theta(s_t)$ is the deterministic action of the mean policy $\bar{\pi}_\theta$ in (3.8).

Algorithm 1 STL-Guided PPO Training on $\mathcal{M}_\varphi$

Require: MDP $\mathcal{M}_\varphi$, number of iterations N_{iter}, rollout batch size B, number of minibatches M, PPO hyperparameters $(c_{\text{clip}}, \lambda_{\text{GAE}}, N_{\text{epoch}})$
Ensure: Trained policy π_θ

1: Initialize policy π_θ and value function V_ψ with random weights
2: Initialize running state normalizer
3: **for** iteration $1, \ldots, N_{\text{iter}}$ **do**
4: Collect a rollout buffer $\mathcal{B}$ of B transitions in $\mathcal{M}_\varphi$ under π_θ across N_{env} parallel environments
5: At each step t, observe $s_t = [x_t^{\text{obs}} \mid s_\varphi(t)]$, return $r_{\text{aux}}(s_t, a_t)$
6: At each terminal step, append $\alpha\, \rho_\varphi(\xi)$ to r_H ▷ Eq. (3.25)
7: Compute advantages $\hat{A}_t$ via GAE (3.27)
8: Compute returns $\hat{G}_t = \hat{A}_t + V_\psi(s_t)$
9: **for** PPO epoch $e = 1, \ldots, N_{\text{epoch}}$ **do**
10: Partition $\mathcal{B}$ into M minibatches $\mathcal{B}_1, \ldots, \mathcal{B}_M$
11: **for** each minibatch $\mathcal{B}_b$, $b = 1, \ldots, M$ **do**
12: Update θ by maximizing the clipped surrogate (2.12) on $\mathcal{B}_b$ with advantages $\hat{A}_t$
13: Update ψ by minimizing $\frac{1}{|\mathcal{B}_b|} \sum_{t \in \mathcal{B}_b} \left(V_\psi(s_t) - \hat{G}_t\right)^2$
14: **end for**
15: **end for**
16: Update running state normalizer with collected data
17: Discard $\mathcal{B}$
18: **end for**
19: **return** π_θ

Remark (Timer augmentation and value estimation). The critic receives the same augmented state s_t as the actor, including the timer signals. These signals provide information about the temporal progress of the STL specification and allow the critic to distinguish states that have different remaining task horizons.

Algorithm 1 performs $N_{\text{epoch}} \cdot M$ gradient steps per iteration and is stated as a generic training call: Round 0 (Algorithm 2, Line 2) invokes it on initial conditions drawn from μ_0 with the plain PPO loss, and each refinement round (Section 3.5) invokes it again on $\tilde{\mu}_0^{(k)}$ with the EWC or PNN augmented loss substituted for the actor update. The values of B, M, N_{epoch}, and N_{env} are reported in Chapter 4.

Training produces a policy that performs well on average, which is what the objective (3.26) rewards. The next section determines whether its violation rate meets the tolerance (E, δ) of Problem 3.1.1 and where any remaining violations occur within $\mathcal{X}_0$.

3.4 Statistical Validation and Counterexample Extraction

This section certifies the trained policy with a finite-sample, distribution-free guarantee and, when certification fails, identifies the region of $\mathcal{X}_0$ where the policy performs worst. A single

calibration set serves both purposes, and the whole procedure is one pass with five steps, each of which corresponds to one panel of Figure 3.3: roll the policy out from a fresh calibration set and run the global PAC test (Figure 3.3a); if the test fails, fit a smooth surrogate to the observed robustness values (Figure 3.3b); score the one-parameter family of candidate regions that the surrogate induces and pick a threshold (Figure 3.3c); candidate regions for different thresholds and the selected threshold (Figure 3.3d); and extract the failure cluster at the selected threshold, turning it into a biased sampling distribution for the next synthesis round (Figure 3.3e). The figure 3.3 has the full counter-example region extraction process, the subsections below give the details of each step.

3.4.1 PAC Certification

Certification at round k is evaluated on the fixed mean policy $\bar{\pi}_{\theta_k}$ of (3.8), and on a calibration set drawn for that round,

$$\mathcal{D}_{\text{cal}}^{(k)} \;=\; \{x_0^{(1)}, \ldots, x_0^{(N_{cal})}\} \overset{\text{i.i.d.}}{\sim} \mu_0, \qquad N_{cal} \;=\; |\mathcal{D}_{\text{cal}}^{(k)}|, \tag{3.29}$$

each $x_0^{(i)} \in \mathcal{X}_0$, following the calibration set of (2.14). The closed-loop trajectory from each element is simulated under $\bar{\pi}_{\theta_k}$, giving the list of robustness values

$$\big[\rho_\varphi(\xi^{\bar{\pi}_\theta}_{x_0^{(i)}})\big]_{i=1}^{N_{cal}} \tag{3.30}$$

plotted in Figure 3.3a, where each rollout appears at its starting pose and is coloured by the sign of its robustness. The size N_{cal} is a per-setup choice, reported in Chapter 5. Counting the negative entries gives

$$n_{\text{viol}} \;=\; \big|\{x_0^{(i)} \in \mathcal{D}_{\text{cal}}^{(k)} \;\mid\; \rho_\varphi(\xi^{\bar{\pi}_\theta}_{x_0^{(i)}}) < 0\}\big|, \tag{3.31}$$

and the policy is certified whenever the Clopper-Pearson condition (2.19) of Section 2.3 holds for this count, that is whenever $\epsilon_{\text{CP}}(n_{\text{viol}}, N_{cal}, \delta) \le E$. This implies $\mathcal{V}(\bar{\pi}_\theta) \le E$ at confidence $1-\delta$, which is the guarantee (3.11) required by Problem 3.1.1. If the test passes, the framework returns $\bar{\pi}_\theta$ and terminates. Otherwise the N_{cal} rollout pairs $\{(x_0^{(i)}, \rho_\varphi(\xi^{\bar{\pi}_\theta}_{x_0^{(i)}}))\}_{i=1}^{N_{cal}}$ pass downstream to localise the failure region.

3.4.2 Robustness Surrogate Landscape and Candidate Region Family

Figure 3.3a shows the robustness scores $\big[\rho_\varphi(\xi^{\bar{\pi}_\theta}_{x_0^{(i)}})\big]_{i=1}^{N_{cal}}$ plotted across the arena. The failures cluster together visibly, but they remain isolated points. Since resampling from a finite point set would collapse training onto those exact points, a continuous region is needed instead.

To obtain this, a surrogate function $\hat{\rho}\colon \mathcal{X}_0 \to \mathbb{R}$ is fit to all N_{cal} observed pairs using radial basis function regression,

$$(\mathbf{Q} + \zeta_{\text{ridge}}\mathbf{I})\,\boldsymbol{\kappa} = \big[\rho_\varphi(\xi^{\bar{\pi}_\theta}_{x_0^{(i)}})\big]_{i=1}^{N_{cal}}, \qquad \hat{\rho}(\chi) = \sum_{j=1}^{N_{cal}} \kappa_j \exp\!\Big(-\tfrac{\|\chi - x_0^{(j)}\|^2}{2\kappa_{\text{ker}}^2}\Big), \tag{3.32}$$

where each weight κ_j is chosen so that $\hat{\rho}$ matches the observed robustness scores. The Gram matrix $\mathbf{Q}_{ij} = \exp(-\|x_0^{(i)} - x_0^{(j)}\|^2/2\kappa_{\text{ker}}^2)$ measures how similar two points from $[\rho_\varphi(\xi^{\bar{\pi}_\theta}_{x_0^{(i)}})]_{i=1}^{N_{cal}}$ are, the bandwidth κ_{ker} controls how broadly each discrete point influences its neighbourhood, and the ridge term $\zeta_{\text{ridge}}\mathbf{I}$ prevents the fit from overfitting to noise in the scores. The free variable $\chi \in \mathcal{X}_0$ is any query point at which $\hat{\rho}$ can be evaluated, not just the N_{cal} simulated ones. As Figure 3.3b shows, the surrogate places a red band around the lower obstacle that covers the failure cluster from Figure 3.3a and fills in the gaps between individual failures smoothly, giving a continuous region to resample from in the next step.

For any threshold $\eta \in \mathbb{R}$, the candidate failure region is the sub-level set

$$\hat{R}(\eta) = \{\chi \in \mathcal{X}_0 \mid \hat{\rho}(\chi) \leq \eta\}, \tag{3.33}$$

three instances of which are drawn in Figure 3.3d. Decreasing η shrinks $\hat{R}(\eta)$ toward the deepest failure cluster and eventually toward a single point, increasing it expands the region until it covers a lot from $\mathcal{X}_0$ and stops being informative. The next task is to select one threshold η^* from this family.

3.4.3 Threshold Selection via a Failure-Concentration Score

A useful counterexample region should contain many failures while being supported by enough observed rollouts. These two objectives conflict, as shown in Figure 3.3d: a small threshold may contain mostly failures but very few samples, while a large threshold contains more samples but also more successful rollouts. An intermediate threshold is therefore required.

For a fixed η, the calibration set gives the number of rollouts inside the region and the number of failures among them,

$$n_{\text{in}}(\eta) = \big|\{i \mid x_0^{(i)} \in \hat{R}(\eta)\}\big|, \qquad n_{\text{fail}}(\eta) = \big|\{i \mid x_0^{(i)} \in \hat{R}(\eta),\ \rho_\varphi(\xi^{\bar{\pi}_\theta}_{x_0^{(i)}}) < 0\}\big|. \tag{3.34}$$

Their ratio estimates the failure probability conditional on starting inside the region,

$$p(\eta) = \mathbb{P}_{x_0 \sim \mu_0}\big[\rho_\varphi(\xi^{\bar{\pi}_\theta}_{x_0}) < 0 \mid x_0 \in \hat{R}(\eta)\big], \tag{3.35}$$

but the raw ratio is unreliable when the region contains few samples: two failures out of three rollouts and two hundred out of three hundred both give $p = 2/3$, and only the second is evidence of clear violation region. The regions are therefore ranked by a lower confidence bound on $p(\eta)$ rather than by the ratio itself. Using the same Clopper-Pearson construction as the global test of Section 3.4.1, now in its lower form,

$$\underline{p}(\eta) \;=\; \begin{cases} 0, & n_{\text{fail}}(\eta) = 0, \\ \text{Beta}^{-1}\big(\delta_{\text{reg}};\ n_{\text{fail}}(\eta),\ n_{\text{in}}(\eta) - n_{\text{fail}}(\eta) + 1\big), & \text{otherwise}, \end{cases} \tag{3.36}$$

where $\text{Beta}^{-1}(\,\cdot\,; a, b)$ is the inverse regularised incomplete beta function and $\delta_{\text{reg}} \in (0, 1)$ is the confidence level of the regional bound. The bound tightens toward $n_{\text{fail}}/n_{\text{in}}$ as the region accumulates rollouts and collapses toward zero when it does not, so a small region has to be almost entirely failures before it can outrank a large one.

Ranking by $\underline{p}(\eta)$ alone still favours whichever region has the highest bounded failure probability, which tends to be a small one tightly wrapped around a cluster. Such a region is a poor resampling target as it supplies few distinct initial conditions and the next round overfits to them. A support factor is therefore included,

$$\eta^* = \underset{\eta \in \{\hat{\rho}(x_0^{(i)})\}_{i=1}^{N_{cal}}}{\operatorname{argmax}} \underline{p}(\eta)\sqrt{n_{\text{in}}(\eta)}, \tag{3.37}$$

whose square root grows with the number of enclosed rollouts but at a decreasing rate, so a region cannot be selected by being large alone. The optimum therefore sits at an interior threshold, as Figure 3.3c shows. Since n_{in} and n_{fail} are computed by checking which rollouts fall inside $\hat{R}(\eta)$, and a rollout enters or leaves the region only when η equals its surrogate value $\hat{\rho}(x_0^{(i)})$, it suffices to evaluate the score only at those N_{cal} values rather than searching over all real-valued thresholds. Sorting those values and sweeping once gives $O(N_{cal} \log N_{cal})$, with one evaluation of (3.36) per candidate.

Remark (Status of the regional bound). The quantity $\underline{p}(\eta)$ is used only to rank candidate regions and select a retraining target. It is not a certificate and it does not need to be one. The certified quantity remains the global violation probability $V(\bar{\pi}_\theta)$, computed on a fresh calibration set at each round.

3.4.4 Counterexample Region Extraction

The threshold η^* fixes one member of the family (3.33), drawn as the dashed contour in Figure 3.3d. Because $\hat{\rho}$ is defined everywhere on $\mathcal{X}_0$, this contour is a continuous area covering the observed failures and the surrounding region where the surrogate predicts similarly low robustness.

The contour is not necessarily a single connected area. The surrogate can fall below η^* in several separate regions, each corresponding to a distinct cluster of failures. Training on their union would spread updates across all clusters rather than concentrating them on the dominant one. The connected region enclosing the most observed failures is therefore selected as $\hat{R}_k$, shown as the orange area in Figure 3.3e. Regions containing fewer than $n_{\min}$ observed rollouts are discarded, since the bound (3.36) is uninformative at small sample sizes. Implementation details are documented in Section 4.4.2.

3.4.5 Biased Resampling

The region $\hat{R}_k$ is where the policy currently fails. Round $k+1$ trains on a distribution biased toward $\hat{R}_k$. With probability β the starting configuration is drawn uniformly from $\hat{R}_k$, and with probability $1-\beta$ it is drawn from μ_0 as usual,

$$\tilde{\mu}_0^{(k)} \;=\; \beta\,\text{Unif}(\hat{R}_k) \;+\; (1-\beta)\,\mu_0, \qquad \beta \in (0,1]. \tag{3.38}$$

The first term concentrates training episodes on the failure region; the second keeps the policy exposed to the full initial condition space so that behaviour outside $\hat{R}_k$ is not degraded. Uniform sampling over $\hat{R}_k$ is possible even though μ_0 is unknown, since $\hat{R}_k$ is a region that can be sampled directly from. The value of β is reported in Chapter 4.

The biased distribution governs training only. Validation in round $k+1$ draws a fresh calibration set $\mathcal{D}_{\text{cal}}^{(k+1)}$ from μ_0, independent of $\mathcal{D}_{\text{cal}}^{(k)}$. This separation is what keeps the PAC certificate (3.11) valid across rounds, the certificate is a statement about μ_0 so calibration samples must come from μ_0 rather than $\tilde{\mu}_0^{(k)}$, and reusing the previous round's rollouts would violate the independence assumption of the binomial bound since the updated policy was modified in direct response to them.

3.5 Continual Learning for Iterative Refinement

The biased distribution $\tilde{\mu}_0^{(k)}$ defines how initial conditions are sampled in round $k+1$. The policy must then be updated to improve performance on $\hat{R}_k$ while retaining the performance achieved on previously certified regions. This section describes how this is done.

Naive fine-tuning $\theta_{k+1} = \arg\max_\theta J_{\hat{R}_k}(\theta)$, where $J_{\hat{R}_k}$ is the objective (3.26) evaluated on initial conditions restricted to $\hat{R}_k$, optimizes for the new failure region but provides no constraint on specification satisfaction elsewhere. What the continual learning literature calls "forgetting a previous task" corresponds, in this framework, to *regression of previously certified STL coverage*. This connection motivates the continual learning mechanisms introduced in Sections 2.4.1 and 2.4.2, which are adapted to the refinement loop below.

3.5.1 Elastic Weight Consolidation

Applying EWC (Section 2.4.1) to the iterative refinement setting, the training loss at round k combines the PPO loss $\mathcal{L}_{\text{PPO}}(\theta) = -L(\theta)$ evaluated on the $\tilde{\mu}_0^{(k)}$ distribution with the Fisher-weighted quadratic penalty of (2.21), so that both terms are minimized jointly. The key adaptation specific to this thesis is online Fisher accumulation with exponential decay,

$$F^{(k)} \;=\; \gamma_F \, F^{(k-1)} \;+\; \hat{F}(\theta_k), \qquad F^{(-1)} = 0, \tag{3.39}$$

where $\hat{F}(\theta_k)$ is the diagonal Fisher estimate at the round-k parameters, computed by the estimator of Section 4.5.2, and $\gamma_F \in [0, 1)$ is a decay factor distinct from the MDP discount γ. Because this framework revisits the full initial condition space at each validation round, the relative importance of parameters shifts as new failure regions are addressed, the decay bounds the Fisher matrix and lets the penalty landscape track that shift while providing some importance to Round 0 weights.

The penalty is a soft constraint, deviations are penalized but no formal guarantee of zero forgetting. Residual forgetting must be measured empirically at each round.

3.5.2 Progressive Neural Networks

Applying PNN (Section 2.4.2), at each refinement round k a new column is initialized with random weights and trained on the current failure region $\hat{R}_k$. The lateral connections to each new column are always taken from the initial column ($k = 0$). This provides a fixed source of previously learned features while allowing the new column to specialise on the refinement

Table 3.1: Comparison of continual learning approaches for iterative policy refinement.

Aspect	**EWC**	**PNN**
Forgetting guarantee	Soft (controlled by λ_{EWC}, γ_F)	Hard (frozen columns)
Network size	Fixed $O(\|\theta\|)$	Grows +1 column per round
Knowledge transfer	Implicit (shared weights)	Explicit (lateral adapters)
Hyperparameters	λ_{EWC}, γ_F	Lateral adapter strength
Deployment model	Single policy	Multi-column with routing

region $\hat{R}_k$ without modifying earlier columns. At deployment, starting configurations in $\hat{R}_k$ are routed to the corresponding column, and all others to column 0. The routing rule is explained in detail in Section 4.5.3.

Because prior columns are frozen, PNN provides theoretically zero forgetting. The cost is linear architecture growth per round.

3.5.3 Comparison and Trade-offs

Neither approach is strictly better than the other. EWC maintains a fixed-size, single-policy architecture, but can introduce some forgetting that must be evaluated after each refinement round. PNN avoids forgetting structurally, but its architecture grows with each new round. For the round counts observed in Chapter 5, at most ten, EWC remains practical with empirically low forgetting. For hard zero-forgetting requirements, PNN is preferred. EWC also requires careful tuning of its regularisation strength, whereas PNN needs no such tuning since forgetting is prevented by construction. Both are carried forward as selectable branches of the refinement stage in the complete algorithm. Table 3.1 compares the two approaches in terms of their relevance to the refinement loop.

3.6 Complete Algorithm

Having developed each component, this section assembles them into the complete algorithm and analyzes its properties.

3.6.1 Algorithm Description

Algorithm 2 presents the full synthesis-validation-refinement pipeline.

Lines 1 and 2 execute Round 0, producing a policy trained on initial conditions drawn from μ_0. The main loop (Lines 3 to 23) alternates between certification and refinement, drawing a fresh calibration set each round (Line 4) so that the independence required by the global bound (Line 8) is respected. The threshold η^* is found exactly by enumeration over the N_{cal} candidate surrogate values (Lines 13 and 14), and the resulting sub-level set is partitioned spatially (Line 15) before being selected as a resampling target (Line 16). The continual learning branch (Lines 21 and 22) fine-tunes the policy while reducing or preventing catastrophic forgetting.

3.6.2 Complexity Analysis

Training requires $O(N_{\text{iter}} \cdot B \cdot H)$ simulator steps per round for PPO. Validation requires $O(N_{cal} \cdot H)$ simulator calls per round. Counterexample-region extraction costs $O(N_{cal}^3)$ for fitting the RBF surrogate and $O(N_{cal} \log N_{cal})$ for the η^* selection sweep. Training and validation both execute on the same pool of N_{env} parallel environments (Chapter 4), so wall-clock time scales as

$$O\left(K_{\max} \cdot \left(\frac{N_{cal} \cdot H}{N_{\text{env}}} + \frac{N_{\text{iter}} \cdot B \cdot H}{N_{\text{env}}}\right)\right). \tag{3.40}$$

For validation, all N_{cal} calibration rollouts are executed in parallel across the N_{env} environments. When $N_{cal} \leq N_{\text{env}}$, the entire calibration set is evaluated in a single pass, when $N_{cal} > N_{\text{env}}$, validation requires $\lceil N_{cal}/N_{\text{env}} \rceil$ passes. The continual-learning step modifies this Round 0 cost:

- **EWC Fisher estimation.** After each task's training phase, the diagonal Fisher $\hat{F}^{(k)}$ is computed over a held-out batch of N_F rollouts, costing $O(N_F \cdot H \cdot |\theta|)$ simulator steps and one backward pass per sample. This cost is paid each round.
- **PNN column training.** Each new round appends a fresh column to the progressive network and freezes all prior columns. Training the new column costs $O(N_{\text{iter}} \cdot B \cdot H)$ per round as before, but lateral connections from the frozen source column introduce a fixed additional $O(|\theta_{\text{col}}|)$ overhead per gradient step, replacing H with $H + |\theta_{\text{col}}|$ in (3.40). In practice PNN takes longer to train as more columns are added, possibly due to the increase in network size.

3.6.3 Convergence and Failure Modes

The framework does not guarantee convergence of the policy to an optimal policy, nor does it guarantee that the resulting policy will achieve a sufficiently tight statistical certificate. Three identifiable failure modes are possible. The first is *insufficient policy capacity*, in which the network architecture lacks the representational power to express a satisfactory policy regardless of how it is trained. The second is *insufficient training budget*, in which the total iteration count N_{iter} is too small for the policy to converge to a good solution. The third is *counterexample region too diffuse to localise*, in which the surrogate surface is flat enough that either no sub-level set passes the minimum-support filter, or the selected region is so small that the biased rollouts it contributes are negligible relative to the full training batch. In all three cases the loop returns the current policy with an empirical violation report after $K_{\max}$ rounds.

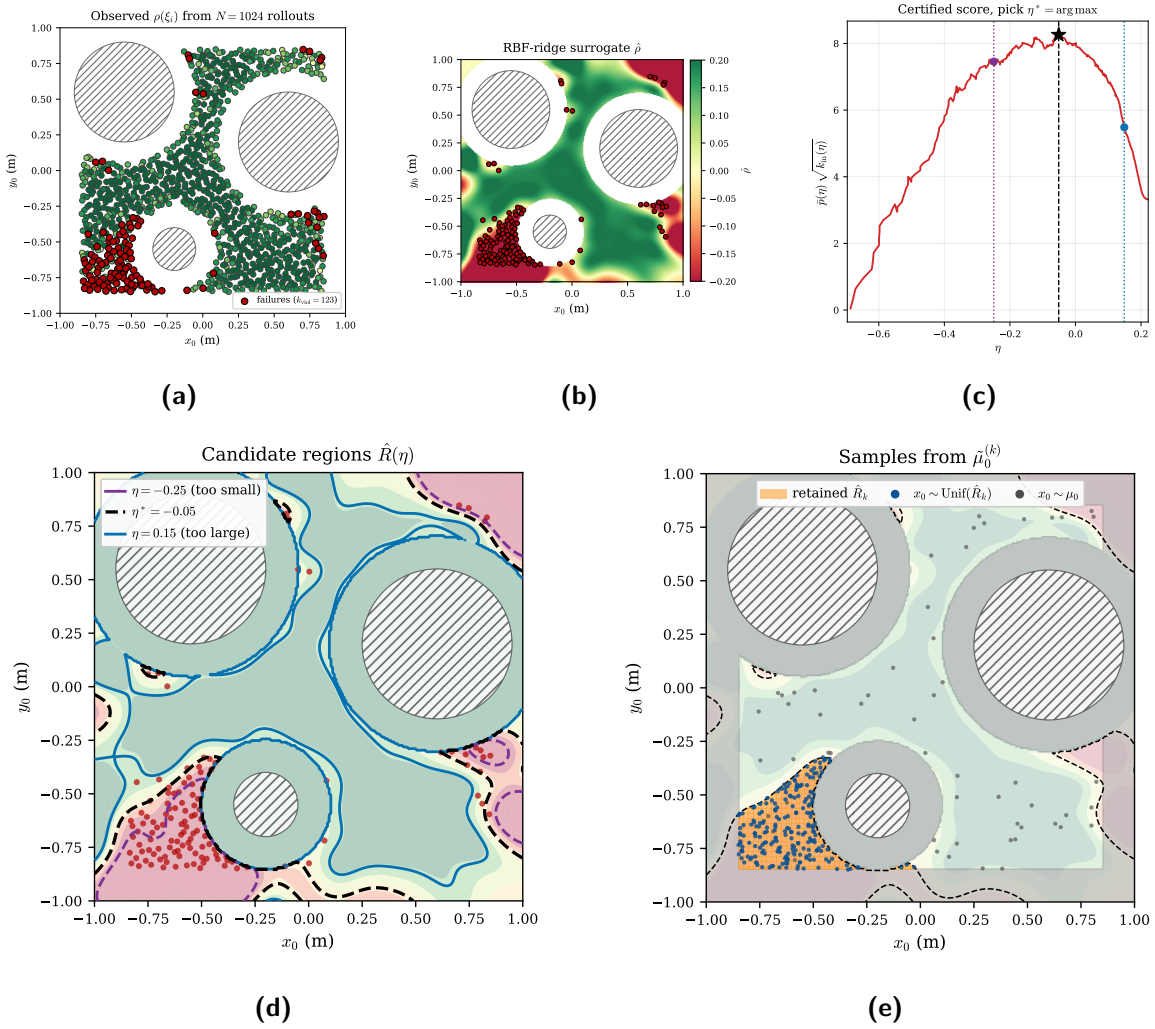


Figure 3.3: Representative figure of five steps of the certification pipeline. (a) Robustness $\rho_\varphi(\xi^{\bar{\pi}_\theta}_{x_0^{(i)}})$ observed from a calibration set of size $|\mathcal{D}^{(k)}_{\text{cal}}| = 1024$, green satisfying and red violating; the violation count feeds the PAC test of Section 3.4.1. (b) The RBF-ridge surrogate $\hat{\rho}$ of Section 3.4.2 fit to those values, turning the scatter into a continuous field. (c) The failure-concentration score of Section 3.4.3 swept over candidate thresholds, with the maximiser η^* marked. (d) The sub-level sets $\hat{R}(\eta)$ at those three thresholds, illustrating why an interior optimum exists. (e) The retained component $\hat{R}_k$ (orange) and a draw from the biased distribution $\tilde{\mu}_0^{(k)}$ of Section 3.4.5, with samples concentrated inside $\hat{R}_k$ and the remainder spread over μ_0.

Algorithm 2 Counterexample-Driven Refinement of RL Controllers

Require: Simulator f, STL specification φ, true initial condition distribution μ_0 with support $\mathcal{X}_0$

Require: Violation tolerance E, confidence δ, calibration set size N_{cal}, max refinement rounds $K_{\max}$

Ensure: Certified policy $\bar{\pi}_{\theta_k}$, or best-effort policy with violation report

Round 0 (training on μ_0)

Initialize policy π_{θ_0} and value function V_{ψ_0}

Train π_{θ_0} via Algorithm 1 on $\mathcal{M}_\varphi$ with initial conditions drawn from μ_0 ▷ §3.3

Iterative Certification Loop

for $k = 0, 1, \ldots, K_{\max}$ **do**

Stage 2a (Global PAC certification)

Draw a fresh calibration set $\mathcal{D}_{\text{cal}}^{(k)} = \{x_0^{(i)}\}_{i=1}^{N_{cal}} \overset{\text{i.i.d.}}{\sim} \mu_0$ ▷ Eq. (3.29)

Roll out the mean policy $\bar{\pi}_{\theta_k}$ from each $x_0^{(i)} \in \mathcal{D}_{\text{cal}}^{(k)}$ ▷ Eq. (3.8)

Collect $\left[\rho_\varphi(\xi_{x_0^{(i)}}^{\bar{\pi}_{\theta_k}})\right]_{i=1}^{N_{cal}}$ ▷ Eq. (3.30)

$n_{\text{viol}} \leftarrow |\{i \mid \rho_\varphi(\xi_{x_0^{(i)}}^{\bar{\pi}_{\theta_k}}) < 0\}|$ ▷ Eq. (3.31)

$\epsilon \leftarrow \epsilon_{\text{CP}}(n_{\text{viol}}, N_{cal}, \delta)$ ▷ Eq. (2.18)

if $\epsilon \leq E$ **then**

return $\bar{\pi}_{\theta_k}$ with (E, δ)-PAC certificate ▷ Eq. (2.19)

end if

Stage 2b (Counterexample region extraction)

Fit RBF surrogate $\hat{\rho}$ to $\{(x_0^{(i)}, \rho_\varphi(\xi_{x_0^{(i)}}^{\bar{\pi}_{\theta_k}}))\}_{i=1}^{N_{cal}}$ ▷ Eq. (3.32)

Evaluate $\underline{p}(\eta) \cdot \sqrt{n_{\text{in}}(\eta)}$ at each $\eta \in \{\hat{\rho}(x_0^{(i)})\}_{i=1}^{N_{cal}}$ ▷ Eqs. (3.36), (3.37)

$\eta^* \leftarrow \hat{\rho}(x_0^{(j^*)})$ with j^* the maximising index

Partition $\hat{R}(\eta^*)$ into connected components

Filter by minimum support, select dominant failure component as $\hat{R}_k$

if no component survives filter **then**

return $\bar{\pi}_{\theta_k}$ with empirical violation report ▷ uniform robustness termination

end if

Stage 3 (Refinement with continual learning)

Construct $\tilde{\mu}_0^{(k)}$ via (3.38)

if EWC: update $F^{(k)}$ via (3.39); train $\pi_{\theta_{k+1}}$ with loss (2.21) on $\tilde{\mu}_0^{(k)}$

if PNN: add column $k{+}1$; freeze columns $0, \ldots, k$; train via (2.22) on $\tilde{\mu}_0^{(k)}$

end for

return $\bar{\pi}_{\theta_{K_{\max}}}$ with violation report ▷ certification not achieved

Chapter 4

Implementation

Chapter 3 developed the training procedure on MDP $\mathcal{M}_\varphi$, a PAC certification test, and a counterexample-driven refinement loop. This chapter describes how each step is realised in simulation. The sections below follow the same order as Chapter 3, covering the simulation platform (Section 4.1), the MDP components (Section 4.2), the training loop (Section 4.3), the certification and counterexample-extraction pipeline (Section 4.4), and the continual-learning approaches (Section 4.5). Numerical values that depend on the robot or the specification are reported in Chapter 5 and Appendix B.

4.1 Simulation Platform

Training and validation both run in NVIDIA Isaac Lab, built on IsaacSim and the PhysX 5 physics engine. Isaac Lab provides a `DirectRLEnv` interface that launches N_{env} independent copies of the environment and steps them in a single GPU kernel call, so that environment stepping, timer updates, robustness evaluation, and policy inference all run in parallel across instances. Only logging at episode boundaries leaves this parallel path. The number of instances is fixed at $N_{\text{env}} = 1024$, the largest that fits in GPU memory on the available hardware, and is held constant across every experiment so that no comparison between methods or rounds is confounded by a change in parallelism.

Training collects one transition from every environment per step. The policy runs at $\Delta t = 0.01\,\text{s}$ with one physics step per control step. With the simulation platform established, the next section describes how each component of the MDP tuple (3.12) is realised within it.

4.2 Realising the MDP

Section 3.2 defined the MDP tuple (3.12) abstractly. Of its five components, two of them need no formal definition. The action space $\mathcal{A} = \mathcal{U}$ follows the actuator limits documented

Source code is available at GitHub Repository.

in Chapter 5, and the transition function is handled by the simulator, which propagates the robot dynamics and its interaction with the environment to produce the successor state from any given state-action pair. The remaining three components are described below.

4.2.1 State Space Construction

At every step the augmented state $s_t = [x_t^{\text{obs}} \mid s_\varphi(t)]$ is computed by concatenating the observation vector x_t^{obs} with the timer vector $s_\varphi(t)$ and passed to the policy network.

The observation contains proprioceptive quantities (position, heading, body velocities) and goal-relative quantities (distance and bearing to the active goal). In setups with tighter obstacle passages, the displacement vector to the nearest obstacle surface is appended. The exact layout per setup is documented in Appendix B.

Each timer is a scalar maintained per environment, updated at every step and reset at the start of each episode. The always fraction $\tau_{i,t}^{\text{always}}$ is updated by incrementing by Δt when $\rho_{\phi_i}(x_t) \geq 0$ and resetting it to zero otherwise, it is frozen at $T_{a,i}$ once the inner $\Box$ operator is satisfied. The eventually fraction $\tau_{i,t}^{\text{eventually}}$ advances by Δt at every step. The latching and reset behaviour defined in Section 3.2.1 are reproduced exactly.

Remark (Fixed-window always fraction not exercised). All specifications in the experiments use the nested $\Diamond\Box$ form, since every $\Diamond$ subformula in the tested specifications contains an inner $\Box$ operator. The assumption of Proposition 3.2.1 is therefore met by construction, no binary satisfaction flag is needed.

4.2.2 Reward and Robustness

The reward signal has two parts. A step-level auxiliary term $r_{\text{aux}}(s_t, a_t)$ is returned at every step, constructed from the same geometric quantities that appear in the predicates of φ so that the shaped signal stays aligned with the specification.

To make the idea of auxiliary rewards concrete, consider a reach-and-stay requirement. From the first step the robot gets a proximity penalty whenever it approaches obstacles or arena walls, discouraging unsafe regions. A distance-progress term rewards reduction in distance to goal at every step, scaled by the elapsed deadline fraction $\tau_{1,t}^{\text{eventually}}$ so that the urgency signal increases as time runs out. On first entry into the goal region the robot receives a one-time bonus, after which the reward structure shifts to a small stay reward that penalises leaving the goal, encoding the inner stay at goal $\Box_{[0,T_a]}$ obligation. This general strategy of shaping rewards to reflect the temporal structure of the specification is inspired by prior work on STL-guided reinforcement learning [7, 9]. A terminal robustness term $\alpha\,\rho_\varphi(\xi)$ is appended at episode end. The exact auxiliary terms and their weights are given per setup in Appendix B.

The robustness values for the $\Diamond$ and $\Box$ operators are computed in parallel and stored in tensors. At termination, the frozen per-operator values are combined according to the Boolean structure of φ via (2.4)–(2.6), giving $\rho_\varphi(\xi)$ exactly.

4.2.3 Initial State, Horizon, and Discount

The initial-condition distribution μ_0 is never represented explicitly. Instead, a rejection sampler draws a candidate position uniformly from the arena, a heading uniformly from $[0, 2\pi)$, and sets all body velocities to zero. Any candidate whose robot footprint overlaps an obstacle is rejected and redrawn. During refinement rounds the sampler switches to the biased mixture $\tilde{\mu}_0^{(k)}$ of (3.38), constructed in Section 4.4.3.

An episode ends on collision or after H steps. Collision falsifies a full-horizon safety operator, so by Remark 3.2.5 this early termination is sound and preserves the sign of $\rho_\varphi(\xi)$. The discount is fixed at $\gamma = 0.99$, keeping returns near-undiscounted over the temporal windows of φ (Section 3.2.5).

With every component of the MDP tuple now in place, the next section describes how Algorithm 1 is executed within the training library.

4.3 Policy Optimisation with rsl-rl

Algorithm 1 is executed through `rsl_rl` [38], an on-policy PPO library designed for GPU-parallel robotic training and shipped as the default RL backend of Isaac Lab. `rsl_rl` was selected because it natively interfaces with Isaac Lab's `DirectRLEnv` stepping loop and handles vectorised rollout collection and advantage computation across N_{env} environments without additional wrapper code. This section records the two adaptations made to the default configuration and the settings used throughout. All settings are reported in Table 4.1.

4.3.1 Training Loop

Each *iteration* is one collect-estimate-update cycle. It fills a rollout buffer with B transitions collected across all N_{env} environments. Environments reset independently on termination. An episode that ends on collision contributes no bootstrap term. An episode cut at the horizon H is bootstrapped by adding $\gamma V_\psi(s_H)$ to the reward at the final step, after which the episode is treated as terminal in the return computation.

The buffer is split into $M = 4$ minibatches and reused for $N_{\text{epoch}} = 4$ *epochs* within each iteration, giving 16 gradient steps before it is discarded.

4.3.2 Practical Implementation Choices

Two implementation choices are made when configuring `rsl_rl`, described below.

KL-adaptive learning rate. `rsl_rl` provides a learning-rate rule that rescales the learning rate after each update based on the KL divergence between the old and new policies. This rule is enabled in Round 0 to speed up initial training by adjusting the step size to the size of each policy update. For refinement rounds $k > 0$, a fixed learning rate of 3×10^{-4} is used instead, since the reduced per-round training budget (Appendix B) makes KL-adaptive rescaling unnecessary and a fixed rate produces more stable updates during fine-tuning.

Table 4.1: PPO settings used in all experiments unless stated otherwise in Chapter 5.

Parameter	Value	Notes
Discount γ	0.99	Section 3.2.5.
GAE decay λ_{GAE}	0.95	Standard bias-variance setting.
Clip c_{clip}	0.2	Library default, retained.
Entropy coefficient c_2	0.005	Exploration mostly driven by the shaped reward.
VF-loss coefficient c_1	1.0	Weight on L^{VF} in (2.13).
Learning rate	Round 0: 10^{-3}	KL-adaptive.
	Refinement: 3×10^{-4}	Fixed for rounds $k > 0$.
Parallel environments N_{env}	1024	Section 4.1.
Rollout batch size B	Appendix B	Transitions per iteration across all environments.
Minibatch reuse	$N_{\text{epoch}} = 4$, $M = 4$	16 gradient steps per iteration.
Network sizes and activations	Appendix B	Separate actor and critic MLPs.

Observation normalisation. Observations are normalised by running statistics stored with the model. The actor and critic maintain separate normalisation statistics and separate model states. This separation is exploited at round boundaries (Section 4.5.1), the value function can be reinitialised without touching the learned policy or its input normaliser.

Training produces a policy that performs well on average over μ_0. The next section determines whether its violation rate meets the tolerance E of Problem 3.1.1 and, if not, where the remaining failures concentrate.

4.4 Certification and Counterexample-Extraction Pipeline

Once a round of training finishes, the mean policy $\bar{\pi}_\theta$ of (3.8) is evaluated on a fresh set of rollouts to determine whether the PAC certification condition is met and, if not, the rollout data is used to extract a counterexample region for refinement. The three subsections below cover validation and certification, surrogate fitting with threshold selection, and region extraction with spawn injection.

4.4.1 Validation and Certification

The PAC certificate of Section 3.4.1 requires the N_{cal} calibration rollouts to be drawn i.i.d. from the true initial-condition distribution μ_0. No variance reduction or low-discrepancy scheme is used, since either would invalidate the independence assumption underlying the Clopper–Pearson bound (2.19). The calibration set size N_{cal} is set per experimental setup and reported in Chapter 5. When $N_{cal} \leq N_{\text{env}} = 1024$, the entire calibration set is evaluated in a single batched pass. When $N_{cal} > N_{\text{env}}$, validation requires $\lceil N_{cal}/N_{\text{env}} \rceil$ passes.

If $\epsilon_{\text{CP}}(n_{\text{viol}}, N_{cal}, \delta) \leq E$, the policy is (E, δ)-PAC certified and the framework terminates. Otherwise, the N_{cal} rollout pairs $\{(x_0^{(i)}, \rho_\varphi(\xi^{\bar{\pi}_\theta}_{x_0^{(i)}}))\}_{i=1}^{N_{cal}}$ pass to the counterexample-extraction

stage described next.

4.4.2 Surrogate Fitting and Threshold Selection

When the PAC test fails, the rollout pairs feed the surrogate-fitting and threshold-selection procedure of Sections 3.4.2–3.4.3.

Surrogate domain and fit. Following Section 3.4.2, the surrogate $\hat{\rho}$ is fit and queried over starting *positions* (q_x, q_y) only. The heading coordinate is excluded for the reason given in Remark 4.4.2 below. $\hat{\rho}$ is evaluated on a 121×121 raster (grid of discrete cells) covering $\mathcal{X}_0^{\text{pos}}$ using ridge-regularised RBF regression (3.32).

Obstacle masking. Raster cells inside obstacle footprints are pinned to the maximum surrogate value. This excludes them from every candidate region $\hat{R}(\eta)$ of (3.33) and prevents the RBF from extrapolating across obstacle interiors.

Threshold selection. The threshold $\eta^\star$ is chosen by the $O(N_{cal} \log N_{cal})$ sweep of (3.37), with $\underline{p}(\eta)$ from (3.36) and the concave support factor $\sqrt{n_{\text{in}}}$. Candidates are restricted to the N_{cal} surrogate values evaluated at the calibration points, since n_{in} and n_{fail} change only as η crosses one of them. The maximising index $j^\star$ identifies the rollout whose surrogate value equals $\eta^\star$. The raster is then evaluated once at the selected threshold rather than at every candidate.

Remark (Heading coordinate excluded from the surrogate). The full starting configuration $x_0 \in \mathcal{X}_0$ includes both position (q_x, q_y) and heading ϑ. Including ϑ as a third coordinate of the surrogate was considered and tested. Failure regions extracted with and without the heading coordinate were qualitatively identical on all experimental setups, because the dominant failure clusters are determined by the robot's position (the violations are from spawn points close to the obstacles) rather than by the initial heading. Heading is therefore excluded, which keeps the surrogate two-dimensional, the raster resolution fixed at 121×121.

4.4.3 Region Extraction and Spawn Injection

Region extraction. The sub-level set $\hat{R}(\eta^\star)$ is stored as a binary mask on the positional raster. A morphological closing fills isolated single-cell gaps introduced by discretisation. The mask is then split into connected components following Section 3.4.4. Any component containing fewer than $n_{\min} = 10$ observed rollouts is discarded. The surviving component with the most observed failures becomes $\hat{R}_k$.

Spawn injection. The training distribution for round $k+1$ is the mixture $\tilde{\mu}_0^{(k)}$ of (3.38), with positions drawn uniformly over $\hat{R}_k$ and headings uniformly in $[0, 2\pi)$, matching the marginal of μ_0 over headings. The mixing weight β is set separately for each continual-learning mechanism in Section 4.5.

Table 4.2: Certification pipeline settings. The target pair (E, δ) and calibration size N_{cal} are set per setup in Chapter 5.

Parameter	Value
Calibration size N_{cal}	setup-dependent
Kernel bandwidth κ_{ker}	floor 0.05 m
Ridge coefficient ζ_{ridge}	10^{-2}
Raster resolution	121×121
Minimum rollout support $n_{\min}$	10 rollouts

The calibration size N_{cal} is larger for setups with bigger arenas to fit a good RBF-surrogate. The kernel bandwidth controls how far each rollout's robustness value spreads across the surrogate surface. It is estimated from the data, with a floor of 0.05 m that prevents the surrogate from fitting noise at individual sample locations. The ridge coefficient adds a small amount of smoothing to keep the surrogate well-behaved. The raster divides the arena floor into cells smaller than the robot, so that narrow gaps between obstacles are not missed. Any connected failure region containing fewer than 10 rollouts is discarded, because so few samples do not provide enough evidence to distinguish a genuine failure cluster from bad luck in the calibration set draw.

With the failure region identified and the biased distribution constructed, the next section describes how the policy is retrained on $\hat{R}_k$ without degrading behaviour elsewhere.

4.5 Continual Learning

Training on the biased distribution $\tilde{\mu}_0^{(k)}$ requires resetting several components of the training state that have converged to the previous distribution. The task-boundary protocol below handles this before either refinement mechanism runs.

4.5.1 Task Boundary Protocol

Four adjustments are applied at every round boundary, identically for EWC and PNN so that the comparison in Chapter 5 is not confounded by initialisation differences.

Value function head. The value function head is reinitialised with random weights and given a 10-iteration warmup during which only V_ψ is updated on transitions from $\tilde{\mu}_0^{(k)}$, so the first policy gradient is computed from advantages already fitted to the new distribution. Because the actor and critic share no parameters, the reset leaves the learned policy untouched.

Optimiser state. The optimiser state is reset to clear Adam moment estimates accumulated on the previous distribution, restoring uniform step sizes.

Exploration noise. Exploration noise is restored to its initial value, since a trained policy explores too little to discover behaviour that works in a region where it currently fails. This affects π_θ only. The mean policy $\bar{\pi}_\theta$, which is the object certified, is unchanged by the noise level.

4.5.2 Elastic Weight Consolidation

EWC (Section 3.5.1) penalises deviations from the previous round's weights, weighted by the diagonal Fisher information matrix. The diagonal Fisher is estimated from a batch of stored transitions as the per-parameter mean of the squared gradient of the Gaussian mean ν_θ. This estimator differs from the exact score-function Fisher (2.20) only by a uniform factor per output dimension that is absorbed into λ_{EWC} (see Section 3.5.1).

The Fisher is accumulated online across rounds with exponential decay $\gamma_F = 0.7$ via (3.39), and the anchor $\theta_k^\star$ of (2.21) moves to the current weights at every round boundary. A static anchor pinned to round zero was considered but found to impede progress in later rounds, since the penalty pulled toward increasingly stale parameters. The penalty is applied to the policy network only.

The mixing weight of (3.38) is $\beta = 0.8$ under EWC, so a fifth of each round's starting configurations are still drawn from μ_0. Because the EWC penalty is soft, these global samples keep earlier behaviour represented in the gradient. The values of λ_{EWC}, the refinement learning rate, and the per-round iteration budget are listed in Appendix B.

4.5.3 Progressive Neural Networks

PNN (Section 3.5.2) adds a column at each refinement round and freezes all prior columns. Each new column receives lateral connections only from the initial column ($k = 0$), providing a fixed source of previously learned features while allowing the new column to specialise on the refinement region $\hat{R}_k$ without modifying earlier columns. The lateral weight matrices $C_\ell^{(m,0)}$ are initialised to small random values (scale 0.01) and have the same width as the source layer. Because freezing makes forgetting structurally impossible, no global samples from μ_0 are required and the mixing weight is $\beta = 1$.

At deployment, each starting configuration is routed by position. For $x_0 \in \hat{R}_k$, column $k+1$ produces the action, and otherwise the initial column is used. When regions from different rounds overlap, the latest column takes precedence. Routing is a pointwise lookup in the raster masks stored alongside the network weights during validation, and the lookup is a single array-indexing operation that adds negligible inference latency.

4.6 Summary

Every experiment in Chapter 5 trains on $N_{\text{env}} = 1024$ parallel environments, the timers of Section 4.2.1, the robustness computation of Section 4.2.2, the `rsl_rl` loop with the settings of Table 4.1, the certification pipeline with the settings of Table 4.2, and the task-boundary protocol of Section 4.5.1. The specification φ and its reward weights, the calibration size N_{cal}, and the per-round training budget, all alongside the results are reported in Chapter 5.

Chapter 5

Experiments and Results

This chapter validates the framework developed in Chapter 3 through simulation experiments on a TurtleBot3 Burger navigating cluttered arenas under two different STL specifications. All training and evaluation runs execute on an NVIDIA GeForce RTX 4060 GPU (8 GB VRAM), an Intel Core i7-14650HX CPU (24 threads, up to 5.0 GHz), and 16 GB of system RAM, with $N_{\text{env}} = 1024$ parallel environments throughout.

Section 5.1 introduces the robotic platform. Sections 5.2 and 5.3 present the two experimental setups. Section 5.4 synthesises the findings and connects them to the three requirements of Problem 3.1.1: that the loop certifies, that refinement is necessary, and that the pipeline remains computationally tractable.

5.1 Robot: TurtleBot3 Burger

Both experimental setups use the TurtleBot3 Burger, a compact differential-drive mobile robot. The policy commands a linear velocity $v \in [-v_{\max}, v_{\max}]$ and an angular velocity $\omega \in [-\omega_{\max}, \omega_{\max}]$, which are converted to individual wheel angular velocities through the standard differential-drive mapping

$$\text{rpm}_{\text{left}} = \frac{2v - \omega\,\text{Track}_{\text{w}}}{2\,\text{Rad}_{\text{w}}}, \qquad \text{rpm}_{\text{right}} = \frac{2v + \omega\,\text{Track}_{\text{w}}}{2\,\text{Rad}_{\text{w}}}, \tag{5.1}$$

where Rad_{w} is the wheel radius and Track_{w} the wheel track. Table 5.1 summarises the physical parameters.

The action space of Section 3.2.2 is $\mathcal{U} = [-1, 1]^2$, so the policy outputs $a_t = [a_{v,t},\, a_{\omega,t}]^\top$, mapped to physical commands by linear scaling:

$$v_t = a_{v,t}\, v_{\max}, \qquad \omega_t = a_{\omega,t}\, \omega_{\max}. \tag{5.2}$$

Normalisation to $[-1, 1]$ improves gradient conditioning and matches the symmetric output activation of the PPO actor network of Section 3.3.3.

Table 5.1: TurtleBot3 Burger physical parameters used in simulation.

Parameter	Symbol	Value
Wheel radius	$\mathrm{Rad_w}$	0.033 m
Wheel track	$\mathrm{Track_w}$	0.160 m
Max linear velocity	$v_{\max}$	0.22 m/s
Max angular velocity	$\omega_{\max}$	2.84 rad/s

5.2 Setup 1: Reach-and-Stay with Obstacle Avoidance

The first setup evaluates the policy on a conjunctive reach-and-stay specification with obstacle avoidance. The robot must navigate from a random initial pose to a fixed goal region, avoid three cylindrical obstacles and the arena boundary at all times, and once inside the goal, stay there continuously for a prescribed duration.

5.2.1 STL Specification and Environment

The robot operates in a square arena of side $d_{\text{arena}} = 2.0\,\text{m}$ centred at the origin, containing three circular obstacles $\mathcal{O}_i$ with centres $\mathbf{o}_i$ and radii rad_i,

$$\{\mathbf{o}_i\} = \{(-0.55, 0.55),\ (0.60, 0.20),\ (-0.20, -0.55)\}\,\text{m}, \quad \{\text{rad}_i\} = \{0.35,\ 0.35,\ 0.15\}\,\text{m}, \quad i = 1, 2, 3,$$

and a goal region centred at $\mathbf{q}_{\mathcal{G}} = (0.0, 0.0)\,\text{m}$ with tolerance $d_{\text{tol}} = 0.20\,\text{m}$. Let $\mathbf{q}_t = [q_{x,t},\ q_{y,t}]^\top$ denote the robot position at step t. The task is specified by

$$\varphi_1 \;=\; \square_{[0,\,45]}\,\phi_{\text{arena}} \;\wedge\; \square_{[0,\,45]}\,\phi_{\text{obst}} \;\wedge\; \lozenge_{[0,\,45]}\,\square_{[0,\,5]}\,\phi_{\text{goal}}, \tag{5.3}$$

with time bounds in seconds. In plain terms, the robot must reach the goal at the centre of the arena and stay there for 5 seconds while always remaining inside the arena and clear of the three obstacles. The specification horizon is $T_\varphi = 45\,\text{s}$, giving $H = T_\varphi/\Delta t = 4500$ simulation steps. Because the inner stay window must close within the horizon, the effective deadline for first goal entry is $T_\varphi - T_{a,1} = 40\,\text{s}$. A trajectory entering the goal later cannot accumulate the required 5 s of occupancy and is scored as a violation. The quantitative robustness of each predicate:

$$\begin{aligned} \rho_{\phi_{\text{arena}}}(x_t) &= \frac{d_{\text{arena}}}{2} - \max(|q_{x,t}|, |q_{y,t}|)\,, \\ \rho_{\phi_{\text{obst}}}(x_t) &= \min_i(\|\mathbf{q}_t - \mathbf{o}_i\| - \text{rad}_i)\,, \\ \rho_{\phi_{\text{goal}}}(x_t) &= d_{\text{tol}} - \|\mathbf{q}_t - \mathbf{q}_{\mathcal{G}}\|. \end{aligned} \tag{5.4}$$

each positive exactly when the corresponding predicate holds.

Specification (5.3) uses the operator classes of Section 3.2.1. The two safety conjuncts are full-horizon $\square$ operators carrying no timer, while the reach-and-stay conjunct is the nested form $\lozenge_{[T_{s,1},T_{d,1}]}\square_{[0,T_{a,1}]}$ with start $T_{s,1} = 0\,\text{s}$ deadline $T_{d,1} = 45\,\text{s}$ and stay duration $T_{a,1} = 5\,\text{s}$. Hence $N_\square = 1$, $N_\lozenge = 1$, so the augmentation of Section 3.2.1 adds two dimensions to the state and two scalar updates per step. The trajectory robustness of φ_1, computed online via the accumulators of Section 3.2.4 and finalised at step H, combines the three conjuncts as

$$\rho_{\varphi_1}(\xi) \;=\; \min\big(\rho_{\phi_{\text{arena}}}(\xi),\ \rho_{\phi_{\text{obst}}}(\xi),\ \rho_{\phi_{\text{goal}}}(\xi)\big). \tag{5.5}$$

The robot spawns at a position drawn uniformly from the free space with 0.15 m clearance from walls and obstacles, heading $\vartheta_0 \sim \mathrm{Unif}[0, 2\pi)$, and zero initial velocity.

5.2.2 State, Reward, and Termination Design

The augmented state $s_t \in \mathbb{R}^{10}$ concatenates an observation $x_t^{\mathrm{obs}} \in \mathbb{R}^8$ (position, heading, velocity, goal distance and bearing) with the two timer channels of Section 3.2.1, $\tau_{1,t}^{\mathrm{eventually}}$ for the $\Diamond_{[0,45]}$ operator and $\tau_{1,t}^{\mathrm{always}}$ for the $\Box_{[0,5]}$ conjunct. The reward follows the decomposition of Equation (3.25) with eight dense auxiliary terms: goal-directed shaping (distance progress with deadline-scaled urgency and bearing alignment), constraint shaping (quadratic proximity penalties for obstacles and walls), a constant step cost, one-time event bonuses for goal entry and safety violations, and the terminal robustness $\alpha\, \rho_{\varphi_1}(\xi)$. Episodes terminate on arena exit, obstacle contact, or timeout at $H = 4500$ steps. The full state specification, reward equations, and termination conditions are documented in Appendix B.1.

5.2.3 Certification and Refinement

This section compares Elastic Weight Consolidation (Section 3.5.1) and Progressive Neural Networks (Section 3.5.2) across refinement rounds. Both methods use the same certification pipeline. The mean policy $\bar{\pi}_\theta$ is (E, δ)-PAC certified when the Clopper–Pearson upper confidence bound satisfies $\epsilon_{\mathrm{CP}}(n_{\mathrm{viol}}, N_{cal}, \delta) \leq E$, with $E = 0.03$ and $\delta = 0.02$.

The calibration set size is held fixed at $N_{cal} = |\mathcal{D}_{\mathrm{cal}}^{(k)}| = 1024$ at every round, so that the Clopper–Pearson bound is evaluated on a constant sample size and rounds remain comparable. At this size and $\delta = 0.02$, the target $E = 0.03$ admits at most $n_{\mathrm{viol}} = 17$ violations. Both refinement methods start from the same Round 0 policy, whose training and evaluation require 509.0 s.

Measuring Forgetting and Forward Transfer

Each refinement round should improve the policy on starting pose inside the counterexample region $\hat{R}_{k-1}$ that motivated refinement, and leave behaviour outside it intact. Forward transfer measures the first effect and forgetting measures the second.

Measuring both directly would require evaluating the round-$(k-1)$ and round-k policies at the same starting positions. Because Section 4.4.1 draws a fresh calibration set $\mathcal{D}_{\mathrm{cal}}^{(k)}$ at every round, the starting positions differ between consecutive sets. For each starting position $(p_x, p_y) \in \mathcal{D}_{\mathrm{cal}}^{(k)}$, its nearest neighbour by Euclidean distance in $\mathcal{D}_{\mathrm{cal}}^{(k-1)}$, denoted $\mathcal{NN}(p_x, p_y)$, serves as a stand-in for the previous round.

Define $\rho^{(k)}(p_x, p_y) := \rho_\varphi(\xi_{(p_x,p_y)}^{\bar{\pi}_{\theta_k}})$, the trajectory robustness achieved by the round-k mean policy from starting position (p_x, p_y). Using whether $\mathcal{NN}(p_x, p_y)$ falls inside $\hat{R}_{k-1}$ to split

$\mathcal{D}_{\text{cal}}^{(k)}$, forgetting and forward transfer are defined as

$$\mathcal{F}_{k,\rho} = \frac{1}{\left|\{\mathcal{NN}(p_x,p_y) \notin \hat{R}_{k-1}\}\right|} \sum_{\mathcal{NN}(p_x,p_y) \notin \hat{R}_{k-1}} [\rho_{k-1}(\mathcal{NN}(p_x,p_y)) - \rho_k(p_x,p_y)]$$
$$(+ = \text{robustness lost outside } \hat{R}_{k-1}), \tag{5.6}$$

$$\mathcal{FT}_{k,\rho} = \frac{1}{\left|\{\mathcal{NN}(p_x,p_y) \in \hat{R}_{k-1}\}\right|} \sum_{\mathcal{NN}(p_x,p_y) \in \hat{R}_{k-1}} [\rho_k(p_x,p_y) - \rho_{k-1}(\mathcal{NN}(p_x,p_y))]$$
$$(+ = \hat{R}_{k-1} \text{ improved}). \tag{5.7}$$

A positive $\mathcal{F}_{k,\rho}$ means the policy lost robustness on positions it previously handled well. A positive $\mathcal{FT}_{k,\rho}$ means the policy improved on the region it was trained on. A well-behaved refinement round has $\mathcal{FT}_{k,\rho} > 0$ and $\mathcal{F}_{k,\rho} \approx 0$.

The nearest-neighbour proxy is not exact since heading at spawn affects trajectory outcomes for a differential-drive robot, even between positions that are close. Small nonzero values of $\mathcal{F}_{k,\rho}$ and $\mathcal{FT}_{k,\rho}$ should therefore be read as measurement noise rather than as evidence of forgetting or transfer. Only large values clearly above this noise floor carry meaning.

Elastic Weight Consolidation

Table 5.2: PAC certification, forgetting, and forward transfer for the EWC policy, evaluated on a calibration set of size $N_{cal} = 1024$ per round at $E = 0.03$, $\delta = 0.02$. Here $\bar{\rho} \pm \sigma_\rho$ is the mean and standard deviation of $\rho_{\varphi_1}(\xi)$, ϵ_{CP} the Clopper–Pearson bound, $\mathcal{F}_{k,\rho}$ forgetting on the untargeted complement, $\mathcal{FT}_{k,\rho}$ forward transfer on $\hat{R}_{k-1}$, and t_{wall} the cumulative wall-clock time including the shared Round-0 baseline cost.

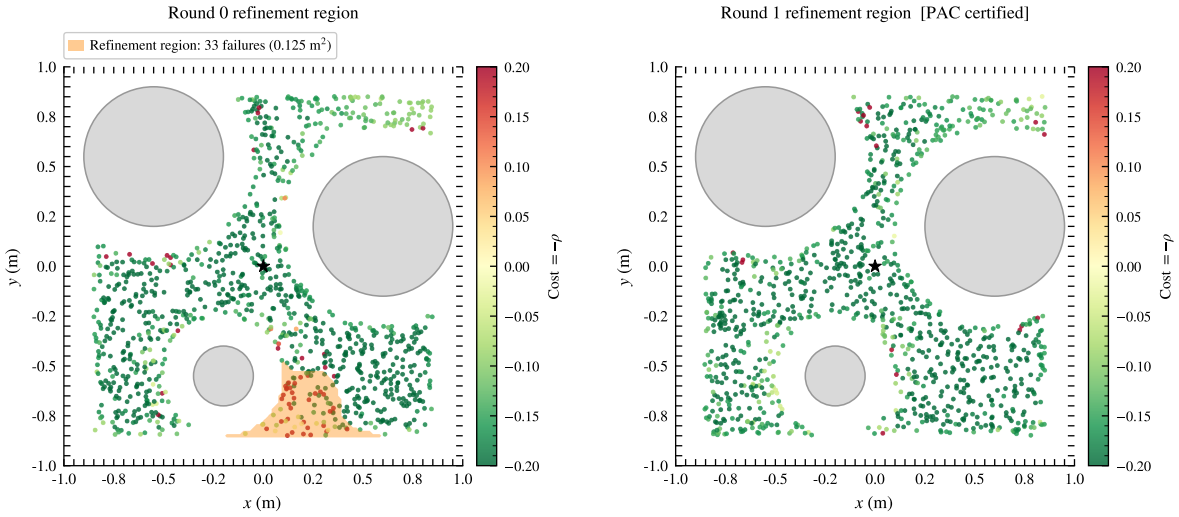
k	Iterations	n_{viol}	$\bar{\rho} \pm \sigma_\rho$	ϵ_{CP}	$\mathcal{F}_{k,\rho}$	$\mathcal{FT}_{k,\rho}$	t_{wall} [s]	Certified
0	2250	56	0.140 ± 0.150	0.0712	—	—	509.0	No
1	3050	14	0.165 ± 0.085	0.0233	+0.0061	0.539	889.7	Yes

For $\lambda_{EWC} = 0.005$, table 5.2 shows that the Round-0 policy records $n_{\text{viol}} = 56$ and $\epsilon_{\text{CP}} = 0.0712$, more than twice the target $E = 0.03$. One round of EWC refinement brings this to $n_{\text{viol}} = 14$ and $\epsilon_{\text{CP}} = 0.0233$, certifying the policy at a cumulative wall-clock time of 889.7 s.

The standard deviation σ_ρ falls 43% from 0.150 to 0.085, while the mean $\bar{\rho}$ rises only 18% from 0.140 to 0.165. Refinement removes the low-robustness tail, the effect that biased resampling (Section 3.4.5) is designed to produce.

Note that $\hat{R}_k$ denotes the counterexample region extracted after round k and used to guide round $k+1$, so $\hat{R}_0$ is the region that motivated the Round 1 refinement.

The forward transfer $\mathcal{FT}_{1,\rho} = 0.539$ confirms a substantial improvement inside $\hat{R}_0$. Round 0 enclosed 33 failures inside $\hat{R}_0$, yet one round removes 42 violations in total. At least nine of the removed violations therefore lay outside $\hat{R}_0$, in the untargeted complement. EWC's shared-weight update propagated beneficial changes across the whole network. Forgetting is $\mathcal{F}_{1,\rho} = +0.0061$, within the noise floor. After one round $\hat{R}_1$ is empty (no component survives the minimum-support filter), so there is no targeted region to measure transfer on.



(a) Round 0 **(b)** Round 1, PAC certified

Figure 5.1: Retained counterexample region $\hat{R}_k$ (orange) and per-rollout robustness $\rho_{\varphi_1}(\xi)$ across refinement rounds under EWC. The colour scale shows $\rho_{\varphi_1}(\xi)$: red indicates large negative robustness (clear violations) and green indicates positive robustness (satisfied trajectories), with yellow marking rollouts close to the satisfaction boundary. In Round 0 the region covers $0.125\,\mathrm{m}^2$ and encloses 33 of the 56 observed failures, concentrated near the lower obstacle. After one refinement round the policy is certified.

Figure 5.1 shows why one round is sufficient. Round 0 failures are not scattered uniformly across the arena but fall inside a single compact region near the lower obstacle, with the remainder below the minimum-support threshold. The counterexample extractor returns one well-localised target, and one round of refinement on that target meets the certification threshold.

Progressive Neural Networks

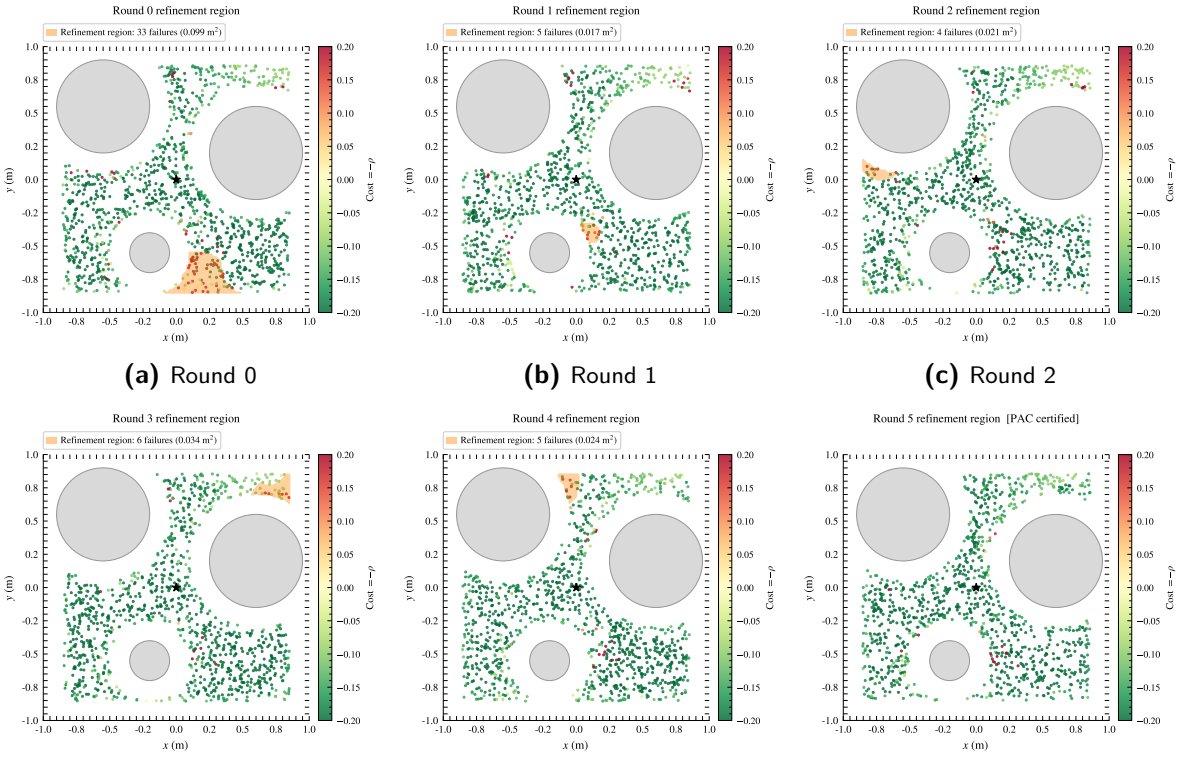
Table 5.3: PAC certification, forgetting, and forward transfer for the PNN policy, evaluated on a calibration set of size $N_{cal} = 1024$ per round at $E = 0.03$, $\delta = 0.02$. Here $\bar{\rho} \pm \sigma_\rho$ is the mean and standard deviation of $\rho_{\varphi_1}(\xi)$, ϵ_{CP} the Clopper–Pearson bound, $\mathcal{F}_{k,\rho}$ forgetting on the untargeted complement, $\mathcal{FT}_{k,\rho}$ forward transfer on $\hat{R}_{k-1}$, and t_{wall} the cumulative wall-clock time including the shared Round-0 baseline cost.

k	Iterations	n_{viol}	$\bar{\rho} \pm \sigma_\rho$	ϵ_{CP}	$\mathcal{F}_{k,\rho}$	$\mathcal{FT}_{k,\rho}$	t_{wall} [s]	Certified
0	2250	56	0.140 ± 0.150	0.0712	—	—	509.0	No
1	3050	30	0.158 ± 0.103	0.0422	+0.0138	0.5608	797.9	No
2	3850	28	0.161 ± 0.102	0.0399	+0.0117	0.4515	1132.3	No
3	4650	22	0.167 ± 0.090	0.0329	+0.0059	0.3702	1525.5	No
4	5450	21	0.167 ± 0.081	0.0317	+0.0079	0.5586	1978.6	No
5	6250	15	0.171 ± 0.063	0.0245	+0.0026	0.4646	2489.8	Yes

Table 5.3 reaches the same certificate within five rounds and 2489.8 s, a 179.8% increase in wall-clock time over EWC for a comparable final margin. Violations fall monotonically (56,

30, 28, 22, 21, 15), so no round trades one failure region for another.

Round 1 removes 26 violations with $\mathcal{FT}_{1,\rho} = 0.5608$, comparable to EWC's gain within the targeted region. The retained region $\hat{R}_0$ enclosed 33 failures, so PNN's first round removes fewer violations than the region contained. The new column improves only within $\hat{R}_0$, and the frozen column 0 handles everything else without change. Forward transfer remains positive but declining across Rounds 2–5, as each new column addresses a progressively smaller residual region. The standard deviation σ_ρ falls 58% from 0.150 to 0.063 over five rounds, reaching a comparable endpoint to EWC's single-round compression.



(a) Round 0 **(b)** Round 1 **(c)** Round 2

(d) Round 3 **(e)** Round 4 **(f)** Round 5, PAC certified

Figure 5.2: Retained counterexample region $\hat{R}_k$ (orange) and per-rollout robustness $\rho_{\varphi_1}(\xi)$ across refinement rounds under PNN. The dominant region shrinks near the same obstacle at every round, reaching certification at Round 5 where no component survives the minimum-support filter.

Figure 5.2 shows the failure region shrinking near the same obstacle at every round without relocating, so the diffuse-failure regime of Section 3.6.3 does not arise on this specification.

Comparison of EWC and PNN

On this setup, EWC certifies in roughly one-third of the wall-clock time and with one-sixth the deployed model size, without measurable forgetting. Both tables share a Round 0 row, so every later difference comes from the refinement mechanism alone. EWC removes 42 violations in one round, exceeding the 33 enclosed in $\hat{R}_0$ and thereby improving configurations it was not directly trained on. PNN removes a comparable 41 across five rounds, 26 of them in Round 1, but fewer than the region contained because its new column is structurally isolated

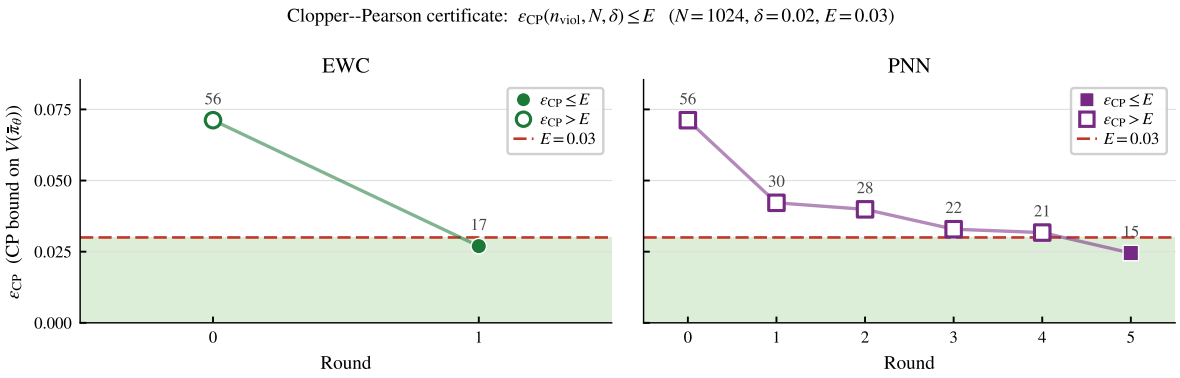


Figure 5.3: Clopper–Pearson bound ϵ_{CP} versus refinement round for EWC (left) and PNN (right), at $N_{cal} = 1024$, $\delta = 0.02$, $E = 0.03$. The dashed red line marks the target E and the shaded green region indicates certification. EWC reaches the certification threshold after one round and PNN requires five.

to the training region. The forgetting columns do not clearly separate the mechanisms: both values sit near the noise floor, and neither method loses more than 1.5% of mean complement robustness at any round. Final quality is close ($\bar{\rho} = 0.165 \pm 0.085$ for EWC against 0.171 ± 0.063 for PNN).

Comparison with TGPO and the No-Refinement Ablation

The framework is additionally compared against TGPO [11], a hierarchical policy-optimisation baseline for STL tasks, and against an ablated variant with the certification-and-refinement loop disabled. TGPO* was implemented on specification (5.3) using the network sizes, optimiser settings, and sampling hyperparameters reported in Appendix A.4 of the original paper, with reward weights taken from the Unicycle column of Appendix A.1. The dynamics, stay duration, and arena scale differ from the original paper, so weights optimised for this task may improve TGPO* beyond what is reported below. The most significant difference with TGPO* is that it does not report or target a PAC certificate on its violation rate; success is measured as an empirical success rate averaged over random seeds. TGPO* reaches $\bar{\rho} = 0.1151 \pm 0.0048$ and costs 3741.0 s. The robustness gap, SR and the compute difference are subject to the tuning caveat above. All methods are evaluated under a repeated-calibration protocol of 500 independently seeded calibration sets of $N_{cal} = 1024$ rollouts each, giving 512,000 rollouts per method. The success rate is defined as

$$\mathrm{SR} = \frac{|\{\xi : \rho_\varphi(\xi) > 0\}|}{N_{cal}}, \tag{5.8}$$

that is, the fraction of rollouts whose robustness is strictly positive.

Both EWC and PNN certify on all 500 seed draws, with no set failing to meet the tolerance $E = 0.03$. The spread of mean robustness across the 500 sets (0.0028 for EWC, 0.0022 for PNN) is roughly 30× smaller than the within-set σ_ρ, showing that the variability is in individual rollouts, not in the calibration draw.

The no-refinement variant highlights the contribution of the certify-and-refine loop. Given 2498.1 s of uninterrupted training, roughly three times EWC's total budget, it reaches $\bar{\rho} =$

Table 5.4: Comparison against TGPO* and the no-refinement ablation on specification (5.3), using 500 independently seeded calibration sets of $N_{cal} = 1024$. Certification is (E, δ)-PAC at $E = 0.03$, $\delta = 0.02$, reported as achieved only when every one of the 500 sets certifies.

Method	$\bar{\rho} \pm \sigma_\rho$	Sets with SR > 0.97	End-to-end time [s]	Certified
EWC (Round 1)	0.1654 ± 0.0028	500/500	889.7	Yes
PNN (Round 5)	0.1692 ± 0.0022	500/500	2489.8	Yes
Ours, no refinement	0.1581 ± 0.0039	134/500	2498.1	No
TGPO*	0.1151 ± 0.0048	0/500	3741.0	No

0.1581 ± 0.0039 but has SR > 0.97 on only 134/500, the policy is uncertified. The gap is not about how much compute was used but where it was spent. PPO optimises expected robustness (3.10) and has no mechanism to control the violation-rate tail that ϵ_{CP} responds to. Targeted refinement reduces that tail in a way that extended uniform training does not.

5.3 Setup 2: Multi-Goal Navigation with Obstacle Avoidance

The second setup tests the framework on a structurally harder specification. The robot must first reach one of two goal regions and stay there, then proceed to a third goal and stay there for a longer duration, all while avoiding five rectangular obstacles in an arena four times the area of Setup 1. Three challenges arise that were not present in Setup 1: the policy must resolve an unspecified disjunctive choice between two geographically opposite goals, the temporal structure is phased with deadline management across phases, and the obstacle geometry creates tighter passages. The phased deadlines double the episode length and hence the per-round training and validation cost.

5.3.1 STL Specification and Environment

The robot operates in a square arena of side $d_{\mathrm{arena}} = 4.0\,\mathrm{m}$ centred at the origin, containing five axis-aligned rectangular obstacles $\mathcal{O}_i$ with centres $(o_{x,i}, o_{y,i})$ and half-widths $(h_{x,i}, h_{y,i})$:

$\mathcal{O}_1 : (0.00,\, -0.15),\ (0.35,\, 0.35), \quad \mathcal{O}_2 : (1.30,\, 0.20),\ (0.14,\, 0.40), \quad \mathcal{O}_3 : (0.15,\, 1.25),\ (0.40,\, 0.15),$
$\mathcal{O}_4 : (-1.40,\, 0.50),\ (0.30,\, 0.12), \quad \mathcal{O}_5 : (0.50,\, -1.50),\ (0.40,\, 0.10).$

Three goal regions are defined by their centres and a shared tolerance $d_{\mathrm{tol}} = 0.20\,\mathrm{m}$:

$$\mathbf{q}_{\mathcal{G}_1} = (1.5,\, -1.3)\,\mathrm{m}, \qquad \mathbf{q}_{\mathcal{G}_2} = (-1.4,\, 1.4)\,\mathrm{m}, \qquad \mathbf{q}_{\mathcal{G}_3} = (1.3,\, 1.4)\,\mathrm{m}.$$

Let $\mathbf{q}_t = [q_{x,t},\, q_{y,t}]^\top$ denote the robot position at step t. The task is specified by

$$\varphi_2 \;=\; \underbrace{\Box_{[0,\,90]}\big(\phi_{\mathrm{arena}} \wedge \phi_{\mathrm{obst}}\big)}_{\text{safety}} \;\wedge\; \underbrace{\Big(\Diamond_{[0,\,42]}\,\Box_{[0,\,5]}\,\phi_{\mathcal{G}_1} \;\vee\; \Diamond_{[0,\,45]}\,\Box_{[0,\,5]}\,\phi_{\mathcal{G}_2}\Big)}_{\varphi_{\mathrm{p1}}\text{: phase 1}} \;\wedge\; \underbrace{\Diamond_{[0,\,80]}\,\Box_{[0,\,7]}\,\phi_{\mathcal{G}_3}}_{\varphi_{\mathrm{p2}}\text{: phase 2}}, \tag{5.9}$$

with all time bounds in seconds. In plain terms, the robot must first reach and stay at either of two intermediate goals ($\mathcal{G}_1$ or $\mathcal{G}_2$) for 5 seconds, then navigate to a third goal ($\mathcal{G}_3$) and

stay there for 7 seconds, while avoiding all five obstacles throughout. The disjunction in φ_{p1} requires the policy to commit to one of two geographically opposite goals. Which goal to visit must be learned from the reward signal alone, since the specification does not prescribe the choice. The specification horizon is $T_\varphi = 90\,\mathrm{s}$, giving $H = T_\varphi/\Delta t = 9000$ simulation steps at $\Delta t = 0.01\,\mathrm{s}$. The predicate robustness functions are

$$\begin{aligned} \rho_{\phi_{\mathrm{arena}}}(x_t) &= \tfrac{d_{\mathrm{arena}}}{2} - \max(|q_{x,t}|,\, |q_{y,t}|), \\ \rho_{\phi_{\mathrm{obst}}}(x_t) &= \min_i\, d_{\mathrm{box}}(\mathbf{q}_t,\, \mathcal{O}_i) - d^{\mathrm{tol}}_{\mathrm{safe}}, \\ \rho_{\phi_{\mathcal{G}_j}}(x_t) &= d_{\mathrm{tol}} - \|\mathbf{q}_t - \mathbf{q}_{\mathcal{G}_j}\|, \end{aligned} \tag{5.10}$$

where $d_{\mathrm{box}}(\mathbf{q},\, \mathcal{O}_i)$ is the signed distance from $\mathbf{q}$ to the axis-aligned box $\mathcal{O}_i$ (positive outside),

$$d_{\mathrm{box}}(\mathbf{q},\, \mathcal{O}_i) \;=\; \big\|\max(|\mathbf{q} - \mathbf{o}_i| - \mathbf{h}_i,\; \mathbf{0})\big\| \;+\; \min\Big(\max(|q_x - o_{x,i}| - h_{x,i},\; |q_y - o_{y,i}| - h_{y,i}),\; 0\Big), \tag{5.11}$$

and $d^{\mathrm{tol}}_{\mathrm{safe}} = 0.10\,\mathrm{m}$ is a safety margin. The trajectory robustness $\rho_{\varphi_2}(\xi)$ is computed from the accumulators of Section 3.2.4 following the Boolean structure of (5.9).

Specification (5.9) uses the operator classes of Section 3.2.1 as follows. The safety conjunct is a full-horizon $\Box$ carrying no timer. Phase 1 (φ_{p1}) places two nested $\Diamond\Box$ operators under a disjunction, contributing two eventually timers and two always timers, and phase 2 (φ_{p2}) is a single nested $\Diamond\Box$ contributing one of each. Hence $N_\Box = 3$, $N_\Diamond = 3$, and the augmented state carries six timer channels. The augmented state, reward structure, and termination conditions are documented in Appendix B.2.

5.3.2 Certification and Refinement with EWC

The certification pipeline follows Setup 1, with the calibration set doubled to $N_{cal} = |\mathcal{D}^{(k)}_{\mathrm{cal}}| = 2048$ per round, primarily to provide the RBF surrogate with sufficient spatial coverage of the larger arena for reliable failure-region extraction. At $N = 2048$ and $\delta = 0.02$, the target $\varepsilon = 0.03$ admits up to $n_{\mathrm{viol}} = 45$ failures.

Table 5.5: PAC certification, forgetting, and forward transfer for the EWC policy on Setup 2, evaluated at each refinement round with $E = 0.03$ and $\delta = 0.02$. Notation follows Table 5.2. All t_{wall} values are cumulative from the start of Round-0 training.

k	Iterations	n_{viol}	$\bar{\rho} \pm \sigma_\rho$	ϵ_{CP}	$\mathcal{F}_{k,\rho}$	$\mathcal{FT}_{k,\rho}$	t_{wall} [s]	Certified
0	5000	142	0.104 ± 0.089	0.0818	—	—	3360.0	No
1	7000	215	0.101 ± 0.109	0.1198	+0.0174	0.1974	5243.8	No
2	9000	257	0.103 ± 0.120	0.1413	+0.0228	0.2155	6789.3	No
3	11000	116	0.120 ± 0.086	0.0681	+0.0234	0.2968	8408.3	No
4	13000	105	0.120 ± 0.081	0.0622	+0.0149	0.2474	9979.4	No
5	15000	98	0.123 ± 0.078	0.0585	+0.0116	0.2644	11490.7	No
6	17000	51	0.132 ± 0.059	0.0330	+0.0037	0.2730	12995.6	No
7	19000	45	0.137 ± 0.058	0.0297	−0.0014	0.1411	14509.1	Yes

Table 5.5 is the only run in either setup where violations rise before they fall. From Round 0 to Round 2, violations increase from 142 to 257 (+81%) and ϵ_{CP} reaches 0.1413, the largest value

recorded across both setups. The forgetting values $\mathcal{F}_{1,\rho} = +0.0174$ and $\mathcal{F}_{2,\rho} = +0.0228$ are the largest recorded, showing that EWC's shared-weight update degraded behaviour outside the targeted region despite a fiftyfold increase in λ_{EWC} from Setup 1 (0.001 to 0.05, which places a stronger quadratic penalty on parameter drift from the previous round's weights).

Recovery follows from Round 3 onward. Violations drop from 257 to 116 between Rounds 2 and 3, and forgetting falls sharply as the extracted regions become smaller and more behaviourally homogeneous. By Rounds 6 and 7 the forgetting is negligible and the forward transfer is still positive. The tail-compression pattern from Setup 1 is preserved: σ_ρ falls 35% from 0.089 to 0.058, while $\bar{\rho}$ rises 32% from 0.104 to 0.137. At Round 7 with a cumulative wall-clock time of 14 509.1 s the policy is certified.

Figure 5.4 shows the regression spatially: at Round 2 the targeted region has contracted while new violations appear across the arena. By Round 7 the failures are diffuse, with only 4 of 45 falling inside a small residual region.

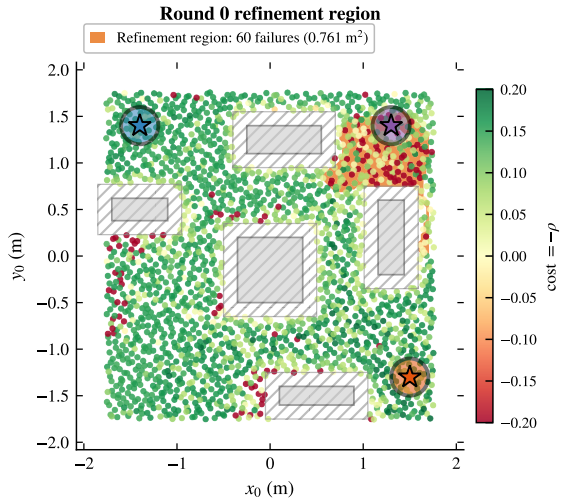
5.3.3 Certification and Refinement with Progressive Neural Networks

Table 5.6: PAC certification, forgetting, and forward transfer for the PNN policy on Setup 2, evaluated at each refinement round with $\varepsilon = 0.03$ and $\delta = 0.02$. Notation follows Table 5.2. Round 0 is the shared initial policy. All t_{wall} values are cumulative from the start of Round-0 training. The certified policy at Round 9 has ten columns.

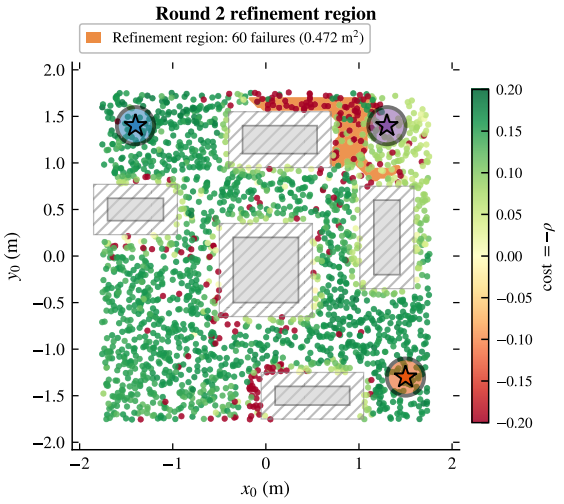
k	Iterations	n_{viol}	$\bar{\rho} \pm \sigma_\rho$	ϵ_{CP}	$\mathcal{F}_{k,\rho}$	$\mathcal{FT}_{k,\rho}$	t_{wall} [s]	Certified
0	5000	142	0.104 ± 0.089	0.0818	—	—	3360.0	No
1	7000	102	0.111 ± 0.078	0.0606	+0.0092	+0.2294	5625.4	No
2	9000	80	0.115 ± 0.070	0.0488	+0.0072	+0.2099	7725.2	No
3	11000	70	0.116 ± 0.067	0.0434	+0.0061	+0.1784	9861.1	No
4	13000	65	0.116 ± 0.066	0.0407	+0.0056	+0.1747	11994.0	No
5	15000	59	0.120 ± 0.060	0.0374	+0.0033	+0.2204	14133.9	No
6	17000	47	0.118 ± 0.063	0.0308	+0.0070	+0.1731	16286.4	No
7	19000	61	0.117 ± 0.067	0.0385	+0.0068	+0.2267	18437.2	No
8	21000	47	0.120 ± 0.062	0.0308	+0.0046	+0.2518	20611.0	No
9	23000	36	0.121 ± 0.059	0.0246	+0.0041	+0.2302	22784.0	Yes

Table 5.6 shows the PNN property freezing is working, at the cost of wall–clock time. Violations fall from 142 to 36 across nine rounds with regression only in round 7, and $\mathcal{F}_{k,\rho}$ stays below +0.0092 throughout, confirming that frozen columns structurally prevent the complement regression exhibited by EWC. Forward transfer is positive at every round, averaging approximately 0.21, reflecting steady improvement within each targeted region.

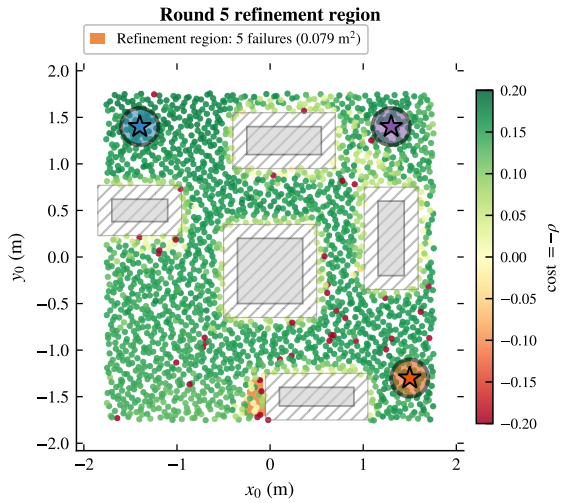
Round 7 shows a modest reversal from 47 to 61 violations, likely because the newly added column was not yet sufficiently trained on the failure region assigned to it. Round 8 corrects this, and Round 9 achieves certification. The cost is in convergence rate and model size, nine rounds are needed against EWC's seven.



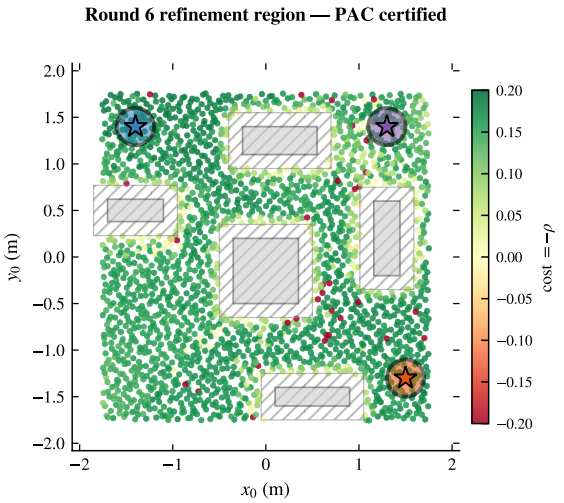
(a) Round 0 (initial policy)



(b) Round 2 (peak regression)

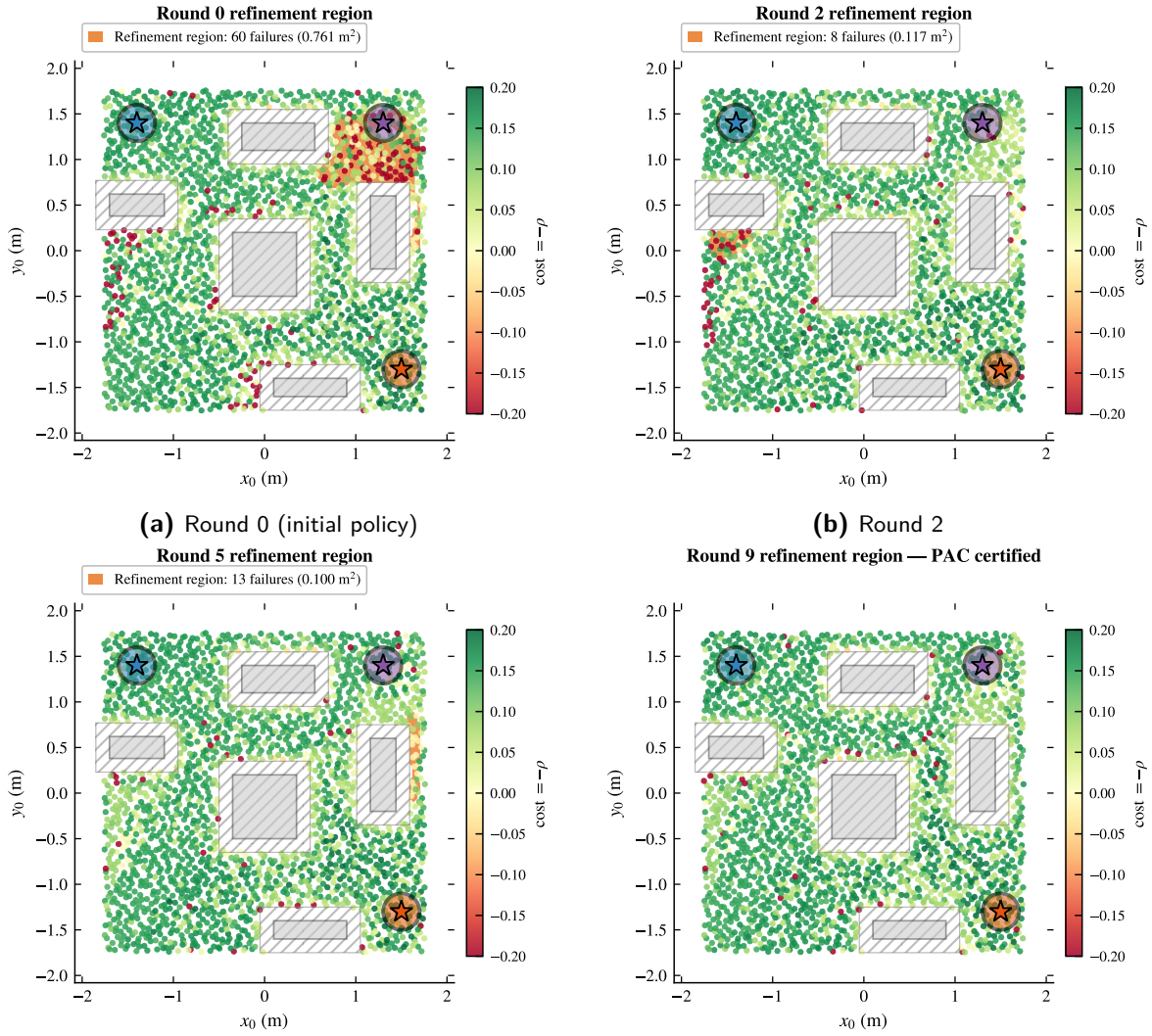


(c) Round 5



(d) Round 7 (certified)

Figure 5.4: Retained counterexample region $\hat{R}_k$ (orange) and per-rollout robustnes $\rho_{\varphi_2}(\xi)$ at selected refinement rounds under EWC for Setup 2. The initial policy (a) concentrates 60 of 142 failures in a $0.761\,\mathrm{m}^2$ region. At peak regression (b), the total violation count rises to 257; many rollouts have robustness values close to zero, appearing yellow rather than deep red, so the spread of degraded behaviour is wider than the colour intensity alone suggests. At (d), policy is certified.



(a) Round 0 (initial policy)

(b) Round 2

(c) Round 5

(d) Round 9 (certified)

Figure 5.5: Retained counterexample region $\hat{R}_k$ (orange) and per-rollout robustness $\rho_{\varphi_2}(\xi)$ at selected refinement rounds under PNN for Setup 2. Violations thin across the arena without the concentration-then-spreading pattern observed under EWC. At certification (d), 36 violations remain, diffusely scattered.

5.3.4 Comparison of EWC and PNN on Setup 2

Tables 5.5 and 5.6 start from the same Round 0, so their divergence comes entirely from the refinement mechanism. The violation trajectories have opposite shapes: EWC is non-monotonic, rising from 142 to 257 before falling to 45, while PNN falls from 142 to 36 regressing only in one of the rounds. EWC's early rounds combine high forgetting with low forward transfer, the rounds degraded the complement without fully resolving failures within the target.

Past Round 3, EWC removes violations faster, certifying in seven rounds and 14 509.1 s against PNN's nine rounds and 22 784.0 s (10× deployed parameters). The same freedom that caused the early regression, EWC's unconstrained shared-weight update, is also what enables faster recovery once the policy aligns with the right behaviour. PNN cannot exploit this but also does not regress. For specifications with disjunctions over geographically distant goals, where the failure region involves very different behaviours, PNN's structural isolation is the safer choice to implement.

5.4 Discussion

The experiments test three claims about the proposed framework: that the train-certify-refine loop produces policies with formally bounded violation rates, that the refinement stage is necessary for certification, and that the pipeline remains computationally tractable. The EWC-versus-PNN comparison is discussed last, as a component study within the refinement stage.

The loop certifies. The key result is that the framework certifies every policy it produces. On Setup 1 certification is achieved after one EWC round (Table 5.2) or five PNN rounds (Table 5.3). On Setup 2 it takes seven EWC rounds (Table 5.5) or nine PNN rounds (Table 5.6). Setup 3 in Appendix A extends the result to a random-goal setting with 20 candidate targets and certifies in three EWC rounds. In every case the certificate is a finite-sample, distribution-free bound on the violation rate computed from rollout data alone. The repeated-calibration protocol of Table 5.4 confirms that certification is a property of the trained policy rather than of one favourable draw: both methods pass on all 500 independently seeded calibration sets, and the cross-set spread in mean robustness is roughly 30× smaller than the within-set standard deviation.

Refinement is necessary. The no-refinement ablation in Table 5.4 provides the strongest evidence for the necessity of the certify-and-refine loop. Given three times EWC's compute budget, uninterrupted PPO training reaches comparable mean robustness ($\bar{\rho} = 0.1581$ vs. 0.1654) but is not PAC certified. The gap is not in average performance, the synthesis objective (3.10) optimises expected robustness effectively. The gap is in the violation-rate tail. PPO cannot identify or prioritise the starting configurations responsible for residual violations. The refinement loop provides this by localizing failures through the surrogate-based extraction of Section 3.4.2, concentrates training on the identified region, and re-validates on a fresh draw. Table 5.4 shows that this targeted approach achieves certification with a third of the

compute. The TGPO* comparison reinforces the claims, regardless of how its robustness or compute compare, TGPO* provides no mechanism to bound its violation rate at a confidence level.

Failures are localisable. The refinement loop is effective only when failures concentrate in identifiable regions of the initial-condition space. In Setup 1, Round 0 failures concentrate near the lower obstacle (Figure 5.1), and the counterexample extractor returns a compact region of $0.125\,\mathrm{m}^2$ enclosing 33 of 56 failures. In Setup 2 the initial failure region covers encloses 60 of 142 failures (Figure 5.4). In both cases the extracted regions shrank across rounds and eventually violations fell in count. Forward transfer is consistently positive across most of the rounds and both approaches, suggesting the refinement loop works well. Section 3.6.3 identifies diffuse failures, spread uniformly across $\mathcal{X}_0$, as the regime where the loop offers no advantage over extended Round 0 training. None of the tested specifications triggered this regime.

The loop is tractable. The wall-clock scaling of Equation (3.40) predicts that per-round cost grows linearly with the episode length H and the calibration set size N. Setup 2 doubles both ($H = 9000$ vs. 4500, $N = 2048$ vs. 1024) and requires more refinement rounds, so the total time grows substantially relative to Setup 1. This is expected as the the doubled horizon comes from the phased specification with longer deadlines. The larger calibration set is needed to provide the RBF surrogate with sufficient spatial coverage of the four-times-larger arena. The disjunction in φ_{p1} and the tighter obstacle passages further increase the number of refinement rounds needed. Despite this, the full EWC pipeline certifies in under four hours on a single consumer GPU. The timer augmentation keeps per-step cost constant by adding only $N_{\square} + N_{\Diamond}$ scalar dimensions to the state, and the Clopper–Pearson bound requires only counting violations, making each validation round substantially cheaper than formal verification methods would require at this policy scale.

EWC versus PNN. Both approaches certify on every setup tested, so the question is not whether refinement works but which approach to use and when. Setup 1 does not clearly separate the two: both certify with comparable final quality, and forgetting values sit near the noise floor. Setup 2 exposes a meaningful divergence. EWC's violation count rises from 142 to 257 in the first two rounds before recovering, driven by forgetting values that exceed the noise floor by an order of magnitude. PNN's violations fall monotonically from 142 to 36 with forgetting values that never exceed $+0.0092$. The difference traces to a structural limitation of the diagonal Fisher approximation, the retained region under φ_{p1} mixes configurations requiring opposite actions, and per-parameter importance scores cannot distinguish the two cases. PNN's frozen columns avoid this entirely, at the cost of more rounds, more wall-clock time, and a deployed model roughly $10\times$ the base parameter count. EWC is preferable when the specification is conjunctive and failure regions are behaviourally homogeneous. PNN is the safer choice when specifications produce qualitatively different behaviour requirements within a single retained region.

The results confirm the framework's three claims and demonstrate applicability to multi-objective STL specifications on a differential-drive robot. The limitations identified, including

sensitivity to failure localisability, the sim-to-real gap, and the absence of training-time safety, are addressed as future work in Chapter 6.

Chapter 6

Conclusions and Future Work

6.1 Concluding Remarks

Reinforcement learning has demonstrated strong empirical performance on robotic tasks, but the lack of guarantees has limited its use in settings where a failure is costly. The approaches reviewed in Chapter 1 either require a dynamics model, scale poorly to deep networks, or produce claims about average performance rather than about the rate of specification violations. This thesis takes a different route. It trains a policy with STL robustness as the objective, certifies it from rollout data alone using a distribution-free bound on the violation rate, and when certification fails, uses the same data to identify where the policy is weak and retrains on those configurations. The result is an iterative train-certify-refine loop that produces policies with a formally bounded violation rate, computed from data, without any assumption on the dynamics or the network architecture.

One of the key claim is that PAC certification combined with counterexample-driven refinement yields certified policies where training on the synthesis objective struggles to do so. The no-refinement ablation in Table 5.4 provides the evidence. Given three times the compute budget, uninterrupted training reaches comparable mean robustness but doesn't provide required PAC certificate. EWC certifies after one refinement round at a third of that budget. The PPO objective optimises expected robustness and does not directly control the violation-rate tail. Extended training reduces violations slowly and uniformly across all starting configurations. Targeted refinement on failure regions compresses the tail specifically, by concentrating gradient updates where the policy is weakest. Uniform training cannot identify or prioritise the starting pose of the robot responsible for the residual violations.

The statistical validation is distribution-free. It requires only that the calibration rollouts are drawn independently from the true initial-condition distribution and makes no assumption about the form of that distribution, the dynamics, or the policy. The Clopper-Pearson bound is exact at finite sample size, not asymptotic. The 500-set repeated calibration in Table 5.4 confirms this empirically. Neither certified policy failed on any of the 500 draws, and the cross-set variability in mean robustness was 30× smaller than the within-set variability. The

randomness that determines whether a policy certifies lies in individual rollout outcomes. Which calibration set happens to be drawn contributes negligibly. This is the empirical counterpart of the bound developed in Section 3.4.1, and it confirms that the certificate is a property of the policy rather than an artefact of one favourable draw.

The counterexample extraction pipeline converts the discrete failure set into a continuous region by fitting an RBF surrogate over starting positions and selecting a threshold that maximises a failure-concentration score. The score balances failure density against sample support using a lower confidence bound on the conditional failure probability, which prevents the extraction from collapsing onto a handful of isolated failures with no nearby samples. In practice, failures were spatially structured near obstacle configurations in all three setups. The extracted regions shrank across rounds.

The entire loop remains computationally tractable on a single consumer GPU. The timer augmentation is what makes this possible: encoding temporal progress as continuous scalar fractions, one per operator, restores the Markov property for satisfaction status at constant per-step cost, with the state dimension growing only by $N_{\square} + N_{\Diamond}$ independent of the specification horizon. This keeps the full train-certify-refine pipeline at 889.7 s on Setup 1 and under four hours on the structurally harder Setup 2. Proposition 3.2.1 establishes that the timers are a sufficient statistic for temporal satisfaction, so feedforward policies without recurrent architectures suffice. The deterministic mean policy makes the only source of randomness in the certificate the draw of the initial condition.

The EWC versus PNN comparison reduces to a practical trade-off between tuning effort and model growth. EWC requires careful tuning of λ_{EWC}, if it is too low forgetting occurs, too high and plasticity is suppressed, and the right value depends on the behavioural complexity of the retained region in ways that are difficult to predict in advance, as the regression in Setup 2 demonstrates. PNN avoids this entirely since freezing prior columns gives hard forgetting guarantees with no penalty coefficient to tune, at the cost of one additional column per round. For specifications with disjunctions or other sources of behavioural heterogeneity within a retained region, PNN is the easier choice to implement.

One assumption held across all experiments that the framework does not enforce. Failures were localisable, meaning they concentrated near obstacle configurations rather than spreading uniformly across the initial-condition space. A task with sparse or fragmented failures could violate this assumption, and the loop would then terminate without a certificate. The surrogate is fit over starting positions only, including the initial heading as a third coordinate was tested but rejected as it produced qualitatively identical failure regions with and without heading. A task where heading or velocity at initialisation strongly influences the failure mode would require a higher-dimensional surrogate and possibly a different type of surrogate.

The certificate is valid when the deployment conditions match the simulation environment used for validation, covering the same arena geometry, obstacle placement, and initial-condition distribution. Any mismatch between simulation and hardware such as unmodelled friction, sensor noise, or physical disturbances is not accounted for and would invalidate the coverage guarantee. Safety during training is also unaddressed since exploration visits unsafe configurations freely, which is acceptable in simulation but would need to be constrained for direct hardware deployment. These gaps are taken up in the next Section.

6.2 Future Work

The most direct extension is to stochastic dynamics with process and measurement noise. But, the difficulty lies in identifying the failure set. Under stochastic dynamics, a starting position may fail only for certain disturbance realisations, so the failure set lives in a higher-dimensional joint space of initial conditions and disturbance sequences. One possible direction is to use a state observer or RNN to estimate the noise levels present when failures occur, and then retrain the policy by injecting disturbances at those identified levels. The general structure of the loop, certifying from rollouts and refining on identified failures, carries over naturally to this setting.

Dynamic obstacles present a similar challenge. The STL semantics and timer construction extend naturally since predicates can depend on any time-varying signal. The difficulty is again in identifying where failures concentrate, since the failure set now depends on the trajectories of other agents rather than on static geometry alone. Conformal prediction for trajectory forecasting could be used to construct high-probability sets over obstacle motion and integrate these into the failure identification step.

A different direction concerns the specification itself and connects back to the continual learning mechanisms evaluated in this thesis. For long-horizon tasks or deeply nested temporal structures, training a single policy end to end on the full STL formula becomes expensive as the episode length grows, and the policy may struggle to resolve competing subgoal deadlines across a long horizon. A hierarchical architecture could address this by decomposing the specification into shorter subtasks, each handled by a lower-level policy, with a high-level planner sequencing them according to the temporal structure of the formula. The continual learning mechanisms evaluated here would transfer naturally since different fragments of the same specification can require qualitatively different behaviours, exactly as different tasks do in the standard continual learning literature. Applying EWC and PNN across fragments of a decomposed specification, where each fragment defines a distinct subtask, would be an interesting use of these approaches and a natural counterpart to the multi-task continual learning benchmarks on which they were originally evaluated.

Safety during training is the most practically significant gap between the current framework and deployment on hardware. A shield or constrained policy update would keep exploration inside a safe set without disturbing certification, since the certificate is computed on whatever policy results from training. Online monitoring can be paired with the current approach, it can be used to predict future violations from the trajectory history, and fallback actions can be taken proactively when the predictor indicates an increased risk of constraint violations in subsequent steps.

Finally, the sensitivity of the loop to its hyperparameters has not been studied. In particular, it is unclear how the number of rounds changes with the EWC penalty λ_{EWC} and the biased-resampling weight β, since only two configurations were tested. A useful practical extension would be to develop a diagnostic that predicts whether a new round is likely to reduce violations before the round is completed. For example, monitoring the variance of policy outputs in the retained region during training could indicate whether the policy is adapting effectively.

Appendix A

Setup 3: Random-Goal Navigation

This appendix reports a third experiment on the same TurtleBot3 Burger platform. At each episode reset, a goal position is sampled uniformly from a set of pre-validated candidates spread across the arena, so the policy must reach and stay at a location it does not know until the episode begins.

A.1 Environment and Specification

The robot operates in a 2.0 m square arena containing two circular obstacles of radius 0.30 m centred at $(-0.55, -0.55)$ and $(0.55, 0.55)$. At each reset, a goal centre is drawn uniformly from 20 pre-set candidates. The robot spawns at a position drawn uniformly from the obstacle-free region with 0.15 m clearance from walls and obstacles, heading $\vartheta_0 \sim \mathrm{Unif}[0, 2\pi)$, and zero initial velocity.

The STL specification is

$$\varphi_3 \;=\; \square_{[0,\,25]}\,\phi_{\text{arena}} \;\wedge\; \square_{[0,\,25]}\,\phi_{\text{obst}} \;\wedge\; \lozenge_{[0,\,25]}\,\square_{[0,\,5]}\,\phi_{\text{goal}}, \tag{A.1}$$

with time bounds in seconds. In plain terms, the robot must reach the episode-specific goal and stay there for 5 seconds while always remaining inside the arena and clear of the two obstacles. The specification horizon is $T_\varphi = 25\,\text{s}$, giving $H = 2500$ simulation steps. The predicate robustness functions follow the same form as (5.4), with ϕ_{goal} evaluated against the sampled goal centre. The augmented state carries two timer channels ($N_e = 1$, $N_a = 1$) exactly as in Setup 1, and the nesting hypothesis of Proposition 3.2.1 is satisfied. The observation is 10-dimensional: position, heading, body velocities, distance and bearing to the current goal, and the two timer signals.

A.2 Results

The certification target is $E = 0.03$, $\delta = 0.02$, with calibration sets of $N_{cal} = 1024$ per round. Both methods start from the same Round 0 policy.

Table A.1: Per-round results for EWC on Setup 3. Wall-clock times are cumulative from Round 0.

k	n_{viol}	Success rate	$\bar{\rho} \pm \sigma_\rho$	ϵ_{CP}	Certified
0	48	0.9531	0.1346 ± 0.1754	0.0592	No
1	35	0.9658	0.1502 ± 0.1566	0.0451	No
2	28	0.9727	0.1398 ± 0.1348	0.0373	No
3	15	0.9854	0.1260 ± 0.1070	0.0225	Yes

Table A.2: Per-round results for PNN on Setup 3.

k	n_{viol}	Success rate	$\bar{\rho} \pm \sigma_\rho$	ϵ_{CP}	Certified
0	49	0.9521	0.1046 ± 0.1767	0.0603	No
1	39	0.9619	0.1081 ± 0.1725	0.0495	No
2	38	0.9629	0.1124 ± 0.1568	0.0484	No
3	38	0.9629	0.1130 ± 0.1498	0.0484	No
4	35	0.9658	0.1154 ± 0.1426	0.0451	No
5	39	0.9619	0.1149 ± 0.1428	0.0495	No
6	24	0.9766	0.1253 ± 0.1089	0.0328	Yes

EWC certifies at Round 3 with $\epsilon_{\text{CP}} = 0.0255$ and a success rate of 0.985. The violation count falls from 47 at Round 0 to 14 at Round 3, following the same tail-compression pattern seen in Setup 1: σ_ρ shrinks while $\bar{\rho}$ rises modestly.

PNN does not certify during training: the Clopper-Pearson bound remains above $E = 0.03$ through all seven rounds, ending at $\epsilon_{\text{CP}} = 0.0308$ at Round 6 with a success rate of 0.9766. However, the 500-set repeated-calibration protocol of Section 5.2.3 tells a different story.

Table A.3: Repeated-calibration results for Setup 3 using 500 independently seeded sets of $N_{cal} = 1024$ rollouts each, at $E = 0.03$, $\delta = 0.02$.

Method	Mean SR	$\bar{\rho} \pm \sigma_\rho$	Sets with SR > 0.97	Certified
EWC (Round 3)	0.9846	0.1654 ± 0.0028	500/500	Yes
PNN (Round 6)	0.9802	0.1589 ± 0.0039	490/500	Yes

Over 500 independent calibration draws, EWC certifies on all 500 sets, while PNN certifies on 490 sets. With $\delta = 0.02$, the PAC framework allows up to 2% of calibration sets to yield a bound above E, so PNN's 10 failing sets are exactly at the permitted non-certification rate. However, this result should not be interpreted as strong evidence that PNN satisfies the target with substantial margin. The fact that 490 of 500 sets certify suggests that its violation rate is close to the certification boundary, with relatively little margin for error. The same effect was observed in the in-training calibration sets during Rounds 1–6, which fell into the non-certifying tail. Thus, the 500-set evaluation indicates that PNN is approximately at the target threshold rather than comfortably below it. This also highlights the role of δ: the guarantee limits the probability of an unfavorable calibration draw, but it does not prevent an individual calibration set from producing a non-certifying bound.

Appendix B

Experimental Configuration

This appendix lists the state layouts, reward definitions, termination conditions, and parameter values of the two setups of Chapter 5, following the notation of Chapters 3 and 4. Settings shared by both setups are not repeated here. PPO settings are in Table 4.1 and certification pipeline settings in Table 4.2.

B.1 Setup 1

Augmented State

The observation $x_t^{\text{obs}} \in \mathbb{R}^8$ of (3.13) carries proprioceptive quantities (indices 0 to 5) and goal-relative features (indices 6 and 7). With the timer vector $s_\varphi(t) \in \mathbb{R}^2$ of (3.19) the augmented state is $s_t \in \mathbb{R}^{10}$, given in Table B.1. Obstacle coordinates are omitted for the reason given in Section 4.2.1.

Table B.1: Setup 1 augmented state $s_t = [x_t^{\text{obs}} \mid s_\varphi(t)] \in \mathbb{R}^{10}$.

Index	Symbol	Description	Unit
0	$q_{x,t}$	Arena-frame x position	m
1	$q_{y,t}$	Arena-frame y position	m
2	ϑ_t	Robot heading (yaw)	rad
3	$v_{x,t}$	World-frame linear velocity, x	m/s
4	$v_{y,t}$	World-frame linear velocity, y	m/s
5	$\omega_{z,t}$	Angular velocity about z	rad/s
6	$d_{\text{goal},t}$	Euclidean distance to goal	m
7	$b_{\text{goal},t}$	Bearing to goal relative to heading	rad
8	$\tau_{1,t}^{\text{eventually}}$	Deadline fraction, Eq. (3.14)	–
9	$\tau_{1,t}^{\text{always}}$	Stay fraction, Eq. (3.18)	–

The bearing $b_{\text{goal},t}$ is the angle from the robot's heading to the goal, wrapped to $(-\pi, \pi]$.

Reward

Eight dense terms are computed at every step and scaled by a global return factor,

$$r_{\text{aux}}(s_t, a_t) \;=\; \lambda_r \sum_{j=1}^{8} r_{j,t}, \tag{B.1}$$

with the terminal robustness $\alpha\,\rho_{\varphi_1}(\xi)$ appended at episode end per (3.25). Table B.2 links each term to the conjunct of (5.3) it shapes.

Table B.2: Setup 1 reward terms. Term 9 is the terminal robustness and is not part of the per-step sum (B.1). Weights are in Table B.3.

j	Term	Conjunct shaped	Definition
1	Distance progress	$\Diamond\Box\,\phi_{\text{goal}}$	Eq. (B.2)
2	Bearing alignment	$\Diamond\Box\,\phi_{\text{goal}}$	$r_{\text{bear}}\cos b_{\text{goal},t}$
3	Obstacle proximity	$\Box\,\phi_{\text{obst}}$	Eq. (B.3)
4	Wall proximity	$\Box\,\phi_{\text{arena}}$	Eq. (B.4)
5	Time-step cost	none	$-r_{\text{step}}$
6	Goal entry bonus	$\Diamond\Box\,\phi_{\text{goal}}$	$r_{\text{entry}}\mathbb{1}[\text{first entry}]$
7	Arena exit penalty	$\Box\,\phi_{\text{arena}}$	$-r_{\text{pen}}\mathbb{1}[\text{arena exit}]$
8	Obstacle contact penalty	$\Box\,\phi_{\text{obst}}$	$-r_{\text{pen}}\mathbb{1}[\text{contact}]$
9	Terminal STL robustness	φ_1 as a whole	$\alpha\,\rho_{\varphi_1}(\xi)$

The distance term rewards reduction in goal distance, amplified as the deadline is consumed. The two constraint terms are quadratic profiles active inside a safety margin.

$$r_{1,t} = r_{\text{dist}}\left(1 + r_{\text{time}}\,\tau_{1,t}^{\text{eventually}}\right)\left(d_{\text{goal},t-1} - d_{\text{goal},t}\right), \tag{B.2}$$

$$r_{3,t} = -r_{\text{obst}}\left(\frac{d_{\text{obst}}^{\text{safe}} - \rho_{\phi_{\text{obst}}}(x_t)}{d_{\text{obst}}^{\text{safe}}}\right)^2 \mathbb{1}\left[\rho_{\phi_{\text{obst}}}(x_t) < d_{\text{obst}}^{\text{safe}}\right], \tag{B.3}$$

$$r_{4,t} = -r_{\text{wall}}\left(\frac{d_{\text{wall}}^{\text{safe}} - \rho_{\phi_{\text{arena}}}(x_t)}{d_{\text{wall}}^{\text{safe}}}\right)^2 \mathbb{1}\left[\rho_{\phi_{\text{arena}}}(x_t) < d_{\text{wall}}^{\text{safe}}\right]. \tag{B.4}$$

Two of these forms were arrived at by fixing observed failures. Without the urgency factor in (B.2) the policy orbited obstacles for marginal proximity gains instead of committing to the goal, undermining the $\Diamond_{[0,45]}$ bound. An earlier version fired the entry bonus on every re-entry, which the policy exploited by oscillating across the goal boundary rather than stabilising inside it, in direct conflict with the $\Box_{[0,5]}$ requirement. Restricting the bonus to a single firing removed this behaviour.

Termination

An episode ends on arena exit ($\max(|q_{x,t}|, |q_{y,t}|) > d_{\text{arena}}/2$), on obstacle collision ($\|\mathbf{q}_t - \mathbf{o}_i\| < \text{rad}_i$ for some i), or on timeout at H steps. The first two are *terminations*: by Remark 3.2.5 stopping early does not change the sign of $\rho_{\varphi_1}(\xi)$, so no value bootstrap is applied. Timeout is a *truncation* and bootstraps from $V_\psi(s_H)$, per Section 4.3.1.

Parameters

Table B.3: Setup 1 parameters. Shared PPO and certification pipeline settings are in Tables 4.1 and 4.2.

Parameter	Symbol	Value
Environment		
Arena side length	d_{arena}	2.0 m
Goal tolerance	d_{tol}	0.20 m
Simulation time step	Δt	0.01 s
Episode length	T_{φ}	45.0 s
Spawn clearance	–	0.15 m
Goal deadline, stay duration	$T_{d,1}$, $T_{a,1}$	45.0 s, 5.0 s
Reward weights		
Distance progress	r_{dist}	30.0
Urgency multiplier	r_{time}	1.0
Bearing alignment	r_{bear}	1.0
Obstacle proximity	r_{obst}	15.0
Obstacle safety margin	$d_{\text{obst}}^{\text{safe}}$	0.25 m
Wall proximity	r_{wall}	12.0
Wall safety margin	$d_{\text{wall}}^{\text{safe}}$	0.20 m
Step cost	r_{step}	0.01
Goal entry bonus	r_{entry}	220.0
Safety and collision penalty	r_{pen}	260.0
Global return scale	λ_r	0.01
Terminal robustness scale	α	200.0
Certification		
Calibration set size	N_{cal}	1024
Target violation rate, confidence	E, δ	0.03, 0.02

B.2 Setup 2

Conventions follow Setup 1. Values shared with it (d_{tol}, Δt, spawn clearance, λ_r, r_{pen}, ε, δ) are repeated in Table B.6.

Augmented State

The observation $x_t^{\text{obs}} \in \mathbb{R}^{10}$ adds obstacle-relative features (indices 8 and 9) to the Setup 1 layout. With six timer channels, one pair per goal, the augmented state is $s_t \in \mathbb{R}^{16}$, given in Table B.4.

Three choices differ from Setup 1. Indices 6 and 7 track the *active* goal: the nearer of $\mathcal{G}_1$ and $\mathcal{G}_2$ during phase 1, switching to $\mathcal{G}_3$ in phase 2 and zeroed once the specification is complete. Indices 8 and 9 give the displacement vector to the nearest obstacle surface, included because

Table B.4: Setup 2 augmented state $s_t = [x_t^{\text{obs}} \mid s_\varphi(t)] \in \mathbb{R}^{16}$.

Index	Symbol	Description	Unit
0	$q_{x,t}$	Arena-frame x position	m
1	$q_{y,t}$	Arena-frame y position	m
2	ϑ_t	Robot heading (yaw)	rad
3	$v_{x,t}$	World-frame linear velocity, x	m/s
4	$v_{y,t}$	World-frame linear velocity, y	m/s
5	$\omega_{z,t}$	Angular velocity about z	rad/s
6	$d_{\text{active},t}$	Distance to active goal	m
7	$b_{\text{active},t}$	Bearing to active goal	rad
8	$d_{o,x,t}$	Relative x to nearest obstacle surface	m
9	$d_{o,y,t}$	Relative y to nearest obstacle surface	m
10, 12, 14	$\tau^{\text{eventually}}_{\mathcal{G}_j,t}$	Deadline fraction for $\mathcal{G}_j$	–
11, 13, 15	$\tau^{\text{always}}_{\mathcal{G}_j,t}$	Stay fraction for $\mathcal{G}_j$, Eq. (B.5)	–

the rectangular obstacles create tighter passages than the circular ones of Setup 1. The always fractions report normalised robustness rather than elapsed time.

$$\tau^{\text{always}}_{\mathcal{G}_j,t} = \text{clip}\left(\frac{\rho^{\text{stay}}_{\mathcal{G}_j,t} + d_{\text{tol}}}{2\,d_{\text{tol}}},\ 0,\ 1\right), \tag{B.5}$$

where $\rho^{\text{stay}}_{\mathcal{G}_j,t}$ is the running minimum predicate value since the robot most recently entered $\mathcal{G}_j$, maintained by the inner accumulator of Section 3.2.4 and rescaled to $[0,1]$. The channel resets on leaving the region and latches at 1 once the stay requirement is met, preserving the semantics of Proposition 3.2.1. After phase 1 acceptance the $\mathcal{G}_1$ and $\mathcal{G}_2$ channels latch and their deadline fractions freeze.

Reward

The per-step terms and the terminal robustness $\alpha\,\rho_{\varphi_2}(\xi)$ form the episode reward $r_{\text{tot}}(\xi)$ of (3.23). Table B.5 lists the terms, where $j \in \{1,2\}$ during phase 1 and $j = 3$ during phase 2. Terms 2, 4, and 5 are zeroed once the specification is satisfied.

The terms with non-trivial structure are the deadline funnel, the two phase bonuses, and the post-specification safety shaping.

$$r_{5,t} = r_{\text{fun}}\tanh\big((d_{\text{ref},t} - d_{\text{active},t})/d_{\text{fun}}\big), \tag{B.6}$$

$$r_{9,t} = \Big(r_{\text{p1}} + r^{\text{early}}_{\text{p1}}\,\text{clip}(1 - t\,\Delta t/T_{d,j},\ 0,\ 1)\Big)\mathbb{1}[\text{phase 1 just accepted}], \tag{B.7}$$

$$r_{10,t} = \Big(r_{\text{p2}} + r^{\text{early}}_{\text{p2}}\,\text{clip}(1 - t\,\Delta t/T_{\varphi},\ 0,\ 1)\Big)\mathbb{1}[\mathcal{G}_3 \text{ just accepted}], \tag{B.8}$$

$$r_{\text{post},t} = r_{\text{post}}\,\text{clip}\big(\min(\rho_{\phi_{\text{arena}}}(x_t),\ \rho_{\phi_{\text{obst}}}(x_t)),\ \pm 0.5\big)\mathbb{1}[\text{spec complete}], \tag{B.9}$$

where $d_{\text{ref},t}$ interpolates linearly from the initial goal distance to d_{tol} over the active deadline, $\hat{\mathbf{u}}_t$ is the unit vector toward the active goal, and $T_{d,j}$ in (B.7) is the deadline of whichever goal

Table B.5: Setup 2 reward terms. Weights are in Table B.6. Terms 9 and 10 fire at most once per phase.

j	Term	Conjunct shaped	Definition
1	Step cost	none	$-r_{\text{step}}$
2	Distance progress	φ_{p1}, φ_{p2}	$r_{\text{dist}}\,\text{clip}(d_{\text{active},t-1} - d_{\text{active},t},\, \pm 0.06)$
3	Robustness progress	φ_2 as a whole	$r_{\text{rob}}\,\text{clip}(\rho_{\varphi_2,t} - \rho_{\varphi_2,t-1},\, \pm 0.06)$
4	Radial velocity	φ_{p1}, φ_{p2}	$r_{\text{rad}}\,\text{clip}\big(\mathbf{v}_t^{\top}\,\hat{\mathbf{u}}_t / v_{\max},\, \pm 1\big)$
5	Deadline funnel	φ_{p1}, φ_{p2}	Eq. (B.6)
6	Obstacle proximity	$\square\,\phi_{\text{obst}}$	Same profile as (B.3), summed over obstacles
7	Control regularisation	none	$r_{\text{lin}}\,a_{v,t}^2 + r_{\text{ang}}\,a_{\omega,t}^2 + r_{\text{smooth}}\,\|a_t - a_{t-1}\|^2$
8	Stay robustness	active $\square$ operator	$r_{\text{stay}}\,\text{clip}(\rho^{\text{stay}}_{\mathcal{G}_j,t},\, \pm 0.10)\,\mathbb{1}[\text{inside } \mathcal{G}_j]$
9	Phase 1 bonus	φ_{p1}	Eq. (B.7)
10	Phase 2 bonus	φ_{p2}	Eq. (B.8)
–	Post-spec safety	$\square(\phi_{\text{arena}} \wedge \phi_{\text{obst}})$	Eq. (B.9)
–	Full-spec bonus	φ_2 as a whole	$r_{\text{spec}}\,\mathbb{1}[\text{spec done},\, \rho_{\varphi_2}(\xi) > 0,\, t = H]$
–	Terminal robustness	φ_2 as a whole	$\alpha\,\rho_{\varphi_2}(\xi)$

was accepted. Both phase bonuses pay more for early satisfaction. Once the specification is complete the goal-directed terms switch off and (B.9) keeps the robot clear of hazards for the rest of the horizon.

Robustness progress (term 3) has no Setup 1 counterpart and was added because the phased structure of φ_2 creates plateaus where per-conjunct shaping stalls while overall robustness can still improve. The obstacle margin is larger (0.30 m against 0.25 m) to account for the tighter passages between rectangular obstacles.

Termination

An episode ends on arena exit, on obstacle collision ($\min_i d_{\text{box},i} < -0.05\,\text{m}$), on phase 1 deadline expiry ($t\,\Delta t > T_{d,2} + T_{a,2}$ with phase 1 not accepted), on phase 2 deadline expiry ($t\,\Delta t > T_{d,3} + T_{a,3}$ with the specification incomplete), or on timeout at $H = 9000$ steps. The first four are terminations and the last a truncation, with the bootstrap rule of Section B.1 unchanged. The two deadline conditions have no Setup 1 counterpart and fire when a goal-reaching subformula can no longer be satisfied within the remaining time.

The predicate $\rho_{\phi_{\text{obst}}}$ of (5.10) turns negative at $d^{\text{tol}}_{\text{safe}} = 0.10\,\text{m}$ while termination fires only at 0.05 m of penetration. Because the safety conjunct records the worst clearance seen over the episode, $\rho_{\varphi_2}(\xi)$ is negative from the first crossing onward and the extra steps change neither its sign nor the certification count.

Parameters

Table B.6: Setup 2 parameters. Shared PPO and certification pipeline settings are in Tables 4.1 and 4.2.

Parameter	Symbol	Value
Environment		
Arena side length	d_{arena}	4.0 m
Goal tolerance	d_{tol}	0.20 m
Obstacle safety margin	$d_{\text{safe}}^{\text{tol}}$	0.10 m
Simulation time step	Δt	0.01 s
Episode length	T_{φ}	90.0 s
Spawn clearance	–	0.15 m
Goal deadlines	$T_{d,1}$, $T_{d,2}$, $T_{d,3}$	42.0, 45.0, 80.0 s
Goal stay durations	$T_{a,1}$, $T_{a,2}$, $T_{a,3}$	5.0, 5.0, 7.0 s
Reward weights		
Step cost	r_{step}	0.03
Distance progress	r_{dist}	30.0
Robustness progress	r_{rob}	40.0
Radial velocity	r_{rad}	1.80
Deadline funnel weight	r_{fun}	1.35
Deadline funnel scale	d_{fun}	0.18 m
Obstacle proximity	r_{obst}	18.0
Obstacle safety margin	$d_{\text{obst}}^{\text{safe}}$	0.30 m
Linear control penalty	r_{lin}	0.001
Angular control penalty	r_{ang}	0.012
Smoothness penalty	r_{smooth}	0.0235
Stay robustness	r_{stay}	20.0
Phase 1 bonus	r_{p1}	200.0
Phase 1 early bonus	$r_{\text{p1}}^{\text{early}}$	80.0
Phase 2 bonus	r_{p2}	320.0
Phase 2 early bonus	$r_{\text{p2}}^{\text{early}}$	120.0
Full-spec completion bonus	r_{spec}	500.0
Post-spec safety weight	r_{post}	1.2
Safety and collision penalty	r_{pen}	260.0
Global return scale	λ_r	0.01
Terminal robustness scale	α	200.0
Certification and refinement		
Calibration set size	N_{cal}	2048
Target violation rate, confidence	E, δ	0.03, 0.02
EWC penalty	λ_{EWC}	0.05
Refinement learning rate	–	2×10^{-4}
Refinement iterations per round	–	2000

Bibliography

[1] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M. Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 81–87, 2014.

[2] Sadra Sadraddini and Calin Belta. Robust temporal logic model predictive control. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 772–779, 2015.

[3] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, page 239–248, New York, NY, USA, 2015. Association for Computing Machinery.

[4] Paulo Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[5] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. Formal methods for control synthesis: An optimization perspective. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:115–141, 2018.

[6] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon temporal logic planning. In *IEEE Conference on Decision and Control (CDC)*, 2010. Related to the TuLiP toolbox ecosystem for temporal-logic-based synthesis and planning.

[7] Derya Aksaray, Austin Jones, Zhaodan Kong, Mac Schwager, and Calin Belta. Q-learning for robust satisfaction of signal temporal logic specifications. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 6565–6570. IEEE, 2016.

[8] Peter Varnai and Dimos V Dimarogonas. On robustness metrics for learning stl tasks, 2020.

[9] Naman Saxena, Sandeep Gorantla, and Pushpak Jagtap. Funnel-based reward shaping for signal temporal logic tasks in reinforcement learning. *IEEE Robotics and Automation Letters*, 9(2):1373–1379, 2024.

[10] Junya Ikemoto and Toshimitsu Ushio. Deep reinforcement learning under signal temporal logic constraints using lagrangian relaxation. *IEEE Access*, 10:114814–114828, 2022.

[11] Yue Meng, Fei Chen, and Chuchu Fan. Tgpo: Temporal grounded policy optimization for signal temporal logic tasks. *arXiv preprint arXiv:2510.00225*, 2025.

[12] Manas Sashank Juvvi, Tushar Dilip Kurne, Vaishnavi J, Shishir Kolathaya, and Pushpak Jagtap. Signal temporal logic compliant co-design of planning and control. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 19204–19209, 2025.

[13] 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 and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[14] Aaron D. Ames, Xiangru Xu, Jessy W. Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2017.

[15] Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient smt solver. volume 4963, pages 337–340, 04 2008.

[16] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Towards verified artificial intelligence, 2020.

[17] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *2013 Formal Methods in Computer-Aided Design*, pages 1–8, 2013.

[18] Alessandro Abate, Daniele Ahmed, Alec Edwards, Mirco Giacobbe, and Andrea Peruffo. Fossil: a software tool for the formal synthesis of lyapunov functions and barrier certificates using neural networks. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, HSCC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[19] He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 686–701, New York, NY, USA, 2019. Association for Computing Machinery.

[20] Xinyu Wang, Luzia Knoedler, Frederik Baymler Mathiesen, and Javier Alonso-Mora. Simultaneous synthesis and verification of neural control barrier functions through branch-and-bound verification-in-the-loop training. In *2024 European Control Conference (ECC)*, pages 571–578, 2024.

[21] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and Aravinda Prasad Sistla, editors, *Computer Aided Verification*, pages 154–169, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[22] Rudi Coppola and Manuel Mazo Jr. On training-conditional conformal prediction and binomial proportion confidence intervals, 2025.

[23] Lars Lindemann, Xin Qin, Jyotirmoy V. Deshmukh, and George J. Pappas. Conformal prediction for stl runtime verification. In *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)*, ICCPS '23, page 142–153, New York, NY, USA, 2023. Association for Computing Machinery.

[24] Lars Lindemann, Yiqi Zhao, Xinyi Yu, George J. Pappas, and Jyotirmoy V. Deshmukh. Formal verification and control with conformal prediction. *ArXiv*, abs/2409.00536, 2024.

[25] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[26] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 92–106. Springer, 2010.

[27] Richard S Sutton, Andrew G Barto, et al. Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134, 1999.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] 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. , 518(7540):529–533, February 2015.

[30] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018.

[31] Vladimir Vovk. Conditional validity of inductive conformal predictors. In Steven C. H. Hoi and Wray Buntine, editors, *Proceedings of the Asian Conference on Machine Learning*, volume 25 of *Proceedings of Machine Learning Research*, pages 475–490, Singapore Management University, Singapore, 04–06 Nov 2012. PMLR.

[32] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017.

[33] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2022.

[34] Maciej Wołczyk, MichałZając, Razvan Pascanu, Ł ukasz Kuciński, and Piotr Miłoś. Continual world: A robotic benchmark for continual reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 28496–28510. Curran Associates, Inc., 2021.

[35] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2022.

[36] Harish Venkataraman, Derya Aksaray, and Peter Seiler. Tractable reinforcement learning of signal temporal logic objectives. In Alexandre M. Bayen, Ali Jadbabaie, George Pappas, Pablo A. Parrilo, Benjamin Recht, Claire Tomlin, and Melanie Zeilinger, editors, *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, volume 120 of *Proceedings of Machine Learning Research*, pages 308–317. PMLR, 10–11 Jun 2020.

[37] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[38] Clemens Schwarke, Mayank Mittal, Nikita Rudin, David Hoeller, and Marco Hutter. Rsl-rl: A learning library for robotics research. *arXiv preprint arXiv:2509.10771*, 2025.

Declaration on the Use of LLM's

In accordance with the TU Delft guidelines on the use of generative artificial intelligence in academic work, the author declares the following regarding the use of AI tools during the preparation of this thesis.

Large language model assistants, including Claude by Anthropic and Google Gemini, were used during the writing process for LaTeX refinement and code support. Google Gemini was also used to create the cover image.

Delft, August 10, 2026

Manas Sashank Juvvi

Glossary

List of Acronyms

CBF Control Barrier Function

CEGAR Counterexample-Guided Abstraction Refinement

CEGIS Counterexample-Guided Inductive Synthesis

CI Confidence Interval

CPU Central Processing Unit

CP Conformal Prediction

DQN Deep Q-Network

EWC Elastic Weight Consolidation

GAE Generalized Advantage Estimation

GPU Graphics Processing Unit

LTL Linear Temporal Logic

LSTM Long Short-Term Memory

MDP Markov Decision Process

MILP Mixed-Integer Linear Program

MLP Multi-Layer Perceptron

MPC Model Predictive Control

PAC Probably Approximately Correct

PNN Progressive Neural Network

PPO Proximal Policy Optimization

RAM Random-Access Memory

RBF Radial Basis Function

ReLU Rectified Linear Unit

RL Reinforcement Learning

RNN Recurrent Neural Network

SAC Soft Actor-Critic

SMT Satisfiability Modulo Theories

STL Signal Temporal Logic

VRAM Video Random-Access Memory

List of Symbols

Note: Only the important symbols recurring throughout the thesis are listed here.

System and Dynamics

$x_t \in \mathcal{X}$	State of the dynamical system at step t
$a_t \in \mathcal{U}$	Action (control input) at step t
f	Transition map, $x_{t+1} = f(x_t, a_t)$; treated as a black box
Δt	Simulation time step
$\xi = x_{0:H}$	Closed-loop trajectory $(x_0, x_1, \ldots, x_H)$
H	Episode length in simulation steps, $H = \lceil T_\varphi / \Delta t \rceil$

Signal Temporal Logic

φ	STL specification formula
ϕ	Predicate; holds at t iff $h(x_t) \geq 0$
$\rho_\varphi(\xi)$	Quantitative robustness; $\rho_\varphi(\xi) \geq 0 \Leftrightarrow \xi \models \varphi$
T_φ	Specification horizon (seconds)
$\Diamond_{[t_1,t_2]}$, $\Box_{[t_1,t_2]}$	Bounded *eventually* and *always* operators
$T_{s,i}$, $T_{d,i}$, $T_{a,i}$	Start, deadline, and stay duration for sub-clause i

MDP, Policy, and Training

$\mathcal{M}_\varphi$	Augmented MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r_{\text{tot}}, T_\varphi, \gamma)$
s_t	Augmented state $[x_t^{\text{obs}} \mid s_\varphi(t)]$
$\tau_i^{\text{ev}},\ \tau_i^{\text{al}}$	Eventually fraction and always fraction (timer signals)
π_θ	Stochastic policy parameterised by θ
$\bar{\pi}_\theta$	Deterministic mean policy used at deployment
$r_{\text{tot}}(\xi)$	Total episode reward, $\alpha\, \rho_\varphi(\xi) + \sum_t r_{\text{aux}}(s_t, a_t)$
r_{aux}	Step-level auxiliary (dense shaping) reward
α	Terminal robustness scaling coefficient
γ	Discount factor
$\hat{A}_t$	GAE advantage estimate at step t
λ_{GAE}	GAE decay parameter

Statistical Certification

$\mathcal{V}(\bar{\pi}_\theta)$	Violation rate, $\mathbb{P}_{x_0 \sim \mu_0}[\rho_\varphi < 0]$
E, δ	PAC targets; policy certified when $\mathbb{P}[\mathcal{V} \leq E] \geq 1 - \delta$
μ_0	True (unknown) distribution over initial conditions
$\mathcal{X}_0$	Support of μ_0
$\mathcal{D}_{\text{cal}}^{(k)}$	Calibration set of N_{cal} i.i.d. rollouts at round k
N_{cal}	Calibration set size
n_{viol}	Number of violating rollouts in $\mathcal{D}_{\text{cal}}^{(k)}$
ϵ_{CP}	Clopper–Pearson upper bound on $\mathcal{V}(\bar{\pi}_\theta)$

Counterexample Extraction and Refinement

$\hat{R}_k$	Counterexample region extracted at round k
$\tilde{\mu}_0^{(k)}$	Biased training distribution, $\beta\, \text{Unif}(\hat{R}_k) + (1-\beta)\, \mu_0$
β	Mixing weight for biased resampling
$k,\ K_{\max}$	Current and maximum refinement round

Continual Learning

$F_j^{(k)}$	Diagonal Fisher information for weight θ_j at round k
λ_{EWC}	EWC regularisation strength
γ_F	Fisher decay factor for online accumulation
$\mathcal{F}_{k,\rho},\ \mathcal{FT}_{k,\rho}$	Forgetting and forward-transfer metrics at round k