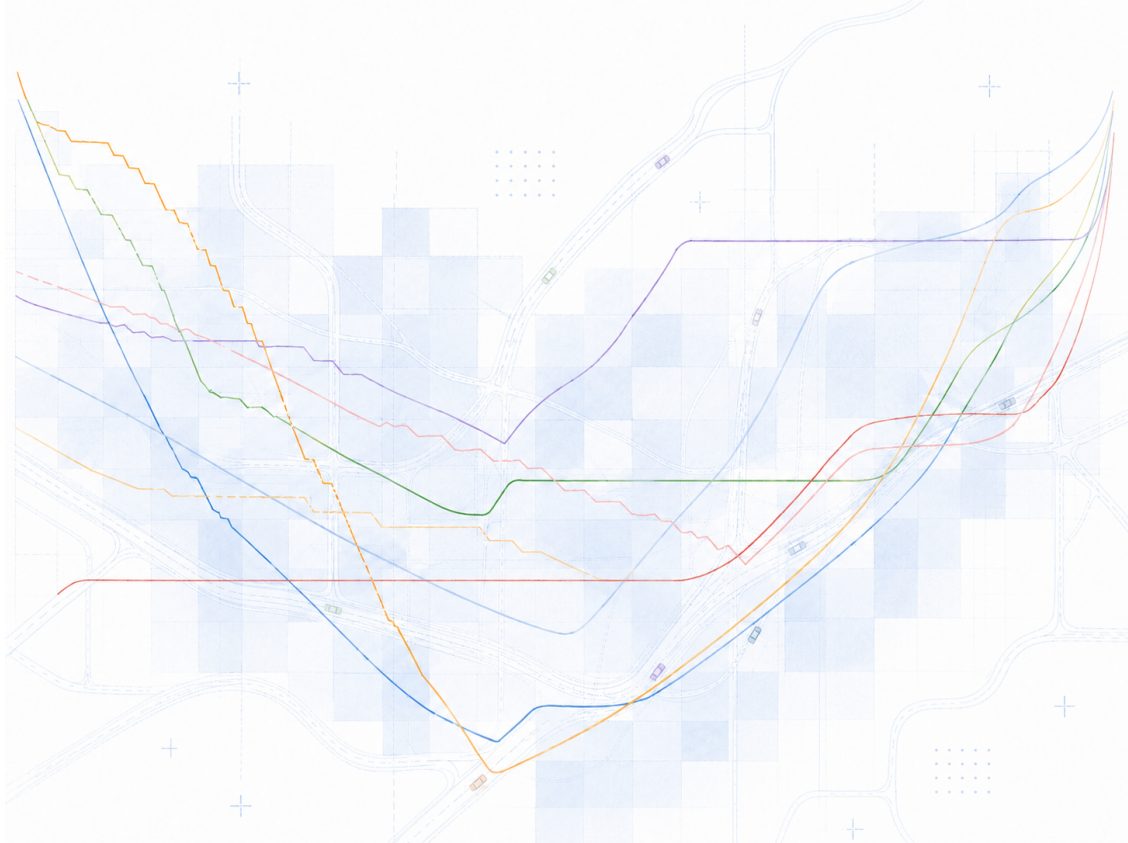




CHALMERS
UNIVERSITY OF TECHNOLOGY



Falsification of Safety-Critical Decision and Control Software

Master's thesis in Systems, Control and Mechatronics

PINGJIE ZHANG & YANG ZHANG

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

MASTER'S THESIS 2026

Falsification of Safety-Critical Decision and Control Software

PINGJIE ZHANG & YANG ZHANG



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

Falsification of Safety-Critical Decision and Control Software
PINGJIE ZHANG & YANG ZHANG

© PINGJIE ZHANG & YANG ZHANG, 2026.

Supervisor: Jonas Krook, Zenseact
Yuvaraj Selvaraj, Zenseact
Examiner: Nikolce Murgovski, Department of Electrical Engineering

Master's Thesis 2026
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover illustration generated with the assistance of ChatGPT, based on the falsification results of this thesis on reinforcement learning-based search for safety-critical collision trajectories

Typeset in L^AT_EX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Falsification of Safety-Critical Decision and Control Software

PINGJIE ZHANG

YANG ZHANG

Department of Electrical Engineering

Chalmers University of Technology

Abstract

Safety testing of autonomous driving systems requires methods that can expose rare but safety-critical failures beyond nominal driving behavior. Since exhaustive real-world testing is infeasible, simulation-based falsification provides a practical way to actively search for scenarios in which a system under test violates desired safety requirements. This thesis investigates deep reinforcement learning for falsification of safety-critical decision and control software in closed-loop simulation. The falsification task is formulated as an adversarial control problem in a simplified longitudinal two-vehicle scenario, where a learning agent controls the target vehicle and attempts to induce collisions with an ego vehicle controlled by the system under test. A TD3-based falsifier is first evaluated against Monte Carlo Tree Search and uniform random sampling to study collision discovery performance across randomly sampled initial conditions. The thesis then extends the framework towards diversity-oriented falsification using a DTW-based(dynamic time wrapping) Series RL method. In this method, previously discovered collision trajectories are stored in a repository and used to reward later agents for finding trajectories with different gap and relative-speed evolution. A collision replay buffer is also introduced to increase the reuse of rare collision-related experiences during training. The results show that the TD3-based falsifier discovers collisions more effectively than the non-learning baselines in the tested setting. They also show that the DTW-based diversity reward leads to a broader and less repetitive set of collision trajectories than the baseline Series RL method. These findings suggest that reinforcement learning can support both efficient failure discovery and diversity-oriented exploration of failure modes, although the conclusions are limited to the simplified simulation setting considered in this thesis.

Keywords: falsification, autonomous driving, reinforcement learning, TD3, trajectory diversity, DTW.

Acknowledgements

We would like to express our sincere gratitude to our supervisors at Zenseact, Jonas Krook and Yuvaraj Selvaraj, for their continuous guidance, technical insights, and support throughout this thesis project. Their expertise in autonomous driving safety and their willingness to engage in detailed discussions have been invaluable in shaping both the direction and the depth of this work.

We would also like to thank our examiner, Professor Nikole Murgovski, at Chalmers University of Technology, for his thoughtful feedback, valuable ideas, and for providing us with many possible perspectives and directions that helped improve this thesis.

Finally, we would like to thank the members of Team Franzjäger and our colleagues at Zenseact for the collaborative environment, as well as our families and friends for their patience and encouragement during this period.

Pingjie Zhang & Yang Zhang, Gothenburg, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

DDPG	Deep Deterministic Policy Gradient
DRL	Deep Reinforcement Learning
DTW	Dynamic Time Warping
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
RAND	Uniform Random Sampling
RL	Reinforcement Learning
SUT	System Under Test
TD	Temporal Difference
TD3	Twin Delayed Deep Deterministic Policy Gradient

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

t	Discrete simulation time-step index
i, j	Generic indices for samples, trajectories, grid cells, or DTW sequence elements
k	Series RL training-stage index
m, n	Lengths of two generic DTW sequences

Sets

$\mathcal{S}$	State space
$\mathcal{A}$	Action space
$\mathcal{C}$	Collision-state set
$\mathcal{E}$	Terminal-state set
$\mathcal{D}$	Replay buffer
$\mathcal{T}_k$	Collision-trajectory repository after stage k
$\mathcal{Q}_k$	Candidate collision-trajectory set at stage k
$\mathcal{G}_i$	Set of phase-space grid cells visited by trajectory τ_i

Parameters

T	Simulation horizon
R_{col}	Positive reward assigned to collision
c_0	Base penalty for non-collision terminal episodes

c_g	Terminal-gap penalty coefficient
λ_{DTW}	Weight of the DTW-based diversity reward

Variables

$s_t^{(3)}$	Three-dimensional state representation $[g_t, v_{\text{ego},t}, v_{\text{tgt},t}]$
$s_t^{(4)}$	Four-dimensional state representation $[p_{\text{ego},t}, v_{\text{ego},t}, p_{\text{tgt},t}, v_{\text{tgt},t}]$
a_t	Adversarial target-vehicle acceleration action
r_t	Scalar reward at time step t
r_t^{base}	Base falsification reward
r_t^{DTW}	DTW-augmented diversity reward
$p_{\text{ego},t}$	Ego-vehicle longitudinal position
$p_{\text{tgt},t}$	Target-vehicle longitudinal position
$v_{\text{ego},t}$	Ego-vehicle longitudinal velocity
$v_{\text{tgt},t}$	Target-vehicle longitudinal velocity
$b_{\text{ego}}^{\text{max}}$	Ego-vehicle maximum braking decelerations
$b_{\text{tgt}}^{\text{max}}$	Target-vehicle maximum braking decelerations
g_t	Longitudinal gap, $p_{\text{tgt},t} - p_{\text{ego},t}$
v_{rel}	Initial closing velocity, $\max(0, v_{\text{ego},0} - v_{\text{tgt},0})$
Δv_t	Relative speed used for trajectory comparison, $v_{\text{tgt},t} - v_{\text{ego},t}$
τ	Collision trajectory
$D_{\text{DTW}}(\cdot, \cdot)$	Dynamic Time Warping distance
$S_{\text{DTW}}(\tau)$	DTW diversity score for trajectory τ
D_{ij}	Jaccard distance between visited cell sets $\mathcal{G}_i$ and $\mathcal{G}_j$

Contents

List of Acronyms	ix
Nomenclature	xi
List of Figures	xvii
List of Tables	xix
1 Introduction	1
1.1 Background	1
1.2 Objective	1
1.3 Research questions	2
1.4 Limitations	2
1.5 Related work	2
1.6 Outline of the thesis	4
2 Theory	5
2.1 Falsification	5
2.2 Markov decision process	5
2.3 Reinforcement learning	6
2.3.1 Q-value estimation	7
2.3.2 Policy gradient	7
2.3.3 Actor-critic algorithm	8
2.3.4 Off-policy learning in actor-critic	9
2.3.5 Deep Deterministic Policy Gradient	10
2.3.6 Twin Delayed Deep Deterministic Policy Gradient	11
2.4 Diversity-oriented search and evaluation	14
2.4.1 Dynamic Time Warping	14
2.4.2 Series Reinforcement Learning	15
2.4.3 Jaccard distance	15
3 System modeling and problem formulation	17
3.1 Simulation framework	17
3.1.1 Dynamic driving environment	18
3.1.2 Adversarial target vehicle	18
3.1.3 Simulated ego-vehicle components and System Under Test	18
3.1.4 Closed-loop interaction	20

3.2	MDP formulation	20
3.2.1	State space	20
3.2.2	Action space	21
3.2.3	Transition dynamics	21
3.2.4	Reward function	21
3.2.5	Terminal conditions	22
3.3	Problem formulation	23
4	Falsification Efficacy: Cross-Scenario Comparison	25
4.1	Test scenarios	25
4.2	TD3 implementation	26
4.3	Baseline methods	28
4.3.1	Monte Carlo Tree Search	28
4.3.2	Uniform random sampling	28
4.4	Difficulty stratification	28
5	Trajectory Diversity with DTW-based Series RL	31
5.1	Four-state TD3 with collision replay buffer	31
5.1.1	Dynamic sampling ratio between buffers	32
5.1.2	Priority sampling within the collision replay buffer	33
5.2	DTW-based Series RL for trajectory diversity	33
5.2.1	Trajectory representation for DTW	34
5.2.2	DTW-based reward	35
5.2.3	Repository update in series RL	35
5.3	Experiment design	36
5.3.1	Experimental protocol	36
5.3.2	Parameter settings	37
5.4	Evaluation metrics	38
5.4.1	DTW-based insertion score	39
5.4.2	Coverage comparison	39
5.4.3	Pairwise Jaccard distance statistics	40
6	Results	41
6.1	Collision discovery performance	41
6.1.1	Collision rate	41
6.1.2	Stratified difficulty analysis	41
6.1.3	Cases study	44
6.2	Trajectory diversity	46
6.2.1	DTW score comparison	48
6.2.2	Coverage comparison	50
6.2.3	Pairwise Jaccard distance statistics	52
7	Exploratory studies and broader implications	55
7.1	Additional Exploratory studies	55
7.1.1	Algorithm selection	55
7.1.2	Reward shaping	55
7.1.3	Action space design	56

7.1.4	Environmental complexity	56
7.2	Generalization to realistic environments	56
7.3	Practical implications	57
8	Conclusion	59
	Bibliography	61

List of Figures

2.1	The classic reinforcement learning interaction loop	6
2.2	Actor-critic data flow and updates	9
3.1	Closed-loop simulation framework	17
3.2	Architecture of the system under test	19
6.1	Per-cell collision-finding success rates of TD3, MCTS, and RAND across 1000 random scenarios, stratified by safety margin and maxi- mum vehicle speed.	43
6.2	Case 1: Only TD3 succeed	45
6.3	Case 2: TD3 and MCTS succeed	45
6.4	Baseline Series RL repository trajectories.	47
6.5	DTW-based Series RL repository trajectories.	49
6.6	Phase-space coverage of the baseline repository	51
6.7	Phase-space coverage of the DTW-based repository	51

List of Tables

2.1	Summary of the modifications introduced by TD3 over DDPG.	13
4.1	Curriculum schedule used to train the TD3 policy	27
4.2	Bin specification for the difficulty stratification.	29
5.1	Training protocol for the trajectory diversity experiment.	37
5.2	TD3 hyperparameters for the trajectory diversity experiment.	38
5.3	Collision-buffer, reward, and diversity-evaluation parameters.	38
6.1	Collision discovery rate	41
6.2	Collision counts on the 3×3 stratified partition of the test set	42
6.3	DTW scores of repository trajectories for the baseline and DTW-based Series RL methods.	48
6.4	Pairwise Jaccard distances D_{ij} for the baseline Series RL repository. .	52
6.5	Pairwise Jaccard distances D_{ij} for the DTW-based Series RL repository.	52
6.6	Summary statistics of pairwise Jaccard distances.	53

1

Introduction

1.1 Background

Verifying the safety of autonomous driving systems is a fundamental challenge: demonstrating reliable behavior to a statistically meaningful level would require hundreds of millions of test kilometers on public roads [19], making simulation-based testing essential.

Falsification offers a systematic approach within this setting, by posing safety testing as an optimization problem that actively searches for scenarios in which the system under test (SUT) violates predefined safety requirements [8]. Reinforcement learning (RL) is well-suited to this adversarial search, as a trained agent can adaptively concentrate exploration on failure-inducing regions of the scenario space [8]. In practice, however, such agents tend to converge on a narrow set of failure modes, repeatedly rediscovering similar scenarios rather than exposing the broader range of system vulnerabilities [10, 31]. Promoting diversity among discovered failure scenarios is, therefore, a central concern of this thesis.

This thesis investigates deep reinforcement learning (DRL) as a tool for automated adversarial falsification of safety-critical advanced driver-assistance systems (ADAS) components, conducted in a closed-loop simulation environment in collaboration with Zenseact.

1.2 Objective

This thesis aims to assess whether DRL-based adversarial scenario generation can effectively expose safety-critical failures in simulation and whether a diversity-aware reward formulation can broaden the range of discovered failure scenarios. The specific objectives are:

- Formulate falsification as an adversarial scenario generation problem and implement a DRL agent for failure discovery.
- Introduce a reward component within a series DRL framework to promote diversity among discovered failure scenarios.
- Evaluate the proposed approach against baseline methods in terms of failure discovery efficiency and scenario diversity.

1.3 Research questions

- RQ1** To what extent can an adversarial agent efficiently discover relevant failure scenarios for the risk-limiting component?
- RQ2** To what extent does incorporating a diversity reward into a DRL framework increase the variety of discovered failure scenarios?
- RQ3** What factors affect the effectiveness of adversarial falsification for discovering relevant failure scenarios?

1.4 Limitations

The framework is evaluated in a one-dimension single-lane simulation scenario. Simplifications in vehicle dynamics and environmental modeling introduce a reality gap. The results are intended to validate algorithmic logic rather than replicate physical-world conditions. The investigation is also limited to DRL-based search strategies, and a systematic comparison against non-learning optimization methods falls outside the scope of this work.

Regarding the interpretation of results, the absence of discovered counterexamples should not be taken as a proof of safety. Finally, the trained agent is not evaluated for zero-shot transfer or adaptation to scenario configurations beyond those used in training.

1.5 Related work

Falsification of cyber-physical system (CPS) seeks counterexamples that violate a safety specification, classically expressed as a signal temporal logic (STL) formula [30]. Tracing its roots to the formal methods community, the dominant paradigm is robustness-guided optimization. Building on robust semantics for temporal logic specifications [13], STL admits a quantitative interpretation whose sign indicates satisfaction and whose magnitude furnishes a smooth objective for black-box search. S-TaLiRo [1] and Breach [11] are the two reference tools. Both connect simulation models with stochastic optimizer to minimize STL robustness over a finite parameterization of the input signal, and their success on industrial benchmarks (powertrain, transmission, AEB) established robustness-guided falsification as the baseline. Refinements have proceeded along several axes: decomposition of conjunctive STL formula into easier sub-goals; converge-aware sampling that interleaves exploitation with explicit exploration. The line continues today with data-driven baselines that learn to falsify without an explicit STL backbone [25]. However, a structural limitation persists. Classical falsifiers are fundamentally open-loop in time, which grows brittle as horizons lengthen and as the SUT contains reactive sub-controllers whose behavior depends on disturbance history.

Inspired by this gap, a second line of work recasts falsification as an MDP and unlocks the machinery of DRL, the adversary becomes a policy that observes the rolling state of the SUT and emits the next disturbance, with a reward signal nudging it towards a safety violation. Its foundational instance is adaptive stress testing (AST), introduced by Lee and Kochenderfer [27] for aircraft collision avoidance and extended by Koren et al. [24] to autonomous-vehicle pedestrian scenarios with using Mont Carlo Tree Search (MCTS) and DRL. Corso et al.’s survey [8] consolidates the framework across DRL and importance sampling. Yamagata et al. [43] formulated robustness-guided falsification of safety properties as a reinforcement learning problem, using STL robustness to guide reward design, and showed that DRL methods can find counterexamples with fewer simulation runs than random sampling, simulated annealing, and cross-entropy baselines.

A discernible trend is the migration of the most active work to the autonomous-driving community, where the SUT is a high-fidelity autonomous vehicle controller and the adversary is surrounding traffic agent. Feng et al.’s dense reinforcement learning approach [37] trains background agent that densify safety-critical states and achieves three orders of magnitude speed-up in failure discovery over naive random testing. Kuutti et al. [26] trained an advantage actor-critic (A2C) adversary to provoke rear-end collisions in a learned ego controller. Yang et al. [44] extend the dense reinforcement learning idea to adaptive multi-agent test environments for connected and automated vehicles. And a taxonomy of these methods is offered by Ding et al.’s survey [10], classifying data-driven, adversarial and knowledge-based scenario generation along a single axis.

A related design choice is whether the falsifier relies on a formal specification at all. Several autonomous-driving falsification studies dispense with STL-style specifications and instead use outcome-based rewards, often augmented with shaping terms based on relative distance, time-to-collision, or collision risk. Doreste et al.’s case study on autonomous driving falsification [12] uses a combined driving quality and collision risk shaping reward to quantify adversarial scenario difficulty. Cai et al.’s work [7], which directly inspired the diversity study in this thesis, likewise operates on a shaped collision reward and penalizes truncation, with no formal specification in the loop. The trade-off is familiar, while outcome rewards are easy to define and need no domain expertise to encode safety, yet they offer weaker gradients than continuous robustness signals.

Exploration, in turn, raises a second concern. Dynamic time warping (DTW) is a widely used similarity metric for time-series trajectories [4] and has been picked up by several falsification frameworks to define a diversity reward. The closest methodological reference for the diversity study is again the Cai et al.’s framework [7], which trains a sequence of RL falsifiers, each falsifier is shaped by a reward that penalizes similarity to the trajectories produced by its predecessors. Ji et al.’s diversity oriented testing framework [18] takes a different route, defining quantitative coverage metrics over the scenario space and steering exploration to maximize them. A deeper difficulty is highlighted by Liu and Feng [29], safety-critical regions are

sparse in high-dimensional input space, so dense novelty signals and sparse rarity signals point in different directions, and naive combinations drift away from precisely the regions that matter.

1.6 Outline of the thesis

The remainder of this thesis is organized as follows:

- Chapter 2 introduces the theoretical background, including falsification, reinforcement learning, TD3, and trajectory-diversity measures.
- Chapter 3 describes the simulation framework, the system under test, and the reinforcement learning formulation of the falsification problem.
- Chapter 4 presents the cross-scenario falsification experiment and the baseline methods used for comparison.
- Chapter 5 introduces the DTW-based Series RL method for diversity-oriented falsification and describes its evaluation metrics.
- Chapter 6 presents the main experimental results on collision discovery and trajectory diversity.
- Chapter 7 Chapter 7 presents exploratory studies and discusses generalization to more realistic environments and the practical implications of the proposed falsification approach.
- Chapter 8 concludes the thesis and discusses limitations and future work.

2

Theory

2.1 Falsification

Demonstrating that a complex system satisfies its safety requirements is a central concern in the development of safety-critical engineered systems. Two complementary approaches are commonly distinguished. Verification attempts to establish that no input to the system produces an unsafe outcome, typically through formal proof or exhaustive analysis. Falsification, by contrast, seeks to expose violations: rather than certifying correctness, it searches for concrete inputs under which the system fails [13].

The object whose specification is to be evaluated is referred to as the system under test (SUT). The SUT does not operate in isolation; its behavior depends on an environment comprising external inputs, other agents, and initial conditions. A complete description of how the SUT and its environment evolve over a finite time horizon is termed a scenario, and simulating a scenario produces a trajectory.

Given an SUT and a specification of unsafe behavior, falsification is the problem of finding a scenario whose trajectory violates the specification. Such a scenario is referred to as a counterexample, and constitutes evidence that the SUT fails to meet its safety requirement.

Two structural features of the falsification problem are worth emphasizing. First, falsification can be applied to systems whose internal structure resists analytical treatment, since it requires only the ability to simulate scenarios and observe their outcomes. Second, falsification is fundamentally an existence problem: a single counterexample suffices as evidence of failure, whereas certifying safety would require reasoning about the entire scenario space. The failure to find a counterexample within a finite search budget therefore does not imply the absence of counterexamples.

2.2 Markov decision process

The interaction between a reinforcement learning agent and its environment is formally captured by a Markov decision process (MDP), defined as the tuple [20]

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma) \tag{2.1}$$

At each discrete timestep t , the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$, and receives a scalar reward $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$. The environment then transitions to a successor state s_{t+1} according to the transition probability $\mathcal{P}(s_{t+1}|s_t, a_t)$. The probability of transitioning to the next state s_{t+1} depends only on s_t and a_t , not on the history of preceding states, which is known as the Markov property.

The agent's goal is to find a policy π that maximizes the expected cumulative reward over time. To ensure this sum remains finite and to reflect the preference for sooner rewards, future rewards are discounted by $\gamma \in [0, 1)$, yielding the return

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \quad (2.2)$$

The utility of executing a policy π from state s_t is then characterized by the value function

$$V^\pi(s_t) = \mathcal{R}(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1}} \mathcal{P}(s_{t+1}|s_t, \pi(s_t)) V^\pi(s_{t+1}) \quad (2.3)$$

where this recursive form is known as the Bellman equation.

2.3 Reinforcement learning

RL is a learning paradigm in which an agent improves its behavior through interaction with an environment. At each timestep, the agent observes the current state, selects an action, and receives a scalar reward signal indicating how desirable that outcome was. By repeating this loop over many iterations, the agent gradually discovers a policy that maximizes its cumulative reward, without requiring explicit supervision or models of how the environment works [39].

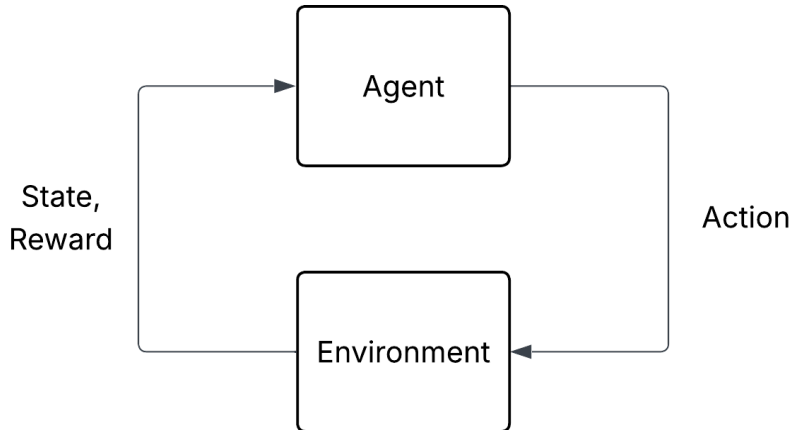


Figure 2.1: The classic reinforcement learning interaction loop: the agent selects an action a_t based on the observed state s_t and receives a reward r_t together with the next state s_{t+1} .

Early RL algorithms represented value functions and policies as lookup tables, which limited their applicability to problems with small, discrete state spaces. The introduction of deep neural networks as function approximators removed this bottleneck.

Neural networks can generalize across high-dimensional and continuous state spaces, enabling RL to scale to complex control problems [33]. The combination of RL and deep neural networks, commonly referred to as DRL, forms the basis of the algorithms presented in the following sections. For the falsification task in this thesis, where the agent must search over a continuous action space, DRL is a suitable choice because of its scalability to high-dimensional problems.

2.3.1 Q-value estimation

A central challenge in RL is to evaluate how good an action is in the given state. This is captured by the action-value function, or Q-function, defined as the expected discounted return when taking action a_t in state s_t and subsequently follows the policy π

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi[G_t | s_t, a_t] \quad (2.4)$$

The definition can be expressed recursively, where the value of the current state-action pair equals the immediate reward plus the discounted value of the next state, a relationship known as the Bellman equation [3, 39].

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_t \sim \mathcal{P}, a_{t+1} \sim \pi}[r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) | s_t, a_t] \quad (2.5)$$

In practice, Q^π is unknown and must be learned from experience. It is approximated by a neural network $Q_\phi(s, a)$ parameterized by ϕ . Two principal approaches exist for learning Q_ϕ . Monte Carlo (MC) methods estimate it by averaging complete episode returns, which provides unbiased estimates but suffers from high variance [39]. Temporal difference (TD) learning [39] instead updates it at each timestep using a bootstrapped target y_t , constructed from the immediate reward and the estimated Q-value of the next state-action pair.

$$y_t = r_t + \gamma Q_\phi(s_{t+1}, a_{t+1}) \quad (2.6)$$

The critic parameters ϕ are trained by minimizing the mean squared error between its current estimate and this target

$$\mathcal{L}(\phi) = \mathbb{E}[(Q_\phi(s_t, a_t) - y_t)^2] \quad (2.7)$$

yielding the gradient-descent update

$$\phi \leftarrow \phi - \alpha_\phi \nabla \mathcal{L}(\phi) \quad (2.8)$$

where α_ϕ is the critic learning rate.

2.3.2 Policy gradient

Having established how the Q-function is estimated, the question becomes how an agent can learn a policy that maximizes the expected return over an episode of horizon T . This objective is formally expressed as

$$J(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{T-1} \gamma^t r_t\right] \quad (2.9)$$

Early approaches addressed this through value-based methods, which derive a policy implicitly by acting greedily with respect to the learned Q-function. However, such methods are fundamentally limited to discrete action spaces [39]. Since the falsification agent must output continuous commands in this problem setting, a direct policy representation is necessary.

Policy gradient methods address this by parameterizing the policy directly and optimizing J through gradient ascent. For a stochastic policy $\pi_\theta(a_t|s_t)$ representing a probability distribution over actions, the policy gradient theorem [40] yields

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot Q^{\pi_\theta}(s_t, a_t) \right] \quad (2.10)$$

For a deterministic policy $\mu_\theta : \mathcal{S} \rightarrow \mathcal{A}$, which maps states directly to actions without sampling from a distribution, the deterministic policy gradient theorem [38] provides an expression without requiring knowledge of the transition dynamics $\mathcal{P}$

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{\mu_\theta} [\nabla_\theta \mu_\theta(s_t) \cdot \nabla_a Q^{\mu_\theta}(s_t, a)|_{a=\mu_\theta(s_t)}] \quad (2.11)$$

Intuitively, this updates θ in the direction that increases the Q-value of the actions selected by the policy: the gradient flows through Q-function with respect to the action, then through the policy μ_θ with respect to θ . Compared to the stochastic form, the deterministic gradient avoids integration over the action space and produces a low-variance estimate.

The actor parameters θ are updated by gradient ascent on J

$$\theta \leftarrow \theta + \alpha_\theta \nabla_\theta J(\theta) \quad (2.12)$$

where α_θ is the actor learning rate.

2.3.3 Actor-critic algorithm

The previous two sections developed the two ingredients required for learning in continuous action spaces: a parameterized Q-function Q_ϕ that evaluates the quality of state-action pairs, and a parameterized policy μ_θ whose parameters are updated via the policy gradient. These two components are naturally combined in the actor-critic framework, originally introduced by Barto, Sutton, and Anderson [2] and later formalized and analyzed theoretically by Konda and Tsitsiklis [22]. In this framework, the policy μ_θ acts as the actor that selects actions, while the Q-function Q_ϕ acts as the critic that evaluates them.

The actor and critic are trained jointly. The critic minimizes the TD error (2.7) to improve its estimate of the actor's value, while the actor ascends the gradient (2.11) to select actions that the critic rates highly. This mutual dependence is the defining feature of actor-critic methods, each component provides the learning signal for each other.

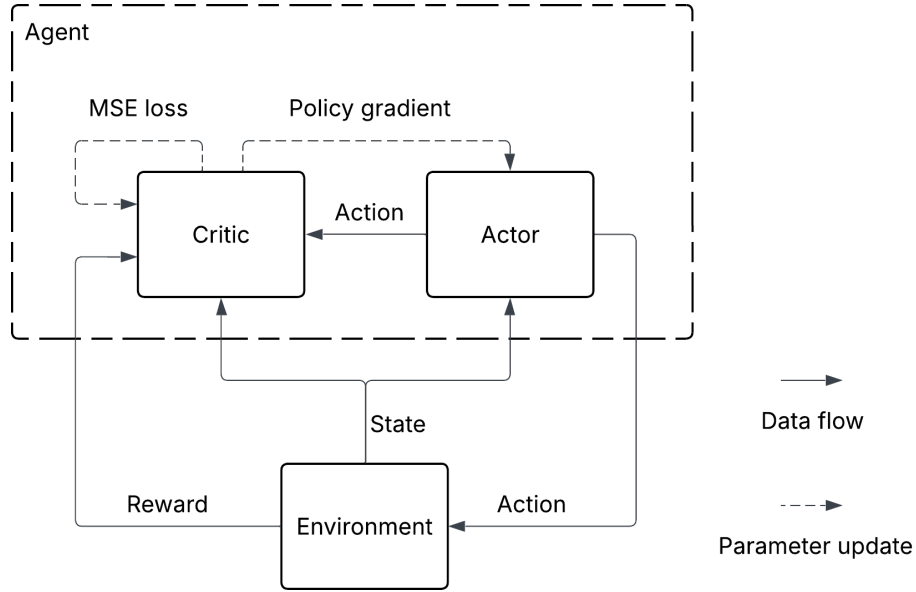


Figure 2.2: Interaction logic and update mechanism of Actor-Critic for continuous action spaces. The agent contains two parameterized networks: the actor maps the observed state to an action and the critic estimates the expected return of the selected state-action pair. The critic provides the learning signal for the actor, while the actor selects the actions whose values the critic learns to estimate – the mutual dependence that defines actor-critic methods.

Compared to the pure value-based method such as Q-learning, the explicit actor removes the need to solve $\arg \max_a Q(s, a)$ at each step, which is intractable in continuous action space. Compared to pure policy gradient methods such as REINFORCE [42], which estimate Q^{π_θ} from complete episode returns, the critic provides a bootstrapped and low-variance estimate that can be updated at every timestep. The combination of these two properties has made actor-critic the dominant framework for modern deep reinforcement learning in continuous control, underpinning algorithms such as Asynchronous Advantage Actor-Critic Algorithm (A3C) [32], DDPG [28], and TD3 [14].

2.3.4 Off-policy learning in actor-critic

Off-policy learning refers to the case where the policy used to collect data is different from the policy being optimized [9]. The data-collecting policy is called the behavior policy, and the policy being learned is called the target policy. Rather than learning only from trajectories generated by the current target policy itself, off-policy methods allow learning from behavior-policy data.

A common off-policy objective can be written as

$$J_\beta(\pi_\theta) = \int_S \rho^\beta(s) V^\pi(s) ds \quad (2.13)$$

where $\rho^\beta(s)$ denotes the state distribution induced by the behavior policy and $V^\pi(s)$ is the value of the target policy. This objective therefore evaluates the target policy

under the state distribution generated by the behavior policy, rather than under its own state distribution.

In the actor-critic framework, the critic learns a value estimate for the target policy from behavior-policy trajectories, and the actor updates the target policy based on this estimate. In this way, data collection and policy improvement are separated, which improves flexibility in learning and makes it possible to reuse experience generated by exploratory or earlier policies.

This separation is particularly useful for deterministic methods. Since a deterministic policy does not provide inherent exploration, exploration can instead be handled by a stochastic behavior policy, while the target policy remains deterministic.

2.3.5 Deep Deterministic Policy Gradient

A naive implementation of the actor-critic framework with deep neural networks is unstable for two reasons. First, the TD target in (2.6) depends on the same critic network Q_ϕ that is being updated, which couples the target to the current estimate and can cause the regression problem to diverge [28]. Second, consecutive transitions collected along a trajectory are strongly correlated, violating the independence assumption underlying stochastic gradient descent and introducing bias into the critic's updates. To address these two problems, the Deep Deterministic Policy Gradient (DDPG) algorithm was introduced by Lillicrap et al. [28]. It extends the actor-critic framework to continuous action spaces using deep neural networks, and addresses the problems by adapting two stabilization techniques from the Deep Q-Network [33] to the continuous-action actor-critic setting; target networks and an experience replay buffer.

Target Networks. DDPG maintains two pairs of networks. The online networks, a critic $Q_\phi(s, a)$ and a deterministic actor $\mu_\theta(s)$, are actively updated during training. To stabilize learning, a second pair of target networks, $Q_{\phi'}$ and $\mu_{\theta'}$, with parameters ϕ' and θ' , respectively, is maintained. The TD target (2.6) is then computed using the target networks rather than the online networks

$$y_t = r_t + \gamma Q_{\phi'}(s_{t+1}, \mu_{\theta'}(s_{t+1})) \quad (2.14)$$

Unlike DQN, which copies the online parameters to the target network every fixed number of steps, DDPG updates the target parameters through a soft update using Polyak averaging [28]

$$\phi' \leftarrow \tau_{\text{poly}} \phi + (1 - \tau_{\text{poly}}) \phi', \quad \theta' \leftarrow \tau_{\text{poly}} \theta + (1 - \tau_{\text{poly}}) \theta' \quad (2.15)$$

where $\tau_{\text{poly}} \ll 1$ controls the update rate. The target networks therefore track the online networks slowly, providing a nearly stationary regression target while still allowing the online networks to improve at each step.

Experience Replay Buffer. Rather than training on the most recent transitions, DDPG stores observed transition (s_t, a_t, r_t, s_{t+1}) in a finite capacity replay buffer

$\mathcal{D}$, discarding older transitions once the buffer is full. At each training step, a mini-batch is sampled uniformly at random from $\mathcal{D}$ to compute the critic loss and actor gradient. This decorrelates the training samples and allows each transition to contribute to multiple updates, improving both stability and sample efficiency. Because the deterministic policy gradient does not involve the action distribution under which the data was collected, DDPG can be trained off-policy on transitions drawn from $\mathcal{D}$, even though the policy has changed since those transitions were generated [28].

Exploration. Unlike stochastic policies, which explore naturally by sampling from their action distribution, a deterministic policy μ_θ always selects the same action in the same state and therefore requires an external exploration mechanism. DDPG addresses this by adding noise to the action

$$a_t = \mu_\theta(s_t) + \epsilon \quad (2.16)$$

where ϵ is drawn from a noise process. Lillicrap et al. [28] originally proposed an Ornstein–Uhlenbeck process to generate temporally correlated noise suited to physical control tasks, although subsequent work [14] has shown that uncorrelated Gaussian noise performs comparably in most settings.

Training Procedure. The full DDPG algorithm proceeds by alternating between environment interaction and gradient updates. At each environment step, an action is selected according (2.16) and the resulting transition is added to $\mathcal{D}$. At each gradient step, a mini-batch of B transitions is sampled from $\mathcal{D}$, the target y_t is computed from (2.14), and the critic parameters ϕ are updated by minimizing the empirical TD loss

$$\mathcal{L}(\phi) = \frac{1}{B} \sum_{j=1}^B (Q_\phi(s_j, a_j) - y_j)^2 \quad (2.17)$$

The actor parameters are then updated by ascending the sampled deterministic policy gradient

$$\nabla_\theta J(\mu_\theta) \approx \frac{1}{B} \sum_{j=1}^B \nabla_\theta \mu_\theta(s_j) \cdot \nabla_a Q_\phi(s_j, a) \Big|_{a=\mu_\theta(s_j)} \quad (2.18)$$

Finally, the target networks are updated by (2.15). This procedure enables stable training of deterministic policies in high-dimensional continuous action spaces and made DDPG the first deep actor-critic algorithm to solve a broad range of physical control benchmarks directly from low-level state [28].

2.3.6 Twin Delayed Deep Deterministic Policy Gradient

While DDPG provides a workable actor-critic implementation for continuous control, Fujimoto et al. [14] demonstrated that it is prone to a systematic overestimation bias in the learned critic and to instability during training, both of which can degrade or entirely prevent convergence. The Twin Delayed Deep Deterministic

Policy Gradient (TD3) algorithm addresses these failure modes through three targeted modifications to the DDPG framework: clipped double Q-learning, delayed policy updates, and target policy smoothing. Together, these modifications make TD3 the standard baseline for off-policy deep reinforcement learning in continuous action spaces and the algorithm adopted in this thesis.

Clipped Double Q-learning. Q-learning is known to suffer from overestimation bias arising from the maximization operator applied to noisy value estimates. Any positive estimation error for a given action is preferentially propagated through the max operation, leading to systematically inflated Q-values [16]. Fujimoto et al. [14] showed both theoretically and empirically that this bias persists in the actor-critic setting, where the deterministic policy plays a role analogous to the max operator: the actor is updated to select actions that the critic rates highly, a process that amplifies any positive errors in Q_ϕ . To mitigate this, TD3 adopts a variant of the double Q-learning idea originally proposed by van Hasselt [16]. Two independent critics Q_{ϕ_1} and Q_{ϕ_2} are maintained, each with its own target network $Q_{\phi'_1}$ and $Q_{\phi'_2}$, and the TD target is computed using the minimum of the two target critics

$$y_t = r_t + \gamma \min_{i=1,2} Q_{\phi'_i}(s_{t+1}, \mu_{\theta'}(s_{t+1})) + \epsilon_{\text{sm}} \quad (2.19)$$

where ϵ_{sm} is the smoothing noise introduced below. Both critics are then trained against this shared target by minimizing the corresponding TD losses

$$\mathcal{L}(\phi_i) = \frac{1}{B} \sum_{j=1}^B (Q_{\phi_i}(s_j, a_j) - y_j)^2, \quad i = 1, 2 \quad (2.20)$$

Taking the minimum biases the target downward, which is preferable to overestimation; an underestimated value is not propagated through the policy update in the same self-reinforcing way, since the actor will not be pushed toward actions whose value is pessimistically estimated [14].

Delayed Policy Updates. A second source of instability in DDPG arises from the coupling between the actor and the critic. If the actor is updated using a critic that has not yet converged, the resulting policy gradient is computed against a noisy value estimate, and the updated policy in return changes the distribution of state-action pairs the critic must evaluate, creating a feedback loop that can amplify errors. TD3 addresses this by updating the actor and the target networks at a lower frequency than the critics, for every d critic updates, the actor and all three target networks are updated once. The actor update uses only the first critic Q_{ϕ_1}

$$\nabla_{\theta} J(\mu_{\theta}) \approx \frac{1}{B} \sum_{j=1}^B \nabla_{\theta} \mu_{\theta}(s_j) \cdot \nabla_a Q_{\phi_1}(s_j, a) \Big|_{a=\mu_{\theta}(s_j)} \quad (2.21)$$

followed by the soft target updates

$$\phi'_i \leftarrow \tau_{\text{poly}} \phi_i + (1 - \tau_{\text{poly}}) \phi'_i, \quad i = 1, 2; \quad \theta' \leftarrow \tau_{\text{poly}} \theta + (1 - \tau_{\text{poly}}) \theta' \quad (2.22)$$

Fujimoto et al. recommend $d = 2$ as a default. Delaying the policy update allows the critic to reduce its estimation error before that error is propagated into the policy,

and reduces the variance of each policy update by ensuring it is computed against a more stable value estimate [14].

Target Policy Smoothing. A deterministic actor combined with a learned critic can exploit narrow peaks in the value estimate, producing a brittle policy that depends on inaccuracies of Q_ϕ rather than on genuine high-value actions. To regularize the critic against this failure mode, TD3 adds a small amount of clipped noise ϵ_{sm} , drawn from a Gaussian with standard deviation $\tilde{\sigma}$ and clipped to $[-c, +c]$, to the target action when computing the TD target

$$\epsilon_{\text{sm}} \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, +c) \quad (2.23)$$

with the clipping range c ensuring that the perturbation remains within a neighborhood of the target action. The effect is to enforce a smoothness prior on the critic: actions in a small neighborhood of $\mu_{\theta'}(s_{t+1})$ are required to have similar values, which prevents the policy from exploiting sharp, spurious peaks and yields a more robust estimate. The technique is a close relation in spirit to the SARSA update, which evaluates the action actually taken rather than the maximizing one [14].

Taken together, the three modifications target a single failure mode of DDPG, the propagation of critic estimation error into the policy. Clipped double Q-learning mitigates positive bias at the target, delayed updates ensure that the policy is updated against a more stable critic, and target policy smoothing prevents the policy from exploiting narrow peaks in the value estimate. The differences between the two algorithms are summarized in Table 2.1.

Table 2.1: Summary of the modifications introduced by TD3 over DDPG.

Component	DDPG	TD3
Critic	Single critic Q_ϕ	Twin critics Q_{ϕ_1}, Q_{ϕ_2} ; target uses $\min_{i=1,2} Q_{\phi'_i}$
Actor and target updates	Every gradient step	Every d critic updates (typically $d = 2$)
Target action	$\mu_{\theta'}(s_{t+1})$	$\mu_{\theta'}(s_{t+1}) + \epsilon_{\text{sm}}$, with $\epsilon_{\text{sm}} \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, +c)$

The complete TD3 algorithm therefore maintains six networks together with a replay buffer. At each environment step, an action is selected by the online actor μ_θ with added Gaussian exploration noise and the resulting transition is stored in the replay buffer $\mathcal{D}$. Each gradient step samples a mini-batch from $\mathcal{D}$, computes the smoothed and clipped target (2.19), and updates both critics via (2.20). Every d steps, the actor is then updated via (2.21) and all three target networks via (2.22). The full procedure is given in Algorithm 1.

Algorithm 1 TD3

```

Initialize critic networks  $Q_{\phi_1}, Q_{\phi_2}$ , and actor network  $\mu_\theta$  with random parameters
 $\phi_1, \phi_2, \theta$ 
Initialize target networks  $\phi'_1 \leftarrow \phi_1, \phi'_2 \leftarrow \phi_2, \theta' \leftarrow \theta$ 
Initialize replay buffer  $\mathcal{D}$ 
for  $t = 0$  to  $T - 1$  do
    Select action with exploration noise  $a_t = \mu_\theta(s_t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma)$ 
    Execute  $a_t$  and observe reward  $r_t$  and next state  $s_{t+1}$ 
    Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $\mathcal{D}$ 
    Sample  $B$  transitions  $(s, a, r, s')$  from  $\mathcal{D}$ 
    Smoothed target action:  $\tilde{a} \leftarrow \mu_{\theta'}(s') + \epsilon_{\text{sm}}, \quad \epsilon_{\text{sm}} \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, +c)$ 
    Compute clipped double-Q target:  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\phi'_i}(s', \tilde{a})$ 
    Update each critic:  $\phi_i \leftarrow \arg \min_{\phi_i} \frac{1}{B} \sum (Q_{\phi_i}(s, a) - y)^2, \quad i = 1, 2$ 
    if  $t \bmod d = 0$  then
        Update actor by deterministic policy gradient:
         $\nabla_\theta J(\mu_\theta) \approx \frac{1}{B} \sum \nabla_\theta \mu_\theta(s) \cdot \nabla_a Q_{\phi_1}(s, a) \Big|_{a=\mu_\theta(s)}$ 
        Soft-update target networks:
         $\phi'_i \leftarrow \tau_{\text{poly}} \phi_i + (1 - \tau_{\text{poly}}) \phi'_i, \quad i = 1, 2$ 
         $\theta' \leftarrow \tau_{\text{poly}} \theta + (1 - \tau_{\text{poly}}) \theta'$ 
    end if
end for

```

2.4 Diversity-oriented search and evaluation

This section introduces three concepts used for the trajectory diversity analysis in this thesis: Dynamic Time Warping (DTW), Series Reinforcement Learning, and Jaccard distance. They provide the basis for trajectory comparison, sequential agent training, and diversity evaluation.

2.4.1 Dynamic Time Warping

DTW is a dynamic-programming-based method for measuring the distance between two temporal sequences. It is useful when two sequences have different lengths or when similar patterns occur at different temporal rates [35].

Given two sequences

$$X = (x_1, x_2, \dots, x_m), \quad Y = (y_1, y_2, \dots, y_n), \quad (2.24)$$

DTW computes an alignment between their elements by minimizing the accumulated matching cost.

The local cost between two elements is defined as

$$c(x_i, y_j) = \|x_i - y_j\|_2. \quad (2.25)$$

Let $A(i, j)$ denote the minimum accumulated cost for aligning the partial sequences $(x_1, \dots, x_i)$ and $(y_1, \dots, y_j)$. The accumulated cost is computed recursively as

$$A(i, j) = c(x_i, y_j) + \min\{A(i-1, j), A(i, j-1), A(i-1, j-1)\}. \quad (2.26)$$

The DTW distance between X and Y is then

$$D_{\text{DTW}}(X, Y) = A(m, n). \quad (2.27)$$

2.4.2 Series Reinforcement Learning

Sequential training is a strategy for discovering multiple failure cases by training a series of agents instead of a single agent [7]. After each stage, the failure cases found by the current agent are stored and used as prior information for later stages. This allows later agents to search not only for successful failures, but also for failures that differ from those already discovered.

Let $\mathcal{T}_{k-1}$ denote the set of failure trajectories found before stage k . A new agent is trained at stage k using information from $\mathcal{T}_{k-1}$, and a newly selected failure trajectory is added to form $\mathcal{T}_k$. In general, the update can be written as

$$\mathcal{T}_k = \mathcal{T}_{k-1} \cup \{\tau_k^*\}, \quad (2.28)$$

where τ_k^* is the selected failure trajectory at stage k .

2.4.3 Jaccard distance

Jaccard similarity is a measure of overlap between two sets [17]. Given two finite sets A and B , the Jaccard similarity is defined as

$$\text{Jac}(A, B) = \frac{|A \cap B|}{|A \cup B|}. \quad (2.29)$$

The value is equal to 1 when the two sets are identical, and 0 when they have no common elements. The corresponding Jaccard distance is defined as

$$D_{\text{Jac}}(A, B) = 1 - \text{Jac}(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}. \quad (2.30)$$

A larger Jaccard distance indicates less overlap between the two sets.

In this thesis, Jaccard distance is used in the trajectory diversity evaluation to compare the overlap between the phase-space grid cells visited by different collision trajectories.

3

System modeling and problem formulation

This chapter introduces the simulation setting and problem formulation used in this thesis. The purpose is to connect the theoretical background introduced earlier with the reinforcement learning methods used in the following chapters. First, the closed-loop simulation framework is described, including the dynamic driving environment, the simulated components of the ego vehicle, the SUT, and the adversarial target vehicle. Then, the falsification task is cast as an MDP by defining the state space, action space, transition dynamics, reward and terminal conditions. Finally, building on these components, the falsification task is stated as a complete policy optimization problem.

3.1 Simulation framework

The simulation framework describes the closed-loop setting used to generate adversarial traffic scenarios. It consists of the longitudinal driving environment, the simulated components of the ego vehicle, the ego vehicle controlled by the output from the SUT, and the adversarial target vehicle controlled by the RL agent. Figure 3.1 illustrates the information flow between these components.

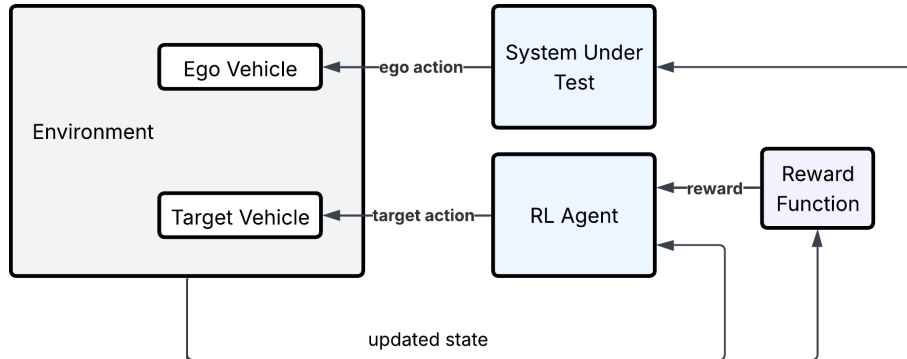


Figure 3.1: Closed-loop interaction between the environment, SUT, and RL agent. The SUT controls the ego vehicle, while the RL agent controls the target vehicle. The updated traffic state is returned to the SUT, agent, and reward function at each simulation step.

3.1.1 Dynamic driving environment

A simplified longitudinal two-vehicle scenario is considered in this work. The two vehicles are modeled as double integrators with acceleration as an input, and position and velocity as outputs. The target vehicle is initially located ahead of the ego vehicle, and only motion along the driving direction is modeled. Therefore, the interaction between the two vehicles is described in a one-dimensional setting.

Reverse motion is not allowed for either vehicle. Accordingly, the velocity of each vehicle is constrained to remain non-negative throughout the simulation. If a braking command would reduce the velocity below zero, the velocity is clipped at zero instead. When the velocity reaches zero, the vehicle remains stationary until a positive acceleration is applied.

The simulation is performed in discrete time with a time step of 0.025 s. At each step, the longitudinal states of both vehicles are updated according to their acceleration commands and the vehicle dynamics.

3.1.2 Adversarial target vehicle

The target vehicle acts as the controllable adversarial vehicle. At each simulation step, the adversarial agent selects a longitudinal acceleration command for the target vehicle. The target-vehicle acceleration is constrained to

$$a_{\text{tgt},t} \in [-4, 4] \text{ m/s}^2.$$

By controlling the target vehicle, the adversarial agent generates external behaviors that may expose safety-critical failures of the system under test.

3.1.3 Simulated ego-vehicle components and System Under Test

The software components that control the ego vehicle conform to a sense, plan, and act architecture [15]. Due to the simple one-dimensional environment, all components have a limited scope. Sensing is performed by a perception component whose main task is to report how far ahead the road is free from vehicles. The plan component is divided into two sub-components: one for calculating the maximum allowed safe velocity (called Calculator), and one for calculating an acceleration request that keeps the ego velocity below the maximum allowed safe velocity (called Controller). The act component consists of the brake actuator. Figure 3.2 shows the software components of the ego vehicle and the signal flow in the closed-loop system with the ego and target vehicle dynamics.

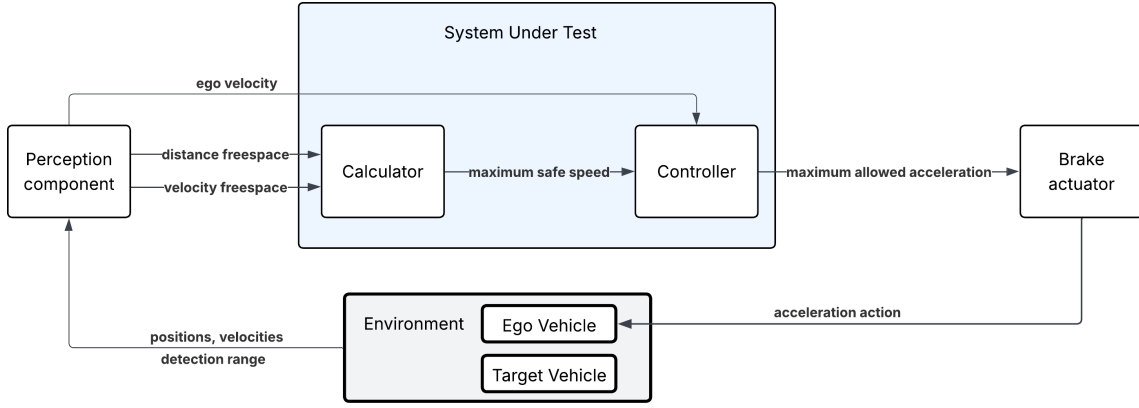


Figure 3.2: Architecture and signal flow of the simulated ego-vehicle system. The perception component provides the distance and velocity of the target vehicle. Based on this information, the Calculator computes a maximum safe speed. The Controller compares this value with the current ego velocity and calculates a maximum allowed acceleration, which the brake actuator applies to the ego vehicle. Meanwhile, the RL agent controls the target vehicle to force the SUT to make unsafe decisions.

The objective of this work is to find failures of the plan component during closed-loop operations when the rest of the system behaves according to a certain set of assumptions. Therefore, the system under test is the safety-critical software in the plan component. That is, the Calculator and the Controller are the two and only sub-components that compose the SUT. From this point of view, both the perception component and the brake actuator are considered parts of the environment, and thus they are both simulated components.

The sense component is assumed to have some limitation for how far away a vehicle may be reliably detected, so there is an upper limit to how far ahead it can report the road as free. If a vehicle is detected within this detection range, the sense component reports the relative distance to the vehicle and its velocity. The perception simulator implements this behavior by calculating the relative distance to the target vehicle, subject to the assumed maximum perception range. That is, if the target vehicle is further away than 50 m, the perception simulator will report a distance of 50 m and a velocity of 0 m/s. When the target vehicle is closer than 50 m, the perception simulator will report the relative distance and the target’s velocity.

The SUT’s objective is to avoid collisions with the target vehicle by constraining the maximum allowed acceleration of the ego vehicle. That is, the SUT can request braking to avoid collisions between the ego and target vehicles. The SUT will assume that the target vehicle at any point in time may brake as much as it can (-4 m/s^2). From the target’s current relative position, velocity, and worst case acceleration, the SUT calculates a stop position of the target vehicle. The ego vehicle can be assumed to be able to brake as much as 6 m/s^2 (an acceleration of -6 m/s^2), so calculating backwards from the target’s stop position, the Calculator part of the SUT can calculate a maximum safe speed. Then the Controller part of the SUT calculates an acceleration request that is ensured not to result in a velocity greater

than the safe speed, neither now nor at the next time step.

The calculations in the SUT are similar in spirit to the calculations of the minimum following distance discussed by Koopman et. al. [23]. As is pointed out in that work, the ego and target vehicle might collide while both are moving if the braking capacity of the ego vehicle is greater than that of the target vehicle (as is the case here). Accounting for this collision type is intentionally left out of the SUT to ascertain that the falsification methods have a failure case to find.

This thesis is based on a simplified, non-production research and testing framework developed for academic purposes only. The SUT and the associated assumptions do not represent the design, performance, or implementation of any production system at Zenseact.

3.1.4 Closed-loop interaction

At each simulation step, the adversarial agent observes the current traffic state and selects an action for the target vehicle. The environment then propagates the system dynamics, while the ego vehicle responds according to its internal control logic. This closed-loop interaction continues until a terminal condition is reached, such as a collision or the end of the simulation horizon.

3.2 MDP formulation

Based on the simulation framework and falsification objective described above, the task is formulated as an MDP as introduced in Section 2.2.

3.2.1 State space

The state represents the longitudinal interaction between the ego vehicle and the target vehicle. In this thesis, two state representations are used for different experimental purposes.

For the collision discovery experiment, a compact three-dimensional state representation is used:

$$s_t^{(3)} = [g_t, v_{\text{ego},t}, v_{\text{tgt},t}], \quad (3.1)$$

where $g_t = p_{\text{tgt},t} - p_{\text{ego},t}$ is the gap between the target vehicle and the ego vehicle, and $v_{\text{ego},t}$ and $v_{\text{tgt},t}$ are their longitudinal velocities. This representation directly captures the relative distance and the vehicle speeds relevant to collision discovery.

For the trajectory diversity experiment, a four-dimensional state representation is used:

$$s_t^{(4)} = [p_{\text{ego},t}, v_{\text{ego},t}, p_{\text{tgt},t}, v_{\text{tgt},t}], \quad (3.2)$$

where $p_{\text{ego},t}$ and $p_{\text{tgt},t}$ denote the longitudinal positions of the ego vehicle and the target vehicle, respectively. This representation preserves the absolute positions and

velocities of both vehicles, which is useful for recording and comparing complete collision trajectories.

3.2.2 Action space

The action of the adversarial agent is the longitudinal acceleration applied to the target vehicle. The action space is

$$\mathcal{A} = [-4, 4] \text{ m/s}^2, \quad (3.3)$$

so the action at time step t is the target-vehicle acceleration

$$a_t = a_{\text{tgt},t} \in \mathcal{A}. \quad (3.4)$$

This action directly determines the target vehicle behavior in the simulation. In contrast, the ego vehicle is not directly controlled by the agent, but responds according to SUT.

3.2.3 Transition dynamics

In the MDP formulation, the transition dynamics are denoted by $\mathcal{P}$. They describe how the environment moves from the current state s_t to the next state s_{t+1} after the agent applies an action a_t .

The transition is generated directly by the closed-loop simulator. At each simulation step, the simulator takes the current state and action as input and returns the next state:

$$s_{t+1} = F_{\text{sim}}(s_t, a_t), \quad (3.5)$$

where F_{sim} represents the overall simulator dynamics.

Therefore, although the MDP uses $\mathcal{P}$ to denote the transition dynamics, in practice each transition is obtained from the simulator. The resulting transition sample is

$$(s_t, a_t, r_t, s_{t+1}, \delta_t), \quad (3.6)$$

where δ_t indicates whether a terminal condition has been reached. These transitions are stored in the replay buffer and used for TD3 training.

3.2.4 Reward function

The reward function specifies the optimization objective of the adversarial agent. Since the falsification goal is to reach the collision set $\mathcal{C}$, the most direct choice would be to reward collision only. However, the SUT is explicitly designed to prevent collisions, so a randomly initialized adversarial policy can hardly triggers one. Under a collision-only reward, nearly all early trajectories would therefore receive an identical zero return, providing no gradient information to distinguish near-misses from non-critical behavior. Such a sparse reward makes learning progress depend

entirely on the exploration noise accidentally producing a collision.

To address this and inspired by the reward shaping of AST [24], we design the reward to prioritize collision discovery while still providing informative feedback for trajectories that do not result in collision. Those unsuccessful falsification attempts are ranked by how close they come to a collision. Collisions are treated as the primary failure events and are assigned a positive reward. For episodes end without a collision, they are penalized in proportion to the remaining gap at termination.

Let $\mathcal{E}$ denote the set of terminal states where the episode reaches its endpoint. The base reward is defined as

$$r_t^{\text{base}} = \begin{cases} R_{\text{col}}, & \text{if } s_t \in \mathcal{C}, \\ -c_0 - c_g g_t, & \text{if } s_t \notin \mathcal{C} \text{ and } s_t \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \quad (3.7)$$

where $R_{\text{col}} \gg 0$ is the collision reward, $c_0 > 0$ defines the base penalty for non-collision terminal states, and $c_g > 0$ scales the terminal gap penalty. The gap term serves as a proxy for the distance to the failure set. Among non-falsifying trajectories, those ending closer to collision obtain a higher return, so the policy is steadily pushed toward more critical interactions even before the first collision is found.

Note that the shaping is applied only at termination. All intermediate steps receive zero reward. A per-step gap penalty might seem more informative, but it would encourage the target vehicle to stay close to the ego vehicle throughout the episode. The accumulated penalty could then dominate the collision reward, and the discovered failure modes would be biased toward prolonged tailgating behavior. With terminal-only shaping, the return depends solely on the episode outcome. Since $R_{\text{col}} > 0 > -c_0$, every collision trajectory strictly dominates every non-collision trajectory, and the shaping therefore improves learnability without altering what the agent ultimately pursues.

3.2.5 Terminal conditions

Each episode is terminated under one of two conditions.

The first terminal condition is the occurrence of a collision between the ego vehicle and the target vehicle. In the proposed framework, such a collision is regarded as a failure case, meaning that the adversarial agent has successfully generated a safety-critical scenario.

The second terminal condition is reaching the predefined simulation horizon. If no collision occurs, the episode ends after 600 simulation steps. With a simulation time step of 0.025 s, this corresponds to 15 s of simulated time. This finite horizon ensures that each episode remains bounded in duration, while still allowing sufficient time for the interaction between the two vehicles to develop.

3.3 Problem formulation

With the state space, action space, transition dynamics, reward function, and terminal conditions defined, the falsification task can be written as a policy optimization problem. At each time step, the adversarial actor observes the current state s_t , selects a target vehicle's acceleration action a_t , and the closed-loop simulator generates the next state. The adversarial agent seeks a deterministic policy μ_θ that drives the closed-loop system into the collision set $\mathcal{C}$:

$$\begin{aligned} \theta^* &= \arg \max_{\theta} \mathbb{E}_{s_0 \sim \rho_0} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right] \\ \text{s.t. } & s_{t+1} = F_{\text{sim}}(s_t, a_t), \quad t = 0, \dots, T-1, \\ & a_t = \mu_\theta(s_t) \in \mathcal{A}, \quad t = 0, \dots, T-1, \\ & r_t = \mathcal{R}(s_t, a_t, s_{t+1}), \quad t = 0, \dots, T-1, \end{aligned} \tag{3.8}$$

where ρ_0 denotes the initial state distribution, $\gamma \in [0, 1)$ is the discount factor introduced in Sector 2.2, and T is the episode length. Since both the policy and the simulator are deterministic, the expectation is taken only over the initial state. Since the reward assigns high return to trajectories that reach $\mathcal{C}$ and penalizes non-collision terminal states, maximizing this return corresponds to searching for target vehicle's behaviors that falsify the SUT. In our settings, a trajectory is falsifying if and only if $s_t \in \mathcal{C}$ for some $t \leq T$.

4

Falsification Efficacy: Cross-Scenario Comparison

A practical falsifier of safety-critical control systems should be evaluated not on whether it can discover a collision in some carefully chosen scenario, but on how often it succeeds when applied to the broad set of conditions under which the system is intended to operate. A method that finds a collision for one initial condition but fails on the next provides little assurance about its usefulness as a verification tool, since real deployment exposes the system to a continuous range of traffic situations. This chapter therefore evaluates each falsification method on a benchmark of $N = 1000$ randomly sampled initial conditions, and reports the fraction of scenarios in which a collision is successfully discovered within a fixed per-attempt budget.

The benchmark compares the proposed reinforcement learning policy against two reference methods. The first is uniform random sampling, which constitutes the lower-bound baseline that employs neither a learned policy nor structured exploration. The second is Monte Carlo Tree Search, which performs structured tree-based exploration over a discretized action space but does not employ a learned policy. Comparing the success rate of the trained policy with these two references thus separates the marginal contribution of structured exploration from that of policy learning.

Because the difficulty of finding a collision varies considerably across initial conditions, an aggregate success rate alone may obscure where each method’s strength lies. The test set is therefore partitioned along two physically meaningful axes that characterize scenario difficulty. The first measures the available kinematic safety margin under worst-case ego braking by both vehicles, and the second captures the maximum vehicle speed at the start of the encounter. The per-cell success rate within this partition reveals whether differences between methods are uniform across the scenario distribution or concentrated in particular regimes.

4.1 Test scenarios

Each scenario in the benchmark is described by a four-dimensional initial condition vector $\mathbf{s} = (p_{\text{ego}}, v_{\text{ego}}, p_{\text{tgt}}, v_{\text{tgt}})$ specifying the longitudinal position and velocity of the ego vehicle and of the target. The ego vehicle starts at the origin so $p_{\text{ego}} = 0$, and the remaining components are drawn independently from uniform distributions

$v_{\text{ego}} \sim \mathcal{U}(0, 36)$ m/s, $p_{\text{tgt}} \sim \mathcal{U}(20, 50)$ m, and $v_{\text{tgt}} \sim \mathcal{U}(0, 36)$ m/s.

We call a scenario avoidable when, for every admissible target trajectory, there exists at least one ego control under which no collision occurs. A candidate initial condition is admitted into the test set only if it satisfies a necessary kinematic condition for this property: that even in the worst case for the ego — both vehicles applying their maximum braking from the initial instant — the ego comes to rest before reaching the target’s final position. Let $b_{\text{ego}}^{\max}, b_{\text{tgt}}^{\max} > 0$ denote the magnitudes of the maximum braking decelerations of the two vehicles. Under maximum braking, the ego covers $v_{\text{ego}}^2/(2b_{\text{ego}}^{\max})$ before stopping and the target covers $v_{\text{tgt}}^2/(2b_{\text{tgt}}^{\max})$. The candidate is therefore accepted when the initial gap exceeds the difference of the two stopping distances,

$$\text{gap} = p_{\text{tgt}} - p_{\text{ego}} > \frac{v_{\text{ego}}^2}{2b_{\text{ego}}^{\max}} - \frac{v_{\text{tgt}}^2}{2b_{\text{tgt}}^{\max}}, \quad (4.1)$$

with the braking bounds set to $b_{\text{ego}}^{\max} = 6$ m/s² and $b_{\text{tgt}}^{\max} = 4$ m/s². When the right-hand side is non-positive — for example when the ego is stationary, or when the target’s stopping distance already exceeds the ego’s — the filter is satisfied by any positive initial gap. Candidates that fail the filter are rejected and resampled until $N = 1000$ admissible configurations have been collected.

The filtering step ensures that every scenario in the test set admits at least one safe control trajectory, so the failure of a falsifier on a given scenario reflects the limitations of the method rather than a kinematic impossibility. The same test set, generated with a fixed random seed, is shared across all methods evaluated in this chapter, which allows per-scenario success and failure to be compared directly between methods rather than only in aggregate.

4.2 TD3 implementation

The proposed falsifier in this thesis is based on the TD3 implementation provided by Stable-Baselines3 [34]. The TD3 architecture, off-policy actor-critic loss, and core training procedure are introduced in Chapter 2 and Chapter 3. Three aspects of the training procedure are specific to the present benchmark and we describe them here in some detail because they reflect deliberate methodological choices rather than peripheral implementation settings.

The policy is trained under a three-stage curriculum that progressively widens the initial condition sampling range across the one million step training horizon, summarized in Table 4.1. Stage 1 restricts the target’s initial position and the speeds of both vehicles to a narrow regime in which a collision can be discovered with relatively short, predictable action sequences, allowing the agent to encounter the sparse positive collision reward early and to internalize the structure of this signal before being asked to generalize. Stage 2 expands the range moderately, requiring the agent to adapt the partially trained policy to a broader operating envelope.

Stage 3 finally covers the full benchmark initial condition range, including high-speed and large-gap configurations in which collision discovery requires sustained coordinated control. The motivation for this schedule is empirical: direct training on the full range produces vanishing positive feedback in the early phase, with the policy spending most of its samples on scenarios where random exploration almost never triggers a collision. Staging the difficulty ensures that informative reward gradients are available throughout training, so that the policy first acquires a working understanding of the collision reward in tractable scenarios and then progressively refines it as the distribution broadens.

Two schedules govern exploration and learning across the curriculum. The action exploration noise decays linearly from $\sigma_{\text{init}} = 0.2$ to $\sigma_{\text{final}} = 0.02$ across the full one-million-step training horizon, independently of stage boundaries, so that the exploration aperture narrows monotonically as the policy matures. The learning rate, by contrast, is interpolated linearly from 3×10^{-4} at the start of each call to the underlying training routine towards 1×10^{-4} at the call’s end. Because the global timestep counter is preserved across stage transitions, the actual learning rate at the start of stages 2 and 3 is the value implied by the global training progress rather than 3×10^{-4} , and is therefore higher than the rate reached at the end of the preceding stage but lower than the initial 3×10^{-4} of stage 1. This produces a modest upward jump in the learning rate at each stage transition, which serves as a brief warm-up that aids adaptation when the policy first encounters the wider scenario distribution introduced by the new stage. The replay buffer used during training is the standard uniform-sampling buffer, in which past transitions are stored without preferential treatment of collision events.

At evaluation time the trained policy is deployed in deterministic inference mode, so that no exploration noise is added to the action output and the policy follows its mean prediction. A single episode of at most 600 simulator steps, corresponding to 15 s of simulated time at the 25 ms control interval, is run on each scenario, and no further learning or scenario-specific adaptation takes place. A scenario is recorded as a success if the gap between the two vehicles reaches zero before the episode terminates.

Table 4.1: Curriculum schedule used to train the TD3 policy for the cross-scenario benchmark. Each stage progressively widens the initial-condition sampling range. The ego vehicle starts at $p_{\text{ego}} = 0$ throughout.

Stage	p_{tgt} (m)	v_{ego} (m/s)	v_{tgt} (m/s)	Training steps
1	30	[0, 16]	[0, 5]	200,000
2	[30, 40]	[5, 26]	[0, 20]	300,000
3	[20, 50]	[0, 36]	[0, 36]	500,000

4.3 Baseline methods

The TD3 method is compared against two reference methods that share no learned component with it. Both baselines operate on the same one-dimensional control interface used by the environment, where at each simulation step a scalar value $a \in [-1, 1]$ is interpreted as the requested target longitudinal acceleration scaled by 4 m/s^2 . Each baseline is invoked once per scenario according to a budget specified in its respective subsection.

4.3.1 Monte Carlo Tree Search

MCTS explores the space of action sequences by incrementally building a search tree whose nodes represent states reached by particular action prefixes from the initial condition [6]. The continuous action interval is discretized into n uniformly spaced bins so that the branching factor of the tree remains tractable. Each iteration of the search consists of $n - 1$ conventional phases. During selection the algorithm descends from the root according to the UCB1 rule [21], which balances exploitation of children with high mean reward against exploration of those that have been visited less often. Expansion adds a single previously untried child to the leaf reached by selection. Simulation performs a uniformly random rollout from the new child until the episode terminates either through a collision or by reaching the maximum timestep horizon. Backpropagation finally credits the leaf and all its ancestors with a binary reward of one for a collision and zero otherwise.

For each scenario the search is allocated a budget of up to 50 iterations and is terminated as soon as a collision is discovered within any rollout. The scenario is recorded as success if any rollout within the budget produces a collision, and as a failure otherwise.

4.3.2 Uniform random sampling

RAND draws an action independently from the uniform distribution on $[-1, 1]$ at every simulation step within a single episode, and uses neither a learned policy nor any search structure. A single episode is run on each scenario as a success if the gap reaches zero within this episode and as a failure otherwise.

4.4 Difficulty stratification

The aggregate collision rate is informative as a single-number summary, but it conflates scenarios of widely differing difficulty under one statistic and so reveals little about where each method’s success or failure originates. To enable a finer reading, the test set is partitioned along two physically interpretable axes that together describe the structure of an encounter at its initial state. The first axis is the available kinematic safety margin, and the second is the maximum initial vehicle speed, both defined below.

The safety margin quantifies how much headroom exists beyond the worst-case kinematic stopping distance. For a scenario with initial gap, it is the dimensionless ratio

$$\text{margin} = \frac{\text{gap}}{d_{\min}^{\text{stop}}}, \quad d_{\min}^{\text{stop}} = \frac{v_{\text{ego}}^2}{2b_{\text{ego}}^{\max}} - \frac{v_{\text{tgt}}^2}{2b_{\text{tgt}}^{\max}}, \quad (4.2)$$

where $d_{\min}^{\text{stop}}$ is the same worst-case required stopping margin that appears in the avoidability filter (Equation 4.1). A margin of unity marks the boundary of physical avoidability and is excluded by the filter, while values above unity indicate increasing kinematic headroom. Configurations in which the target’s stopping distance already exceeds the ego’s ($d_{\min}^{\text{stop}} \leq 0$) admit an arbitrary positive gap and are assigned an effectively infinite margin; these describe a regime in which the ego cannot kinematically overtake the target’s stopping point under maximum braking, and any collision must be elicited through active manipulation of the target’s acceleration. In line with this interpretation we treat them as belonging to the largest margin bin defined below..

The second axis records the maximum initial speed of the encounter, $\max(v_{\text{ego}}, v_{\text{tgt}})$, which characterizes the kinetic energy at stake regardless of which vehicle is faster. The two axes are jointly discretized into three bins each, yielding a 3×3 partition of the test set as summarized in Table 4.2. The margin axis is divided into *tight* ($\text{margin} < 2$), *moderate* ($\text{margin} \in [2, 5)$), and *loose* ($\text{margin} \geq 5$). These edges express conventional notions of kinematic comfort in stopping-distance multiples, with a tight encounter offering less than twice the worst-case stopping distance and a loose encounter offering five times as much or more. The speed axis is divided into *low* ($\max(v_{\text{ego}}, v_{\text{tgt}}) < 20$ m/s), *mid* ($\in [20, 30)$ m/s), and *high* (≥ 30 m/s). The speed edges are placed close to the empirical tertiles of the $\max(v_{\text{ego}}, v_{\text{tgt}})$ distribution induced by the initial condition sampling procedure, so that the marginal sample counts of the three speed bins remain comparable in magnitude.

Table 4.2: Bin specification for the difficulty stratification.

Axis	Bin	Range
Safety margin	tight	$\text{margin} < 2$
	moderate	$\text{margin} \in [2, 5)$
	loose	$\text{margin} \geq 5$
Max initial speed	low	$\max(v_{\text{ego}}, v_{\text{tgt}}) < 20$ m/s
	mid	$\max(v_{\text{ego}}, v_{\text{tgt}}) \in [20, 30)$ m/s
	high	$\max(v_{\text{ego}}, v_{\text{tgt}}) \geq 30$ m/s

5

Trajectory Diversity with DTW-based Series RL

This chapter focuses on the trajectory diversity experiment. While the previous experiment evaluates collision discovery performance, this experiment investigates whether the proposed DTW-based Series RL method can discover collision trajectories with more diverse interaction patterns. The comparison is conducted between a baseline Series RL setting using the base reward and a DTW-based Series RL setting using the proposed diversity reward.

Both settings use the same four-state TD3 configuration with a collision replay buffer. Therefore, the difference between the two settings comes from whether the DTW-based diversity term is included in the reward. This chapter first describes the four-state TD3 training setup, then introduces the DTW-based Series RL procedure, followed by the experiment design and trajectory diversity metrics.

5.1 Four-state TD3 with collision replay buffer

A practical issue in this task is that collision experiences can be easily diluted in the standard replay buffer [24]. During early training, the agent has not yet learned a stable adversarial policy, but collisions may still occur occasionally under specific initial conditions or action sequences. These experiences are important because they describe how a normal interaction can develop into a dangerous scenario. However, as training continues, a large number of non-collision transitions are stored in the replay buffer. The few early collision transitions therefore become rare in the sampling process [45]. As a result, even if the agent has encountered collisions before, it may not reuse these key samples often enough to learn from them effectively.

To mitigate this issue, a separate collision replay buffer is introduced in addition to the standard replay buffer. The design is motivated by prioritized experience replay, where informative transitions are sampled more frequently during training [36]. However, instead of assigning priority according to the TD error, this thesis defines priority based on collision relevance. All transitions are stored in the standard replay buffer, while transitions from episodes that end in collision are also copied into the collision replay buffer. During training, each mini-batch is constructed by sampling from both buffers, which increases the likelihood that collision-related transitions contribute to the TD3 updates and prevents them from being diluted by the larger

number of non-collision transitions. Within the collision replay buffer, transitions closer to the collision time step are assigned higher sampling priority, so that learning focuses more strongly on the part of the episode where the failure develops.

The collision replay buffer does not change the TD3 objective or the actor-critic update rule. It only changes the sampling distribution of training data. In this way, the TD3 agent can repeatedly learn from rare failure cases that have already appeared during training.

5.1.1 Dynamic sampling ratio between buffers

Fixed sampling ratios are not always ideal in replay-based RL, because the importance of different types of replay samples can change during training [41]. Motivated by this idea, this work adjusts the collision-buffer sampling ratio according to the number of stored collision transitions. At the beginning of training, the collision replay buffer may contain only a few transitions, and sampling too many of them may repeatedly expose the agent to a small set of cases. Later in training, when more collision transitions have been collected, a low fixed ratio may not use these informative samples sufficiently. Therefore, the sampling ratio from the collision replay buffer is adjusted according to the current number of stored collision transitions.

Let N_c denote the number of transitions currently stored in the collision replay buffer. The collision sampling ratio is defined as

$$\rho_c(N_c) = \begin{cases} 0, & N_c = 0, \\ \text{clip}(\rho_{\text{base}} + k_\rho \log(1 + N_c), \rho_{\min}, \rho_{\max}), & N_c > 0, \end{cases} \quad (5.1)$$

where $\text{clip}(\cdot)$ constrains the value to the interval $[\rho_{\min}, \rho_{\max}]$.

Here, ρ_{base} is the base sampling ratio, k_ρ controls the growth rate, $\rho_{\min}$ is the minimum ratio once collision transitions are available, and $\rho_{\max}$ is the upper bound that prevents the training batch from being dominated by collision transitions.

For a mini batch of size B , the number of samples drawn from the collision buffer is

$$B_c = \lfloor B\rho_c(N_c) \rfloor, \quad (5.2)$$

and the number of samples drawn from the standard replay buffer is

$$B_r = B - B_c. \quad (5.3)$$

The two subsets are then merged before being used for the TD3 network updates.

This design allows training to rely entirely on the standard replay buffer before any collision has been found. Once collision transitions are available, they are guaranteed to appear in the training batches with at least $\rho_{\min}$ proportion. As more collision transitions are collected, the ratio increases, while the upper bound $\rho_{\max}$ maintains a balance between non-collision transitions and collision transitions.

5.1.2 Priority sampling within the collision replay buffer

After the number of samples from the collision replay buffer is determined, the next step is to decide which transitions should be sampled from it. Instead of sampling collision transitions uniformly, the sampling scheme is motivated by prioritized experience replay, where more informative transitions are sampled more frequently [36]. It is also related to prioritized sequence experience replay, which propagates priority to earlier transitions within the same experience sequence [5]. The key event is the collision at the end of an episode, so transitions from the same collision episode are prioritized according to their temporal distance to the collision moment. Transitions closer to collision receive higher priority because they provide more direct information about how the interaction develops into a failure.

During an episode, transitions are first stored in a temporary episode cache. If the episode ends in collision, all transitions from this episode are added to the collision replay buffer. At this point, the priority of each transition is computed according to its temporal distance to the collision moment. Let k_i denote the number of steps from transition i to the final collision point. The priority of transition i is defined as

$$w_i = \begin{cases} 1 + \kappa_w \frac{K - k_i}{K}, & k_i < K, \\ 1, & k_i \geq K. \end{cases} \quad (5.4)$$

Here, K defines the number of steps before collision that receive additional attention, and κ_w controls the strength of the priority increase. If a transition is outside this window, its priority remains 1. If it is close to the collision point, its priority becomes larger. Therefore, transitions immediately before collision are sampled more often than earlier transitions in the same episode.

When sampling from the collision replay buffer, priorities are normalized over all stored collision transitions:

$$p_i^{\text{col}} = \frac{w_i}{\sum_{j=1}^{N_c} w_j}, \quad (5.5)$$

where N_c is the current number of transitions stored in the collision replay buffer. The priority w_i is fixed when the transition is added to the collision replay buffer. However, the sampling probability p_i^{col} is recomputed during sampling, since the normalization term changes as new collision transitions are added.

5.2 DTW-based Series RL for trajectory diversity

This section introduces the proposed diversity-oriented Series RL framework with DTW. The goal is to extend collision-oriented falsification from simply finding failures to discovering diverse failure trajectories. We first describe how DTW is used to compare collision trajectories, then introduce the DTW-based reward and the sequential training procedure that uses previously discovered trajectories to guide later agents.

5.2.1 Trajectory representation for DTW

In the trajectory diversity experiment, DTW is used to compare collision trajectories in terms of interaction-level features. Instead of applying DTW to the raw vehicle state, each collision trajectory is represented using the gap and relative speed between the ego vehicle and the target vehicle. At each time step t , the feature vector is defined as

$$z_t = [g_t, \Delta v_t], \quad (5.6)$$

where g_t is the gap and Δv_t is the relative speed, computed as the target vehicle speed minus the ego vehicle speed. The trajectory sequence is formed from the states recorded between the initial state of the episode and the terminal collision step.

Thus, a collision trajectory τ with length L_τ is represented as

$$Z(\tau) = (z_1, z_2, \dots, z_{L_\tau}). \quad (5.7)$$

Given two collision trajectories τ^a and τ^b with lengths m and n , their feature sequences are denoted by

$$Z^a = (z_1^a, z_2^a, \dots, z_m^a), \quad Z^b = (z_1^b, z_2^b, \dots, z_n^b). \quad (5.8)$$

The local cost used in DTW is the Euclidean distance between two feature vectors:

$$c(z_i^a, z_j^b) = \|z_i^a - z_j^b\|_2. \quad (5.9)$$

Equivalently, the local cost matrix $C \in \mathbb{R}^{m \times n}$ is constructed as

$$C_{ij} = c(z_i^a, z_j^b) = \sqrt{(g_i^a - g_j^b)^2 + (\Delta v_i^a - \Delta v_j^b)^2}. \quad (5.10)$$

No additional feature-wise normalization is applied before computing DTW. The local cost is computed directly from the gap in meters and the relative speed in m/s.

The accumulated DTW cost matrix A is then computed by dynamic programming. The first element is initialized as

$$A(1, 1) = C_{11}. \quad (5.11)$$

For the remaining entries, the accumulated cost is updated as

$$A(i, j) = C_{ij} + \min\{A(i-1, j), A(i, j-1), A(i-1, j-1)\}, \quad (5.12)$$

where the three predecessor terms correspond to vertical, horizontal, and diagonal moves in the warping path. Boundary values outside the matrix are treated as infeasible, so the alignment starts at $(1, 1)$ and ends at (m, n) .

The DTW distance between trajectories τ^a and τ^b is therefore

$$D_{\text{DTW}}(\tau^a, \tau^b) = A(m, n). \quad (5.13)$$

This representation focuses on the interaction pattern that leads to collision. Therefore, two trajectories are considered similar if they exhibit similar gap and relative-speed evolution.

5.2.2 DTW-based reward

The base reward encourages the adversarial agent to discover collision trajectories, but it only evaluates whether a failure is reached and how close a non-failure trajectory is to collision. It does not distinguish whether a newly discovered collision represents a new failure pattern or simply repeats a previously found one. As a result, repeated training may lead to collision trajectories that are successful but highly similar.

To encourage diverse failure discovery, we augment the collision reward with a DTW-based diversity term [7]. The intuition is that a collision trajectory should receive an additional reward if its temporal interaction pattern is different from previously discovered collision trajectories.

To compute the diversity term, we use a collision trajectory repository $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_N\}$ that stores previously discovered collision trajectories. For a newly generated collision trajectory τ , the DTW distance $D_{\text{DTW}}(\tau, \tau_i)$ is computed between τ and each stored trajectory $\tau_i \in \mathcal{T}$. The DTW-based diversity score is then defined as the smallest of these distances:

$$S_{\text{DTW}}(\tau) = \min_{\tau_i \in \mathcal{T}} D_{\text{DTW}}(\tau, \tau_i). \quad (5.14)$$

A larger value of $S_{\text{DTW}}(\tau)$ means that the generated collision trajectory is farther from the closest trajectory already stored in the repository.

The coefficient λ_{DTW} controls the contribution of this diversity score relative to the base collision reward. The modified reward is defined as

$$r_t^{\text{DTW}} = \begin{cases} R_{\text{col}} + \lambda_{\text{DTW}} S_{\text{DTW}}(\tau), & \text{if } s_t \in \mathcal{C}, \\ -c_0 - c_g g_t, & \text{if } s_t \notin \mathcal{C} \text{ and } s_t \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \quad (5.15)$$

Thus, the DTW-based bonus is added only when the generated trajectory reaches a collision state. Non-collision terminal states and non-terminal states use the same reward as the base reward.

5.2.3 Repository update in series RL

During series training, a collision trajectory repository is maintained across stages to store the selected collision trajectories found by previous agents [7]. At stage k , the repository $\mathcal{T}_{k-1}$ is fixed and used to compute the DTW-based reward for the current agent. The simulation setting and initial condition are kept fixed across stages, while a different random seed is used for training.

Let π_k denote the adversarial policy trained at stage k . Initially, $\mathcal{T}_0 = \emptyset$, so no reference trajectory is available for computing the DTW-based diversity score. Therefore,

the first selected collision trajectory is stored as the initial reference, and its diversity score is left undefined.

After training, two policies may be considered for repository update. The final policy π_k^{final} is the policy obtained at the end of training. In addition, during training, an evaluation callback periodically evaluates the current policy on a fixed evaluation setting. The checkpoint with the highest evaluation performance is saved as the best policy π_k^{best} . Since TD3 uses a deterministic policy for evaluation, each saved policy is rolled out deterministically to obtain a candidate collision trajectory.

The candidate set for stage k is formed from the collision trajectories generated by π_k^{final} and π_k^{best} :

$$\mathcal{Q}_k = \{\tau_k^{\text{final}}, \tau_k^{\text{best}}\}, \quad (5.16)$$

where only trajectories that result in collision are included. If no collision trajectory is found, $\mathcal{Q}_k = \emptyset$ and the repository is not updated at this stage.

For each candidate trajectory $\tau \in \mathcal{Q}_k$, the diversity score $S_{\text{DTW}}(\tau)$ is computed using the current repository $\mathcal{T}_{k-1}$ as the reference set. This means that each candidate is compared with the trajectories already stored in the repository, and the closest stored trajectory determines the candidate’s diversity score.

When $\mathcal{T}_{k-1} = \emptyset$, the first available collision trajectory in $\mathcal{Q}_k$ is selected directly as τ_k^* . Otherwise, the trajectory with the largest diversity score is selected:

$$\tau_k^* = \arg \max_{\tau \in \mathcal{Q}_k} S_{\text{DTW}}(\tau). \quad (5.17)$$

The repository is then updated by appending the selected trajectory:

$$\mathcal{T}_k = \mathcal{T}_{k-1} \cup \{\tau_k^*\}. \quad (5.18)$$

5.3 Experiment design

5.3.1 Experimental protocol

The purpose of this experiment is to evaluate whether adding the DTW-based diversity term to the reward improves the diversity of discovered collision trajectories. We compare two series-training settings: Series RL with the base reward only, and Series RL with the proposed DTW-based reward. The former is used as the baseline, while the latter corresponds to the proposed method.

To ensure a fair comparison, both settings use the same simulation configuration, training budget, evaluation procedure, and sequence of random seeds. Thus, the difference between the two settings comes from whether the DTW-based diversity term is included in the reward.

For both settings, each agent is first trained under its corresponding reward function. After each training stage, the final policy and the best checkpoint saved during training are evaluated deterministically. Collision trajectories obtained from these evaluations are used as candidates for comparison, rather than being collected directly from training rollouts.

After the series training and deterministic evaluation, the collision trajectories stored in each method’s repository are visualized for diversity analysis. For each repository, we plot the time profiles of the ego and target vehicle positions, velocities, gap, and relative speed. These plots show how the interaction evolves before collision and allow us to compare differences in collision timing, closing behavior, and gap evolution.

In addition, we plot the trajectories in the phase space of gap and relative speed. Since gap and relative speed are the two features used in the DTW computation, the phase plot provides a direct visualization of the trajectory patterns measured by the DTW-based diversity score. More separated curves in this phase space indicate that the discovered collision trajectories follow more diverse interaction patterns.

5.3.2 Parameter settings

The main parameter settings used in the trajectory diversity experiment are summarized in Tables 5.1–5.3. The same TD3 training configuration, replay-buffer setting, simulation horizon, and random-seed sequence are used for the baseline Series RL setting and the DTW-based Series RL setting. The only reward-level difference between the two settings is the value of the DTW diversity coefficient λ_{DTW} .

Table 5.1: Training protocol for the trajectory diversity experiment.

Parameter	Value
Number of series stages	10
Maximum training steps per agent	400,000
Training seeds	42–50, 57
Episode horizon	600 steps
Simulation time step	0.025 s
Initial ego state	$p_{\text{ego},0} = 0, v_{\text{ego},0} = 0$
Initial target state	$p_{\text{tgt},0} \sim \mathcal{U}(0, 30), v_{\text{tgt},0} = 0$

Table 5.2: TD3 hyperparameters for the trajectory diversity experiment.

Parameter	Value
Mini-batch size B	256
Discount factor γ	0.99
Learning rate	$3 \times 10^{-4} \rightarrow 1 \times 10^{-4}$
Learning starts	2,000 steps
Standard replay buffer capacity	200,000
Policy delay d	2
Target update rate τ_{poly}	0.005
Target policy noise	0.2
Target noise clip	0.5
Exploration noise	$0.2 \rightarrow 0.02$ over 200,000 steps
Actor and critic hidden layers	64, 64 with tanh activation

Table 5.3: Collision-buffer, reward, and diversity-evaluation parameters.

Parameter	Value
Collision buffer capacity	50,000 transitions
ρ_{base}	0.03
k_{ρ}	0.06
ρ_{min}	0.25
ρ_{max}	0.5
K	80 steps
κ_w	6.0
R_{col}	150
c_0	200
c_g	1.0
λ_{DTW}	0.1
Repository capacity N_{max}	10 trajectories
Gap range for phase grid	$[0, 30]$ m
Relative-speed range for phase grid	$[-14, 0]$ m/s
Grid resolution	$1 \text{ m} \times 1 \text{ m/s}$
Number of grid cells N_{cells}	30×14

5.4 Evaluation metrics

This section defines the metrics used to evaluate trajectory diversity in the diversity experiment. We use three complementary measures. The DTW-based insertion score evaluates how different a newly selected trajectory is from the repository at the time it is added. Phase-space coverage measures how much of the predefined gap and relative-speed space is visited by the repository trajectories. Pairwise Jaccard distance measures the overlap between individual trajectories based on their visited

grid-cell sets. The insertion score evaluates the repository construction process, while coverage and Jaccard distance evaluate the final repository content.

5.4.1 DTW-based insertion score

The DTW diversity score S_{DTW} described above is used during training as part of the reward. To evaluate the repository construction process, the same nearest-neighbor DTW calculation is recorded for the trajectory selected after each training stage. This recorded value is called the insertion score S_{ins} . Thus, S_{DTW} guides the agent during training, while S_{ins} evaluates how different each new repository trajectory is from those already stored.

During repository construction, each newly selected collision trajectory is assigned a DTW-based insertion score. For a trajectory τ_k^* selected at stage k , the score is defined as its nearest-neighbor DTW distance to the repository before insertion:

$$S_{\text{ins}}(\tau_k^*) = \min_{\tau_i \in \mathcal{T}_{k-1}} D_{\text{DTW}}(\tau_k^*, \tau_i). \quad (5.19)$$

This score measures how different the selected trajectory is from the trajectories that have already been stored at the time it enters the repository. When $\mathcal{T}_{k-1} = \emptyset$, the score is undefined because no reference trajectory is available. Therefore, the first repository trajectory is excluded when computing the average insertion score:

$$\bar{S}_{\text{ins}} = \frac{1}{N_\tau - 1} \sum_{k=2}^{N_\tau} S_{\text{ins}}(\tau_k^*). \quad (5.20)$$

This score measures how different the selected trajectory is from the trajectories that have already been stored at the time it enters the repository. Since the score is computed from the gap and relative-speed sequences, a larger insertion score means that the added trajectory differs from its nearest stored neighbor in how the vehicles approach collision. It therefore indicates that the repository is not only adding another successful collision, but also adding a less redundant test scenario, for example one where the gap closes in a different way, the relative speed changes differently before collision, or the collision occurs at a different time.

5.4.2 Coverage comparison

To assess whether the generated collision trajectories represent diverse failure cases rather than repeated variants of the same scenario, trajectory diversity is evaluated in the phase space of gap g and relative speed Δv . The phase space is fixed to $g \in [0, 30]$ m and $\Delta v \in [-14, 0]$ m/s and is discretized into a 30×14 grid, corresponding to cells of 1 m by 1 m/s.

For each trajectory τ_i , each recorded point $(g_t, \Delta v_t)$ is mapped to its corresponding grid cell. Repeated visits to the same cell are counted only once. Let $\mathcal{G}_i$ denote the set of distinct cells visited by trajectory τ_i :

$$\mathcal{G}_i = \{b(g_t, \Delta v_t) \mid (g_t, \Delta v_t) \in \tau_i\}, \quad (5.21)$$

where $b(g_t, \Delta v_t)$ maps a phase-space point to its corresponding grid cell. The phase-space coverage is defined as

$$C_{\text{cov}} = \frac{|\bigcup_{i=1}^{N_\tau} \mathcal{G}_i|}{N_{\text{cells}}}, \quad (5.22)$$

where $N_{\text{cells}} = 30 \times 14$.

This metric measures how much of the predefined phase space is covered by the generated trajectories. Higher coverage means that the repository contains trajectories that pass through more combinations of gap and relative speed, such as different following distances and different closing speeds before collision. This indicates that the repository includes a broader range of pre-collision situations instead of concentrating around a single type of collision.

5.4.3 Pairwise Jaccard distance statistics

Based on the visited grid-cell sets defined above, the pairwise Jaccard distance is used to measure diversity between individual trajectories. For each trajectory pair (i, j) , the Jaccard distance between their visited cell sets $\mathcal{G}_i$ and $\mathcal{G}_j$ is computed as

$$D_{ij} = 1 - \frac{|\mathcal{G}_i \cap \mathcal{G}_j|}{|\mathcal{G}_i \cup \mathcal{G}_j|}. \quad (5.23)$$

A larger D_{ij} indicates that the two trajectories visit more distinct regions of the phase space. The overall pairwise diversity score is defined as the mean pairwise Jaccard distance:

$$D_{\text{mean}} = \frac{2}{N_\tau(N_\tau - 1)} \sum_{i < j} D_{ij}. \quad (5.24)$$

The average is computed only over pairs of different trajectories, with diagonal self-comparisons excluded. A larger D_{ij} indicates that two trajectories visit more distinct regions of the phase space. Therefore, a higher mean pairwise Jaccard distance means that the repository trajectories overlap less in their gap and relative-speed behavior. This indicates lower redundancy among the stored collision scenarios.

Because the simulation uses discrete time steps, the recorded collision state can slightly overshoot the moment when the gap becomes zero, resulting in small negative gap values. These values are clipped to zero before mapping trajectory points to grid cells.

6

Results

6.1 Collision discovery performance

This section reports the outcome of the cross-scenario benchmark defined in Chapter 4. The aggregate collision rate is presented first, followed by a stratified breakdown along the difficulty axes introduced in Section 4.4, and finally two qualitative case studies of individual trajectories.

6.1.1 Collision rate

Across the test set of $N = 1000$ initial conditions, the proposed TD3 policy discovers a collision in 943 scenarios, Monte Carlo Tree Search in 785, and uniform random sampling in 750. The corresponding per-method success rates are summarized in Table 6.1.

TD3 outperforms both reference methods by a substantial margin, with an absolute gap of about sixteen percentage points over MCTS and twenty over RAND. The aggregate rate alone, however, does not reveal which scenarios contribute to this advantage or whether the gap is uniformly distributed across the test set, and so the next subsection partitions the same data by difficulty.

TD3	MCTS	RAND
94.3%	78.5%	75.0%

Table 6.1: Collision discovery rate across $N = 1000$ randomly sampled initial conditions.

6.1.2 Stratified difficulty analysis

To localize where TD3’s advantage over the baseline methods originates, the $N = 1000$ scenarios are partitioned along the safety-margin and maximum-speed axes defined in Section 4.4. Figure 6.1 shows the per-cell success rate of the three methods on the resulting 3×3 grid, and Table 6.2 lists the sample counts and per-method collision counts for each cell.

The proposed TD3 method achieves a 100% success rate on the six cells of the tight and moderate margin rows, while the three loose-margin cells show moderate drops

to 98%, 92%, and 88% for the low-, mid-, and high-speed bins respectively. The two reference methods remain close to TD3 in the tight and moderate margin rows, with RAND showing only minor dips to 94–96% in two low-speed cells. The substantive divergence is confined to the loose margin row. In *(loose, mid)* MCTS reaches 66% and RAND 60% while TD3 retains 92%, opening a gap of roughly thirty percentage points. In *(loose, high)* MCTS reaches 46% and RAND 44% while TD3 holds at 88%, with a gap of about forty percentage points. The remaining loose cell, *(loose, low)*, shows essentially no separation, with MCTS at 99%, RAND at 97%, and TD3 at 98%.

The *(moderate, mid)* and *(moderate, high)* cells contain only $n = 19$ and $n = 9$ scenarios respectively, with *(tight, high)* similarly limited to $n = 19$. This sparsity is structural rather than statistical: the combination of moderate or tight margins with mid- to high-encounter speeds is geometrically narrow under the initial-condition distribution and the RSS-style margin definition. All three methods reach 100% on these under-populated cells, and the per-cell rates therefore cannot be distinguished from 100% at any useful confidence level on the available data; they contribute no signal to the comparative reading that follows.

The breakdown shows that the aggregate gap reported in Table 6.1 originates almost entirely from the mid- and high-speed cells of the loose margin row. In the tight and moderate margin bins, where the controller’s safety margin is already narrow, all three methods can locate a collision. In the *(loose, low)* cell, even with a wide safety margin the encounter speed is low enough that random and shallow-search exploration remain effective. The advantage of the learned policy becomes visible only once both conditions hold, namely a large safety margin that gives the controller substantial reaction room and a high encounter speed at which defeating it requires a sustained, coordinated manipulation of the target’s acceleration. In these two cells the two reference methods cover barely two thirds of the scenarios at mid speed and less than half at high speed, while TD3 retains a 92–88% success rate. This is the regime in which the contribution of TD3 over the baseline methods is most apparent.

Table 6.2: Per-cell sample counts (n) and per-method collision counts on the 3×3 stratified partition of the test set. Each method column reports the number of scenarios in the cell that the corresponding method falsified.

	tight			moderate			loose			Total
	low	mid	high	low	mid	high	low	mid	high	
n	35	57	19	50	19	9	305	299	207	1000
TD3	35	57	19	50	19	9	298	274	182	943
MCTS	35	57	19	50	19	9	302	198	96	785
RAND	33	57	19	48	19	9	296	178	91	750

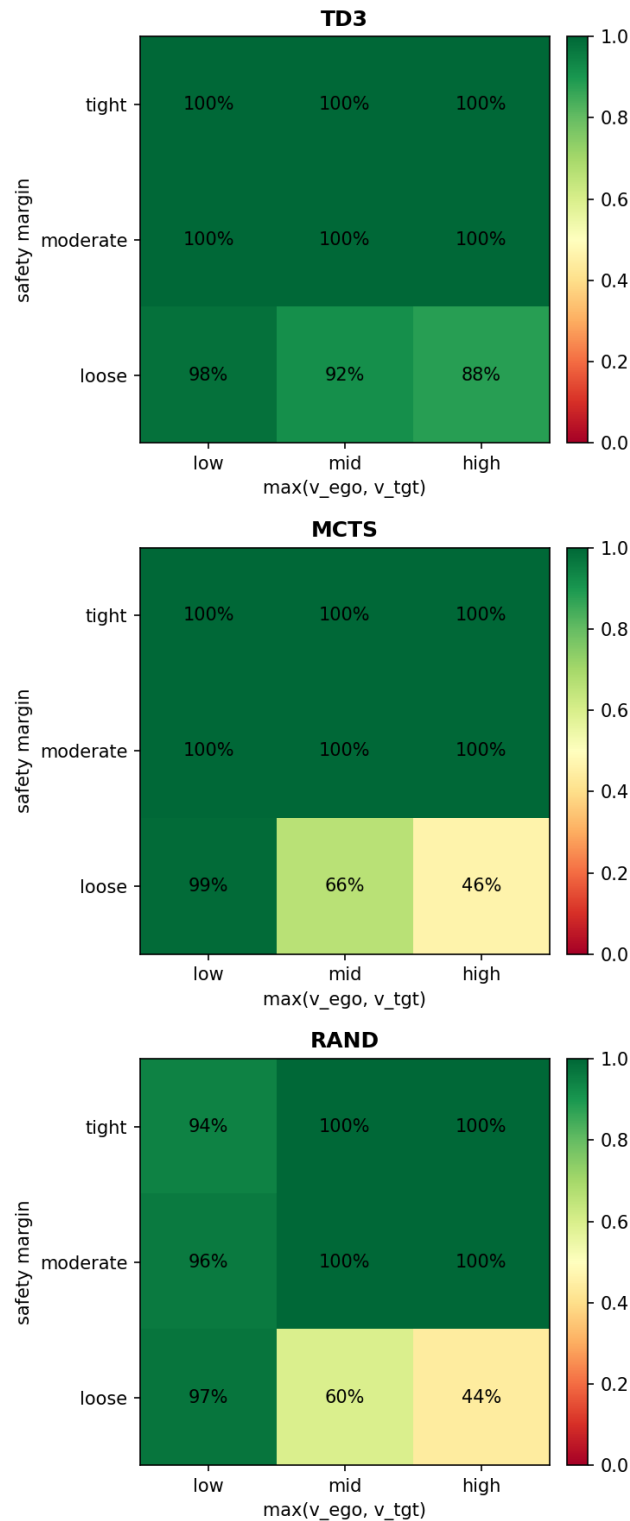


Figure 6.1: Per-cell success rate of TD3, MCTS (50 iterations), and RAND (1 attempt) across 1000 randomly sampled scenarios, stratified by safety margin (rows) and maximum vehicle speed $\max(v_{\text{ego}}, v_{\text{tgt}})$ (columns). Each cell's color reflects the success rate (red→green = 0→1). Margin bins: tight < 2 , moderate $\in [2, 5)$, loose ≥ 5 ; speed bins: low < 20 m/s, mid $\in [20, 30)$, high ≥ 30 m/s.

6.1.3 Cases study

The stratified breakdown identifies where TD3’s advantage is concentrated but does not show what the three methods actually do on the scenarios where they diverge. To make this concrete, we examine two cases from the loose-margin region in detail.

Figure 6.2 shows the first case. The encounter starts with the ego moving slowly, at $v_{\text{ego}} = 5.36$ m/s, while the target is moving at $v_{\text{tgt}} = 35.32$ m/s from a gap of 32.14 m. With the target receding nearly seven times faster than the ego, simply matching its speed already poses a problem; forcing a collision demands more. Under both RAND and MCTS the random commands fail to slow the target consistently, the gap stretches past one hundred meters within three seconds, and both methods exhaust their budget with the two vehicles more than two hundred meters apart. TD3 deviates from this pattern from the very first step. The target’s acceleration is driven negative immediately, and over the following seconds the target velocity is dragged from 35 m/s down towards zero. The ego, in the meantime, is accelerated to a peak speed of about 25 m/s, and once its speed exceeds the (now falling) target’s the gap begins to close, reaching zero at simulated time 13.10 s. The mechanism is plain in the velocity subplot: what distinguishes TD3 from the baselines is the persistence of a directional bias in the target’s acceleration over hundreds of consecutive steps, a pattern unlikely to emerge from independently sampled commands within these budgets.

Figure 6.3 shows the second case, in which TD3 and MCTS both succeed while uniform random sampling fails. The setup is tighter than the first one in initial gap, with the ego starting at $v_{\text{ego}} = 8.13$ m/s and the target at $v_{\text{tgt}} = 24.13$ m/s from a gap of 24.74 m. TD3 reaches a collision at 9.58 s, the successful MCTS rollout requires nearly 12 s, and RAND fails outright, with the target escaping to a final gap of about 70 m. The success of MCTS shows that the class of attack required by these scenarios is not beyond the reach of structured search; given enough rollouts, the tree can discover the same target-deceleration pattern that TD3 finds. The substantial gap in collision time between TD3 and MCTS, on the other hand, shows that the learned policy commits to the attack much earlier and more decisively than the search method, an observation consistent with the lower mean steps-to-collision recorded for TD3 across the benchmark as a whole.

Taken together, the two cases reinforce the reading of the stratified statistics. In the loose-margin regime the controller has substantial reaction room, and the attacks that actually defeat it are extended sequences in which the target is gradually decelerated until the ego can no longer be kept safe. The learned policy synthesizes such sequences directly, in a single deterministic pass over the observation stream; the search and sampling baselines either fail to discover them within their budget or, when they do find them, take considerably more environment interactions to commit to the same pattern.

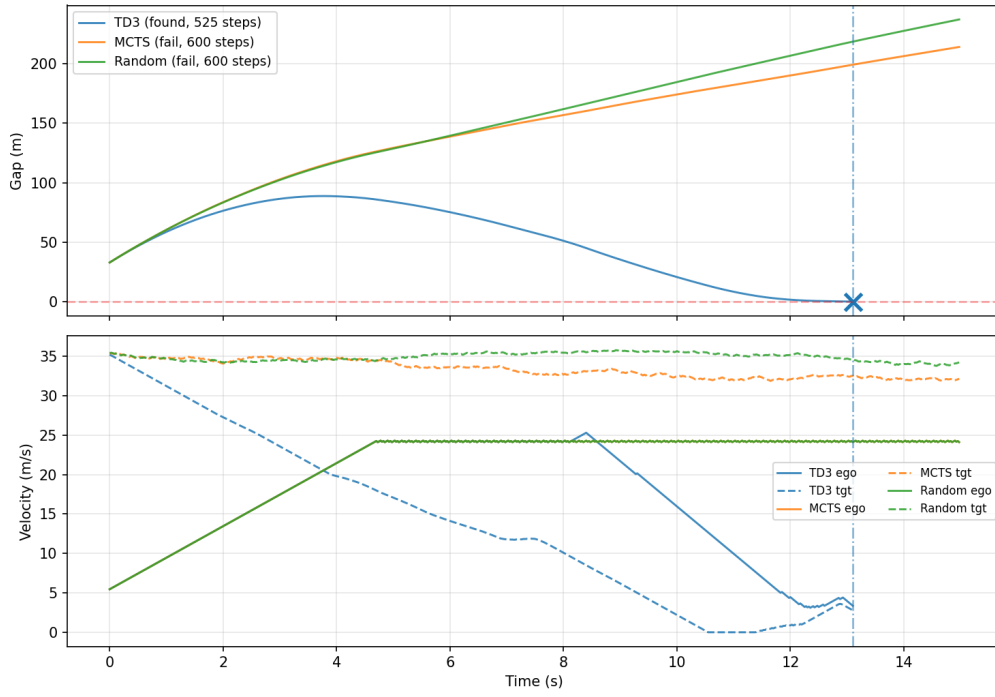


Figure 6.2: Trajectories on the first case study, in which TD3 succeeds while MCTS and RAND fail. The cross marks the moment of collision. And the vertical dash-dotted line indicates the collision time.

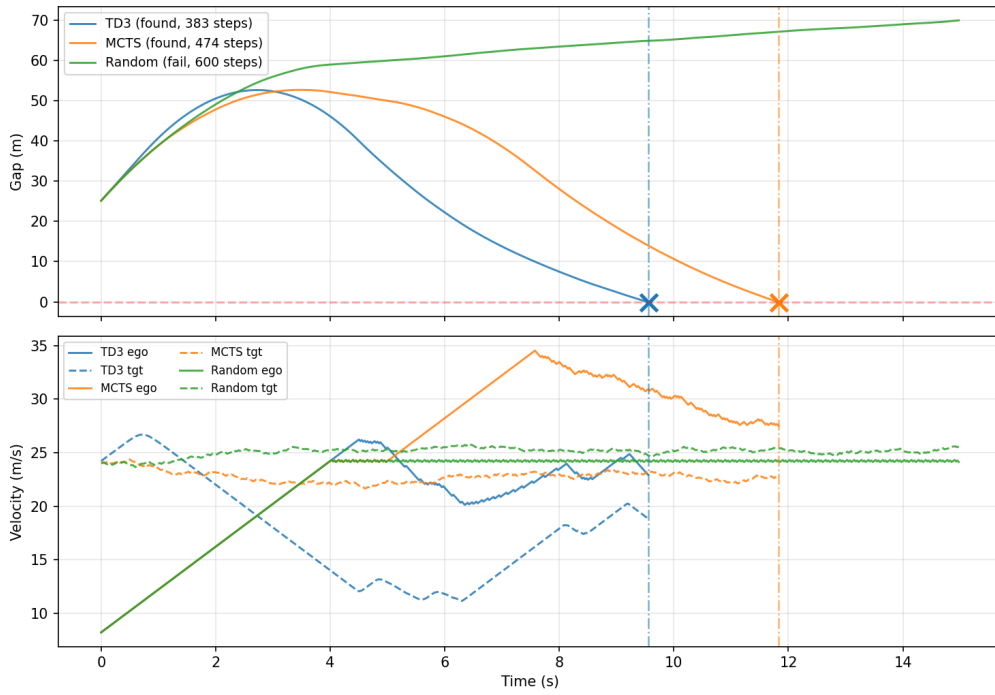


Figure 6.3: Trajectories on the second case study, in which TD3 and MCTS both succeed while RAND fails. The crosses mark the moments of collision. Vertical dash-dotted lines indicate the collision times.

6.2 Trajectory diversity

The results are generated according to the diversity experiment described previously. We compare the baseline Series RL method and the DTW-based Series RL method under the same fixed initial condition $(p_{\text{ego}}, v_{\text{ego}}, p_{\text{tgt}}, v_{\text{tgt}}) = (0, 0, 30, 0)$. For each method, 10 agents are trained sequentially, and the same sequence of random seeds is used in the baseline and DTW-based settings. After each agent finishes training, the final policy and the best checkpoint are evaluated deterministically, and the selected collision trajectory is added to the corresponding repository following the procedure described in Section 5.2.3. Therefore, the two repositories are constructed using the same training schedule and evaluation protocol, with the main difference being whether the DTW-based reward is used during training.

The trajectories in each repository are visualized by plotting position, velocity, gap, and relative speed over time. In the position and velocity plots, solid lines represent the ego vehicle and dashed lines represent the target vehicle. The relative speed is computed as the target vehicle speed minus the ego vehicle speed. The gap and relative speed are the two interaction features used in the DTW computation.

Figure 6.4 shows the repository trajectories obtained by the baseline Series RL method. From the position profiles, the stored trajectories can be roughly separated according to their collision time and final collision position. The red trajectory forms a late collision group with the largest final collision position, and the light blue trajectory forms another delayed collision group. The light purple and light green trajectories can be regarded as a third group, as they collide at similar positions and times. The remaining trajectories form an early collision group, with collisions occurring within a shorter time range and at lower positions.

The velocity profiles provide another view of these groups. The late collision trajectories are associated with different velocity evolution patterns, such as maintaining high speed for a longer period, decelerating before collision, or evolving with a nearly steady speed difference. Apart from small local variations in some trajectories, such as the light orange and pink curves, the velocity plot shows a grouping pattern that is consistent with the position plot.

In the gap plot, the four groups become more explicit because collision directly corresponds to the gap reaching zero. The red trajectory keeps the gap close to the initial value for the longest time and then closes it late. The light blue trajectory also shows a delayed collision, but its gap starts decreasing earlier than the red trajectory. The light purple and light green trajectories form a medium-duration group, with nearly linear gap reduction before collision. The remaining trajectories belong to the early-collision group, where the gap decreases rapidly and reaches zero within a similar time range.

The relative speed plot further explains these gap patterns. Trajectories with similar gap evolution also tend to have similar relative speed profiles, while the delayed

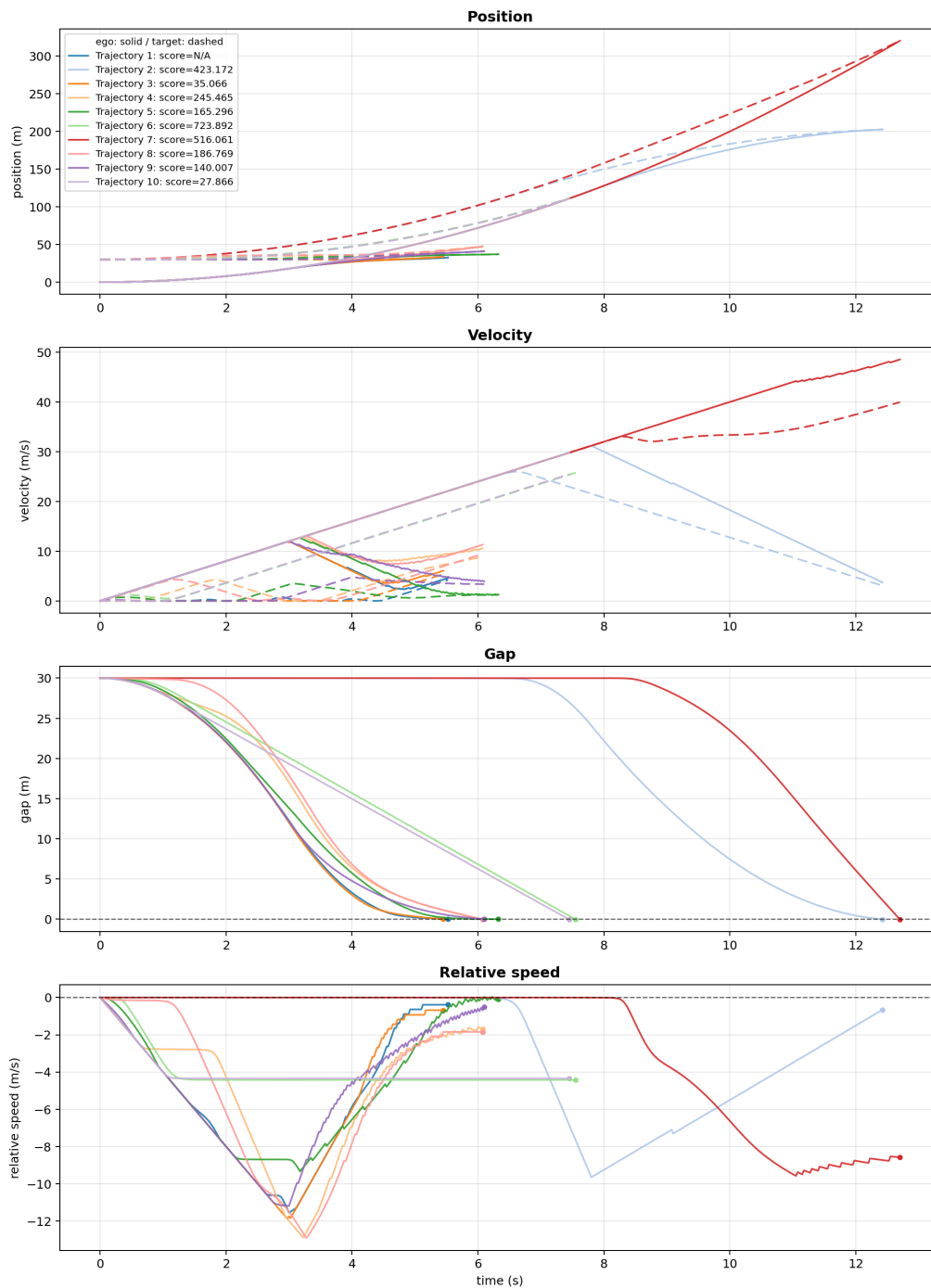


Figure 6.4: Repository trajectories obtained by the baseline Series RL method. The plots show position, velocity, gap, and relative speed over time for the stored collision trajectories. Solid lines denote the ego vehicle and dashed lines denote the target vehicle in the position and velocity plots. Many trajectories follow similar shapes, indicating repetition in the repository.

trajectories show distinct changes in relative speed before collision. Overall, the baseline repository contains several visually distinguishable trajectory types, but a large portion of the stored trajectories still concentrate in a similar early collision pattern.

Figure 6.5 shows the repository trajectories obtained by the DTW-based Series RL method. In the position plot, the trajectories reach collision at different times and locations. This indicates that the repository contains multiple collision outcomes, rather than only trajectories that end at similar positions or after similar durations. The velocity plot shows that these collision outcomes are produced through different speed evolution patterns. Some trajectories collide after a short interaction with relatively low vehicle speeds, while others involve longer acceleration phases and higher vehicle speeds at the collision moment. This indicates that the DTW-based repository differs not only in collision timing and position, but also in the ego and target vehicle speeds when collision occurs.

The relative speed plot also shows clear variation across trajectories. The trajectories differ not only in the trend of relative speed over time, but also in the relative speed at the collision moment. This means that the stored collision cases involve different closing dynamics before collision. Since gap and relative speed are the features used in the DTW computation, the variation observed in these two plots directly reflects the diversity encouraged by the proposed reward.

6.2.1 DTW score comparison

In addition to the trajectory plots, Table 6.3 compares the DTW-based insertion scores defined in Section 5.4.1. The first trajectory in each repository is used as the initial reference. Since there is no previously stored trajectory for comparison, its DTW score is undefined and is not included in the average score. Therefore, the score comparison starts from Trajectory 2.

Table 6.3: DTW scores of repository trajectories for the baseline and DTW-based Series RL methods.

Trajectory	Baseline	DTW-based
2	423.172	704.433
3	35.066	149.557
4	245.465	435.727
5	165.296	271.402
6	723.892	662.462
7	516.061	464.653
8	186.769	281.734
9	140.007	353.869
10	27.866	303.736
Average	273.733	403.064

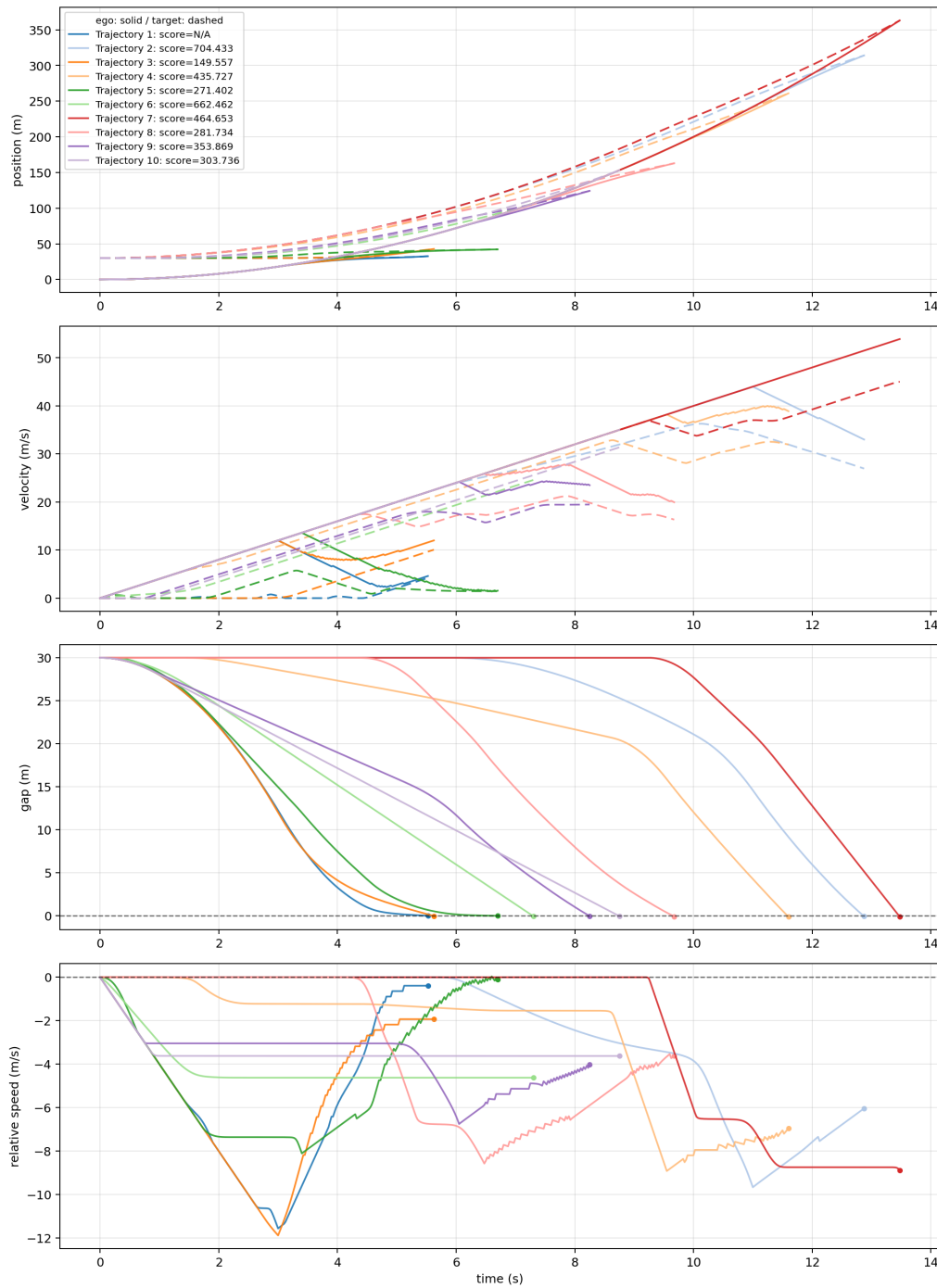


Figure 6.5: Collision trajectories stored in the DTW-based Series RL repository. The plots show position, velocity, gap, and relative speed over time. Solid lines denote the ego vehicle and dashed lines denote the target vehicle in the position and velocity plots. The trajectories show greater variation in collision time and pre-collision dynamics.

As shown in Table 6.3, the average insertion score increases from 273.733 for the baseline to 403.064 for the DTW-based method. Since each score measures the distance to the closest trajectory already in the repository, this increase means that newly added trajectories are generally less similar to previously stored ones in their gap and relative-speed evolution. The method therefore discovers more varied ways in which collisions develop, rather than repeatedly adding small variations of the same interaction. For testing, this provides a broader set of pre-collision behaviors under which the response of the SUT can be examined.

It should be noted that these scores are insertion-time scores. A high score means that the trajectory was far from the repository when it was added, not necessarily that it is the farthest trajectory from all other trajectories in the final repository. After later trajectories are added, the nearest neighbor of an earlier trajectory may change. Therefore, the DTW score comparison should be interpreted as evidence of how the repository was progressively expanded, rather than as a final pairwise diversity ranking among all stored trajectories.

6.2.2 Coverage comparison

To further evaluate trajectory diversity, we analyze the coverage of the repository trajectories in the phase space of gap and relative speed. This phase space is directly related to the DTW-based comparison because gap and relative speed are the two features used to represent each trajectory. The phase space is divided into grid cells, and a cell is marked as visited if at least one repository trajectory passes through it. Therefore, a larger number of visited cells indicates that the repository covers a broader range of interaction states.

Figures 6.6 and 6.7 show the visited grid cells for the baseline and DTW-based repositories, respectively. The colored curves represent the repository trajectories in the phase space, while the shaded cells indicate the union of all visited grid cells.

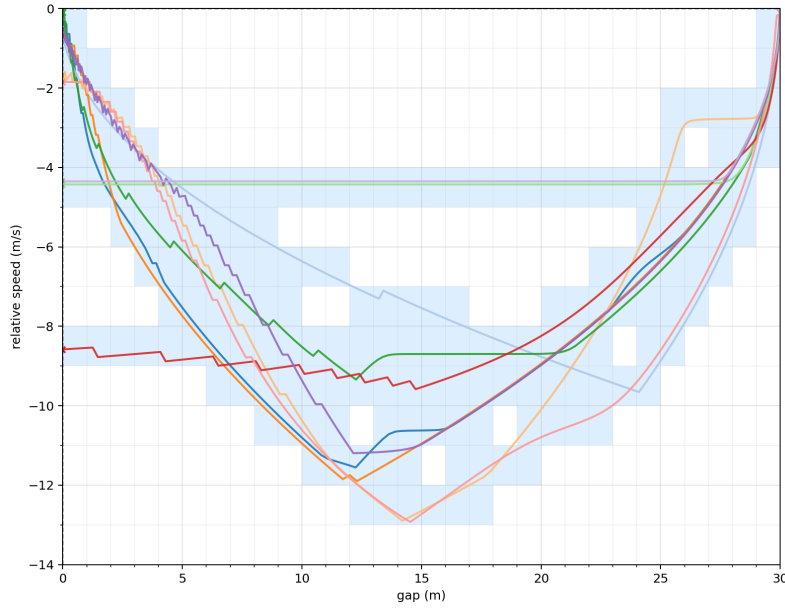


Figure 6.6: Collision trajectories from the baseline Series RL repository in the phase space of gap and relative speed. Colored curves represent individual trajectories, and shaded cells indicate regions visited by at least one trajectory. The repository visits 167 of 420 cells, corresponding to 39.76% coverage, with several trajectories concentrated in similar regions.

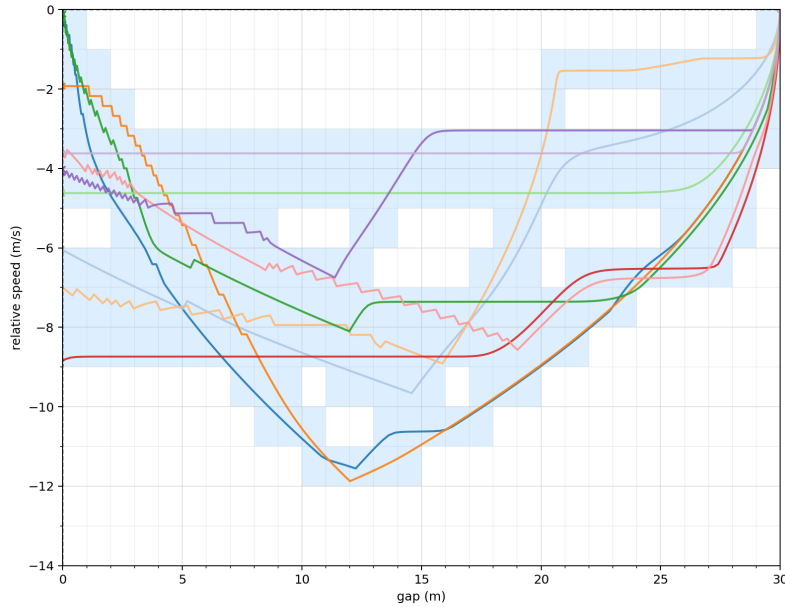


Figure 6.7: Collision trajectories from the DTW-based Series RL repository in the phase space of gap and relative speed. Colored curves represent individual trajectories, and shaded cells indicate regions visited by at least one trajectory. The repository visits 190 of 420 cells, corresponding to 45.24% coverage. This is 5.48 percentage points higher than the baseline, indicating broader coverage of pre-collision interaction patterns.

The DTW-based repository visits 23 more cells than the baseline, corresponding to an increase of 5.48 percentage points. This means that its trajectories pass through more combinations of gap and relative speed before collision. The increase therefore reflects broader diversity in the interaction states reached, rather than only differences in trajectory length or collision time. In practical testing, these additional regions correspond to a wider range of following distances and closing speeds under which the SUT is evaluated.

6.2.3 Pairwise Jaccard distance statistics

To further examine trajectory overlap, Tables 6.4 and 6.5 report the pairwise Jaccard distance matrices computed from the visited grid cells. A larger D_{ij} indicates less overlap between two trajectories in the phase space of gap and relative speed. Since the pairwise Jaccard distance is symmetric, i.e., $D_{ij} = D_{ji}$, each trajectory pair is computed only once, although the results are displayed as symmetric matrices for readability. The diagonal entries are omitted because they correspond to comparing each trajectory with itself.

Table 6.4: Pairwise Jaccard distances D_{ij} for the baseline Series RL repository.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
T1	-	0.893	0.173	0.800	0.707	0.893	0.767	0.874	0.594	0.877
T2	0.893	-	0.875	0.859	0.773	0.853	0.882	0.708	0.789	0.884
T3	0.173	0.875	-	0.793	0.712	0.889	0.757	0.869	0.554	0.871
T4	0.800	0.859	0.793	-	0.798	0.913	0.867	0.543	0.705	0.897
T5	0.707	0.773	0.712	0.798	-	0.861	0.764	0.845	0.703	0.890
T6	0.893	0.853	0.889	0.913	0.861	-	0.897	0.893	0.877	0.086
T7	0.767	0.882	0.757	0.867	0.764	0.897	-	0.902	0.746	0.912
T8	0.874	0.708	0.869	0.543	0.845	0.893	0.902	-	0.817	0.921
T9	0.594	0.789	0.554	0.705	0.703	0.877	0.746	0.817	-	0.859
T10	0.877	0.884	0.871	0.897	0.890	0.086	0.912	0.921	0.859	-

Table 6.5: Pairwise Jaccard distances D_{ij} for the DTW-based Series RL repository.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
T1	-	0.929	0.530	0.952	0.740	0.878	0.882	0.848	0.911	0.907
T2	0.929	-	0.915	0.750	0.895	0.944	0.868	0.963	0.870	0.862
T3	0.530	0.915	-	0.952	0.753	0.877	0.880	0.831	0.882	0.905
T4	0.952	0.750	0.952	-	0.877	0.957	0.897	0.962	0.944	0.941
T5	0.740	0.895	0.753	0.877	-	0.900	0.855	0.692	0.919	0.914
T6	0.878	0.944	0.877	0.957	0.900	-	0.955	0.866	0.776	0.902
T7	0.882	0.868	0.880	0.897	0.855	0.955	-	0.700	0.957	0.955
T8	0.848	0.963	0.831	0.962	0.692	0.866	0.700	-	0.746	0.913
T9	0.911	0.870	0.882	0.944	0.919	0.776	0.957	0.746	-	0.627
T10	0.907	0.862	0.905	0.941	0.914	0.902	0.955	0.913	0.627	-

To summarize the pairwise Jaccard distance matrices, Table 6.6 reports the minimum, maximum, mean, and standard deviation of D_{ij} over all trajectory pairs.

The DTW-based repository has a higher mean pairwise Jaccard distance than the baseline, increasing from 0.785 to 0.866. This indicates that the stored trajectories overlap less in the phase space of gap and relative speed. The minimum distance also increases from 0.086 to 0.530, showing that the baseline contains at least one highly

Table 6.6: Summary statistics of pairwise Jaccard distances.

Method	Min D_{ij}	Max D_{ij}	Mean D_{ij}	Std. D_{ij}
Baseline	0.086	0.921	0.785	0.169
DTW-based	0.530	0.963	0.866	0.094

similar trajectory pair, while even the most similar pair in the DTW-based repository remains more separated. The DTW-based repository also has a lower standard deviation, indicating that the Jaccard distances among its trajectory pairs vary less. Therefore, the improvement is not only caused by a few very different trajectories. Instead, most trajectories in the DTW-based repository remain reasonably separated from each other. As a result, the final repository is less redundant. Its stored collision scenarios cover more distinct parts of the phase space, instead of repeatedly visiting the same region. This supports the conclusion that the DTW-based reward improves trajectory diversity not only by adding a few outlying trajectories, but by producing a repository in which most trajectories differ from each other. For testing, this means that a repository of the same size contains fewer near-duplicate cases and represents more distinct interaction patterns.

7

Exploratory studies and broader implications

This chapter presents additional observations from the development of the falsification framework and discusses the broader implications of the proposed approach. The exploratory studies address the third research question by identifying factors that affected learning stability, collision discovery, and the types of failure scenarios found. These observations are not intended as controlled benchmark comparisons. The chapter then discusses how the framework could be extended to more realistic environments and how it could be used in practical automotive software testing.

7.1 Additional Exploratory studies

7.1.1 Algorithm selection

The main experiments use TD3 because the falsification task has a continuous action space and benefits from an off-policy method that can reuse previous simulation experience. During development, PPO was also tested to check whether collision discovery was possible with a different policy-gradient algorithm. In these preliminary runs, PPO could discover some failure scenarios, but the experiments were not controlled enough to support a direct comparison with TD3. PPO was therefore not included as a main baseline in the final evaluation. The observation is only used to indicate that the adversarial formulation itself is not tied exclusively to TD3, while the quantitative conclusions of the thesis are based on the TD3 experiments.

7.1.2 Reward shaping

Because the reward signal determines what behavior the adversarial agent is encouraged to learn, several reward formulations were considered during development. In addition to terminal feedback, process reward terms were introduced to provide feedback before the end of an episode. These included a gap-related reward, which encouraged the agent to reduce the distance between the ego vehicle and the target vehicle, and a time penalty, which encouraged failures to occur earlier in the episode.

These additional reward terms appeared to make learning easier in some settings because the agent received intermediate feedback instead of only receiving feedback at the terminal state. At the same time, they seemed to influence which scenarios

were discovered. By rewarding small gaps and early failures, the reward function may have biased the search toward particular failure patterns. In the explored settings, process rewards appeared helpful for learning, but they also seemed to guide the search toward a narrower set of failure patterns.

7.1.3 Action space design

Another design choice was how much control the adversarial agent should have over the simulated interaction. During development, this was examined by comparing a target-only agent with an agent that also proposed an ego-vehicle signal, while the final ego acceleration remained constrained by the SUT.

The two configurations led to different discovered trajectories, indicating that the choice of output signals affects the types of scenarios found. The two-output configuration was also harder to train, possibly because the larger output space and the intervention of the SUT made the learning problem more complex. This suggests that the effectiveness of adversarial falsification depends not only on the learning algorithm, but also on how the agent’s outputs are connected to the simulated system.

7.1.4 Environmental complexity

A further question was how the method behaved when the simulator was made more realistic than the simplified benchmark used for the main experiments. An additional setting included brake delay in the environment and in the ego vehicle response. This made the interaction between the agent, the environment, and the SUT more complex. Under this setting, successful collision-inducing behavior was only observed when using a process reward. However, training was less stable and more sensitive to parameter choices and random seeds. For example, a policy that succeeded under one seed could fail to learn collision-inducing behavior under another.

These observations suggest that increasing the complexity of the simulation environment has an ambiguous effect. Additional factors can create more possible paths to failure, but they also enlarge the learning problem and can make training less stable. Therefore, a more complex environment does not necessarily make falsification easier or harder in a simple way. Its effect depends on whether the additional failure opportunities outweigh the increased difficulty of learning in a larger and more complex search space.

7.2 Generalization to realistic environments

The experiments in this thesis were conducted in a simplified longitudinal environment with one ego vehicle and one adversarial target vehicle. The results show that the trained policy can handle different initial positions and velocities within the ranges considered during training. However, the experiments do not show that

the policy can transfer directly to different road layouts, vehicle models, or traffic situations.

In principle, the same RL-based falsification framework could be extended to more realistic two-dimensional environments. Such environments could include lateral motion, curved or multi-lane roads, more realistic vehicle dynamics, sensor uncertainty, and delayed vehicle responses. The state representation would then need to include additional information, such as lateral position, heading, lane occupancy, and the relative motion of nearby actors. Similarly, the action space could include steering and lane-change decisions in addition to longitudinal acceleration. The general closed-loop formulation would still apply, but the reward function, state representation, and choice of RL algorithm would need to be redesigned for the extended environment.

The framework could also be extended to control multiple surrounding vehicles or other road users, such as pedestrians. This would allow the falsifier to search for failures caused by coordinated interactions between several actors. However, every additional actor introduces more state variables and possible actions. The agent must then determine not only how each actor should behave, but also how their actions should be coordinated to produce a critical scenario. Pedestrians and vehicles also have different dynamics and behavioral constraints, making it more difficult to define actions that are both adversarial and realistic.

The effectiveness of RL-based falsification also depends on the design of the simulator. The simulator determines what the agent can observe and control, and how vehicle dynamics, sensor uncertainty, and response delays are modeled. A simplified simulator may produce failures that depend on unrealistic assumptions, while a more realistic simulator requires more computational resources and may make training more difficult. The brake-delay experiment in this thesis illustrates this dependency, as even a relatively small increase in environmental complexity made training less stable and more sensitive to the reward design and random seed. Therefore, the practical scalability of the proposed approach depends on both the RL method and the simulation environment.

7.3 Practical implications

The proposed approach could be used as an offline testing tool for safety-critical automotive software. Instead of relying only on manually designed test cases, an RL agent could search the simulation environment for interactions that lead to failures. Once trained, the policy could be applied to different initial conditions to generate critical scenarios with relatively few simulation runs.

The diversity-based method could further help produce test cases with different interaction patterns. The discovered trajectories could be stored in a repository, reviewed by engineers, and converted into regression tests. These tests could then be repeated after changes to the SUT to determine whether previously discovered

failures have been corrected or reintroduced.

However, the generated scenarios should be checked for physical and behavioral realism before being used in practice. A failure discovered in a simplified simulator may not occur in a higher-fidelity environment or in the real world. In addition, failure to discover a collision does not prove that the SUT is safe. The proposed method should therefore complement other verification and testing methods rather than replace them.

8

Conclusion

This thesis investigated deep reinforcement learning as a method for falsifying safety-critical decision and control software in closed-loop simulation. The falsification task was formulated as an adversarial control problem in which a learning agent controls a target vehicle and searches for behaviors that can drive the system under test into collision. In line with the research questions introduced in Chapter 1, the study examined whether an adversarial RL agent can efficiently discover relevant failure scenarios, whether a DTW-based diversity reward can increase the variety of discovered collision trajectories, and which formulation and training factors affect the effectiveness of RL-based falsification.

The first research question concerned collision discovery across relevant test scenarios. The TD3-based falsifier was evaluated on 1000 randomly sampled and physically avoidable initial conditions. It achieved a collision discovery rate of 94.3%, compared with 78.5% for MCTS and 75.0% for uniform random sampling. This shows that, in the tested longitudinal scenario, the learned adversarial policy was more effective at exploiting the closed-loop interaction than the two non-learning baselines. The stratified analysis further showed that evaluating only an aggregate success rate is insufficient, since the difficulty of falsification depends strongly on the initial safety margin and vehicle speeds.

The second research question asked whether diversity-aware training could broaden the set of discovered failures. To address this, the thesis introduced a DTW-based Series RL method that rewards newly discovered collision trajectories according to their dissimilarity from trajectories already stored in a repository. Compared with the baseline Series RL method, the DTW-based method produced a more diverse repository of collision trajectories. The average insertion-time DTW score increased from 273.733 to 403.064, the phase-space coverage increased from 39.76% to 45.24%, and the mean pairwise Jaccard distance increased from 0.785 to 0.866. These results indicate that the diversity reward reduced repeated discovery of similar collision patterns and encouraged broader exploration of failure trajectories.

The third research question focused on factors that affect the effectiveness of adversarial falsification. The exploratory studies showed that effectiveness depends on how the problem is formulated. Several factors, including the reward function, agent outputs, simulation environment, and learning setup, affected both the training process and the discovered scenarios. Because these factors interact, it is difficult to attribute the observed behavior to any single design choice. These observations

highlight the importance of considering the full falsification setup when designing and evaluating adversarial testing methods.

This work was limited to a simplified longitudinal traffic scenario with a small number of controllable actors and selected environment parameters. Future work should evaluate the method in richer traffic scenarios, including more surrounding vehicles, additional sources of uncertainty, and more realistic vehicle dynamics. Future studies should also investigate how the discovered failure scenarios can be integrated into a continuous testing pipeline for safety-critical automotive software.

Bibliography

- [1] Yashwanth Annpureddy, Che-Yuan Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6605 of *Lecture Notes in Computer Science*, pages 254–257. Springer, 2011.
- [2] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.
- [3] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [4] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, AAAIWS’94*, page 359–370. AAAI Press, 1994.
- [5] Marc Brittain, Josh Bertram, Xuxi Yang, and Peng Wei. Prioritized sequence experience replay. *arXiv preprint arXiv:1905.12726*, 2019.
- [6] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.
- [7] Xuan Cai, Xuesong Bai, Zhiyong Cui, Peng Hang, Haiyang Yu, and Yilong Ren. Adversarial stress test for autonomous vehicle via series reinforcement learning tasks with reward shaping. *IEEE Transactions on Intelligent Vehicles*, 10(2):832–847, 2025.
- [8] Anthony Corso, Robert Moss, Mark Koren, Ritchie Lee, and Mykel Kochenderfer. A survey of algorithms for black-box safety validation of cyber-physical systems. *Journal of Artificial Intelligence Research*, 72, October 2021.
- [9] Thomas Degris, Martha White, and Richard S. Sutton. Off-policy actor-critic. In *Proceedings of the 29th International Conference on International Conference on Machine Learning, ICML’12*, page 179–186, Madison, WI, USA, 2012. Omnipress.
- [10] Wenhao Ding, Chejian Xu, Mansur Arief, Haohong Lin, Bo Li, and Ding Zhao. A survey on safety-critical driving scenario generation—a methodological perspective. *IEEE Transactions on Intelligent Transportation Systems*, 24(7):6971–6988, 2023.
- [11] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Proceedings of the 22nd International Conference on*

- Computer Aided Verification*, CAV'10, page 167–170, Berlin, Heidelberg, 2010. Springer-Verlag.
- [12] Andréa Doreste, Matteo Biagiola, and Paolo Tonella. Adversarial testing with reinforcement learning: A case study on autonomous driving. In *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 293–304, 2024.
 - [13] Georgios Fainekos and George Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 2009.
 - [14] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 2018.
 - [15] Erann Gat, R Peter Bonasso, and Robin Murphy. On three-layer architectures. In David Kortenkamp, R Peter Bonasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, pages 195–210. MIT Press, Cambridge, MA, USA, 1998.
 - [16] Hado van Hasselt. Double q-learning. In *Proceedings of the 24th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'10, page 2613–2621, Red Hook, NY, USA, 2010. Curran Associates Inc.
 - [17] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et du jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.
 - [18] Xinyu Ji, Lei Xue, Zhijian He, and Xiapu Luo. Autonomous driving system testing via diversity-oriented driving scenario exploration. *ACM Trans. Softw. Eng. Methodol.*, 34(7), August 2025.
 - [19] Nidhi Kalra and Susan M Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? Technical Report RR-1478-RC, RAND Corporation, Santa Monica, CA, 2016.
 - [20] Mykel J. Kochenderfer, Christopher Amato, Girish Chowdhary, Jonathan P. How, Hayley J. Davison Reynolds, Jason R. Thornton, Pedro A. Torres-Carrasquillo, N. Kemal Üre, and John Vian. *Decision Making Under Uncertainty: Theory and Application*. The MIT Press, 07 2015.
 - [21] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning (ECML)*, pages 282–293. Springer, 2006.
 - [22] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 1008–1014. MIT Press, 1999.
 - [23] Philip Koopman, Beth Osyk, and Jack Weast. Autonomous vehicles meet the physical world: RSS, variability, uncertainty, and proving safety. In *Computer Safety, Reliability, and Security: 38th International Conference, SAFECOMP*. Springer, 2019.
 - [24] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J. Kochenderfer. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–7, 2018.
 - [25] Atanu Kundu, Sauvik Gon, and Rajarshi Ray. Data-driven falsification of cyber-physical systems, 2025.

-
- [26] Sampo Kuutti, Saber Fallah, and Richard Bowden. Training adversarial agents to exploit weaknesses in deep control policies, 2020.
 - [27] Ritchie Lee, Mykel J. Kochenderfer, Ole J. Mengshoel, Guillaume P. Brat, and Michael P. Owen. Adaptive stress testing of airborne collision avoidance systems. 2015.
 - [28] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv: Learning*, 2015.
 - [29] Henry X Liu and Shuo Feng. Curse of rarity for autonomous vehicles. *Nature Communications*, 15(1):4808, 2024.
 - [30] Oded Maler and Dejan Ničković. Monitoring temporal properties of continuous signals. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 3253 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2004.
 - [31] Quentin Mazouni, Helge Spieker, Arnaud Gotlieb, and Mathieu Acher. Testing for fault diversity in reinforcement learning. In *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024)*, page 136–146. ACM, April 2024.
 - [32] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML’16*, page 1928–1937. JMLR.org, 2016.
 - [33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dhharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.
 - [34] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
 - [35] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.
 - [36] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.
 - [37] Xintao Yan Haojie Zhu Zhengxia Zou Shengyin Shen Henry X. Liu Shuo Feng, Haowei Sun. Dense reinforcement learning for safety validation of autonomous vehicles. *Nature*, 615(7953):620–627, 2023. Publisher: Nature Publishing Group.
 - [38] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings*

- of the 31st International Conference on International Conference on Machine Learning - Volume 32, ICML'14, page I–387–I–395. JMLR.org, 2014.
- [39] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018.
- [40] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pages 1057–1063. MIT Press, 2000.
- [41] Luiz Felipe Vecchietti, Minah Seo, and Dongsoo Har. Sampling rate decay in hindsight experience replay for robot control. *IEEE Transactions on Cybernetics*, 52(3):1515–1526, 2022.
- [42] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992.
- [43] Yoriyuki Yamagata, Shuang Liu, Takumi Akazaki, Yihai Duan, and Jianye Hao. Falsification of cyber-physical systems using deep reinforcement learning. *IEEE Transactions on Software Engineering*, 47(12):2823–2840, 2021.
- [44] Jingxuan Yang, Ruoxuan Bai, Haoyuan Ji, Yi Zhang, Jianming Hu, and Shuo Feng. Adaptive testing environment generation for connected and automated vehicles with dense reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 26(4):5135–5145, 2025.
- [45] Shangdong Zhang and Richard S. Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se



CHALMERS
UNIVERSITY OF TECHNOLOGY