VERIFICATION-GUIDED SAFETY-REPAIR OF LEARNED CONTROLLERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Formal verification can certify that a learned controller is safe. But what should be done when verification fails? Simply *suppressing actions* that lead to unsafe next states is insufficient because the dangerous action that ultimately caused the safety violation might have occurred many steps earlier. *Retraining* modifies the policy globally, thereby potentially introducing new failures. Instead, we introduce *ReachECO*, a verification-guided repair procedure that exploits the information of failed multi-step reachability analysis for synthesizing explicit constraints on controller behavior. For Differentiable Weightless Controllers, the repairs compile into exact circuit patches that leave the controller unchanged everywhere outside the targeted policy cells. We prove *soundness* and *termination* of the repair procedure for ReachECO, as well as, assuming exhaustive continuation search, *completeness* relative to the reachability abstraction. Empirically, we demonstrate ReachECO's ability to successfully repair otherwise unsafe controllers in three continuous-control benchmarks: ACC, Docking2D and Car2D.

1 INTRODUCTION

Deep Reinforcement learning (RL) has been extensively used to train agents that interact with digital environments, such as computer games (Vinyals et al., 2019). Recently, RL has also become an important part of post-training for large-language-models (Ouyang et al., 2022). The original inspiration for RL, however, was to synthesize controllers for agents that interact with the physical world, such as cyber-physical, robotic, and autonomous systems (Rupprecht & Wang, 2022; Tang et al., 2025). Training these with RL is especially appealing when their desired behavior is difficult to hardcode, but can be expressed through a reward function (Sutton & Barto, 1998).

While powerful, using RL for training systems that are meant to interact in real-world environments also has some downsides. In particular, empirically optimizing the reward function does not provide any formal *guarantees* on the resulting controller (Alshiekh et al., 2018; Brunke et al., 2022; Krasowski et al., 2023). This is particularly consequential in safety-critical applications, such as cyber-physical systems, where failures can have severe physical consequences. Stating safety requirements is often straightforward. For example, an autonomous vehicle should not collide with another road user, a robotic system should not enter a forbidden region, or a physical process should remain within prescribed operating limits (Zhao et al., 2023). However, *enforcing* them is far from straightforward, as the consequences of an action may only become apparent several steps later. Arguing about safety therefore requires *closed-loop* reasoning about the evolution of the system (controller as well as environment) over time.

A common approach to closed-loop verification of neural-network control systems is to compute the set of reachable states, or an over-approximation of it, by repeatedly propagating sets through the neural-network controller and the system dynamics (Xiang & Johnson, 2018; Ivanov et al., 2019; 2021; Dutta et al., 2019; Tran et al., 2020; Huang et al., 2022a; Schilling et al., 2022; Lopez et al., 2023; Sidrane et al., 2022; Tian et al., 2024). If the set of all reachable states does not intersect with the unsafe set, the controller can indeed be certified to be safe. What, however, can be done if the sets do intersect? **This is the central question we address in this work: given a trained controller that cannot be certified to be safe, how can it be *repaired* such that it subsequently passes closed-loop safety verification?**

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Existing approaches to *closed-loop controller repair* can be broadly categorized into two categories: *local* and *trajectory-level*. *Local methods* derive safety repairs from single-step constraints or counterexamples (Yang et al., 2022; Santa Cruz et al., 2022). However, a state might be safe itself, but have no available action that will keep it safe in the next state. For instance, a car approaching an obstacle may already be traveling too fast for braking to avoid a later collision. Consequently, single-step repair shifts the difficulty of reasoning about the system's temporal dynamics to the availability of additional safety information that summarizes future behavior, such as a larger *losing set*, i.e., a set of states from which an unsafe state is eventually unavoidable. Computing such sets exactly, however, can be computationally intractable and is even formally undecidable for sufficiently general hybrid systems (Henzinger et al., 1995).

Trajectory-level methods influence the multi-step behavior of the controller, such that it avoids not only unsafe regions of the state space but also regions that inevitably lead to unsafe states at a later step (Zhou et al., 2020; Hamilton et al., 2022; Lu et al., 2023; Lyu et al., 2023; Lu et al., 2025). Such methods typically consider individual trajectories and do not consider the verifier results during the repair. Hence, they can only establish post-hoc formal guarantees for *all* possible trajectories. Furthermore, they generally rely on fine-tuning to repair the original neural network, thereby forfeiting any guarantees of termination or progress (Boetius et al., 2023). One exception is Lu et al. (2023), which searches directly over counterfactual controller behaviors. Its guarantee, however, is tied to the analyzed run-time violation rather than to all trajectories from an initial-state region.

In this work, we take inspiration from the setting of static feed-forward networks. There, neural-network repair can be formulated directly as a problem of enforcing explicit constraints on the output: verification may identify inputs or regions whose outputs must change, and the repaired network is required to satisfy these constraints. This approach enables strong formal guarantees on the repair procedure (Goldberger et al., 2020; Sotoudeh & Thakur, 2021; Fu & Li, 2022; Tao et al., 2023). However, extending this idea to closed-loop systems is nontrivial because the required modification may lie several time steps before the observed safety violation, and the repair constraint must account for the resulting future behavior.

Our main contribution, ReachECO, solves this problem in an effective yet efficient way for abstraction-based controllers. ReachECO uses closed-loop reachability analysis to derive repair constraints over future behavior, bringing constraint-based repair and its formal guarantees to the multi-step closed-loop setting.

Specifically, we formulate ReachECO for *abstraction-based neural-network controllers* (Jin et al., 2022; Tian et al., 2024; 2023) and instantiate it with Differentiable Weightless Controllers (DWCs) (Kresse & Lampert, 2026b). These are controllers that have an exact boolean circuit representation, such that repairs can be formulated as exact local circuit patches. We then prove termination and, for an exhaustive variant, completeness relative to the environment over-approximation. Finally we evaluate ReachECO on several closed-loop control benchmarks.

2 Verification-Guided Closed-Loop Repair

We now present **Reach**ability-derived **E**ngineering **C**hange **O**rder (ReachECO), a verification-guided procedure that derives controller-repair constraints from failed multi-step closed-loop reachability analysis. The name is inspired by engineering change orders (ECOs), which modify an existing circuit implementation to satisfy revised design requirements while preserving as much of the original implementation as possible (Lin et al., 1995; Krishnaswamy et al., 2009). Similarly, ReachECO seeks to restore certifiability through localized patches to the controller rather than replacing it globally. Before we do so, we introduce the main concepts necessary to reason about the properties of discrete systems that operate in a continuous environment, in particular their safety. Formal definitions and additional details are given in Appendix C.

Abstraction-based Control. We study a control task with continuous state space $\mathcal{X} \subseteq \mathbb{R}^d$ and system dynamics governed by a differential equation, $\dot{x} = f(x, a)$, where $a \in \mathcal{A}$ is a control action. We consider controllers $\pi : \mathcal{X} \to \mathcal{A}$ that are evaluated every Δ time units, with the selected action held constant between controller updates (called *zero-order hold*). While cyber-physical systems necessarily have to process real-valued inputs, because they operate in continuous (real-world) environments, it has been observed that forming *abstractions* of the continuous state-space into finite sets

can often preserve near-optimal behavior (Abel, 2019). Similarly, it has been observed that strong *quantization* of a neural controller's input space often incurs only limited performance degradation for standard continuous-control tasks (Kresse & Lampert, 2026a;b).

Such abstractions can substantially simplify formal verification: controllers defined over a finite input partition avoid much of the complexity associated with having to bound neural network activations over continuous input regions, enabling significantly more scalable closed-loop reachability analysis (Jin et al., 2022; Tian et al., 2023; 2024; Yang et al., 2025). Consequently, in this work, we focus on *piecewise-constant abstraction-based controllers* (Def. 2, Appendix C), which partition the state space into finitely many axis-aligned *policy cells* $\mathcal{Q}$ and assign a single action $a \in \mathcal{A}$ to each cell $q \in \mathcal{Q}$.

Differentiable Weightless Controllers Importantly to maintain efficiency, the use of abstraction-based controllers is not contradictory to gradient-based learning. In particular, *Differentiable Weightless Controllers (DWCs)* (Kresse & Lampert, 2026b) can be trained with gradient descent but nevertheless admit an exact representation as a Boolean circuit. Their quantized input encoding creates the finite policy partition $\mathcal{Q}$, while their quantized outputs lead to a finite action alphabet $\mathcal{A}$. Hence, DWCs provide a natural instance of the abstraction-based controller class considered here. DWC circuits consist primarily of Boolean lookup tables (LUTs). A 6-input LUT (LUT6), as commonly provided by modern FPGAs, stores the truth table of an arbitrary Boolean function of six inputs, requiring only $2^6 = 64$ bits (AMD, 2026). Consequently, DWC inference can be implemented directly as Boolean circuit evaluation without floating-point multiplication. Despite this strongly quantized representation, DWCs have been shown to learn policies of comparable quality as more conventional neural-network controllers on challenging continuous-control tasks, such as Humanoid (Kresse & Lampert, 2026b). Hence, we instantiate our framework for DWCs. In Section 2.3, we exploit the exact circuit representation to realize repairs as bit-exact local patches of the original controller.

Abstraction-Based Closed-Loop Reachability Analysis. *Closed-loop reachability analysis* propagates sets of possible states through the controller and system dynamics. Given a set $B \subseteq \mathcal{X}$ and action a, we assume *sound* over-approximations $\overline{\mathrm{Post}}(B, a)$ of the one-step successor set and $\overline{\mathrm{Tube}}(B, a)$ of all states visited before the next controller step, where *soundness* means that no trajectory is omitted that the true system starting in B under a could take. Every over-approximation is further enclosed by an axis-aligned hyperrectangle in $\mathbb{R}^d$, which we call a *box*, to enable efficient propagation by the verifier that performs the reachability analysis.

For an abstraction-based controller, let every box B be contained in exactly one policy cell, which we denote by $q(B) \in \mathcal{Q}$. Hence, the controller assigns a single exact action $\pi(q(B))$ to all states in B. For such a box, we denote by $\mathrm{Succ}(B, a)$ a set of *successor boxes* by computing $\overline{\mathrm{Post}}(B, a)$ and, because the post-image may intersect multiple of the policy cells, subsequently splitting along policy-cell boundaries. For scalability, if multiple boxes fall into the same policy cell, they may also be merged again (Jin et al., 2022; Tian et al., 2023; 2024; Yang et al., 2025).

Now, starting from an initial set of boxes, $\mathcal{X}_0$, repeated application of Succ yields a sound over-approximation of the closed-loop reachable set. Let $\overline{\mathcal{X}}_k$ denote the collection of reachable boxes at step k, then we define, for any time horizon $T \in \mathbb{N}$:

$$\overline{\mathrm{Reach}}_{\pi,T}(\mathcal{X}_0) := \bigcup_{k=0}^{T-1} \bigcup_{B \in \overline{\mathcal{X}}_k} \overline{\mathrm{Tube}}(B, \pi(q(B))). \tag{1}$$

Consequently, $\overline{\mathrm{Reach}}_{\pi,T}(\mathcal{X}_0)$ over-approximates every state visited by the closed-loop system from $\mathcal{X}_0$ during the first T control steps. We now have all components to formally define our central object of interest: *certified safety*.

Definition 1 (Certified safety)**.** *A controller π is called* certified safe to horizon $H \in \mathbb{N}$*, if*

$$\overline{\mathrm{Reach}}_{\pi,H}(\mathcal{X}_0) \cap \mathcal{U} = \emptyset, \tag{2}$$

where $\mathcal{U} \subseteq \mathcal{X}$ is the set of unsafe states. We call π certified safe to horizon $H = \infty$ *if for some step k the reachable-set computation establishes an inductive fixed point,*

$$\bigcup \overline{\mathcal{X}}_{k+1} \subseteq \bigcup \overline{\mathcal{X}}_k, \qquad \textit{and} \qquad \overline{\mathrm{Reach}}_{\pi,k+1}(\mathcal{X}_0) \cap \mathcal{U} = \emptyset. \tag{3}$$

Algorithm 1 REACHECO

Input: π, $\mathcal{X}_0$, $\underline{\mathcal{L}}$, budget T, backend $\mathcal{W}$
$\pi' \leftarrow \pi$; $\Gamma \leftarrow \emptyset$; $\overline{\mathcal{X}}_0 \leftarrow \mathcal{X}_0$; $k \leftarrow 0$
while $k < T$ **do**
  $(\overline{\mathcal{X}}_{k+1}, \mathcal{F}) \leftarrow \textsc{ReachStep}(\overline{\mathcal{X}}_k, \pi', \underline{\mathcal{L}})$
  if $\mathcal{F} \neq \emptyset$ **then**
    $(s, \pi', \Delta\underline{\mathcal{L}}) \leftarrow \textsc{BRepair}(\mathcal{F}, \underline{\mathcal{L}}, \mathcal{W}, k)$
    $\underline{\mathcal{L}} \leftarrow \underline{\mathcal{L}} \cup \Delta\underline{\mathcal{L}}$ {certified losing}
    if $s = \textsc{Unknown}$ **then**
      return UNKNOWN
    end if
    $k \leftarrow \textsc{EarliestAffected}(\pi', \Delta\underline{\mathcal{L}})$
  else if $\bigcup \overline{\mathcal{X}}_{k+1} \subseteq \bigcup \overline{\mathcal{X}}_k$ **then**
    return (π', ∞) {inductive invariant}
  else
    $k \leftarrow k + 1$
  end if
end while
return (π', T)

Algorithm 2 BREPAIR. Γ is the set of committed constraints and is stored across calls.

Input: next step failing boxes $\mathcal{F}, \underline{\mathcal{L}}, \mathcal{W}, t$
$\overline{\Gamma} \leftarrow \emptyset$; $L_0 \leftarrow \underline{\mathcal{L}}$
while true do
  $(\Gamma_w, \mathcal{F}_o, \Delta\underline{\mathcal{L}}) \leftarrow \textsc{FindCont}(\mathcal{F}, \mathcal{W})$
  $\underline{\mathcal{L}} \leftarrow \underline{\mathcal{L}} \cup \Delta\underline{\mathcal{L}}$; $\overline{\Gamma} \leftarrow \overline{\Gamma} \cup \Gamma_w$
  if $\mathcal{F}_o = \emptyset$ **then**
    $(s, \pi', \mathcal{F}_o, \Gamma, \overline{\Gamma}) \leftarrow \textsc{Joint}(\Gamma, \overline{\Gamma})$
    if $s = \textsc{Sat}$ **then**
      $\Gamma \leftarrow \Gamma \cup \overline{\Gamma}$
      return $(\textsc{Repair}, \pi', \underline{\mathcal{L}} \setminus L_0)$
    else if $s = \textsc{Unknown}$ **then**
      return $(\textsc{Unknown}, \pi', \underline{\mathcal{L}} \setminus L_0)$
    end if
  end if
  if $\exists (B, h) \in \mathcal{F}_o : \ h = t + 1$ **then**
    return $(\textsc{Unknown}, \pi', \underline{\mathcal{L}} \setminus L_0)$
  end if
  $\mathcal{F} \leftarrow \textsc{Backtrack}(\mathcal{F}_o)$
end while

In addition, for a reach-avoid task, any trajectory that has entered the absorbing goal region is subsequently assumed to be safe.

A controller that has been proven *certified safe* is also safe under the actual dynamics f. Conversely, however, the inability to certify a controller does not imply that it is necessarily unsafe. This, however, is not a shortcoming of this specific definition. It is known that *exact reachability*, and therefore safety, is not only computationally intractable but formally *undecidable* for sufficiently general hybrid systems (Henzinger et al., 1995).

Controller Repair. When a controller fails safety verification, a natural next step is to *repair* it. Formally, we define a *repair instance* as an abstraction-based controller π_0 together with sound propagation operators $\overline{\mathrm{Post}}$ and $\overline{\mathrm{Tube}}$ for the system dynamics, an initial set $\mathcal{X}_0 \subseteq \mathcal{X}$, an unsafe set $\mathcal{U} \subseteq \mathcal{X}$, an optional absorbing goal set $\mathcal{X}_\mathcal{G} \subseteq \mathcal{X}$, and a verification budget T. The goal is to obtain a controller π' that is *certified safe* in the sense of Definition 1, while retaining as much of the behavior of the original controller π_0 as possible. Repairs may assign each policy cell a new action from a finite repair alphabet $a \in \mathcal{A}_{cont} \subseteq \mathcal{A}$ or keep the original action $\pi_0(q)$.

2.1 REACHECO

ReachECO enables effective yet efficient controller repair. Algorithm 1 and 2 provide pseudo-code, based on a number of arguments and subroutines that we will introduce in this section. Appendix C gives additional details and formal definitions. We provide a high-level sketch of ReachECO's working principles and how it achieves local repair in a closed-loop setup.

At its core, ReachECO alternates between forward closed-loop reachability analysis and backward repair search. It maintains the sets of reachable boxes $\overline{\mathcal{X}}_0, \ldots, \overline{\mathcal{X}}_k$ computed under the current controller together with a certified losing set $\underline{\mathcal{L}}$. The latter contains regions for which the algorithm has established that no admissible continuation can avoid a future safety violation. Initially, $\underline{\mathcal{L}}$ may contain only the unsafe set $\mathcal{U}$ and can be expanded as additional losing regions are certified. Throughout $\underline{\mathcal{L}}$ is a subset of the *true* losing set $\mathcal{L}^*$. We call action a *one-step safe* from a box B if its inter-sample tube avoids $\mathcal{U}$ and its post-image avoids the current losing set underapproximation:

$$\mathrm{SafeStep}(B, a) \ :\Longleftrightarrow \ \overline{\mathrm{Tube}}(B, a) \cap \mathcal{U} = \emptyset \ \wedge \ \overline{\mathrm{Post}}(B, a) \cap \underline{\mathcal{L}} = \emptyset. \tag{4}$$

Suppose forward verification encounters one or more boxes $B \in \overline{\mathcal{X}}_k$ for which $\mathrm{SafeStep}(B, \pi(q(B)))$ is not fulfilled. ReachECO collects these failing boxes as the initial *repair*

obligations $\mathcal{F}$. We then search for alternative controller actions that admit a safe continuation. The continuation backend $\mathcal{W}$ determines which candidate actions are explored. If no continuation exists from a failing box, then intervening at that time step is already too late, and ReachECO backtracks to $\overline{\mathcal{X}}_{k-1}$. The required *safe* continuation horizon, however, increases by one as we want to establish a safe continuation up to $k+1$, precisely where the initial safety violation was encountered. This backtracking and forward search process continues until a safe continuation is found for every failure or the initial set is reached. During this search, ReachECO may also certify additional regions as losing and add them to $\underline{\mathcal{L}}$.

Each successful continuation search yields a repair *constraint*. Each constraint ensures a safe continuation from the obligation that generated it. Constraints discovered across different failures and repair rounds are accumulated and must ultimately be satisfied by a new controller π'. This joint realization is nontrivial because independently derived constraints may require incompatible actions in the same policy cell. Section 2.2 describes how ReachECO resolves such conflicts while preserving progress of the repair procedure. Once a jointly realizable repair is found, the controller is updated. Because this change may alter previously computed reachable sets, ReachECO resumes forward verification from the earliest time step affected by either the controller modification or an expansion of $\underline{\mathcal{L}}$.

In summary, **at the core of ReachECO lies an intuitive procedure of alternating forward propagation with lazy backtracking. Our main contribution lies in how safety failures are handled when they are encountered: resolving conflicts while ensuring finite progress and, under exhaustive-search assumptions, completeness relative to the verifier abstraction.**

2.2 Obligations, Constraints and Conflicts

We next describe how ReachECO represents and combines the repairs discovered during backward search. We first introduce *obligations*, which encode the need to establish a safe continuation from a reachable region. We then show how successful continuation searches are stored as *repair constraints*. Finally, we describe how constraints from different failures are jointly resolved while ensuring progress of the repair procedure.

Obligations. An obligation is a pair (B, h), where B is a box and h is the number of future control steps for which a safe continuation must be established. The initial obligation set $\mathcal{F}$ is obtained from the boxes that violated the SafeStep predicate during forward reachability propagation. Each one is initialized with horizon $h = 1$. Starting from B, FindCont uses the same sound set propagation as forward verification, but explores alternative actions proposed by $\mathcal{W}$. Thus, for an obligation (B_0, h), the search has the following alternating structure:

$$\begin{aligned} &\exists a_0 \ : \ \mathrm{SafeStep}(B_0, a_0) \\ &\quad \wedge \ \forall B_1 \in \mathrm{Succ}(B_0, a_0) \ \exists a_1 \ : \ \mathrm{SafeStep}(B_1, a_1) \\ &\qquad\qquad \vdots \\ &\quad \wedge \forall B_{h-1} \in \mathrm{Succ}(B_{h-2}, a_{h-2}) \ \exists a_{h-1} \ : \ \mathrm{SafeStep}(B_{h-1}, a_{h-1}). \end{aligned} \tag{5}$$

The backend $\mathcal{W}$ determines which candidate actions are explored. An exhaustive backend considers every action in $\mathcal{A}_{\mathrm{cont}}$. This leads to a finite-horizon safety game-like search (Tabuada, 2009).

If FindCont cannot establish an h-step safe continuation from B, the obligation is returned as unresolved ($\mathcal{F}_o$). BackTrack then replaces it by obligations on its predecessor box at the preceding time step and increases the required continuation horizon to $h+1$. Thus, every backtracking step searches for an earlier intervention while accounting for one additional future step. If no safe continuation can be established from time step 0, ReachECO returns Unknown: under the employed reachability abstraction and continuation backend, no feasible continuation could be established from that obligation.

Constraints. A successful continuation search is represented by a finite game DAG $\mathcal{G}$ containing the safe continuations discovered by the backend $\mathcal{W}$. Each node v corresponds to a box B_v and stores the candidate actions that admit a safe continuation for the remaining horizon h_v. At the root node v_0, these are exactly the box and horizon of the corresponding obligation. The successor boxes

generated by each action, that were established to satisfy Equation 5, appear as child nodes with horizon $h_v - 1$.

Together with its root time step i, the game forms a *repair constraint* $\gamma = (\mathcal{G}, i)$. By retaining all feasible continuations found by $\mathcal{W}$, rather than committing immediately to a single one, the subsequent joint realization step has flexibility when combining repairs from different failures. We call γ *anchored* if $i = 0$, since its root has reached the initial time step and cannot be backtracked further. ReachECO maintains two sets of constraints. The *committed* constraints Γ were established in previous calls to BREPAIR, whereas the *current* constraints $\overline{\Gamma}$ were generated during the present call. This distinction matters during conflict resolution because committed constraints where generated under an earlier controller.

Conflict Resolution. Once every current obligation has produced a repair constraint, JOINT searches for a single stationary controller that satisfies all accumulated constraints. Satisfying a constraint means selecting one of its continuations: whenever a game node is reached, the controller must choose one of the actions stored at that node. Since every box belongs to a policy cell $q \in \mathcal{Q}$, realizing a continuation ultimately imposes action requirements on these cells.

Constraints generated independently need not be jointly realizable. In particular, different continuations may require incompatible actions in the same policy cell, while the repaired controller must assign exactly one action to each cell. We encode this finite joint-realization problem using CP-SAT (Perron & Didier, 2025), requiring one action per policy cell and satisfaction of every repair constraint. Among feasible assignments, we prefer controllers that remain close to the original policy.

Accordingly, JOINT has three possible outcomes: SAT, returning a jointly satisfying controller; BACKTRACK, returning a set of conflicting non-anchored constraints as new obligations $\mathcal{F}_o$; or UNKNOWN, if the conflict cannot be resolved because it involves only anchored constraints. If the constraints are jointly infeasible, ReachECO first backtracks conflicting non-anchored constraints generated in the current repair round. If the conflict instead involves *anchored* constraints, non-anchored committed constraints may be discarded. Anchored constraints are never discarded. Consequently, ReachECO returns UNKNOWN only when repair fails at the initial time step: either no safe continuation can be established for an anchored obligation, or the anchored constraints are jointly infeasible under the employed abstraction and backend.

2.3 ECO PATCH COMPILATION

The repair from JOINT returns a set of policy cells $\mathcal{Q}_r \subseteq \mathcal{Q}$, where actions differ from the original controller. The repair is realized as:

$$\pi'(x) = \pi_0(x) + \sum_{q_r \in \mathcal{Q}_r} g_{q_r}(x)\, \Delta_r, \qquad (6)$$

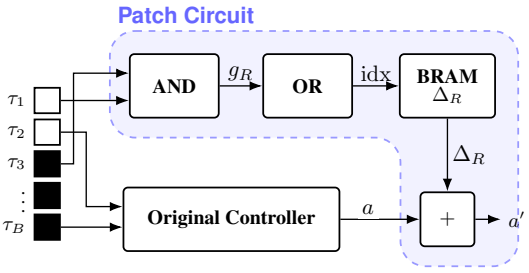

Figure 1: ECO patch architecture. Existing thermometer predicates are AND-reduced to implement the policy-cell identification g_R, from these an address selecting the stored correction Δ_R is obtained. The correction is applied exactly and only to repaired policy cells.

For DWCs, an initial thermometer encoding (Carneiro et al., 2015) provides the threshold predicates needed to identify each policy cell. We therefore implement each g_{q_r} as an AND reduction over these bits. The resulting one-hot guards are encoded into a memory address selecting the corresponding correction Δ_r, which is added to the DWC's integer output. Address zero stores the zero correction, so the controller is unchanged outside the repaired cells. The final BRAM-based output decoding stage of the DWC remains unchanged. Figure 1 illustrates the constructed patch. It is similar to support-network repair in REASSURE (Fu & Li, 2022) and lookup-based patch architectures (Tao et al., 2023), but realizes the repair exactly as a digital circuit. Note that the patch can be executed largely in parallel with the original controller. Circuit-size estimation and implementation details are given in Appendix F.2.

2.4 THEORETICAL GUARANTEES

The purpose of ReachECO is to provide controllers with formal guarantees; hence, the repair procedure itself should also admit formal guarantees. We state these below as theorems. The corresponding proofs and implementation assumptions are given in Appendix D. Soundness applies to the full procedure. For simplicity, termination and completeness are stated for fixed $\underline{\mathcal{L}} = \mathcal{U}$ and without merging (see Appendix C, implementation details). Extending the proofs to monotone losing-set and merge-refinement updates requires additional bookkeeping.

Theorem 1 (Soundness)**.** *Suppose ReachECO returns (π', H) with $H \in \{T, \infty\}$ rather than* UNKNOWN*. Then π' is certified safe to horizon H (Definition 1). In particular, no trajectory of the closed-loop system under π' starting in $\mathcal{X}_0$ enters $\mathcal{U}$ during the first H control steps, and for all time if $H = \infty$.*

Theorem 2 (Termination)**.** *Assume that the initial set has a finite abstract representation, and that the policy partition $\mathcal{Q}$, the continuation alphabet $\mathcal{A}_{\text{cont}}$, and the verification horizon T are finite. Furthermore, assume* JOINT *always returns a repair if there exists one under the constraints. Consider $\underline{\mathcal{L}} = \mathcal{U}$ fixed and no merging. Then ReachECO terminates and returns either (π', H), with $H \in \{T, \infty\}$, or* UNKNOWN*.*

Here $\mathrm{Safe}_T(B, \pi)$ denotes that every transition in the unmerged T-step unfolding from B under π satisfies $\mathrm{SafeStep}$; its recursive definition is given in Appendix C. We denote with $\mathcal{A}^{\mathcal{Q}}$ the set of all controllers with the action alphabet $\mathcal{A}$ over the grid $\mathcal{Q}$.

Theorem 3 (Completeness under exhaustive search)**.** *Under the assumptions of Theorem 2, let $\mathcal{X}_0$ be the fixed initial box representation, containing every anchored root. Use the exhaustive backend over the original action alphabet $\mathcal{A}$. If some $\overline{\pi} \in \mathcal{A}^{\mathcal{Q}}$ satisfies*

$$\forall B \in \mathcal{X}_0 : \quad \mathrm{Safe}_T(B, \overline{\pi}),$$

then ReachECO returns a controller $\pi' \in \mathcal{A}^{\mathcal{Q}}$ that is certified safe (Definition 1).

ReachECO attempts to synthesize a controller that remains close to the original policy π_0 in terms of the L_1-distance in action space. However, finding an optimal repair is generally computationally hard: even minimal strategy repair in finite reachability games is NP-complete (Gaillard et al., 2023). Hence, ReachECO employs heuristics and accepts non-optimal assignments after a search timeout.

3 EXPERIMENTS

We investigate three tasks, showing different properties of ReachECO across them: *Adaptive Cruise Control (ACC)* (Fulton & Platzer, 2018; Teuber et al., 2024), *Docking2D* from the AINNCS competition (Lopez et al., 2025), and *Car2D*, which uses the NAV dynamics from (Lopez et al., 2025). We briefly summarize Docking2D and Car2D here and defer their full analysis to Appendix A. All experiments were performed on an AMD EPYC 9534 (Zen 4) CPU, with an NVIDIA L40S GPU used where applicable. In particular, ReachECO's reachability analysis runs on the GPU.

3.1 ADAPTIVE CRUISE CONTROL

Setup. A controlled vehicle must avoid a constant-velocity lead vehicle while maintaining comfortable acceleration. Relative distance and velocity are denoted by p_{rel} and v_{rel}, respectively, with $v_{\text{rel}} < 0$ indicating that the vehicles approach each other. We train five DWCs following Kresse & Lampert (2026b). The system has a known losing set $\mathcal{L}^*$ (Figure 2, everything below dotted black line): states above its boundary can avoid collision by braking, whereas in states below the line, an eventual collision is unavoidable. We initialize ReachECO with boxes entirely outside $\mathcal{L}^*$ satisfying $v_{\text{rel}} < 0$ and $p_{\text{rel}} < \mathcal{P}_{max}$, covering $\approx 94\%$ of the corresponding safe region. The unsafe set $\mathcal{U}$ is defined as $p_{rel} \leq 0$, corresponding to a rear-end collision between the lead and controlled vehicle. All ReachECO runs use $T = 200$, and all runs reach a closed-loop fixed point within this budget, certifying infinite-horizon safety. Details of the dynamics and training are given in Appendix E.

Losing set knowledge reduces runtime. We cumulatively evaluate four levels of losing set knowledge: (i) no $\underline{\mathcal{L}}$ expansion, (ii) $\underline{\mathcal{L}}$ expansion, (iii) initialization with an offline-computed $\underline{\mathcal{L}}$ fixed point, and (iv) the exact braking invariant. $\mathcal{W}$ uses 11 discrete actions including zero and both extrema. We additionally evaluate a single-action full-braking backend $\mathcal{W}_{brake}$, under (ii).

Table 1 shows that stronger safety information reduces repair runtime. For $\mathcal{P}_{max} = 30$, runtime drops from 366s without losing-set learning to 244s with online learning, 26s with an underapproximated initialized losing set, and 3s with $\mathcal{L}^*$. The full-braking and invariant additionally scales to $\mathcal{P}_{max} = 100$ (Table 2). At this initial set size, games of depth-11+ are necessary, making $\mathcal{W}$ infeasible. As shown in Figure 6 in the Appendix: losing-set expansion eliminates a large number of deep continuation games. At $\mathcal{P}_{max} = 30$, this reduces depth-3–7 games by a large amount, while the knowledge of $\mathcal{L}^*$ requires only depth-1 games.

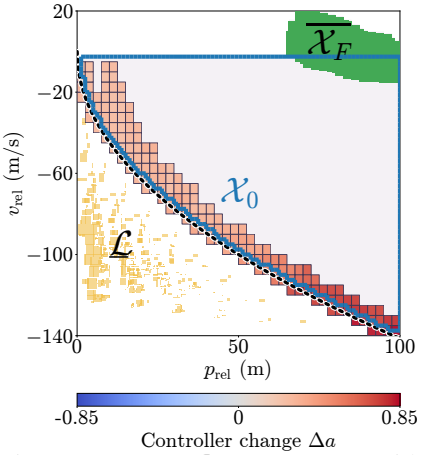

Figure 2: ACC ($\mathcal{P}_{max} = 100$, with $\mathcal{W}_{brake}$): Showing $\mathcal{X}_0$, the repaired cells with the associated magnitude change, and the final fixed-point $\overline{\mathcal{X}}_F$. The learned losing set $\underline{\mathcal{L}}$ is shown in orange. Changes in the controller are localized close to the losing set boundary, which ReachECO was not provided with in this run.

Table 1: ACC: verification + repair runtime [s] at $\mathcal{P}_{max} = 30$ over 5 seeds.

Method	t [s]
No losing-set	366 ± 253
Online losing-set	244 ± 253
Initial losing-set	26 ± 28
$\mathcal{W}_{brake}$	4 ± 1
$\mathcal{L}^*$	3 ± 1

Repairs are localized. ReachECO successfully repairs all five DWC seeds and increases their mean reward from 2434 to 3861; per-seed results are reported in Table 5 in the Appendix. Qualitatively, Figure 2 shows that the resulting modifications concentrate near the boundary of $\mathcal{L}^*$, even when ReachECO is provided only with the unsafe set $\mathcal{U}$. Table 2 further shows that access to $\mathcal{L}^*$ and the more general backend $\mathcal{W}$ reduces both the number of modified policy cells and the mean action change.

Other benchmarked methods require $\mathcal{L}^*$ or do not certify. We compare against methods with and without access to $\mathcal{L}^*$. When $\mathcal{L}^*$ is available, we construct counterexample-guided repair loops using either BBReach (Tian et al., 2024) for DWCs or NCubeV (Teuber et al., 2024) for ReLU controllers. Verification identifies failing regions, based on which the controller is repaired and reverified until the first successful certificate is obtained. For supervised repair, we fine-tune on the identified failures together with safe transitions from the original controller to reduce unintended changes; full details are given in Appendix E. For the ReLU controller, we additionally combine NCubeV with REASSURE (Fu & Li, 2022), which constructs a local patch network from the identified unsafe regions. We cannot certify the resulting patched network with NCubeV because its architecture is outside the verifier's supported network class, and therefore report the resulting repair size without a closed-loop certificate. Among the methods that obtain a certificate of safety, ReachECO is faster and produces smaller behavioral changes than the supervised fine-tuning baselines, requiring only a few hundred additional LUT6s to repair the DWC. REASSURE style patches yield a smaller amout of changes, This is at least in part due to the fact that the underlying ReLU network is already safer than the initial DWCs. Nevertheless, it requires a patch size of 2350 additional *floating-point* parameters.

In a second setup, we assume that only the unsafe set is known; for this setting, we use STL-Gym (Hamilton et al., 2022) to fine-tune the original controller for an additional 200,000 time steps. Figure 5 in the Appendix shows the resulting reward and empirical safety across training. Training is unstable across seeds: in some runs reward degrades substantially, and none of the resulting controllers achieves 100% empirical safety.

3.2 Docking2D and Car2D: Takeaways

The main analysis of these two case studies is deferred to Appendix A, we highlight main results.

Docking2D: Reachability enables *highly* localized patches. Docking2D has a large continuous state space and long closed-loop horizons. Nevertheless, ReachECO certifies all our three DWC controllers, till docked, while modifying only a small number of cells. For the seed exhibiting the

Table 2: ACC at $\mathcal{P}_{max} = 100$. t denotes the runtime for verification plus repair. If verification fails, t reports repair time only. Runtime and repair size are reported over five trained model seeds. † denotes a ReLU network. For ReachECO, *Param.* denotes the number of patch LUT6s; for NCubeV+REASSURE, results are reported only for the network from (Teuber et al., 2024), and *Param.* denotes the number of additional floating-point parameters. The first column indicates the losing set information provided: $\mathcal{L}^*$ denotes the known losing set, $\mathcal{U}$ denotes only the unsafe set.

			Repair			
	Method	Verified	t [s]	Param.	Change [%]	mean $\lvert\Delta\rvert$
$\mathcal{L}^*$	ReachECO	✓	$2_{\pm 1}$	$223_{\pm 48}$	$8.0_{\pm 1.9}$	$0.035_{\pm 0.013}$
	BBReach + Supervised	✓	$65_{\pm 10}$	0	$96.9_{\pm 7.0}$	$0.335_{\pm 0.125}$
	NCubeV + Supervised†	✓	$1760_{\pm 1188}$	0	$47.77_{\pm 4.51}$	$0.036_{\pm 0.012}$
	NCubeV + REASSURE†	?	88	2350	1.2	0.003
$\mathcal{U}$	ReachECO ($\mathcal{W}_{\text{brake}}$)	✓	$22_{\pm 4}$	$340_{\pm 26}$	$12.4_{\pm 3.2}$	$0.056_{\pm 0.023}$
	STLGym†	×	$53_{\pm 5}$	0.0	$38.5_{\pm 0.3}$	$0.120_{\pm 0.036}$

most empirically unsafe transitions, only 3,326 policy cells are changed, corresponding to less than $10^{-3}\%$ of the overall policy cells.

Car2D: enabling certification through small changes. The original controller exhibits no unsafe behavior in 1,000 sampled rollouts, yet closed-loop reachability fails to certify it, identifying a potential safety violation after three timesteps. ReachECO repairs the controller in 292 s, requiring continuation games of up to depth 3 while modifying only 42 policy cells. Several repaired regions are almost never visited by the sampled trajectories. These regions may correspond either to rare behaviors missed by sampling or to conservatism introduced by the reachability overapproximation; in either case, modifying them is sufficient to make the controller certifiable.

4 Summary and Discussion

We introduced ReachECO, which enables targeted repair of abstraction-based neural network controllers to fulfill formally certified safety conditions. On a technical level, ReachECO brings the constraint-based view established in static neural-network repair to the closed-loop setting by explicitly accounting for the future behavior induced by controller modifications. Repair constraints are derived lazily from failed forward reachability analysis and subsequent finite-horizon continuation reasoning. This formulation enables strong formal guarantees on the repair procedure, including termination and, for an exhaustive variant, completeness relative to the reachability abstraction. For the special class of Differentiable Weightless Controllers (DWCs), we further showed that the resulting repairs can be compiled into exact circuit patches. Because ReachECO only modifies regions encountered during closed-loop reachability analysis, the resulting repairs can remain highly localized in the controller's input space. At the circuit level, the additional logic consists primarily of LUT6s, allowing the repair to be realized as a lightweight overlay on the original controller.

Some limitations remain: ReachECO relies on *deep continuation games*, which can become computationally expensive for long-horizon problems. This can be mitigated by additional prior information, such as knowledge of the losing region. Our experiments show that even partial losing-set information can substantially reduce the required search. Even though repair patches are highly localized in the controller's input space, many of them might be needed for originally unsafe controllers. A promising direction for future work is to exploit additional structure during patch compilation.

Our current implementation additionally relies on relatively coarse, benchmark-specific post-image and inter-sample tube computations chosen for efficient GPU execution. Tighter GPU-based reachability procedures could reduce conservatism, while more general tube constructions would make the approach easier to transfer across dynamical systems. Finally, our experiments consider relatively low-dimensional systems, as is common in closed-loop reachability benchmarks. Scaling both verification and the additional repair search to higher-dimensional dynamics remains the most important future challenge.

AI USE STATEMENT

In this work, we used generative AI tools for developing conceptual frameworks, formulating mathematical claims, assisting in the writing of the presented proofs, for providing feedback on research methodology and experiments, implement methods, and interpreting results. We have not used generative AI tools for proposing or refining hypotheses, supporting qualitative and thematic data analysis, or providing critical ingredients for proving mathematical claims. Synthetic data sets generation, assisting with translation, cleaning and reformatting datasets are not applicable to this work.

Additionally, we used generative AI tools to assist with figure generation, code prototyping, drafting and polishing text, brainstorming, searching for information, and proposing keywords for the paper. The tube overapproximations used in this work were initially derived with assistance from generative AI tools. We reviewed all AI-assisted work. The original prototype implementation was extensively audited, rewritten, and restructured, and AI-provided technical information was carefully reviewed and independently verified by at least one of the authors. We also conducted conventional literature searches to verify and complement AI-assisted information retrieval. We take responsibility for the final content of this work, including text, claims, code, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Upon acceptance, we will release the source code for ReachECO and all experiments. Implementation details are provided in Appendix C. We have provided proofs of our Theorems in Appendix D. Furthermore, we have provided detailed descriptions of all case study environments we investigate in Appendix E.

REFERENCES

David Abel. A theory of state abstraction for reinforcement learning. In *Conference on Artificial Intelligence (AAAI)*, 2019.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Conference on Artificial Intelligence (AAAI)*, 2018.

Matthias Althoff, Olaf Stursberg, and Martin Buss. Reachability analysis of linear systems with uncertain parameters and inputs. In *IEEE Conference on Decision and Control (CDC)*, 2007.

AMD. *UltraScale Architecture Libraries Guide*. Advanced Micro Devices, Inc., 2026.

Jean-Pierre Aubin. A survey of viability theory. *SIAM Journal on Control and Optimization*, 1990.

Alan T. L. Bacellar, Zachary Susskind, Mauricio Breternitz Jr., Eugene John, Lizy K. John, Priscila M. V. Lima, and Felipe M. G. França. Differentiable weightless neural networks. In *International Conference on Machine Learning (ICML)*, 2024.

Osbert Bastani. Safe reinforcement learning with nonlinear dynamics via model predictive shielding. In *American control conference (ACC)*, 2021.

Fabian Bauer-Marquart, David Boetius, Stefan Leue, and Christian Schilling. Specrepair: Counter-example guided safety repair of deep neural networks. In *International Symposium on Model Checking Software (SPIN)*, 2022.

Luca Benvenuti, Davide Bresolin, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Emanuele Mazzi, Alberto Sangiovanni-Vincentelli, and Tiziano Villa. Reachability computation for hybrid systems with Ariadne. *IFAC Proceedings Volumes*, 2008.

Dimitri Bertsekas. Infinite time reachability of state-space regions by using feedback control. *IEEE Transactions on Automatic Control (TAC)*, 1972.

Franco Blanchini. Set invariance in control. *Automatica*, 1999.

Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis: Runtime enforcement for reactive systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 533–548. Springer, 2015.

David Boetius, Stefan Leue, and Tobias Sutter. A robust optimisation perspective on counterexample-guided repair of neural networks. In *International Conference on Machine Learning (ICML)*, 2023.

Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 2022.

Tiziana Calamoneri, Pierre Gaillard, Giacomo Paesani, and Giuseppe Perelli. Strategy repair in reachability games via a graph quotientation. In *Scandinavian Symposium on Algorithm Theory (SWAT)*, 2026.

Hugo CC Carneiro, Felipe MG França, and Priscila MV Lima. Multilingual part-of-speech tagging with weightless neural networks. *Neural Networks*, 2015.

Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural Lyapunov control. *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Puja Chaudhury, Alexander Estornell, and Michael Everett. Learning verifiable control policies using relaxed verification. In *IEEE Conference on Decision and Control (CDC)*, 2025.

William Hillick Clohessy and R. S. Wiltshire. Terminal guidance system for satellite rendezvous. *Journal of the Aerospace Sciences*, 1960.

Davide Corsi, Guy Amir, Andoni Rodríguez, César Sánchez, Guy Katz, and Roy Fox. Verification-guided shielding for deep reinforcement learning. In *Reinforcement Learning Conference (RLC)*, 2024.

Luca De Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. *Theoretical Computer Science (TCS)*, 2007.

Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2019.

Alec Edwards, Andrea Peruffo, and Alessandro Abate. Fossil 2.0: Formal certificate synthesis for the verification and control of dynamical models. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2024.

Georgios E Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science (TCS)*, 2009.

Feisi Fu and Wenchao Li. Sound and complete neural network repair with minimality and locality guarantees. In *International Conference on Learning Representations (ICLR)*, 2022.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, 2018.

Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *Conference on Artificial Intelligence (AAAI)*, 2018.

Pierre Gaillard, Fabio Patrizi, and Giuseppe Perelli. Strategy repair in reachability games. In *European Conference on Artificial Intelligence (ECAI)*, 2023.

Ben Goldberger, Guy Katz, Yossi Adi, and Joseph Keshet. Minimal modifications of deep neural networks using verification. In *Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, 2020.

Nathaniel Hamilton, Preston K. Robinette, and Taylor T. Johnson. Training agents to satisfy timed and untimed signal temporal logic specifications with reinforcement learning. In *International Conference on Software Engineering and Formal Methods (SEFM)*, 2022.

Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *ACM Symposium on Theory of Computing (STOC)*, 1995.

Haimin Hu, Mahyar Fazlyab, Manfred Morari, and George J. Pappas. Reach-SDP: Reachability analysis of closed-loop systems with neural network controllers via semidefinite programming. In *IEEE Conference on Decision and Control (CDC)*, 2020.

Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. POLAR: A polynomial arithmetic framework for verifying neural-network controlled systems. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2022a.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research (JMLR)*, 2022b.

Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2019.

Radoslav Ivanov, Taylor Carpenter, James Weimer, Rajeev Alur, George Pappas, and Insup Lee. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *International Conference on Computer Aided Verification (CAV)*, 2021.

Peng Jin, Jiaxu Tian, Dapeng Zhi, Xuejun Wen, and Min Zhang. Trainify: A CEGAR-driven training and verification framework for safe deep reinforcement learning. In *International Conference on Computer Aided Verification (CAV)*, 2022.

Bettina Könighofer, Julian Rudolf, Alexander Palmisano, Martin Tappler, and Roderick Bloem. Online shielding for reinforcement learning. *Innovations in Systems and Software Engineering*, 2022.

Hanna Krasowski, Jakob Thumm, Marlon Müller, Lukas Schäfer, Xiao Wang, and Matthias Althoff. Provably safe reinforcement learning: Conceptual analysis, survey, and benchmarking. *Transactions on Machine Learning Research (TMLR)*, 2023.

Fabian Kresse and Christoph H. Lampert. Learning quantized continuous controllers for integer hardware. In *Learning for Dynamics and Control Conference (L4DC)*, 2026a.

Fabian Kresse and Christoph H. Lampert. Differentiable weightless controllers: Learning logic circuits for continuous control. *International Conference on Machine Learning (ICML)*, 2026b.

Smita Krishnaswamy, Haoxing Ren, Nilesh Modi, and Ruchir Puri. DeltaSyn: An efficient logic difference optimizer for ECO synthesis. In *Proceedings of the 2009 International Conference on Computer-Aided Design (ICCAD)*, 2009.

Chih-chang Lin, Kuang-Chien Chen, Shih-Chieh Chang, Malgorzata Marek-Sadowska, and Kwang-Ting Cheng. Logic synthesis for engineering change. In *ACM/IEEE Design Automation Conference (DAC)*, 1995.

Diego Manzanas Lopez, Sung Woo Choi, Hoang-Dung Tran, and Taylor T. Johnson. NNV 2.0: The neural network verification tool. In *International Conference on Computer Aided Verification (CAV)*, 2023.

Diego Manzanas Lopez, Matthias Althoff, Luis Benet, Samuel Coogan, Marcelo Forets, Akash Harapanahalli, Taylor T. Johnson, Tobias Ladner, Christian Schilling, Huan Zhang, and Xiangru Zhong. ARCH-COMP25 category report: artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In *International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH)*, 2025.

Pengyuan Lu, Ivan Ruchkin, Matthew Cleaveland, Oleg Sokolsky, and Insup Lee. Causal repair of learning-enabled cyber-physical systems. In *IEEE International Conference on Assured Autonomy (ICAA)*, 2023.

Pengyuan Lu, Matthew Cleaveland, Oleg Sokolsky, Insup Lee, and Ivan Ruchkin. Repairing neural network-based control policies with safety preservation. *Research Directions: Cyber-Physical Systems*, 2025.

Deyun Lyu, Jiayang Song, Zhenya Zhang, Zhijie Wang, Tianyi Zhang, Lei Ma, and Jianjun Zhao. Autorepair: Automated repair for AI-enabled cyber-physical systems under safety-critical conditions. *arXiv preprint arXiv:2304.05617*, 2023.

Benedikt Maderbacher, Stefan Schupp, Ezio Bartocci, Roderick Bloem, Dejan Ničković, and Bettina Könighofer. An adaptive, provable correct simplex architecture. *International Journal on Software Tools for Technology Transfer*, 2025.

Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *International symposium on formal techniques in real-time and fault-tolerant systems (FTRTFT)*, 2004.

Udayan Mandal, Guy Amir, Haoze Wu, Ieva Daukantas, Fletcher Lee Newell, Umberto J. Ravaioli, Baoluo Meng, Michael Durling, Milan Ganai, Tobey Shim, Guy Katz, and Clark Barrett. Formally verifying deep reinforcement learning controllers with lyapunov barrier certificates. In *Formal Methods in Computer-Aided Design (FMCAD)*, 2024.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

Laurent Perron and Frédéric Didier. CP-SAT v9.12, 2025. URL `https://developers.google.com/optimization/cp/cp_solver/`.

André Platzer. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning*, 2008.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research (JMLR)*, 2021.

Umberto J Ravaioli, James Cunningham, John McCarroll, Vardaan Gangal, Kyle Dunlap, and Kerianne L. Hobbs. Safe reinforcement learning benchmark environments for aerospace control systems. In *IEEE Aerospace Conference (AERO)*, 2022.

Timothy Rupprecht and Yanzhi Wang. A survey for deep reinforcement learning in markovian cyber–physical systems: Common problems and solutions. *Neural Networks*, 2022.

Ulices Santa Cruz, James Ferlez, and Yasser Shoukry. Safe-by-repair: A convex optimization approach for repairing unsafe two-level lattice neural network controllers. In *IEEE Conference on Decision and Control (CDC)*, 2022.

Christian Schilling, Marcelo Forets, and Sebastián Guadalupe. Verification of neural-network control systems by integrating Taylor models and zonotopes. In *Conference on Artificial Intelligence (AAAI)*, 2022.

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

Keyi Shen and Glen Chou. Parallel differentiable reachability for learning and planning with certified neural dynamics and controllers. In *Robotics: Science and Systems (RSS)*, 2026.

Chelsea Sidrane, Amir Maleki, Ahmed Irfan, and Mykel J. Kochenderfer. Overt: An algorithm for safety verification of neural network control policies for nonlinear systems. *Journal of Machine Learning Research (JMLR)*, 2022.

Matthew Sotoudeh and Aditya V. Thakur. Provable repair of deep neural networks. In *ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI)*, 2021.

Xinyu Sun, Wanwei Liu, Shangwen Wang, Tingyu Chen, Ye Tao, and Xiaoguang Mao. AutoRIC: Automated neural network repairing based on constrained optimization. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2025.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press Cambridge, 1998.

Paulo Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes. *Annual Review of Control, Robotics, and Autonomous Systems*, 2025.

Zhe Tao, Stephanie Nawas, Jacqueline Mitchell, and Aditya V. Thakur. Architecture-preserving provable repair of deep neural networks. *Proceedings of the ACM on Programming Languages (PACMPL)*, 2023.

Samuel Teuber, Stefan Mitsch, and André Platzer. Provably safe neural network controllers via differential dynamic logic. *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

Jiaxu Tian, Dapeng Zhi, Si Liu, Peixin Wang, Cheng Chen, and Min Zhang. Boosting verification of deep reinforcement learning via piece-wise linear decision neural networks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Jiaxu Tian, Dapeng Zhi, Si Liu, Peixin Wang, Guy Katz, and Min Zhang. Taming reachability analysis of dnn-controlled systems via abstraction-based training. In *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2024.

Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification (CAV)*, 2020.

Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 2019.

Panagiotis Vlantis, Leila Bridgeman, and Michael Zavlanos. Failing with grace: Learning neural network controllers that are boundedly unsafe. In *Learning for Dynamics and Control Conference (L4DC)*, 2023.

Yixuan Wang, Chao Huang, Zhaoran Wang, Zhilu Wang, and Qi Zhu. Verification in the loop: Correct-by-construction control learning with reach-avoid guarantees. *arXiv preprint arXiv:2106.03245*, 2021.

Weiming Xiang and Taylor T. Johnson. Reachability analysis and safety verification for neural network control systems. *arXiv preprint arXiv:1805.09944*, 2018.

Junfeng Yang, Min Zhang, Xin Chen, and Qin Li. Formal verification of probabilistic deep reinforcement learning policies with abstract training. In *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2025.

Xiaodong Yang, Tom Yamaguchi, Hoang-Dung Tran, Bardh Hoxha, Taylor T. Johnson, and Danil Prokhorov. Neural network repair with reachability analysis. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2022.

Emily Yu, Djordje Žikelić, and Thomas A. Henzinger. Neural control and certificate repair via runtime monitoring. In *Conference on Artificial Intelligence (AAAI)*, 2025.

Hengjun Zhao, Xia Zeng, Taolue Chen, and Zhiming Liu. Synthesizing barrier certificates using neural networks. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2020.

Weiye Zhao, Tairan He, Rui Chen, Tianhao Wei, and Changliu Liu. State-wise safe reinforcement learning: A survey. *International Joint Conference on Artificial Intelligence (IJCAI)*, 2023.

Weichao Zhou, Ruihan Gao, BaekGyu Kim, Eunsuk Kang, and Wenchao Li. Runtime-safety-guided policy repair. In *International Conference on Runtime Verification (RV)*, 2020.

Table 3: Docking2D: $\mathcal{X}$ reports empirical safety for 1000 rollouts from the 130 meter starting circle. $\mathcal{X}_0$ reports empirical safety from the initial region (before repair). t reports the overall time for both repair and verification. tanh denotes the controller utilized in (Lopez et al., 2025). Param reports the increase in size in terms of LUT6s of the repaired controller. Note that naive repair is a lower bound.

	Emp. safe [%]		Reward				Param [LUT6]	
π	$\mathcal{X}$	$\mathcal{X}_0$	original	after repair	Verified	t [h]	ours	naive
tanh	95.2	100.0	3.096 ± 0.069	·	×	·	·	·
1	100.0	100.0	3.066 ± 0.067	3.066 ± 0.067	✓	0.69	+38	+92k
2	94.8	100.0	2.994 ± 0.061	2.994 ± 0.061	✓	1.07	+0	+87k
3	98.4	99.8	3.003 ± 0.148	3.059 ± 0.066	✓	8.09	+7k	+120k

A Additional Experiments

A.1 Docking2D

Setup. Docking2D is a challenging benchmark for neural-network safety controller verification. For example, none of the entries to the ARCH-COMP AINNCS competition in 2025 was able to certify the full provided initial region (Lopez et al., 2025). We consider a larger initial region, covering the competition's region, and, moreover, verify the reach-avoid property proposed in (Mandal et al., 2024), rather than only the original 40-step ARCH-COMP specification. The resulting set propagation requires more than *850 time steps* before all sets have converged to the goal region around the origin.

We train three DWC policies with PPO; training details are given in Appendix E. For continuation search, $\mathcal{W}$ uses a backup controller that targets the docking point $(0, 0)$ at a desired speed of $0.18\mathrm{m/s}$ and proposes the corresponding discrete action. The trained DWCs achieve reward close to the competition tanh policy, with some seeds remaining empirically unsafe before repair. We split the initial set into four smaller initial boxes. To reduce repeated recomputation, we use a greedy restart strategy: after a repair, verification resumes from the depth at which the corresponding continuation game was constructed rather than from the earliest potentially affected time step. Since certification is reported only after a subsequent repair-free pass over all initial boxes, this optimization does not weaken the final guarantee. Our main insights are:

Depth-1 games are insufficient. When performing repair, we observe a total of 35 depth-2 games. These occur when the intersample tube does not allow a safe continuation, requiring backtracking.

Repairs are *highly* localized. Table 3 reports reward, repair, and verification time as well as additional LUT6s for patch circuits. Seed 1 is the only controller that is empirically safe over the entire evaluation region, yet still requires modifications to 5 policy cells before it can be certified. Seed 2 exhibits empirical failures outside the certified initial region but none from $\mathcal{X}_0$; ReachECO can certify it without having to modify the controller at all. Seed 3 is empirically unsafe and requires the largest circuit patch, adding roughly 7k LUT6s. Nevertheless, only 3,326 policy cells are modified, corresponding to less than $10^{-3}\%$ of the overall cells. Thus, even when the compiled circuit patch is comparatively large, the behavioral modification remains extremely local. Figure 3 visualizes these changes and shows that they are concentrated primarily around the docking region.

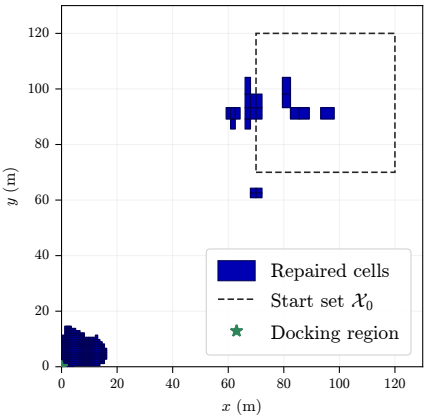

Figure 3: Docking2D: repaired regions of the DWC seed 3 projected onto the (x, y) plane.

Reachability information *substantially* reduces patch size. We compare against a naive patching scheme that does not use reachability-derived information, but instead patches every fully safe policy cell whose transition can enter the unsafe set from anywhere in the safe region. As shown in Table 3, the resulting patches are substantially larger than those produced by ReachECO, requiring up to several orders of magnitude more LUT6s. Moreover, these numbers are conservative: the naive construction considers only cells fully contained in the safe set and therefore underapproximates the patch that would be required over the entire boundary.

A.2 CAR2D

Setup. We investigate a modified version of the nonlinear NAV benchmark from the ARCH-COMP AINNCS competition (Lopez et al., 2025). We train a DWC controller with TD3 (Fujimoto et al., 2018), with specifics given in Appendix E. The controller must reach the goal region $x > 6$ while avoiding the two obstacles. Figure 4 shows the initial set and the obstacles. During repair, we modify only the steering output and consider a repair alphabet consisting of three increasingly large steering corrections in each direction, together with the two extreme steering actions.

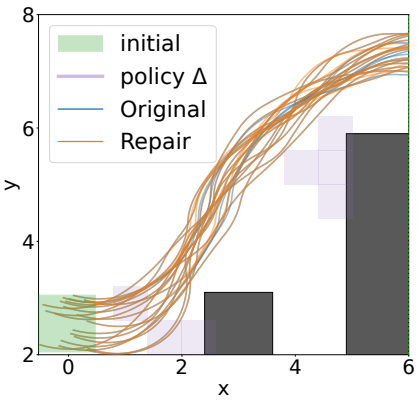

Figure 4: Car2D rollouts before and after repair, with modified policy cells highlighted.

Repair restores certifiability. The original controller is empirically safe over 1000 rollouts sampled from the initial set, exhibiting no observed unsafe trajectories. Nevertheless, formal verification with our BBReach implementation finds a reachable-set intersection with the unsafe set at time step $t = 3$. We subsequently apply ReachECO. During repair, ReachECO constructs 1368 depth-1, 90 depth-2, and 7 depth-3 continuation games. After 292 s, ReachECO returns a repaired controller modifying 42 policy cells that passes verification. Figure 4 compares rollouts of the original and repaired controllers and visualizes the modified policy cells. Notably, the upper repaired regions are virtually absent from the sampled rollouts. Their modification is nevertheless required for the reachability analysis to pass.

B APPENDIX: RELATED WORK

Shielding. Shielding enforces safety at runtime by monitoring the nominal controller and overriding actions when necessary (Bloem et al., 2015; Alshiekh et al., 2018). Shields may be synthesized offline or perform additional safety reasoning online (Bastani, 2021; Könighofer et al., 2022). Corsi et al. (2024) use offline verification to identify regions in which an online shield needs to be active. More recently, Maderbacher et al. (2025) propose an architecture in which a performance-optimized controller is permitted to act only inside a certified recoverable region, while a baseline controller intervenes otherwise.

Our repaired controller is closely related to a sparse, statically compiled shield: it overrides the nominal policy only on a finite set of state regions. The main distinction is that ReachECO uses failed closed-loop reachability verification to discover the required interventions offline, incorporates them directly into the nominal controller, and reverifies the resulting controller. Consequently, deployment requires neither runtime safety reasoning nor invocation of a separate backup controller.

Closed-Loop Verification of Neural Network Controllers Safety verification of neural-network controlled systems is often performed with reachability analysis: an over-approximation of the set of reachable closed-loop states is propagated forward in time and checked for intersection with an unsafe set. A large ecosystem of methods and tools implements this basic recipe with different set representations and abstractions of the network (Xiang & Johnson, 2018; Ivanov et al., 2019; 2021; Dutta et al., 2019; Tran et al., 2020; Lopez et al., 2023; Huang et al., 2022a; Hu et al., 2020; Schilling et al., 2022; Sidrane et al., 2022; Tian et al., 2023; 2024). The ARCH-COMP AINNCS category

provides a yearly comparison between some of these tools (Lopez et al., 2025). A second line of research focuses on certificate-based verification, reducing the need to argue about entire trajectories to single-step queries on the certificate (Chang et al., 2019; Zhao et al., 2020; Mandal et al., 2024; Edwards et al., 2024). However, this adds the need to both synthesize an appropriate certificate and subsequently check it.

Another reachability-related line of work trains controllers that are constant on a finite partition of the state space, so that closed-loop verification never has to perform over-approximation of the neural network. Jin et al. (2022) introduce Trainify, where a neural network controller is trained in a CEGAR loop that iteratively refines an abstraction of the state space until the control objectives can be fulfilled; since actions are constant per abstract state, neural network verification is skipped entirely and model checking is performed. Tian et al. (2024) instead fix an initial abstraction, train on it, and perform reachability analysis (BBReach), again exploiting that constant actions make over-approximating the network unnecessary. Tian et al. (2023) extend the controller class to piecewise-affine decision networks, enabling coarser abstractions, and Yang et al. (2025) extend abstraction-based controller verification to probabilistic safety.

Constraint based Repair. Counterexample-guided finetuning is not guaranteed to terminate (Boetius et al., 2023). Hence, there has been work to provide stronger repair guarantees by modifying the internal parameters of a network based on input-output constraints (Goldberger et al., 2020; Sotoudeh & Thakur, 2021; Bauer-Marquart et al., 2022; Tao et al., 2023; Sun et al., 2025) or adding patches to it (Fu & Li, 2022). These approaches reason about the network specification itself rather than the closed-loop system dynamics.

Local Controller Repair. Encompasses methods that rely on *single* step reasoning, either performing repair based on counterexamples (Santa Cruz et al., 2022) or one-step reachability sets (Yang et al., 2022). Teuber et al. (2024) utilize a control envelope to transform the global controller safety problem into a local one. However, this requires the envelope to be available. Santa Cruz et al. (2022), perform constraint optimization at a known counterexample while safeguarding certain global properties. Mandal et al. (2024) reduce the multistep repair problem to a single-step one by fine-tuning a Lyapunov-barrier certificate together with the controller; this shifts the computational complexity to checking the certificate.

Trajectory-Level Controller Repair. Many global controller repair methods rely on parameter fine-tuning, often based on STL robustness (Maler & Nickovic, 2004; Fainekos & Pappas, 2009). Examples are STLGym (Hamilton et al., 2022) and ISAR (Lu et al., 2025). Other methods rely on discovering failing trajectories and supervised fine-tuning (Lyu et al., 2023) or direct changes to the policy discovered through exhaustive reasoning (Lu et al., 2023). Lastly, Zhou et al. (2020) and Yu et al. (2025) utilize a runtime monitor and perform fine-tuning of the controller based on runtime feedback.

Set-based training A set of methods attempts to improve the likelihood that a set-based closed-loop proof will succeed (Wang et al., 2021; Vlantis et al., 2023; Chaudhury et al., 2025; Shen & Chou, 2026). In these methods set-based information is included in the training loss, aiming to reduce over-approximation during verification.

Circuit Rectification Repairing a deployed circuit with a small patch instead of re-synthesizing it is long-standing practice in logic design, where engineering change orders (ECOs) rectify an implementation towards a revised specification with minimal modification (Lin et al., 1995; Krishnaswamy et al., 2009). Our patch compilation adopts this philosophy for learned controllers, with one key difference: the revised specification is not written by a designer but generated automatically, in the form of action-correction constraints discovered by a closed-loop reachability verifier.

Strategy Repair Synthesis based on discrete-reachability games has a long history (De Alfaro et al., 2007). However, even for discrete state-spaces, prior work on repair has addressed the problem of *strategy repair* only relatively recently (Gaillard et al., 2023; Calamoneri et al., 2026). ReachECO addresses an analogous problem for learned controllers that operate on continuous dynamics, however, in our setting the underlying game is not available a priori: relevant transitions are constructed lazily from failed closed-loop reachability verification.

C Formal Definitions and Algorithm Details

C.1 Reachability

As in the main body of text: we study a task with a continuous state space $\mathcal{X} \subseteq \mathbb{R}^d$ and system dynamics governed by a differential equation, $\dot{x} = f(x, a)$, where $a \in \mathcal{A}$ is a control action.

Now, let $\Phi_t^a(s)$ denote the state reached after time t from s under constant input a. For a set of states, $S \subseteq \mathcal{X}$, and an action $a \in \mathcal{A}$, define

$$\mathrm{Post}(S, a) := \{\Phi_\Delta^a(x) \mid x \in S\}, \qquad \mathrm{Tube}(S, a) := \{\Phi_s^a(x) \mid x \in S,\ s \in [0, \Delta]\}. \tag{7}$$

Because of the continuous nature of the underlying dynamics, exact reachability cannot be computed in general. We therefore rely on *sound over-approximations*.

$$\overline{\mathrm{Post}}(S, a) \supseteq \mathrm{Post}(S, a), \qquad \overline{\mathrm{Tube}}(S, a) \supseteq \mathrm{Tube}(S, a).$$

C.2 Problem Setting

We now formally define *piecewise-constant abstraction-based controller*, as they are the controller type studied throughout.

Definition 2 (Piecewise-constant abstraction-based controller)**.** *A controller, $\pi : \mathcal{X} \to \mathcal{A}$, is called* piecewise-constant abstraction-based*, if $\mathcal{A} \subseteq \mathbb{R}^m$ is finite and exposes a finite axis-aligned policy cell partition $\mathcal{Q}$ of $\mathcal{X}$ such that for any $q \in \mathcal{Q}:\ x, x' \in q \Rightarrow \pi(x) = \pi(x')$. Hence, π equivalently induces a finite map $\pi : \mathcal{Q} \to \mathcal{A}$, and we write $\mathcal{A}^{\mathcal{Q}} := \{\pi \mid \pi : \mathcal{Q} \to \mathcal{A}\}$ for the set of all controllers over the fixed policy partition $\mathcal{Q}$ and action alphabet $\mathcal{A}$.*

Let us now formally define the repair problem.

Definition 3 (Repair instance)**.** *A* repair instance *consists of an abstraction-based controller π_0, a finite set of continuation actions $\mathcal{A}_{\mathrm{cont}} \subseteq \mathcal{A}$, sound propagation operators $\overline{\mathrm{Post}}$ and $\overline{\mathrm{Tube}}$, an initial set $\mathcal{X}_0 \subseteq \mathcal{X}$, given as a collection of boxes, an unsafe set $\mathcal{U} \subseteq \mathcal{X}$, an optional absorbing goal set $\mathcal{X}_\mathcal{G} \subseteq \mathcal{X}$, and a verification budget T.*

Definition 4 (Repair class)**.** *Given a repair instance, set $\mathcal{A}(q) = \{\pi_0(q)\} \cup \mathcal{A}_{\mathrm{cont}}$ for every $q \in \mathcal{Q}$. Then*

$$\Pi_R := \prod_{q \in \mathcal{Q}} \mathcal{A}(q)$$

is the repair class *of π_0. It contains all controllers that either retain the original action or select a continuation action independently on each policy cell.*

Definition 5 (Certified repair)**.** *A* certified repair *of a repair instance is a controller $\pi' \in \Pi_R$ that is certified safe according to Definition 1.*

C.3 Controllability and Losing Regions

Given a failing reachable region, repair must determine how many time steps before the reported violation an intervention is needed. Whether an intervention at a given state can still prevent the violation depends on whether safety can be maintained from the resulting successor states. For the infinite time case this is captured by the notion of *controlled invariance* (Bertsekas, 1972; Blanchini, 1999); its maximal set corresponds to the *viability kernel* in viability theory (Aubin, 1990) and to the *winning region* of a safety game (Tabuada, 2009).

We define controllability relative to the finite continuation alphabet $\mathcal{A}_{\mathrm{cont}}$ used by repair.

Definition 6 (Controlled invariant set)**.** *A set $C \subseteq \mathcal{S}$, with $\mathcal{S} := \mathcal{X} \setminus \mathcal{U}$, is* controlled invariant *if*

$$\forall x \in C\ \exists a \in \mathcal{A}_{\mathrm{cont}}: \qquad \mathrm{Tube}(\{x\}, a) \cap \mathcal{U} = \emptyset\ \wedge\ \mathrm{Post}(\{x\}, a) \subseteq C.$$

The maximal controlled invariant subset of $\mathcal{S}$ is denoted $\mathcal{C}^*$.

Dually, let $\mathcal{L}^* := \mathcal{X} \setminus \mathcal{C}^*$ denote the *losing set*, i.e., the states from which safety cannot be maintained indefinitely using only actions from $\mathcal{A}_{\mathrm{cont}}$. If a controlled invariant set is known, it yields a one-step

condition for repair: the repaired action must keep the current transition safe and map all successors into the invariant.

When $\mathcal{C}^*$ is unavailable, the one-step repair condition above cannot be applied directly. ReachECO instead starts from a sound under-approximation $\underline{\mathcal{L}} \subseteq \mathcal{L}^*$, which may initially contain only the unsafe set $\mathcal{U}$.

Hence, we define the SAFESTEP predicate in eq. 4 as the verifiers analogue to the ideal one-step condition above. Since $\underline{\mathcal{L}}$ may be a strict under-approximation of $\mathcal{L}^*$, however, satisfying $\mathrm{SafeStep}$ may still lead to a safety violation in later time steps.

For later use, let us now additionally define the predicate Safe_h. For a box B contained in policy cell $q(B)$ and a fixed controller π, $\mathrm{Safe}_0(B, \pi) := \mathrm{true}$ and, for $h \geq 1$, define

$$\begin{aligned} \mathrm{Safe}_h(B, \pi) \ :\Longleftrightarrow\ & \mathrm{SafeStep}(B, \pi(q(B))) \\ & \wedge\ \forall B' \in \mathrm{Succ}(B, \pi(q(B))) : \mathrm{Safe}_{h-1}(B', \pi). \end{aligned} \tag{8}$$

Thus Safe_h checks the continuation under π itself.

C.4 ALGORITHM DETAILS

Forward Verification. ReachECO uses our implementation of BBReach for forward reachability analysis (Tian et al., 2024), to propagate the current repaired controller π' in REACHSTEP. At time step k, the initial failing frontier, the set of *obligations*, is

$$\mathcal{F} := \left\{ (B, 1) \;\middle|\; B \in \overline{\mathcal{X}}_k \ \wedge\ \neg \mathrm{SafeStep}(B, \pi'(q(B))) \right\}. \tag{9}$$

A nonempty frontier triggers BREPAIR. After a repair or an expansion of $\underline{\mathcal{L}}$, verification resumes from the earliest affected time step. If a reachable-set fixed point is reached before the horizon budget T, we obtained an unbounded safety certificate. Furthermore, intersections with the absorbing goal are removed from further reachability steps.

Notes on Merging. Merging boxes that belong to the same policy cell is essential for the scalability of BBReach-based reachability analysis (Tian et al., 2024). Without merging, every propagated box has a unique parent in the preceding timestep. Merging destroys this property: a merged box may contain successors originating from several distinct parent boxes.

To retain the efficiency of merging while recovering an exact lineage when a safety violation is encountered, ReachECO records the candidate parents contributing to each merged box. Upon detecting a failing merged box, it replays only these candidate parents from the preceding timestep, with merging disabled. Thus, this refinement is performed lazily and only along the portion of the reachability computation relevant to the safety violation. If no safety violation remains after replay, the original violation was induced by merging. The corresponding obligation is removed from the failing frontier, and a *merge mask* is updated with the timestep and policy-cell identifier at which merging must be disabled in subsequent forward propagations.

Otherwise, replay yields one or more genuinely failing unmerged boxes, each with a unique parent. Consequently, BACKTRACK is only applied to unmerged boxes: for an obligation (B, h), if B_{pred} is the unique parent of B in the preceding timestep, BACKTRACK replaces (B, h) by $(B_{\mathrm{pred}}, h + 1)$. Repeating this operation therefore recovers a unique predecessor lineage despite the use of merging during the original forward propagation.

Repair Constraints. Recall a *repair constraint* $\gamma = (\mathcal{G}, i)$, with root timestep i and game DAG $\mathcal{G}$. Let $\mathcal{A}_v$ denote the set of actions stored at node v, which allow a safe continuation. For a controller π, the constraint is satisfied recursively as follows: at a node v, the action $a = \pi(q(B_v)) \in \mathcal{A}_v$, and, if $h_v > 1$, the same condition must hold for every child node corresponding to a box $B' \in \mathrm{Succ}(B_v, a)$. We write

$$\pi \models \gamma$$

if this condition holds from the root of $\mathcal{G}$.

For a set of constraints Γ, we write

$$\pi \models \Gamma \quad \Longleftrightarrow \quad \forall \gamma \in \Gamma :\ \pi \models \gamma. \tag{10}$$

Losing-Set Expansion A box B may be certified losing under the continuation alphabet if:

$$\forall a \in \mathcal{A}_{\text{cont}},\ \forall B' \in \mathrm{Succ}(B, a) : \quad B' \subseteq \underline{\mathcal{L}}. \tag{11}$$

Then $B \subseteq \mathcal{L}^*$ and it may be added to $\underline{\mathcal{L}}$. If we find a losing box, our implementation greedily attempts to certify the whole policy cell, that contains the certified losing box, as losing.

Joint Realization. JOINT searches for a controller $\pi' \in \Pi_R$ such that $\pi' \models \Gamma \cup \overline{\Gamma}$. Γ denotes the *committed* constraints from prior repair rounds, while $\overline{\Gamma}$ represents the *current* constraints that were generated under the current π'. As all constraints are derived independently they need not be jointly realizable, since each policy cell must be assigned a single action. We encode their joint realization as a CP-SAT problem using OR-Tools (Perron & Didier, 2025). Binary variables $w_{q,a}$ select one action per cell, and y_v indicates activation of a game node v:

$$\sum_{a \in \mathcal{A}(q)} w_{q,a} = 1 \quad \forall q \in \mathcal{Q}, \qquad\qquad y_{\mathrm{root}(\gamma)} = 1 \quad \forall \gamma \in \Gamma \cup \overline{\Gamma}, \tag{12}$$

where $\mathrm{root}(\gamma)$ denotes the root node v_0 of the game DAG $\mathcal{G}$ of $\gamma = (\mathcal{G}, i)$.

Every active node must select a stored safe action, and selecting an action activates all corresponding child nodes: for every node v of a game in $\Gamma \cup \overline{\Gamma}$,

$$\begin{aligned} y_v \implies & \bigvee_{a \in \mathcal{A}_v} w_{q(B_v),a}, \\ y_v \wedge w_{q(B_v),a} \implies & y_{v'} \quad \text{for every } a \in \mathcal{A}_v \text{ and every child } v' \text{ of } v \text{ induced by } a. \end{aligned} \tag{13}$$

Joint-Realization Heuristics. Optimal joint realization can be expensive, so JOINT employs several heuristics. For a set of repair constraints Σ, define its *support* as the set of policy cells occurring in any of its game DAGs,

$$\mathrm{supp}(\Sigma) := \{q(B_v) \mid \gamma = (\mathcal{G}, i) \in \Sigma,\ v \text{ is a node of } \mathcal{G}\}\,. \tag{14}$$

We call $\mathrm{supp}(\overline{\Gamma})$ the *current constraint cells*.

JOINT first restricts the realization search to the current constraint cells, such that policy cells outside $\mathrm{supp}(\overline{\Gamma})$ are not modified. If no feasible joint realization is found, the search is repeated over

$$\mathrm{supp}(\Gamma \cup \overline{\Gamma}),$$

allowing cells occurring in both committed and current constraints to be modified.

Among feasible assignments, JOINT searches for one minimizing the distance to the original controller. We limit this optimization to $60\,\mathrm{s}$ and, if optimality has not been established by then, accept the best feasible assignment found.

Additional implementation details. The original BBReach implementation uses Ariadne (Benvenuti et al., 2008) for post-image propagation. We instead implement propagation directly on the GPU using interval arithmetic for $\overline{\mathrm{Post}}$ and sound continuous-time bounds for $\overline{\mathrm{Tube}}$. For efficiency, boxes in the same policy cell are merged by their bounding box. This is a more aggressive merging scheme than employed in (Tian et al., 2024). Benchmark-specific propagators are given in Appendix E. We perform each tube check for 100 segments, escalating to a 1000 for any tubes where violations were found. Only after this refinement do we accept a tube as failing. Furthermore, after a certified repair is found, the repair is localized to only cells visited by the repaired controller. This information can be simply extracted from the stored forward propagation sets. Last, accounting for floating-point errors is outside the scope of this work.

D APPENDIX: THEOREMS

Verifier assumptions. We build on the sound reachability procedure of Tian et al. (2024). Our proofs rely on the following assumptions:

1. REACHSTEP is covering: for every accepted step k and every $B \in \overline{\mathcal{X}}_k$, $\mathrm{Post}(B, \pi'(q(B))) \subseteq \bigcup \overline{\mathcal{X}}_{k+1}$, where $\overline{\mathrm{Post}}$ and $\overline{\mathrm{Tube}}$ are sound.
2. REACHSTEP reports every source box whose abstract tube intersects $\mathcal{U}$ or whose abstract post-image intersects $\underline{\mathcal{L}}$. Hence,
$$\mathcal{F} = \emptyset \implies \forall B \in \overline{\mathcal{X}}_k : \mathrm{SafeStep}(B, \pi'(q(B))).$$
3. EARLIESTAFFECTED returns a rewind time step no later than any previously verified step affected by a controller modification, an expansion of $\underline{\mathcal{L}}$, or a refinement of the reachability computation with respect to merging; the subsequent sets are recomputed.
4. Every obligation (B, h) in the backward cone (the initial obligations of Eq. 9 and every obligation produced from them by BACKTRACK) satisfies $\neg\mathrm{Safe}_h(B, \pi)$, where π is the controller that generated the failing frontier.
5. FINDCONT evaluates candidate actions with the same $\mathrm{SafeStep}$ predicate and Succ relation as the forward verifier: every action a stored at a game node v satisfies $\mathrm{SafeStep}(B_v, a)$, and, if $h_v > 1$, *all* boxes in $\mathrm{Succ}(B_v, a)$ appear as children of v with horizon $h_v - 1$.
6. All calls to REACHSTEP, JOINT, FINDCONT, EARLIESTAFFECTED, BACKTRACK terminate. In particular JOINT either returns UNKNOWN, SAT with a valid repair, or BACKTRACK, with $\mathcal{F}_o \neq \emptyset$.
7. Assume further that each abstract successor and predecessor computation returns finitely many boxes and that the continuation backend $\mathcal{W}$ terminates.
8. For reach–avoid tasks we treat the goal set $\mathcal{X}_\mathcal{G}$ as terminal: a trajectory that enters $\mathcal{X}_\mathcal{G}$ (satisfying the safety condition at entry) has completed the task and is not modeled further. Certified safety then means: every closed-loop trajectory from $\mathcal{X}_0$ avoids $\mathcal{U}$ until it enters $\mathcal{X}_\mathcal{G}$. Non-entering trajectories avoid $\mathcal{U}$ till T.

For simplicity, we state the guarantees with respect to the unmerged reachability computation. Merging is used only as a scalability optimization in the implementation. Whenever a merged box contributes to a failing or unresolved frontier, BACKTRACK replays the corresponding reachability step without merging and future forward steps retain this refinement. Thus all repair constraints are derived from the unmerged predecessor relation.

Theorem 1 (Soundness)**.** *Suppose ReachECO returns (π', H) with $H \in \{T, \infty\}$ rather than* UNKNOWN*. Then π' is certified safe to horizon H (Definition 1). In particular, no trajectory of the closed-loop system under π' starting in $\mathcal{X}_0$ enters $\mathcal{U}$ during the first H control steps, and for all time if $H = \infty$.*

Proof. By assumption 1, every accepted reachability step soundly covers the sampled nonterminal successors, and its inter-sample tubes are disjoint from $\mathcal{U}$ by assumption 2. Whenever π', $\underline{\mathcal{L}}$, or the reachability computation changes, EARLIESTAFFECTED causes every potentially invalidated step to be recomputed (assumption 3). Thus, at termination, all steps in the accepted prefix are valid for the returned controller π'.

If the algorithm returns (π', T), every step $j = 0, \ldots, T-1$ has been accepted. Hence,
$$\overline{\mathrm{Reach}}_{\pi', T}(\mathcal{X}_0) \cap \mathcal{U} = \emptyset,$$
so π' is certified safe to horizon T by Definition 1.

If the algorithm instead returns (π', ∞) at step k, every step $j = 0, \ldots, k$ has been accepted, and the fixed-point test establishes
$$\bigcup \overline{\mathcal{X}}_{k+1} \subseteq \bigcup \overline{\mathcal{X}}_k.$$
Together with
$$\overline{\mathrm{Reach}}_{\pi', k+1}(\mathcal{X}_0) \cap \mathcal{U} = \emptyset,$$
this establishes certified safety to horizon ∞ by Definition 1.

By assumption 1, the computed sets and tubes cover the concrete trajectories throughout the accepted prefix. In the fixed-point case, assumption 1 and the inclusion above allow the accepted transition

from $\bigcup \overline{\mathcal{X}}_k$ to be applied inductively. Its tube checks therefore establish safety for all subsequent steps (assumption 2).

For reach-avoid tasks, this argument applies to trajectories that have not terminated: every successor outside $\mathcal{X}_G$ remains covered by the propagated sets. Trajectories removed upon safe goal entry have completed the task under assumption 8. Thus concrete trajectories avoid $\mathcal{U}$ throughout the certified horizon, subject to this terminal-goal convention. □

Lemma 1 (Failure constraints eliminate the current controller)**.** *Let $\pi \in \Pi_R$ be the controller whose forward propagation generated a failure, and let $\gamma = (\mathcal{G}, i)$ be a repair constraint generated from a backward obligation (B, h), where $\mathcal{G}$ has horizon h. Then*

$$\pi \not\models \gamma.$$

Consequently, for every Γ such that $\pi \models \Gamma$,

$$\Pi(\Gamma \cup \{\gamma\}) \subsetneq \Pi(\Gamma),$$

where

$$\Pi(\Sigma) := \{\pi' \in \Pi_R \mid \pi' \models \Sigma\}.$$

Proof. Since (B, h) lies in the backward cone of a failure of π (assumption 4),

$$\neg \mathrm{Safe}_h(B, \pi).$$

By soundness of the continuation-game constraint (assumption 5),

$$\pi' \models \gamma \implies \mathrm{Safe}_h(B, \pi').$$

Hence

$$\pi \not\models \gamma.$$

If additionally $\pi \models \Gamma$, then

$$\pi \in \Pi(\Gamma) \setminus \Pi(\Gamma \cup \{\gamma\}),$$

and therefore

$$\Pi(\Gamma \cup \{\gamma\}) \subsetneq \Pi(\Gamma).$$

□

Lemma 2 (Termination of backward repair)**.** *Assume a finite verification horizon T, finitely branching predecessor expansion, a terminating continuation backend $\mathcal{W}$, and finite committed constraint sets. Then every call to* BREPAIR *terminates.*

Proof. Consider an arbitrary call to BREPAIR caused by a failure at time step $0 \le t < T$. Suppose, for contradiction, that the call has an infinite execution trace.

Let $\mathcal{T}_t$ be the backward unfolding of the failing frontier. Its nodes are occurrences of box-horizon pairs (B, h) and

$$(B, h) \to (B', h + 1), \qquad B' \in \textsc{Backtrack}(B).$$

The initial frontier is finite, every node has finite out-degree (assumption 7), and

$$1 \le h \le t + 1 \le T.$$

Hence

$$|\mathcal{T}_t| < \infty.$$

Consider an arbitrary iteration of the assumed infinite trace. Since the trace does not terminate, neither BREPAIR nor JOINT returns REPAIR or UNKNOWN (assumption 6).

Thus the iteration can continue only by

1. returning one or more current constraints for further backtracking, or
2. removing one or more previously committed non-anchored constraints in favor of an anchored current constraint.

In Case 1, each returned pair (B, h) is replaced by predecessors $(B', h+1)$. Hence the corresponding occurrence advances strictly in the finite tree $\mathcal{T}_t$. A current constraint that is returned for backtracking is removed from $\overline{\Gamma}$ and therefore cannot be backtracked again at the same occurrence. Thus Case 1 can occur only finitely many times.

In Case 2, at least one constraint from the committed set present at entry to the call is removed. Since this set is finite, Case 2 can also occur only finitely many times.

Hence the assumed infinite trace cannot exist. Therefore every call to BREPAIR terminates. □

Theorem 2 (Termination)**.** *Assume that the initial set has a finite abstract representation, and that the policy partition $\mathcal{Q}$, the continuation alphabet $\mathcal{A}_{\mathrm{cont}}$, and the verification horizon T are finite. Furthermore, assume* JOINT *always returns a repair if there exists one under the constraints. Consider $\underline{\mathcal{L}} = \mathcal{U}$ fixed and no merging. Then ReachECO terminates and returns either (π', H), with $H \in \{T, \infty\}$, or* UNKNOWN.

Proof. By Lemma 2, every call to BREPAIR terminates. It remains to show that only finitely many such calls can occur.

Since $\mathcal{Q}$ and $\mathcal{A}_{\mathrm{cont}}$ are finite,

$$|\Pi_R| \le (|\mathcal{A}_{\mathrm{cont}}| + 1)^{|\mathcal{Q}|} < \infty.$$

Let Γ_r be the committed constraint set immediately before the r-th call to BREPAIR, let

$$A_r := \{\gamma \in \Gamma_r \mid \gamma \text{ is anchored}\},$$

and let π_r be the current controller. By construction of JOINT (assumption 6, trivial for $r = 0$),

$$\pi_r \models \Gamma_r. \tag{15}$$

Consider an arbitrary successful call r in this sequence.

There are two cases.

Case 1: no committed constraint is removed. Then

$$\Gamma_r \subseteq \Gamma_{r+1}, \qquad \text{and hence} \qquad \Pi(A_{r+1}) \subseteq \Pi(A_r).$$

At least one constraint γ generated from the current failure is newly committed. By Lemma 1,

$$\pi_r \not\models \gamma.$$

Together with 15,

$$\pi_r \in \Pi(\Gamma_r) \setminus \Pi(\Gamma_{r+1}),$$

and therefore

$$\Pi(\Gamma_{r+1}) \subsetneq \Pi(\Gamma_r). \tag{16}$$

Case 2: a committed constraint is removed. By construction, this is permitted only in favor of a newly generated anchored constraint γ; anchored constraints are never discarded. Hence

$$A_r \subsetneq A_{r+1}.$$

Again, by Lemma 1,

$$\pi_r \not\models \gamma.$$

Since $A_r \subseteq \Gamma_r$, equation 15 gives $\pi_r \models A_r$, so

$$\pi_r \in \Pi(A_r) \setminus \Pi(A_{r+1}),$$

and therefore

$$\Pi(A_{r+1}) \subsetneq \Pi(A_r). \tag{17}$$

By equation 17 in Case 2, and by equation 16 together with $\Pi(A_{r+1}) \subseteq \Pi(A_r)$ in Case 1, every successful call strictly decreases

$$(|\Pi(A_r)|,\ |\Pi(\Gamma_r)|)$$

in lexicographic order. Since

$$\Pi(A_r), \Pi(\Gamma_r) \subseteq \Pi_R$$

and $|\Pi_R| < \infty$, only finitely many calls to BREPAIR can occur while $\underline{\mathcal{L}}$ is fixed.

Therefore only finitely many calls to BREPAIR occur in total. Between two calls, forward verification advances over the finite horizon $0, \ldots, T$. Hence ReachECO terminates.

□

Theorem 3 (Completeness under exhaustive search)**.** *Under the assumptions of Theorem 2, let $\mathcal{X}_0$ be the fixed initial box representation, containing every anchored root. Use the exhaustive backend over the original action alphabet $\mathcal{A}$. If some $\overline{\pi} \in \mathcal{A}^{\mathcal{Q}}$ satisfies*

$$\forall B \in \mathcal{X}_0 : \quad \mathrm{Safe}_T(B, \overline{\pi}),$$

then ReachECO returns a controller $\pi' \in \mathcal{A}^{\mathcal{Q}}$ that is certified safe (Definition 1).

Proof. Since $\mathcal{A}_{\mathrm{cont}} = \mathcal{A}$, we have $\Pi_R = \mathcal{A}^{\mathcal{Q}}$. Let $G(B, h)$ be the exhaustive continuation game. For $\pi \in \Pi_R$, equation 8 and induction on h give

$$\mathrm{Safe}_h(B, \pi) \implies \pi \models G(B, h) :$$

the action $\pi(q(B))$ passes $\mathrm{SafeStep}$, every successor admits its continuation under π, and, since the exhaustive backend explores every action of $\mathcal{A}$, the game retains this action and all its successor branches at every node (assumption 5).

Every anchored constraint $\gamma = (\mathcal{G}, 0)$ has $B_{v_0} \in \mathcal{X}_0$ and remaining horizon $h \leq T$ (obligations never exceed horizon $t + 1 \leq T$, cf. line 15 of Algorithm 2). Since Safe_T implies Safe_h for $h \leq T$, the witness satisfies $\mathrm{Safe}_h(B_{v_0}, \overline{\pi})$ and hence γ. This holds for both committed and current constraints, so

$$\overline{\pi} \models \{\gamma \in \Gamma \cup \overline{\Gamma} : \gamma \text{ is anchored}\}.$$

Thus no obligation at time step zero lacks a winning continuation, and no anchored subset is jointly infeasible. Hence the algorithm cannot return UNKNOWN. By Theorem 2 it terminates and returns some (π', H) with $H \in \{T, \infty\}$; by Theorem 1, π' is certified safe to horizon H, in particular safe for the first T control steps. □

E APPENDIX: ENVIRONMENTS

E.1 LINEAR ACC

System Dynamics. Our implementation follows (Teuber et al., 2024). Let p_t denote the distance from the controlled vehicle to the lead vehicle and v_t their relative velocity, with $v_t < 0$ indicating that the vehicles are approaching each other. Under a relative acceleration a_t held constant over one sampling interval, the dynamics are

$$\begin{bmatrix} p_{t+1} \\ v_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2 \\ \Delta t \end{bmatrix} a_t.$$

We use a sampling period of $\Delta t = 0.1\,\mathrm{s}$ and relative acceleration bounds $a_t \in [-100, 100]\,\mathrm{m/s^2}$. The normalized controller action $u_t \in [-1, 1]$ determines $a_t = 100 u_t\,\mathrm{m/s^2}$; positive actions correspond to ego-vehicle braking.

We consider collision avoidance, requiring $p(\tau) > 0$ at all times, including between samples. For this specification, the analytic braking controlled-invariant region is

$$\mathcal{C}_{\mathrm{brake}} = \left\{ (p, v) : p > \frac{\max(-v, 0)^2}{2 a_{\max}} \right\}, \qquad a_{\max} = 100\,\mathrm{m/s^2}.$$

The condition ensures that the available distance exceeds the stopping distance under maximal braking when the vehicles are approaching each other.

Tube over-approximation. Under a constant relative acceleration a, the continuous-time trajectory from an initial state (p_0, v_0) is

$$p(\tau) = p_0 + v_0\tau + \tfrac{1}{2}a\tau^2, \qquad v(\tau) = v_0 + a\tau, \qquad \tau \in [0, \Delta t].$$

For an initial box $B = [\underline{p}, \overline{p}] \times [\underline{v}, \overline{v}]$, position is monotone in both initial coordinates for $\tau \geq 0$. Consequently, its lower and upper bounds over a time segment $[\tau_0, \tau_1]$ are

$$\underline{p}_{\text{Tube}} = \min_{\tau \in [\tau_0, \tau_1]} \left(\underline{p} + \underline{v}\tau + \tfrac{1}{2}a\tau^2\right), \qquad \overline{p}_{\text{Tube}} = \max_{\tau \in [\tau_0, \tau_1]} \left(\overline{p} + \overline{v}\tau + \tfrac{1}{2}a\tau^2\right).$$

Each extremum is obtained by evaluating the corresponding quadratic at the segment endpoints and, when $a \neq 0$, at its stationary point $\tau_* = -v_0/a$ if that point lies within the segment. Velocity bounds follow from the endpoint extrema of $\underline{v} + a\tau$ and $\overline{v} + a\tau$. Together, these bounds form an axis-aligned enclosure of every trajectory starting in B throughout the segment.

Reinforcement Learning. Again following (Teuber et al., 2024), all controllers are trained with PPO (Schulman et al., 2017) for a total of 200,000 time steps. In contrast to Teuber et al. (2024), we train with cleanrl (Huang et al., 2022b) instead of stable-baselines-3 (Raffin et al., 2021). Our five seeds results are reported in Table 5. We utilize the hyperparameters given in Table 4.

Table 4: ACC: training hyperparameters.

	Hyperparameter	Value
PPO	Total time steps	200000
	Learning rate	$3 \cdot 10^{-4}$ (annealed)
	Parallel environments	1
	Rollout steps	2048
	Batch	2048
	Update epochs	10
	Discount γ	0.99
	GAE λ	0.95
	Clipping coefficient	0.2
	Clipped value loss	true
	Advantage normalization	true
	Entropy / value-loss coefficient	0.0 / 0.5
	Max gradient norm, target KL	0.5, none
DWC	Thermometer bits per dimension	81
	Thermometer type	uniform
	Grid domain	$[-2.5, 202.5]$ m, ± 205 m/s
	LUT layers $\times$ width	2×128
	LUT fan-in	6
	Initial $\log \alpha$	-0.6931
	Final `tanh` squashing	off
	Critic layers $\times$ width, act.	4×64, ReLU

For the ReLU baseline, we use the 4-layer MLP from (Teuber et al., 2024) with 64 units per layer, and for the DWC baseline we use a 2-layer DWC with 128 LUT-6 per layer. The per-step reward function is defined as $r_t = 10(1 - |u|)$, encouraging small action changes. In addition, a large penalty of -2000 is applied for collisions. When the distance between the cars grows larger then $p_{rel} > 200$, a penalty of -600 is given ending the episode.

E.1.1 Baselines

In Table 2, Change [%] denotes the fraction of states for which the repaired controller output differs from the original controller output by more than 10^{-6}. For ReLU networks, this quantity is estimated using 100,000 Monte Carlo samples; for DWCs, it is computed exactly. Mean Δ denotes the average magnitude of the controller-output change over the changed states.

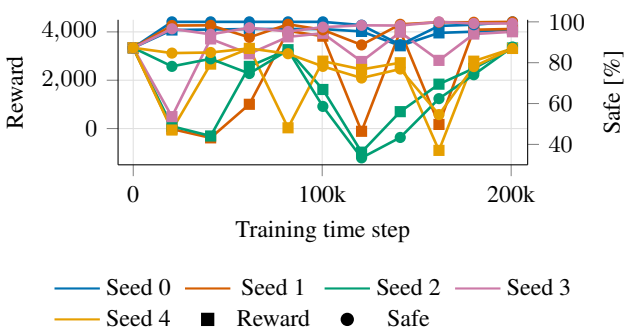

Figure 5: ACC reward and empirical safety during STL-Gym fine-tuning across five seeds (Hamilton et al., 2022). None of the five seeds achieves empirical safety when evaluated from a small region slightly above the invariant boundary. Both reward and empirical safety vary across seeds and training checkpoints.

BBReach + Supervised This baseline adapts the repair procedure of Yang et al. (2022). We first perform multi-step closed-loop reachability analysis (Yang et al. (2022) perform only one-step analysis) using our BBReach implementation. Whenever verification reaches the unsafe set, we collect the failing one-step transition and add the predecessor states to a supervised repair dataset, assigning a target action of 1.05, corresponding to full braking. To mitigate degradation of the original policy, we additionally collect 100,000 safe transitions generated by the unrepaired DWC and retain their original actions as supervised targets.

We then retrain the controller for 1000 gradient steps with batch size 256, sampling each minibatch equally from the repair and preservation datasets. After retraining, we rerun closed-loop reachability analysis and repeat this procedure until the controller verifies successfully for a horizon of 50 time steps. Finally, we run the ReachECO verifier requiring that no further repair constraints are generated. We report the total runtime of this complete repair-and-reverification loop.

The invariant is required as only one-step repair can be performed with this technique.

STLGym fine-tuning. We adapt the dense robustness reward of Hamilton et al. (2022) to the ACC specification

$$\phi = \mathbf{G}_{[0,420]}(p_{\text{rel}} > 0),$$

evaluated at the sampled states of each episode. Starting from the same pretrained NCubeV ReLU PPO controller, we perform five independent fine-tuning runs with a budget of 200,000 environment steps each. At step t, PPO receives

$$r_t = \frac{1}{P} \min_{0 \le k \le t} p_{\text{rel},k},$$

where $P = 100$. To expose the controller to difficult initial states, we draw 10% of training resets slightly inside the analytic braking boundary. We evaluate each run at ten checkpoints during training. Every checkpoint has sampled trajectories from near-boundary starts that reach $p_{\text{rel}} \le 0$. We therefore do not attempt a formal safety certificate.

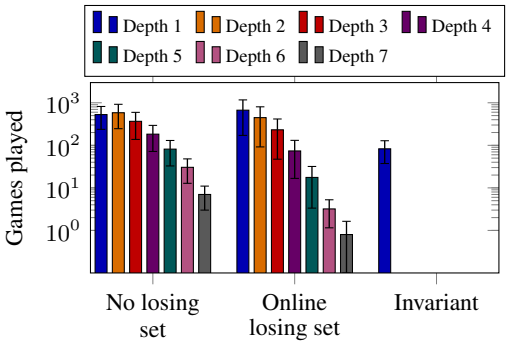

Figure 6: ACC game depths for different losing set configurations.

The 5 seed runs are shown in Figure 5.

NCubeV + REASSURE This baseline uses the NCubeV framework introduced by Teuber et al. (2024) together with their four-layer ReLU MLP. NCubeV utilizes a provably safe control envelope derived using differential dynamic logic (Platzer, 2008) to reduce infinite-horizon closed-loop safety

Table 5: ACC evaluation over 1,000 rollouts per controller. Reward is mean ± standard deviation. Unsafe denotes sampled collisions. All controllers, including the reference and invariant-repaired DWCs, use the same initial-state sampling protocol.

Seed	Reward	Unsafe (%)	Architecture
Teuber et al. (2024)	3254.16 ± 2001.49	12.70	ReLU
Before repair			
1	2439.83 ± 2588.00	25.60	DWC
2	1957.28 ± 2776.69	33.40	DWC
3	3183.04 ± 1928.62	12.20	DWC
4	2090.53 ± 2645.38	29.80	DWC
5	2499.29 ± 2386.20	22.10	DWC
After invariant repair			
1	3943.34 ± 40.08	0.00	DWC
2	3916.66 ± 38.61	0.00	DWC
3	3895.16 ± 39.77	0.00	DWC
4	3797.67 ± 37.74	0.00	DWC
5	3754.33 ± 48.61	0.00	DWC

verification to neural-network verification against the envelope. REASSURE (Fu & Li, 2022) repairs ReLU networks by constructing localized patch networks around identified buggy regions.

We combine the two approaches as follows. NCubeV identifies regions in which the controller violates the safe control envelope, and we provide these regions to REASSURE together with the desired controller outputs. REASSURE then constructs local patches enforcing the desired behavior. For ACC, full braking provides such a safe target action for states inside the controllable region.

NCubeV + Supervised This baseline is similar to BBReach + Supervised; NCubeV identifies unsafe regions from which the supervised repair dataset is generated (each sample having its correct output set to 1.05). Additionally, an equal number of safe transitions (100,000) is produced by the unrepaired neural network. With this data, the neural network is retrained until a loss threshold is reached. This procedure is repeated until NCubeV can not provide any unsafe regions and verifies the safety of the controller.

E.1.2 Extended Experimental Evaluation

Figure 6 shows the depth of the games played with $\mathcal{P}_{max} = 30$. Learning the losing set decreases the number of deeper games. Table 5 shows performance of the 5 DWC seeds before and after invariant repair, as well as the ReLU baseline from Teuber et al. (2024).

After repair, all seeds are certified and hence, there are no remaining unsafe empirical trajectories.

E.2 Docking2D

System Dynamics. Planar spacecraft docking is modeled using the linear Clohessy-Wiltshire-Hill equations (Clohessy & Wiltshire, 1960). Let $p_t = (x_t, y_t)$ denote the relative position and $v_t = (v_{x,t}, v_{y,t})$ the relative velocity of the spacecraft. Under a thrust $u_t = (u_{x,t}, u_{y,t})$ held constant over one sampling interval, the dynamics can be written as

$$\dot{x}_t = Ax_t + Bu_t,$$

with state $x_t = (x_t, y_t, v_{x,t}, v_{y,t})^\top$ and

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 3n^2 & 0 & 0 & 2n \\ 0 & 0 & -2n & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1/m & 0 \\ 0 & 1/m \end{bmatrix}.$$

As in prior work, we use a sampling period of $\Delta t = 1\,\mathrm{s}$, spacecraft mass $m = 12\,\mathrm{kg}$, and orbital mean motion $n = 0.001027\,\mathrm{s}^{-1}$. The normalized controller action $u_t \in [-1, 1]^2$ determines the

thrust components in newtons. These dynamics and parameters follow (Lopez et al., 2025; Ravaioli et al., 2022; Mandal et al., 2024).

Specification We follow the AINNCS competition benchmark (Lopez et al., 2025), which requires

$$\|v(\tau)\|_2 \le v_{\text{offset}} + 2n\|p(\tau)\|_2, \qquad v_{\text{offset}} = 0.2\,\text{m/s},$$

at all times, including between controller samples. We take as initial set

$$\mathcal{X}_0 = [70, 120]^2 \times [-0.28, 0.28]^2$$

which is larger than in (Lopez et al., 2025).

The terminal goal is specified following (Mandal et al., 2024):

$$\mathcal{X}_{\mathcal{G}} = \{(x, y, v_x, v_y) : |x| \le 0.35\,\text{m},\ |y| \le 0.35\,\text{m}\}.$$

We remove any partial-box entering the goal set that satisfies the velocity bound from further reachability propagation, considering it successfully docked.

Tube over-approximation. To account for the continuous-time evolution between controller samples, we augment the state with the constant control input,

$$z = (x, y, v_x, v_y, u_x, u_y)^\top.$$

Since $\dot{u}_x = \dot{u}_y = 0$, the augmented dynamics are linear,

$$\dot{z} = Kz, \qquad K = \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix}.$$

Hence, for a constant action over one sampling interval, the trajectory is given exactly by

$$z(\tau) = e^{K\tau} z_0, \qquad \tau \in [0, \Delta t].$$

Thus, continuous-time propagation only requires evaluating the matrix exponential of the constant 6×6 matrix K.

For an initial state box B, the augmented initial set is

$$Z = B \times \{u\} = [\underline{z}, \overline{z}].$$

At any fixed time τ, let

$$M(\tau) = e^{K\tau}.$$

The exact axis-aligned enclosure of the propagated box is

$$\underline{z}(\tau) = M^+(\tau)\underline{z} + M^-(\tau)\overline{z}, \qquad \overline{z}(\tau) = M^+(\tau)\overline{z} + M^-(\tau)\underline{z},$$

where $M^+ = \max(M, 0)$ and $M^- = \min(M, 0)$ are taken elementwise. Evaluating these bounds at $\tau = \Delta t$ and retaining the first four coordinates gives the next-state box. To conservatively enclose the continuous-time reachable tube between controller samples, we partition $[0, \Delta t]$ into time segments, following standard continuous-time over-approximation constructions systems (Althoff et al., 2007). For a segment $[\tau_0, \tau_1]$, define

$$c = \frac{\tau_0 + \tau_1}{2}, \qquad \delta = \frac{\tau_1 - \tau_0}{2}.$$

Let $[\underline{z}(c), \overline{z}(c)]$ denote the reachable box at the segment midpoint. For any $\tau \in [\tau_0, \tau_1]$, the deviation from the midpoint state can be conservatively bounded componentwise by

$$r = \left(e^{|K|\delta} - I\right) \max\left(|\underline{z}(c)|, |\overline{z}(c)|\right).$$

Hence, the reachable tube over the entire segment is contained in

$$[\underline{z}(c) - r,\ \overline{z}(c) + r].$$

The first four coordinates therefore give an axis-aligned over-approximation of all states reached from B during the segment.

Table 6: Docking2D: training hyperparameters.

	Hyperparameter	Value
PPO	Total time steps	2000000
	Learning rate	$5 \cdot 10^{-5}$ (constant)
	Parallel environments	4
	Rollout steps per environment	1000
	Batch / minibatch	4000
	Update epochs	30
	Discount γ	0.968559
	GAE λ	0.928544
	Clipping coefficient	0.3
	Clipped value loss	true
	Advantage normalization	true
	Entropy / value-loss coefficient	0.0 / 1.0
	Max gradient norm, target KL	0.5, 0.01
DWC	Thermometer bits per dimension	161
	Thermometer type	Gaussian (x, y), uniform (v_x, v_y)
	Grid domain	± 125 m, ± 0.5 m/s
	LUT layers $\times$ width	2×1024
	LUT fan-in	6
	Initial $\log \alpha$	-0.6931
	Final `tanh` squashing	on
	Critic layers $\times$ width, act.	2×64, tanh

Reinforcement Learning. We utilize the environment and reward functions of Ravaioli et al. (2022), and train three seeds for two million environment steps each. Our initial state lies on the circle of radius $130\,\mathrm{m}$ around the dock, and evaluation is run from this same set. Table 6 summarizes the hyperparameters used for training the DWC controller with PPO.

E.3 CAR2D

System dynamics. The cars state is comprised of $s = (x, y, \theta, v)$, planar position, heading, and speed. The controller outputs an acceleration and turn command $u = (u_a, u_\omega) \in [-1, 1]^2$. Each command is held constant for $\Delta t = 1\,\mathrm{s}$ and scaled to physical acceleration $a = 0.25u_a$ and turn rate $\omega = 0.5u_\omega$. The continuous dynamics are the same as the NAV benchmark in AINNCS (Lopez et al., 2025).

$$\dot{x} = v\cos\theta, \qquad \dot{y} = v\sin\theta, \qquad \dot{\theta} = \omega, \qquad \dot{v} = a.$$

Hence, $\theta(\tau) = \theta_0 + \omega\tau$ and $v(\tau) = v_0 + a\tau$, while

$$x(\tau) = x_0 + \int_0^\tau (v_0 + at)\cos(\theta_0 + \omega t)\,dt,$$
$$y(\tau) = y_0 + \int_0^\tau (v_0 + at)\sin(\theta_0 + \omega t)\,dt.$$

Specification. The unsafe set consists of contact with any obstacle or violation of $|\theta| \le \pi$. The two rectangular obstacles, expressed as Cartesian products of x and y intervals, are

$$O_1 = [2.4, 3.6] \times [1.9, 3.1],$$
$$O_2 = [4.9, 6.1] \times [-1, 5.9].$$

For the $1\,\mathrm{m} \times 1\,\mathrm{m}$ initial-box experiment, we verify

$$\mathcal{X}_0 = [-0.53, 0.47] \times [2.05, 3.05] \times [-0.35, -0.25] \times [0.94, 0.98].$$

The goal set is $x \ge 6$. The verification budget was set to $T = 100$ controller steps. After a safe transition, any portion of a post-image box contained in $\mathcal{X}_G$ is removed from further propagation.

Tube over-approximation. For a fixed controller action, the closed-form flow described above is used to compute an interval enclosure of the state at any fixed time $\tau \in [0, \Delta t]$. Evaluating this enclosure at $\tau = \Delta t$ gives the post-image box. To conservatively enclose the continuous-time trajectory between controller samples, we partition $[0, \Delta t]$ into time segments $[\tau_0, \tau_1]$. For each segment, we first evaluate the closed-form flow at τ_0, obtaining a box

$$B(\tau_0) = X(\tau_0) \times Y(\tau_0) \times \Theta(\tau_0) \times V(\tau_0).$$

Since

$$\dot{x} = v\cos\theta, \qquad \dot{y} = v\sin\theta,$$

we have

$$|\dot{x}| \le |v|, \qquad |\dot{y}| \le |v|.$$

Let

$$V_{\max} = \max_{\substack{v_0 \in [\underline{v}, \overline{v}] \\ \tau \in [\tau_0, \tau_1]}} |v_0 + a\tau|.$$

Because $v_0 + a\tau$ is affine in both v_0 and τ, this maximum is attained at one of the four endpoint combinations. Writing $h = \tau_1 - \tau_0$, every trajectory over the segment therefore satisfies

$$x(\tau) \in X(\tau_0) + [-hV_{\max}, hV_{\max}], \qquad y(\tau) \in Y(\tau_0) + [-hV_{\max}, hV_{\max}].$$

Heading and speed are affine in time,

$$\theta(\tau) = \theta_0 + \omega\tau, \qquad v(\tau) = v_0 + a\tau,$$

so their bounds over $[\tau_0, \tau_1]$ are obtained directly from their endpoint extrema. The Cartesian product of these four intervals forms a sound axis-aligned over-approximation of the reachable tube over the segment.

Controller training. The DWC controller was trained with TD3 (Fujimoto et al., 2018) for 120,000 environment steps. The training reward is progress towards the environment's goal box, minus a per-step and action penalty, with an additional reward for reaching the goal and a penalty for unsafe termination. The trained policy uses a $20 \times 20 \times 10 \times 10$ policy-grid.

F Circuit Size

F.1 Circuit Size Estimation of a DWC Policy

A DWC consists of four stages: (1) a thresholding layer producing the thermometer encoding, (2) a stack of LUT layers (in this work always two), (3) a popcount per action dimension, and (4) a final lookup mapping the popcount to an action value (Kresse & Lampert, 2026b).

We report circuit size in six-input lookup tables (LUT6), and we count only stages (2) and (3). Stage (1) is an input encoding whose cost depends on the fixed-point width of the raw observation (Kresse & Lampert, 2026b) rather than only on the policy. Stage (4) is a table, reported separately as memory (M), assuming each final output word is 16-bit: $M = L_2 \times 16bit$, with L_2 being the last layers output width.

LUT layers. A DWC layer L_ℓ is a set of k-input lookup tables, with $k = 6$ throughout. Each gate is by construction one LUT6, so the layers contribute

$$S_{\text{LUT}} \;=\; \sum_{\ell} |L_\ell|. \tag{18}$$

Popcount. Let N denote the number of final-layer outputs feeding one action dimension and A the number of action dimensions. We follow the construction of (Bacellar et al., 2024) (see Appendix E in their work):

$$n_{\text{FA}}(N) \;\approx\; N - \log_2\!\left(\tfrac{N}{3}\right) - 2, \qquad\qquad n_{\text{HA}}(N) \;\approx\; \tfrac{N}{3} - 1. \tag{19}$$

With $n_{\text{FA}}(N)$ being the number of full adders and $n_{\text{HA}}(N)$ the number of half adders needed for a popcount of N bits. Bacellar et al. (2024) then calculate the NAND2 equivalent of this to compare to ASIC implementations. We instead calculated the LUT6 equivalent.

Each of the adders can be realized by a single dual-output LUT6. Hence

$$S_{\text{pop}} \;=\; A(n_{\text{FA}}(N) + n_{\text{HA}}(N)) \tag{20}$$

and the policy size is $S_{\text{pol}} = S_{\text{LUT}} + S_{\text{pop}}$.

Table 7 gives the circuit size estimates for the different investigated DWCs.

Table 7: Circuit size estimates, in LUT6 equivalents, for the DWCs used in each benchmark. S_{LUT} counts the gates of the LUT layers, S_{pop} the adders realizing the popcount, and $S_{\text{pol}} = S_{\text{LUT}} + S_{\text{pop}}$ the complete policy.

Benchmark	S_{LUT}	S_{pop}	S_{pol}	Memory
ACC	256	162	418	2 Kibit
Docking2D	2048	1346	3394	16 Kibit
Car2D	1024	664	1688	8 Kibit

F.2 Circuit Size Estimation for a Patch

As described in Section 2.3, a repair modifies a set of policy cells $\mathcal{Q}_r \subseteq \mathcal{Q}$, with each modified cell q_r receiving an integer action correction Δ_r.

For circuit implementation, repaired policy cells with identical corrections may be grouped exactly into disjoint axis-aligned rectangles with respect to the controller's input partition. Let $\mathcal{R} = \{R_1, \ldots, R_C\}$ denote the resulting rectangles, with

$$R_c = \prod_{d=1}^{D} [\ell_{c,d}, u_{c,d}],$$

where $\ell_{c,d}$ and $u_{c,d}$ denote the lower and upper policy-cell indices of rectangle R_c along input dimension d. Since all cells within R_c have the same correction, let $\Delta_c \in \mathbb{Z}^A$ denote this correction.

The patch consists of four components: rectangle guards implemented from the existing thermometer predicates, a one-hot-to-address encoder, a memory storing the corrections, and one signed adder per action dimension. Address zero is reserved for the zero correction, so the original controller output is left unchanged outside the repaired regions.

Guard decoder. Let input dimension d have K_d thermometer thresholds $t_{d,1} < \cdots < t_{d,K_d}$, inducing $K_d + 1$ intervals. The corresponding thermometer bits are

$$\tau_{d,j}(x) := \mathbf{1}[x_d \geq t_{d,j}], \qquad j = 1, \ldots, K_d.$$

Let $c_d(x) \in \{1, \ldots, K_d + 1\}$ denote the index of the interval containing x_d. For an interval of cell indices $[\ell, u]$,

$$c_d(x) \geq \ell \iff \tau_{d,\ell-1}(x) = 1, \qquad c_d(x) \leq u \iff \tau_{d,u}(x) = 0,$$

where the first condition is omitted for $\ell = 1$ and the second for $u = K_d + 1$.

Hence, membership in any one-dimensional interval $[\ell, u]$ can be implemented by a single LUT6 using at most two thermometer bits already computed by the controller.

Let $\mathcal{I}_d$ denote the set of distinct intervals used along dimension d. Since identical interval predicates can be shared across rectangles, the one-dimensional guards require at most $\sum_{d=1}^{D} |\mathcal{I}_d|$ LUT6s. The D interval predicates defining each rectangle are then combined using a tree of 6-input LUTs. Let

$$\tau_6(f) = \begin{cases} 0, & f \leq 1, \\ \left\lceil \dfrac{f-1}{5} \right\rceil, & f > 1, \end{cases}$$

denote the number of LUT6s required to reduce f Boolean inputs to a single output. For C rectangles, the total guard logic therefore satisfies

$$S_{\text{guard}} \;\leq\; \sum_{d=1}^{D} |\mathcal{I}_d| + C\,\tau_6(D). \tag{21}$$

Address encoder. The correction Δ_c of rectangle R_c is stored at address c, so the address is $W = \lceil \log_2(C+1) \rceil$ bits wide and bit b is the disjunction of the one-hot lines whose address has that bit set, of fan-in $f_b = |\{c \in \{1, \ldots, C\} : \mathrm{bit}_b(c) = 1\}|$. Realising each as an independent balanced tree of six-input ORs,

$$S_{\mathrm{enc}} \;\le\; \sum_{b=0}^{W-1} \tau_6(f_b), \tag{22}$$

Adders. For each action dimension a, define

$$\Delta_a^{\min} := \min_{c=1,\ldots,C} (\Delta_c)_a, \qquad \Delta_a^{\max} := \max_{c=1,\ldots,C} (\Delta_c)_a.$$

Let N_a denote the maximum integer output of the original controller in action dimension a, and let w_a be the smallest integer such that

$$-2^{w_a-1} \le \Delta_a^{\min}, \qquad N_a + \Delta_a^{\max} \le 2^{w_a-1} - 1. \tag{23}$$

Using one LUT6 per carry-chain bit, the additions require

$$S_{\mathrm{add}} = \sum_{a=1}^{A} w_a \quad \text{LUT6s.} \tag{24}$$

Memory. The correction memory contains $C + 1$ words, including the zero correction at address zero, and therefore requires

$$M = (C+1) \sum_{a=1}^{A} w_a \quad \text{bits.} \tag{25}$$

Total. Excluding the correction memory, the estimated patch size is

$$S_{\mathrm{patch}} = S_{\mathrm{guard}} + S_{\mathrm{enc}} + S_{\mathrm{add}} \quad \text{LUT6s.} \tag{26}$$