SYNTHESIZING LOCAL CONTROL BARRIER FUNCTIONS FOR HYBRID SYSTEMS WITH LLM-ASSISTED SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

A control barrier function (CBF) certifies that a controller can keep a system within a set of safe states. Manually constructing a CBF for a given system often requires intuition about the function's structure and parameters and is challenging in general. For hybrid systems with mode switching, a single barrier for all modes may be overly conservative. Using per-mode local barriers can reduce this conservatism, but they introduce an additional constraint that at every mode switch, a safe state must remain safe in the next mode. We propose a framework to synthesize per-mode local CBFs satisfying the mode-switching safety constraint. In our framework, a large language model proposes and refines candidate barriers based on feedback from a satisfiability solver checking the conditions for certification. We present case studies in which the framework successfully synthesizes per-mode CBFs. In each case study, the solver proves that the synthesized functions satisfy both the CBF conditions for each mode and the safety constraint at every mode switch. The proposed framework succeeds even when a naively-prompted baseline without solver feedback fails to produce any certified CBFs.

1 INTRODUCTION

Safety for a controlled system can be defined by the property that the system's state never leaves some safe set. A control barrier function (CBF) is a tool to certify that a controller can keep the state in a given set, and it serves as a run-time filter on an existing controller (Ames et al., 2019; Brunke et al., 2022). Methods of safe reinforcement learning also use CBFs (Gu et al., 2024). An important presupposition to deploy the run-time filter and safe reinforcement learning is that a CBF for the system is already available. However, finding a CBF by hand for each specific system often requires nontrivial intuition about the structure and the parameters of the system (Dawson et al., 2023) and is challenging in general.

In this paper, we focus on hybrid systems that switch between discrete modes with different dynamics and input ranges. The transition of the modes follows the property of systems, and a controller usually cannot choose when the mode switches. In general, using a single, common barrier function for all the modes tends to be over-conservative (Yang et al., 2024a) and thus inefficient; instead, using different barriers for each mode can be more efficient. Such per-mode local barriers, however, bring additional complication because at mode transition, a safe state in one mode may then go outside the safe set of the next mode. Yang et al. (2024a) secure such mode-switching safety by reachability computation (Mitchell et al., 2005; Bansal et al., 2017), but the method assumes that CBFs for each mode are given in advance, which may limit the applicability in practice.

Automatic synthesis of barrier functions has been studied by several researchers. In counterexample-guided inductive synthesis (CEGIS; Dawson et al., 2023), a learner proposes a candidate certificate, and a verifier either proves the candidate or returns a counterexample. MLES (Hu et al., 2026) designs control policies with an evolutionary search in which a large language model (LLM) writes candidates. In BarrierBench (Taheri et al., 2026), an LLM writes barrier certificates inside a verifier loop. These methods basically suppose systems with a single vector field or synthesize a single certificate for all modes (Peruffo et al., 2021), and automatic synthesis of per-mode local CBFs for hybrid, switching systems remains underexplored.

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.

We propose a framework to synthesize local CBFs for hybrid systems, in which an LLM writes candidate barriers for each mode and an SMT solver verifies the candidates. The LLM first gives the symbolic structure of each barrier and declares which coefficients are free, a numerical fitter tunes those coefficients, and the SMT solver dReal (Gao et al., 2013) checks every condition of the certificate. The certificate covers not only the standard CBF conditions within each mode but also the mode-switching safety constraint. When a candidate fails, the verifier returns the status of each condition, and these statuses drive an evolutionary search adapted from MLES (Hu et al., 2026). In our case studies, the framework produces certificates for a rover with two terrains and for adaptive cruise control with a dry and an icy road.

Our contribution mainly lies in the following two points, which distinguish the proposed method from previous ones. First, the certificate includes safety at switches, so the framework synthesizes per-mode local CBFs for hybrid systems, which, to our knowledge, existing synthesis methods have not addressed and which Yang et al. (2024a) assumed to be given. Second, unlike MLES (Hu et al., 2026), which ranks candidates by a numerical score, in our method the evolutionary search is adapted to work directly with the discrete status of the verifier, ranking candidates by which conditions remain unproved. The proposed framework enables us to obtain local CBFs for hybrid systems without hand design, along with a machine-checked proof that covers every switch.

2 Background

2.1 Control barrier functions

Consider a control-affine system $\dot{x} = f(x) + g(x)u$ with state $x \in X \subseteq \mathbb{R}^n$ and input $u \in U \subseteq \mathbb{R}^m$. We suppose that under a locally Lipschitz control law $u = k(x)$, the trajectory from x_0 exists and is unique on a maximal time interval $I(x_0) = [0, \tau_{\max})$. A continuously differentiable function $h : X \to \mathbb{R}$ defines a *safe set* $C = \{x \in X : h(x) \geq 0\}$, in which the state should remain for safety of the system. A set of states, $A \subset X$, is *forward invariant* if $x(t) \in A$ for all $t \in I(x_0)$ whenever $x_0 \in A$, and the system is *safe* if the safe set C is forward invariant.

The function h can work as a *control barrier function* (CBF) when every state in C satisfies

$$\sup_{u \in U} \nabla h(x) \cdot \big(f(x) + g(x)u\big) \;\geq\; -\alpha\big(h(x)\big) \tag{1}$$

for a continuous and strictly increasing function α s.t. $\alpha(0) = 0$. By enforcing (1) not only in C but also outside a compact C, we can ensure that C is asymptotically stable. In fact, it is a standard practice to ask for (1) to hold at every state of some set $D \supset C$ (Ames et al., 2019; Yang et al., 2024a). However, such a function satisfying the condition on the entire region of D may not exist or may be hard to construct (Shaw Cortez et al., 2022). We thus focus on establishing the condition (1) within C alone. Under mild assumptions one can show that satisfying (1) on C keeps a trajectory in C while the trajectory stays in the interior of X (see Appendix A for details).

2.2 Hybrid systems

We define hybrid systems following the definition in Yang et al. (2024a). A *hybrid automaton* is a tuple $H = \langle X, Q, U, \{U_q\}, \{F_q\}, \mathrm{Guard}\rangle$, where X and U are the state and control spaces, and

- Q is a finite set of modes;
- $U_q \subseteq U$ is the set of inputs admissible in mode $q \in Q$;
- F_q denotes the vector field of the flow in mode q; i.e., $F_q(x, u) = f_q(x) + g_q(x)u$, with some $f_q : X \to \mathbb{R}^n$, $g_q : X \to \mathbb{R}^{n\times m}$ and $u \in U_q$; and
- $\mathrm{Guard}(q, q') \subseteq X$ denotes the set of states that trigger the switch from mode q to mode q'.

Figure 1a depicts a simple example of such a system. We write $q \to q'$ for the transition from q to q' when $\mathrm{Guard}(q, q') \neq \emptyset$, following the notation of Yang et al. (2024a). Each U_q bounds every component of the input from below and above. Appendix A.2 details other technical assumptions.

We want to use CBFs in such hybrid, switching systems. Let us denote the per-mode safe set by $C_q = \{x \in X : h_q(x) \geq 0\}$, with some function h_q, for mode q. A *local CBF* for mode q is a CBF for the dynamics F_q defined by f_q, g_q, and U_q. Local CBFs alone do not keep a hybrid system safe because the state at the instance of mode transition $q \to q'$ may lie outside $C_{q'}$ (Yang et al., 2024a).

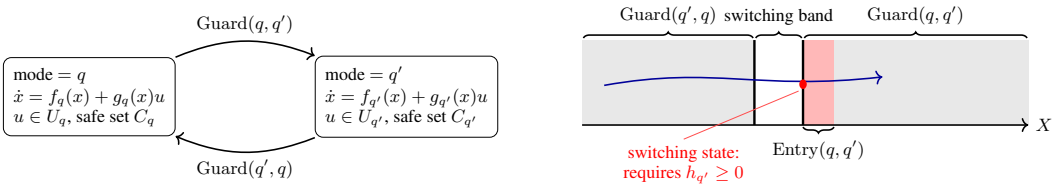


Figure 1: *(a)* A hybrid automaton with two modes; the mode switches when the state enters $\mathrm{Guard}(q, q')$ or $\mathrm{Guard}(q', q)$. *(b)* The two guards of (a) in the state space, separated by the switching band. The red point denotes a switching state of $q \to q'$, where the switch condition requires $h_{q'} \geq 0$. The red strip is the entry set $\mathrm{Entry}(q, q')$, on which the switch condition is asked.

3 Local CBF Synthesis

3.1 Switching structure

Let us first define a finer structure of how the mode switching happens. Note that it may not be a precise physical depiction of every real switching phenomena, as switching happens for various reasons in practice. Nevertheless, it is useful for rigorous analysis of the proposed framework, and we believe it does not impose any unrealistic structures.

We refer to the states at which switching happens exactly as *switching states* (the red point in Figure 1b). To ensure the safety of a hybrid automaton, every switching state (of $q \to q'$) must satisfy $h_{q'}(x) \geq 0$, for all q, q' between which the mode may switch. As we cannot precisely enumerate all the switching states in advance, we will instead ask for $h_{q'}(x) \geq 0$ at x in the *entry set*, $\mathrm{Entry}(q, q') \subseteq \mathrm{Guard}(q, q')$, the strip of the guard along its boundary through which a trajectory of mode q enters the guard. The entry set depends only on the guard and the dynamics of mode q, so it can be fixed before any barriers or controller are chosen. Every switching state of $q \to q'$ lies in the corresponding entry set (see Appendix A for details).

We suppose that $\mathrm{Guard}(q, q')$ and $\mathrm{Guard}(q', q)$ are separated by a nonempty *switching band* (the unshaded gap in Figure 1b), a set of states that lies in neither guard. If there was no switching band, a switching state of $q \to q'$ could have been in $\mathrm{Guard}(q', q)$, and the mode would have switched back at once. The notion of switching band is useful to exclude such a pathological switching behavior. Because the band separates the two guards, $\mathrm{Entry}(q, q')$ is not directly adjacent to $\mathrm{Guard}(q', q)$. From this separation, consecutive switches are a positive time apart (see Appendix A.4).

Finally, we define the *mode domain* of mode q as $X_q = \overline{X \setminus \bigcup_{q'} \mathrm{Guard}(q, q')}$, the closure of the set of states that lie in no guards leaving q. Mode q is active only in X_q, so in the certification of barriers, we seek the invariance, unsafe, and coverage conditions of mode q only on X_q.

3.2 Problem setting

A problem instance is specified by the following notions: an extended class-$\mathcal{K}_\infty$ function α, an initial state set X_I, an initial mode $q_0 \in Q$, an *avoid set* X_{av}, and an *operating set* $X_{\mathrm{op},q}$ for each $q \in Q$. An avoid set X_{av} is a set of states that the system must not enter, e.g., areas occupied by obstacles; a user should specify such an avoid set. We suppose we can also specify operating sets $X_{\mathrm{op},q}$ for $q \in Q$, in which the system operates to achieve meaningful control goals. A local CBF should be synthesized so that its safe set subsumes $X_{\mathrm{op},q}$; otherwise, the synthesized CBF may specify an overly conservative safe set, e.g., a region only slightly larger than X_I, with which the system remains near initial states and never reaches meaningful goals. Considering $X_{\mathrm{op},q} \subset C_q$ is thus useful as a guardrail to prevent such pathological cases and to obtain a meaningful outcome by barrier synthesis.

Given a problem instance $\langle \alpha, X_I, q_0, X_{\mathrm{av}}, \{X_{\mathrm{op},q}\} \rangle$, a CBF synthesis method should produce a set of barriers $h := \{h_q\}_{q \in Q}$ and an *unsafe set* X_U that covers the avoid set. (h, X_U) is said to be *certified* when every condition in Table 1 is met. Note that the conditions are verified if they hold at every state in relevant sets, not only at sampled states. The problem of barrier synthesis is to search for (h, X_U) such that the conditions are verified to hold.

Table 1: Conditions that a certified (h, X_U) satisfies. A bracketed name is instantiated per mode, per transition, or per subset of the avoid set. See Appendix A.1 for more formal definitions.

	domain of verification	condition	motivation
init	$X_I \cap X_{q_0}$	$h_{q_0}(x) \geq 0$	start safe
init_in_mode	X_I	$x \notin$ guards of q_0	make q_0 well-defined
invariance[q]	$C_q \cap X_q$	Equation (1)	remain in C_q at mode q
unsafe[q]	$X_U \cap X_q$	$h_q(x) < 0$	make C_q disjoint from X_U
coverage[q]	$X_{\mathrm{op},q} \cap X_q$	$h_q(x) \geq 0$	prevent over-conserv. C_q
switch[$q \to q'$]	$\mathrm{Entry}(q, q') \cap C_q$	$h_{q'}(x) \geq 0$	keep system safe at switch
obstacle_coverage[i]	i-th region of X_{av}	$x \in X_U$	avoid states in X_{av}

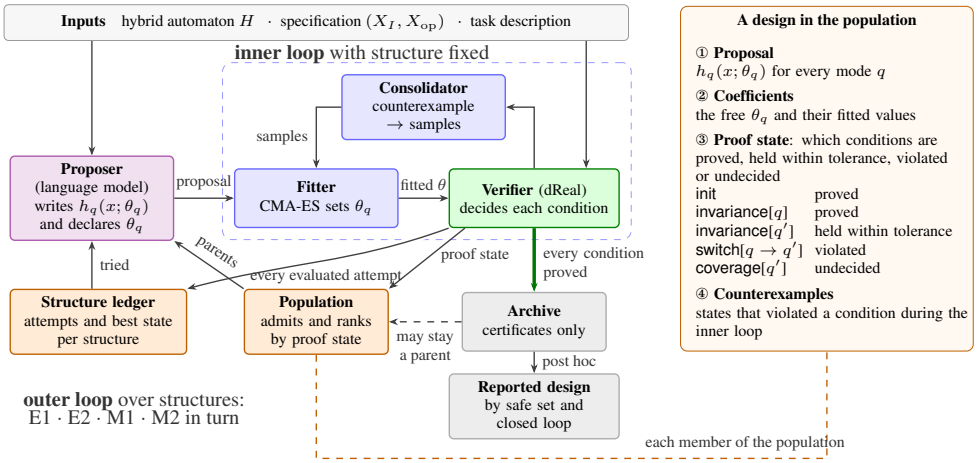


Figure 2: Framework to synthesize local CBFs. The inner loop fits undetermined parameters of a proposal with the counterexamples from the verifier. The outer loop proposes new structures using the population and the structure ledger. A design enters the archive when every condition is met.

3.3 Overall Framework

We propose a framework to search for local CBFs for switching systems. The inputs to the framework are threefold: a hybrid automaton $H = \langle X, Q, U, \{U_q\}, \{F_q\}, \mathrm{Guard}\rangle$, a problem specification $\langle \alpha, X_I, q_0, X_{\mathrm{av}}, \{X_{\mathrm{op},q}\}\rangle$, and a task description in natural language. The output is, if the algorithm ends successfully, a certified (h, X_U) satisfying the conditions in Table 1.

The overall architecture is shown in Figure 2. The framework comprises outer and inner loops. The outer loop searches for the structure of CBFs, and the inner loop adjusts the parameters of the candidate and verifies the conditions with a fixed structure. In the following, we refer to a function expression $h_q(x;\theta_q)$ with some undetermined parameters θ_q (and an unsafe set X_U) as a *proposal*; it specifies the structure of a candidate function. On the other hand, we refer to a pair of a proposal and a concrete value of θ_q as a *design*; it is an instantiation of a candidate function, ready for verification.

3.3.1 Inner Loop: Parameter Fitting and Verification

The inner loop (the blue dashed box of Figure 2) is based on the idea of CEGIS (Abate et al., 2018) used in FOSSIL (Abate et al., 2021a). It comprises the fitter, the verifier, and the consolidator.

Fitter. Given a proposal, the fitter adjusts the undetermined parameters so that the conditions in Table 1 are less violated at sampled states. The parameter fitting is done with samples given by the consolidator in the previous iteration of the inner loop. The fitter uses CMA-ES to minimize each condition's violation at the sample states (see Appendix B.1.1 for details). The fitter runs only if the verifier returns a counterexample with the initial guess, or if some parameter has no initial guess.

Verifier. Given a design from the fitter, the verifier checks if each condition in Table 1 is met by searching for a state that violates it. We use dReal (Gao et al., 2013) as the verifier. It is a δ-complete solver, that is, a state it returns may satisfy a condition by a margin smaller than a tolerance δ.

For each condition, if no condition-violating states are found, it is regarded to hold in the corresponding domain and is marked as *proved*. If a violating state (i.e., a *counterexample*) is found, we measure the largest violation of the condition in two ways: by gradient ascent from the counterexample, and by sampling on the condition's domain. We mark the condition as *violated* if the largest violation is at least 20δ, and *held within tolerance* otherwise. A condition held within tolerance does not count as proved because the small violation may still be real. A condition remains *undecided* when dReal runs out of time. These statuses for all conditions together form the design's *proof state*.

Consolidator. Following the structure of FOSSIL, the consolidator compiles information for the fitter in the next iteration of the inner loop. It adds the following three pieces of information to the samples used by the fitter: the found counterexamples, a bunch of points sampled around the counterexamples, and the end points of the gradient ascent. A counterexample held within tolerance enters here only when every remaining counterexample is also held within tolerance.

Comparison to FOSSIL. FOSSIL (Abate et al., 2021a) synthesizes certificates by training neural networks. Its learner adjusts the network weights using gradients to satisfy the certificate conditions at sampled states. Our inner loop instead adjusts parameters of a symbolic expression proposed by the LLM with CMA-ES. If fitting does not yield a certificate, our outer loop can change the terms of the expression and try the new structure. See Appendix B.1.2 for more discussion.

3.3.2 Outer loop: LLM-powered evolutionary search

The outer loop seeks structures to examine in the inner loop by an evolutionary search, as in MLES (Hu et al., 2026), powered by the proposer, the population, the structure ledger, and the archive.

Proposer. A specifically-prompted LLM writes a symbolic expression of $h_q(x;\theta_q)$ for every mode and declares which part of the parameter θ_q is undetermined, with some initial guess of the values. The LLM also writes a candidate of the unsafe set X_U. Along with the proposal $h_q(x;\theta_q)$ and X_U, it gives a short rationale stating the intent of the proposal. Appendix E gives the prompt templates, including the instruction that fixes the geometry of X_U.

More specifically, the h-proposing part uses a set of prompting instructions (*operators*) based on those of MLES: two exploration operators (E1 & E2) and two modification operators (M1 & M2). They take turns, E1, E2, M1, M2, one design per outer iteration; one lacking its parents passes.

- E1 (new structures) is asked to write a design whose structure is not recorded in the structure ledger, targeting the condition on which the recorded structures fail.
- E2 (shared structure) draws two parents from the population and writes a design that preserves their shared structure.
- M1 (term and structure changes) draws one parent and is asked to use the counterexamples and violations to identify and change the terms responsible for each violation. If all of the parent's conditions are proved, it is asked to enlarge the safe sets, keeping every condition proved.
- M2 (numerical changes) draws one parent and is prompted to preserve its structure while changing fixed constants, promoting them to free parameters, or setting their initial values.

Population. The population is a set of designs from which the proposer draws parent(s). Following Hu et al. (2026), the draw is done by exponential rank selection with probability mass $p_i \propto \exp(-c \cdot \mathrm{rank}_i)$ for each design. We draw without replacement and use $c = 0.5$. The rank of each design is determined by the lexicographic order of the following tuple key:

$$\big(\#\text{violated} + \#\text{undecided},\ \#\text{held-within-tolerance},\ \#\text{persistently-refuted},\ \max\{\text{violation}\}\big),$$

where #violated, #undecided, and #held-within-tolerance are the numbers of conditions in the corresponding status returned by the verifier at the last verification. #persistently-refuted is the number of conditions for which every verification after a fit returned a counterexample. $\max\{\text{violation}\}$ is taken over violated conditions at the last counterexamples. See Appendix B.2.1 for more details.

Two designs are regarded as the same when their fitted polynomial h_q agree term by term in every mode up to a positive factor, or when their proposed expressions of h_q agree in every mode modulo free parameters and numeric coefficients. The population keeps the highest-ranked design among those regarded as the same, and the one with the larger safe sets when both are certified.

Structure ledger. The structure ledger is a log of the design skeletons (i.e., the structure of functions $\{h_q\}$) that the framework has ever evaluated. We refer to an ever-evaluated design corresponding to a structure as the structure's *attempt*. A line of the ledger records how many attempts a structure has received. For a structure that was never certified, the line keeps its highest-ranked attempt, and for a certified one the certified design with the largest safe sets. The ledger records the numbers of the violated, undecided, and held-within-tolerance conditions, as well as the conditions for which the inner loop of the attempt returned a counterexample. The proposer reads how often the best-ranked structures in the ledger have been tried and which conditions returned counterexamples.

Archive. A design enters the archive when the inner-loop verifier proves every condition in Table 1 and its safe set is non-empty. A run of the framework ends by reporting one of the archived designs, if any. As all the archived designs are certified, the reported design is chosen by the size of the safe set and by the closed loop control performance under a CBF-QP controller (see Appendix B.3).

Comparison to MLES. MLES (Hu et al., 2026) uses an LLM to write programs that choose control inputs. It ranks candidate policies by numerical performance scores and gives the LLM visual summaries of their behavior to guide revisions. On the other hand, our outer loop searches for symbolic barrier expressions. Our inner loop fits their free coefficients and checks the certificate conditions. We rank candidates first by the discrete verifier status: how many conditions are violated or undecided. The failed conditions and counterexamples further guide revisions of the barriers.

Comparison to BarrierBench. BarrierBench (Taheri et al., 2026) asks an LLM to write a barrier certificate for a system without mode switching, together with a controller if it has inputs. Its verifier checks the candidate, and the LLM revises its coefficients or structure using failed conditions and earlier attempts. Our framework synthesizes a local CBF for each mode of a hybrid, switching system. It checks that admissible inputs exist to satisfy the CBF inequality within each mode, as well as that a switching state in the source safe set also belongs to the next mode's safe set.

3.3.3 Analysis

We analyze the safety property of the result of the proposed framework in Appendix A. Here we give an informal sketch of the analysis. Suppose that the framework returns a certified (h, X_U), and a controller continuously enforces the CBF inequality in the active mode. Under some regularity and switching assumptions, every trajectory from $X_I \cap \operatorname{int} X$ remains in its current mode's safe set and outside X_U, while it stays in $\operatorname{int} X$. It also avoids unsafe switching states (i.e., states in a guard that are safe in the current mode but outside the safe set of a possible next mode). When several guards are reached at once, this guarantee holds for every transition that can fire.

4 Experiments

4.1 Experimental setting

We report two case studies of synthesizing local CBFs for hybrid systems: a two-terrain rover (`Rover`) and the adaptive cruise control (`ACC`) example of Yang et al. (2024a). In the `Rover` task, a rover moves on the ground that has firm and loose parts (i.e., two modes; see Figure 3), which differ in the drag and traction properties. An obstacle lies in the loose part near the boundary, and a safe control should avoid the obstacle. The `ACC` task has two modes, dry and icy parts of a road, which differ in the drag and braking authority. The system's state $x = [p, v, d]$ comprises a car's position p, speed v, and distance d to the car in front. The safe sets are defined with v and d, such that the car should not be too fast and too close to the car ahead, differently on dry and icy roads. Appendix D gives the detailed description of the dynamics and the problem specifications.

For the `Rover` task, we run the proposed framework with the gpt-oss-120b model (with temperature 0) as the LLM backbone, for 60 outer-loop iterations. For the `ACC` task, we run the framework once, with GPT-5.5, for 50 outer-loop iterations. Appendix B.6 further details the experimental setting.

To our knowledge, no existing method directly produces the same output from the given system and specification: local CBFs certified both within modes and across switches. MLES (Hu et al., 2026) searches for control policies, BarrierBench (Taheri et al., 2026) constructs certificates for systems without mode switching, and the methods of Yang et al. (2024a;b) take local CBFs as given input.

4.2 CASE: TWO-TERRAIN ROVER

For the `Rover` task we ran nine trials with different random seeds, out of which six runs produced at least one certificate. Figure 3a shows the design from a successful run. It displays the largest x-direction speed $v_x^{\max}(p_x, p_y)$ tolerated by the corresponding barriers, derived from $h_q(p_x, p_y, v_x^{\max}(p_x, p_y), 0) \geq 0$, i.e., how fast the synthesized CBF of each mode allows the rover to move eastward. There we also show a simulated closed-loop trajectory under the CBF-QP controller built on this design, starting near the center of X_I.

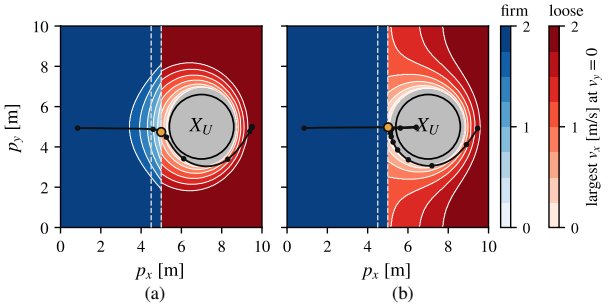
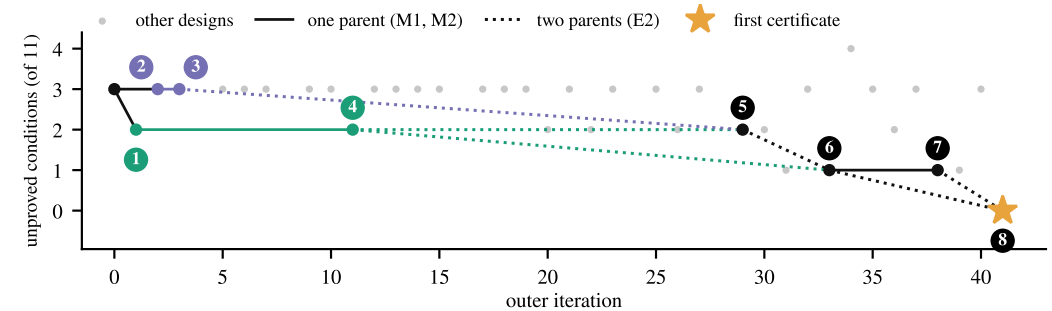


Figure 3: (*a*) Design reported for `Rover`. The largest speed $v_x^{\max}$ tolerated by h_{firm} and h_{loose} is shown. The **black curve** is a trajectory by CBF-QP. The orange point is the switching state. (*b*) A design certified w/o switch condition, close to local CBFs synthesized independently. It admits full speed up to the guard, and the trajectory enters X_U.

For example, let us look at what happens near $p_y = 5$ of Figure 3a. The firm-ground barrier h_{firm} admits less speed near the guard at $p_x = 5$, compared to the loose-ground barrier h_{loose}. More specifically, on the guard h_{firm} admits only $v_x^{\max} \approx 0.48$ m/s, whereas h_{loose} admits $v_x^{\max} \approx 0.66$ m/s there. The verifier proved the switching-safety condition (i.e., switch[firm $\rightarrow$ loose]), so every switching state that lies in the firm-ground mode's safe set, C_{firm}, also lies in the safe set of the loose-ground mode, C_{loose}.

Figure 4 follows the result shown in Figure 3a back through its parents. The vertical axis is the number of the unproved conditions among 11, and a design with zero unproved conditions is certified. M2 at ❸ and ❹ keeps the symbolic structure of the parent and changes only the numbers. At ❺, E2 combines the two branches. At ❻, E2 changes only the numbers, after which invariance[loose] is the only condition left. At ❼, M1 brings in products of a velocity and a position, and at ❽ the verifier proves all the 11 conditions. The iteration continued after this first certificate, and the design reported in Figure 3, from outer iteration 50, covers a larger safe set than that at ❽ of iteration 41.



❶ **outer 1, M1**
Change: + vx, vy (linear terms, both modes)
Rationale: "small linear terms in vx,vy … so the control input can affect the barrier"

❷ **outer 2, M1**
Change: firm: + (px−0.5); loose: (px−4.5) → (px−5.0)
Rationale: "a linear term for firm … The added longitudinal terms raise h near the mode-switch boundaries"

❸ **outer 3, M2**
Change: same form as outer 2; new initial values
Rationale: "We raise the firm-mode longitudinal weight … we lower the loose-mode radius offset r0_loose"

❹ **outer 11, M2**
Change: same form as outer 1; r0_loose 4.267 → 3.9
Rationale: "decreasing the radius offset r0_loose, which raises h at the firm→loose guard … leaving all other parameters unchanged"

❺ **outer 29, E2**
Change: firm: the terms of both parents; the obstacle term gets a weight k_obst in both modes
Rationale: "a shared obstacle-distance term that forces h<0 inside the inflated crater"

❻ **outer 33, E2**
Change: same form as outer 29; new initial values

❼ **outer 38, M1**
Change: + (px−7)vx, (py−5)vy (both modes)
*Rationale: "To strengthen the input effect where invariance was marginal, we add position-velocity cross terms (px-7)*vx and (py-5)*vy with free gains"*

❽ **outer 41, E2 (first certificate)**
Change: loose: + (px−4.5), so both modes now have the same terms
Rationale: "The same structural template is used for both modes, with separate free gains."

Figure 4: Designs on the way to the first certificate for `Rover`. Dots trace the ancestors of the first certified design (orange star). Each card gives the change of the form and quotes the LLM's rationale.

Comparison to baseline with naive prompting. We also ran a naive baseline, to measure how much of the result comes from the loop rather than the LLM's capability alone. We ran the same LLM only with the prompt used in the initial outer-loop iteration of the proposed framework, meaning that the LLM received no counterexamples, no parent designs, and no information from the structure ledger. The inner loop was unchanged, so the counterexamples were still used to fit the parameters of each proposal. We ran the LLM with the same random seeds as in the main experiment, with the same verifier, for 60 iterations, at temperature 0 as in the main experiment. Each run yielded 58 to 60 distinct completions. However, *none* of the nine runs of such a baseline produced a certificate (cf. six of the nine runs succeeded with the proposed framework).

Comparison to baseline with no switch condition. Figure 3b shows a design synthesized without the switch condition, approximately corresponding to synthesizing local CBFs for each mode independently, without taking the switching structure into account. Since the obstacle lies entirely on loose ground, the conditions for firm ground alone allow its barrier to admit full speed up to the guard. The shown trajectory then crosses into loose ground, where control authority is lower, and enters X_U. The switching state must lie in C_{loose} to guarantee the safety of the loose ground's barrier, and the conditions within each mode alone do not ensure it. The switch condition is in fact necessary to prevent such a failure case, coupling the two barriers so that safety in one mode also constrains the approach in another mode.

4.3 Case: Adaptive Cruise Control

For the ACC task, the framework gave the first certificate at outer iteration 2. We continued the iteration for 50 steps, and finally the archive had 32 certified designs. In the reported design, the barriers for the two modes (dry and ice) read

$$h_{\text{dry}} = d - 1.80\,v - \left(6.63 \times 10^{-2} + 1.87 \times 10^{-2}\,v\right) p \left(0.0625\,v - 1\right)^2 - 2.66 \times 10^{-3}\,v^3 \left(0.045455\,v - 1\right)^2 - 1, \quad \text{and}$$

$$h_{\text{ice}} = d - 1.80\,v - 2.37 \times 10^{-4}\,v^3\,(v - 22) - 6.005,$$

where the coefficients are normalized by that of d. It is interesting that only h_{dry} depends on the position p. Figure 5a draws the corresponding C_{ice} and a slice of C_{dry} on the (v, d) plane, for the $p = 0$ slice. It also draws a trajectory by the CBF-QP controller from the initial state $(v, d) = (30, 90)$.[1] The trajectory reaches the guard at a switching state, staying inside C_{ice}.

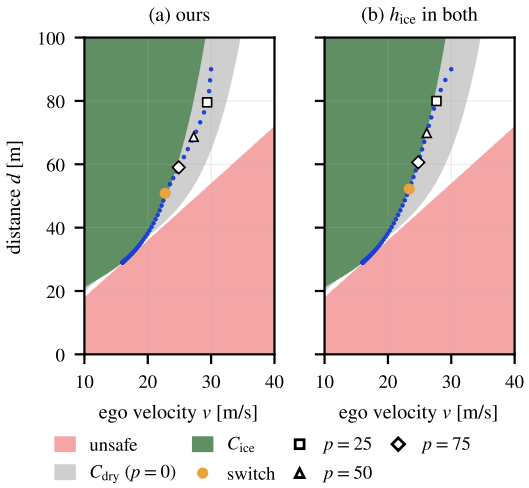


Figure 5: (a) Design reported for ACC and a trajectory from the upper-right. (b) Using h_{ice} in both modes.

Let us see what happens if we use the reported h_{ice} for both modes as a global CBF, following an analysis done by Yang et al. (2024a). In this case, the verifier proves every condition but one: the initial set does not lie in C_{ice}, so the global barrier does not certify the specification. The result is shown in Figure 5b. The initial state, $(v, d) = (30, 90)$, is outside C_{ice}, and the trajectory enters C_{ice} only after the car reaches the icy road. Moreover, in Figure 5b, early on the dry road the speed v is overly conservative (i.e., slower) compared to the case with local CBFs in Figure 5a (see also Appendix D.4, where we repeat the comparison with different gain parameters in the α function). The same observations on the difference between local and global CBFs are reported by Yang et al. (2024a). Note that Yang et al. (2024a) designed the two barriers by hand; our framework synthesizes them automatically.

[1] While the certificate was obtained with $\alpha(s) = 0.4s$, we set the CBF-QP controller with $\alpha(s) = 2s$. Such deviation is fine because only the invariance condition includes α, and increasing α's gain from 0.4 to 2 only loosens the condition; i.e., it holds with $\alpha(s) = 2s$ at every state where it was proved with $\alpha(s) = 0.4s$.

5 Related work

Synthesis of barrier certificates. Synthesizing certificates of barriers and related quantities, such as Lyapunov functions, has been studied with several approaches (Dawson et al., 2023). Prajna et al. (2007) construct polynomial barrier certificates by sum-of-squares programming. Robey et al. (2020) learn a CBF from expert demonstrations, with guarantees under Lipschitz assumptions on the dynamics. Dawson et al. (2022) learn a robust control Lyapunov barrier function with a model of the dynamics. FOSSIL (Abate et al., 2021a;b; Edwards et al., 2024), the method of Chang et al. (2019), and that of Peruffo et al. (2021) train neural networks with the CEGIS framework (Abate et al., 2018). Verdier & Mazo (2018) run CEGIS with genetic programming over symbolic expressions and the SMT solver dReal. These methods fix the structure of the certificate as a polynomial template or a network, or search expressions with genetic programming. On the other hand, in our framework, an LLM writes the structure of each barrier. Moreover, our search ranks designs directly by the discrete status of each condition, where Verdier & Mazo (2018) rank candidates by a weighted sum of numerical fitness values.

LLM-powered search has been applied to various targets, such as program evolution (Lehman et al., 2024), mathematical and algorithmic discovery (Romera-Paredes et al., 2024; Novikov et al., 2025; Lange et al., 2026), heuristic design (Ye et al., 2024), control policies (Hu et al., 2026), and models of dynamical systems (Holt et al., 2024). In this context, BarrierBench (Taheri et al., 2026) is the most relevant to our study because it uses LLMs in a verifier loop for barrier certificates. In principle, however, the method introduced in BarrierBench is for systems with a single vector field, with no consideration of switching and safety there.

CBFs for hybrid systems. CBFs and safe control of hybrid systems have been studied in control theory. Prajna et al. (2007) synthesize a barrier certificate for each mode, and Peruffo et al. (2021) and FOSSIL (Abate et al., 2021a) a single certificate for all modes, for hybrid systems that have no control input. Ravanbakhsh & Sankaranarayanan (2017) and Girard et al. (2010) deal with switched systems, where a controller chooses which mode's dynamics act, and no mode has a control input of its own. Verdier & Mazo (2020) synthesize a controller together with a Lyapunov-barrier function, which certifies that the state reaches a goal set without leaving a safe set. Lindemann et al. (2021) and Robey et al. (2021) learn one hybrid CBF for all modes from data of safe behavior. Few studies synthesize safe controllers for hybrid systems with local CBFs. Yang et al. (2024a;b) consider local CBFs but suppose some (initial) barriers are given in advance, and no fully automatic synthesis is discussed.

6 Conclusion

We present a framework to synthesize local per-mode CBFs for switching systems, with a proof of safety not only within each mode but also at switching moments. An LLM proposes and revises symbolic barrier expressions, while an inner loop fits their free coefficients and verifies the certificate conditions. The case studies show that local CBFs can be synthesized without hand design. The proposed search succeeds where naive prompting fails. Moreover, the proposed framework certifies the safety at switching moments, without which the overall hybrid system may indeed become unsafe.

A practical limitation of the current study is that the framework has been applied only to systems with a small number of distinct modes, mainly because of the difficulty of preparing realistic many-mode systems for controlled experiments. A direction of future study includes application to systems with a larger number of modes and more complex behavior of dynamics and control inputs.

AI use statement

In this work, we used generative AI tools to help develop the conceptual framework, to formulate the mathematical claims, to provide critical ingredients for proving them and to assist in writing the proofs, to propose and refine hypotheses, to design and give feedback on the methodology and experiments, to implement the method, to assist with translation, to support qualitative analysis of the run records, and to interpret results. We did not use them to generate synthetic data sets. We also used them for coding, creating the figures, suggesting experimental parameters, brainstorming,

searching for information, drafting paper sections, improving readability, identifying and summarizing literature, formatting references, suggesting the structure of the paper, and proposing its title.

We have reviewed all AI-assisted work. The numbers in the paper were regenerated from the run logs by scripts, and the authors checked them against the logs. The proofs were drafted with AI assistance and read and checked by the authors. For every cited work, an author read the passages that support what we state about it, and we recorded those passages. Language models are also a component of the method (§3); a design they propose is reported only after the solver has proved it. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

Every run launches from a frozen snapshot of the code with its resolved configuration recorded beside the log. Appendix E reproduces the prompts as sent; Appendix A gives the proof of Proposition 1; Appendix D.3 lists a certified design in full. The code, configurations and run logs will be released.

References

Alessandro Abate, Cristina David, Pascal Kesseli, Daniel Kroening, and Elizabeth Polgreen. Counterexample Guided Inductive Synthesis Modulo Theories. In *Computer Aided Verification*, pp. 270–288, 2018.

Alessandro Abate, Daniele Ahmed, Alec Edwards, Mirco Giacobbe, and Andrea Peruffo. FOSSIL: A software tool for the formal synthesis of lyapunov functions and barrier certificates using neural networks. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, HSCC '21, pp. 1–11, 2021a.

Alessandro Abate, Daniele Ahmed, Mirco Giacobbe, and Andrea Peruffo. Formal Synthesis of Lyapunov Neural Networks. *IEEE Control Systems Letters*, 5(3):773–778, 2021b.

Aaron D. Ames, Xiangru Xu, Jessy W. Grizzle, and Paulo Tabuada. Control Barrier Function Based Quadratic Programs for Safety Critical Systems. *IEEE Transactions on Automatic Control*, 62(8): 3861–3876, 2017.

Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control Barrier Functions: Theory and Applications. In *2019 18th European Control Conference (ECC)*, pp. 3420–3431, 2019.

Jean-Pierre Aubin. *Viability Theory*. Birkhäuser, Boston, 2009. ISBN 978-0-8176-4909-8 978-0-8176-4910-4.

Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J. Tomlin. Hamilton-Jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 2242–2253, 2017.

F. Blanchini. Set invariance in control. *Automatica*, 35(11):1747–1767, 1999.

Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig. Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(Volume 5, 2022):411–444, 2022.

Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural Lyapunov Control. In *Advances in Neural Information Processing Systems*, volume 32. 2019.

Charles Dawson, Zengyi Qin, Sicun Gao, and Chuchu Fan. Safe Nonlinear Control Using Robust Neural Lyapunov-Barrier Functions. In *Proceedings of the 5th Conference on Robot Learning*, pp. 1724–1735. 2022.

Charles Dawson, Sicun Gao, and Chuchu Fan. Safe Control With Learned Certificates: A Survey of Neural Lyapunov, Barrier, and Contraction Methods for Robotics and Control. *IEEE Transactions on Robotics*, 39(3):1749–1767, 2023.

Alec Edwards, Andrea Peruffo, and Alessandro Abate. Fossil 2.0: Formal Certificate Synthesis for the Verification and Control of Dynamical Models. In *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '24, pp. 1–10, 2024.

Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT Solver for Nonlinear Theories over the Reals. In *Automated Deduction – CADE-24*, pp. 208–214, 2013.

Antoine Girard, Giordano Pola, and Paulo Tabuada. Approximately Bisimilar Symbolic Models for Incrementally Stable Switched Systems. *IEEE Transactions on Automatic Control*, 55(1): 116–126, 2010.

Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, and Alois Knoll. A Review of Safe Reinforcement Learning: Methods, Theory and Applications, 2024. arXiv:2205.10330.

Samuel Holt, Tennison Liu, and Mihaela van der Schaar. Automatically Learning Hybrid Digital Twins of Dynamical Systems. *Advances in Neural Information Processing Systems*, 37:72170–72218, 2024.

Qinglong Hu, Tong Xialiang, Mingxuan Yuan, Fei Liu, Zhichao Lu, and Qingfu Zhang. Multi-modal LLM-assisted Evolutionary Search for Programmatic Control Policies. In *The Fourteenth International Conference on Learning Representations*, 2026.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards Open-Ended and Sample-Efficient Program Evolution. In *The Fourteenth International Conference on Learning Representations*, 2026.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution Through Large Models. In *Handbook of Evolutionary Machine Learning*, pp. 331–366. 2024.

Lars Lindemann, Haimin Hu, Alexander Robey, Hanwen Zhang, Dimos Dimarogonas, Stephen Tu, and Nikolai Matni. Learning Hybrid Control Barrier Functions from Data. In *Proceedings of the 2020 Conference on Robot Learning*, pp. 1351–1370. 2021.

I.M. Mitchell, A.M. Bayen, and C.J. Tomlin. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 50(7): 947–957, 2005.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. arXiv:2506.13131.

Andrea Peruffo, Daniele Ahmed, and Alessandro Abate. Automated and Formal Synthesis of Neural Barrier Certificates for Dynamical Models. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 12651, pp. 370–388. 2021.

Stephen Prajna, Ali Jadbabaie, and George J. Pappas. A Framework for Worst-Case and Stochastic Safety Verification Using Barrier Certificates. *IEEE Transactions on Automatic Control*, 52(8): 1415–1428, 2007.

Hadi Ravanbakhsh and Sriram Sankaranarayanan. A Class of Control Certificates to Ensure Reach-While-Stay for Switched Systems. *Electronic Proceedings in Theoretical Computer Science*, 260: 44–61, 2017.

Alexander Robey, Haimin Hu, Lars Lindemann, Hanwen Zhang, Dimos V. Dimarogonas, Stephen Tu, and Nikolai Matni. Learning Control Barrier Functions from Expert Demonstrations. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pp. 3717–3724, 2020.

Alexander Robey, Lars Lindemann, Stephen Tu, and Nikolai Matni. Learning Robust Hybrid Control Barrier Functions for Uncertain Systems. *IFAC-PapersOnLine*, 54(5):1–6, 2021.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Wenceslao Shaw Cortez, Xiao Tan, and Dimos V. Dimarogonas. A Robust, Multiple Control Barrier Function Framework for Input Constrained Systems. *IEEE Control Systems Letters*, 6:1742–1747, 2022.

Ali Taheri, Alireza Taban, Sadegh Soudjani, and Ashutosh Trivedi. BarrierBench: Evaluating Large Language Models for Safety Verification in Dynamical Systems. In *Proceedings of The 8th Annual Learning for Dynamics and Control Conference*, pp. 640–661. 2026.

Sander Tonkens and Sylvia Herbert. Refining Control Barrier Functions through Hamilton-Jacobi Reachability. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 13355–13362, 2022.

Cees F. Verdier and Manuel Mazo. Formal Synthesis of Analytic Controllers for Sampled-Data Systems via Genetic Programming. In *2018 IEEE Conference on Decision and Control (CDC)*, pp. 4896–4901, 2018.

Cees F. Verdier and Manuel Mazo. Formal controller synthesis for hybrid systems using genetic programming, 2020. arXiv:2003.14322.

Shuo Yang, Mitchell Black, Georgios Fainekos, Bardh Hoxha, Hideki Okamoto, and Rahul Mangharam. Safe Control Synthesis for Hybrid Systems through Local Control Barrier Functions. In *2024 American Control Conference (ACC)*, pp. 344–351, 2024a.

Shuo Yang, Yu Chen, Xiang Yin, George J. Pappas, and Rahul Mangharam. Learning Local Control Barrier Functions for Hybrid Systems, 2024b. arXiv:2401.14907.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution. In *Advances in Neural Information Processing Systems*, volume 37, pp. 43571–43608. 2024.

CONTENTS OF APPENDICES

A Analysis **14**

A.1 Notation and the conditions 14

A.2 Assumptions 15

A.3 Main statement 15

A.4 Preparation 16

A.5 Proof of main results 18

A.6 Limitations of the analysis 20

A.7 Why the standard CBF results do not apply here 20

A.8 Assumptions in case studies 21

B Implementation details **22**

B.1 Inner loop 22

B.2 Outer loop 23

B.3 Final output selection 24

B.4 CBF-QP controller 24

B.5 Checks of (A4) and C^1 at selection 24

B.6 Detailed configuration 25

C Evaluation on BarrierBench **27**

D Hybrid case studies **28**

D.1 Specification of two-terrain rover (`Rover`) 28

D.2 Specification of adaptive cruise control (`ACC`) 28

D.3 Certified example for `Rover` 29

D.4 Different gains for `ACC` 30

E Prompts **30**

E.1 Proposer prompt (`Rover`, outer-loop iteration 0) 32

E.2 What M1 appends (`Rover`, outer-loop iteration 1) 35

E.3 Operator blocks E1, E2, M2 and the M2 premises 36

A ANALYSIS

The goal of this section is to prove that a closed loop under local CBFs synthesized by the proposed framework is safe, i.e., it stays in the safe set and outside X_U (Proposition 1).

A.1 NOTATION AND THE CONDITIONS

Notation. Let us first recall some important notations that appeared in §2 and §3. A hybrid, switching system is denoted by $H = \langle X, Q, U, \{U_q\}, \{F_q\}, \mathrm{Guard}\rangle$, where X is the state space, Q is the set of modes, U_q and $F_q(x,u) = f_q(x) + g_q(x)u$ are the input set and the vector field of mode $q \in Q$, respectively, and Guard denotes the family of guard sets defining where mode switches. The mode q has its domain X_q, the closure of the set of states of X in no guard leaving q, and safe set $C_q = \{x \in X : h_q(x) \geq 0\}$. Mode transitions are denoted by $q \to q'$ for $q, q' \in Q$. Remember also the notions of an initial set X_I, an initial mode q_0, operating sets $\{X_{\mathrm{op},q}\}$, an avoid set X_{av}, an unsafe set X_U, and the function α.

For the further analysis here, we add the following structures and notations:

- *Input bounds.* Each U_q is the product of intervals $[\underline{u}_{q,j}, \overline{u}_{q,j}]$ for $j = 1, \ldots, m$, where $\underline{u}_{q,j}$ and $\overline{u}_{q,j}$ are the lower and upper bounds of the j-th input in mode q.
- *Guard function.* We write each guard as the non-negative set of a function $a_{q,q'} : X \to \mathbb{R}$:
$$\mathrm{Guard}(q, q') = \{x \in X : a_{q,q'}(x) \geq 0\}. \tag{2}$$
- *Interior region.* $W_q := \mathrm{int}\, X \cap \mathrm{int}\, X_q$. Such a set W_q is open.
- *Safe set inside the interior region.* $\Omega_q := C_q \cap W_q$.
- *Admissible inputs.* At a state x, the inputs that satisfy the CBF condition (1) are
$$K_q(x) \;=\; \big\{u \in U_q :\; \nabla h_q(x) \cdot F_q(x,u) \;\geq\; -\alpha(h_q(x))\big\}. \tag{3}$$
- *Controller and closed loop.* A controller is a feedback $k_q : X_q \to U_q$ for each mode q. The closed loop of mode q is $F_q^k(x) := F_q(x, k_q(x))$.
- *Entry set.* The *strip* of $\mathrm{Guard}(q, q')$ is the set $\{x \in X : 0 \leq a_{q,q'}(x) \leq \varepsilon\}$, with the width $\varepsilon > 0$. The entry set $\mathrm{Entry}(q, q')$ is defined from the strip in two cases.
  - If $\nabla a_{q,q'}(x) \cdot g_q(x) \neq 0$ at some $x \in X$, the entry set is the whole strip.
  - If $\nabla a_{q,q'}(x) \cdot g_q(x) = 0$ at every $x \in X$, the input cannot change how fast $a_{q,q'}$ changes. The entry set then keeps only the states of the strip that satisfy the *direction condition* $\nabla a_{q,q'}(x) \cdot f_q(x) \geq 0$. These states move into the guard or along its boundary. The condition is non-strict so that it also holds where a trajectory touches the guard tangentially.

  In both cases $\mathrm{Entry}(q, q') \subseteq \mathrm{Guard}(q, q')$, because $a_{q,q'} \geq 0$ on the strip.
- *Regions of avoid set.* The avoid set X_{av} comprises one or more disjoint subsets $X_{\mathrm{av},1}, X_{\mathrm{av},2}, \ldots$.

Moreover, let
$$S_{q,q'} := \mathrm{Guard}(q, q') \cap C_q \setminus C_{q'}. \tag{4}$$
This is the unsafe switching set (Yang et al., 2024a, Def. 8), comprising the states in the guard that lie in C_q but outside $C_{q'}$. At switching, a state should not be in this set.

The conditions. Below is the formal definition of the conditions in Table 1. For each condition, dReal receives its negation. The condition is *proved* when dReal returns UNSAT, the answer that the negation has no solution (Gao et al., 2013). The tolerance δ enters only the other answer, δ-sat,

so an UNSAT answer is exact for every δ.

$$\textsf{init}[q_0]: \quad \forall x \in X_I \cap X_{q_0} : \; h_{q_0}(x) \geq 0 \tag{5}$$

$$\textsf{init_in_mode}[q_0]: \quad X_I \cap \mathrm{Guard}(q_0, q') = \emptyset \text{ for every } q_0 \to q' \tag{6}$$

$$\textsf{invariance}[q]: \quad \forall x \in C_q \cap X_q : \; \sup_{u \in U_q} \nabla h_q(x) \cdot F_q(x,u) \;\geq\; -\alpha(h_q(x)) \tag{7}$$

$$\textsf{unsafe}[q]: \quad \forall x \in X_U \cap X_q : \; h_q(x) < 0 \tag{8}$$

$$\textsf{coverage}[q]: \quad \forall x \in X_{\mathrm{op},q} \cap X_q : \; h_q(x) \geq 0 \tag{9}$$

$$\textsf{switch}[q \to q']: \quad \forall x \in \mathrm{Entry}(q, q') \cap C_q : \; h_{q'}(x) \geq 0 \tag{10}$$

$$\textsf{obstacle_coverage}[i]: \quad \forall x \in X_{\mathrm{av},i} \cap X : \; x \in X_U \tag{11}$$

We note the following points on the list:

- As U_q is a product of closed bounded intervals, it is compact. The expression in (7) is affine, hence continuous, in u. So the supremum is attained, and condition (7) holds at x exactly when $K_q(x)$ is non-empty. The same two facts give the supremum a closed form (see below). Condition (7) is thus a first-order formula in x alone, like the other conditions.
- Condition (6) is asked only when the conditions are restricted to mode domains. Condition (11) is asked once for each avoid region $X_{\mathrm{av},i}$.
- The inequality in (8) is strict. With a non-strict inequality, a state with $h_q = 0$ could lie in both C_q and X_U.

Supremum over input sets. Condition (7) takes a supremum over U_q. This supremum has a closed form. The expression in (3) is affine in u, and U_q is a product of intervals, one per input. The supremum therefore sets each u_j to the end of its interval that makes $(\nabla h_q \cdot g_q)_j \, u_j$ largest. Let c_j and r_j be the center and the half-width of the j-th interval. Then

$$\sup_{u \in U_q} \nabla h_q \cdot g_q \, u \;=\; \sum_j \Big[c_j \, (\nabla h_q \cdot g_q)_j + r_j \, \big|(\nabla h_q \cdot g_q)_j\big| \Big].$$

The solver receives this expression with its absolute values. The fitter evaluates the same expression. The solver and the fitter therefore agree on whether an admissible input exists.

A.2 Assumptions

(A1) f_q and g_q are locally Lipschitz, and h_q is continuously differentiable (C^1).

(A2) X and every U_q are compact.

(A3) The guards and the switches have the following properties.
- Each function $a_{q,q'}$ of (2) is continuously differentiable.
- The mode switches at the instant the state enters a guard.
- The continuous state does not jump at a switch.

(A4) $\nabla h_q(x) \neq 0$ at every $x \in W_q$ with $h_q(x) = 0$.

(A5) The controller k_q is locally Lipschitz on W_q, and $k_q(x) \in K_q(x)$ at every $x \in C_q \cap X_q$.

(A6) The input at time t is $k_q(x(t))$. The controller does not hold its input between sample times.

(A7) α is an extended class-$\mathcal{K}_\infty$ function.

(A8) No state of an entry set lies in a guard that leaves the next mode: $\mathrm{Entry}(q, q') \cap \mathrm{Guard}(q', q'') = \emptyset$ for every pair of transitions $q \to q'$ and $q' \to q''$.

See Appendix A.6 for the discussion on how these assumptions are realistic.

A.3 Main statement

Proposition 1 (safety of the closed loop)**.** *Assume (A1)–(A8), and suppose that the solver has proved* (5)*,* (6)*,* (7)*,* (8) *and* (10)*. Let $x_0 \in X_I \cap \mathrm{int}\, X$ be an initial state, and let $x(\cdot)$ be the closed-loop trajectory on H starting at initial mode and state (q_0, x_0). Moreover,*

- *Let $J(x_0) = [0, t_J)$ be the longest interval on which $x(\cdot)$ exists and stays in* $\operatorname{int} X$.
- *Let $q_0, q_1, \ldots$ be the modes that the trajectory visits, and let $0 = \tau_0 < \tau_1 < \ldots$ be its switching times, finitely or infinitely many. Mode q_i is active from τ_i to τ_{i+1}. If q_i is the last mode visited, set $\tau_{i+1} = t_J$.*
- *Suppose that if the state enters two guards at the same instant, either transition may fire.*

Then, for every visited mode q_i and every $t \in [\tau_i, \tau_{i+1}] \cap J(x_0)$,

(1) $x(t) \in C_{q_i}$, that is, $h_{q_i}(x(t)) \geq 0$; and

(2) $x(t) \notin X_U$.

Proposition 2 (no unsafe switch)**.** *Under the assumptions of Proposition 1, take a visited mode q_i, a transition $q_i \to q'$ and a time $t \in [\tau_i, \tau_{i+1}] \cap J(x_0)$. Then $x(t) \notin S_{q_i,q'}$.*

These statements say that, by using the CBF synthesized by the proposed framework, the trajectory remains safe within each mode (i.e., x is in safe set C_q and is out of unsafe set X_U), as well as at the timing of mode switching (i.e., x is not in the unsafe switching set in (4)).

A.4 Preparation

The proof of Proposition 1 relies on four auxiliary results, stated as Lemmas 3–6.

- Lemmas 3 and 4 show that the closed loop of one mode keeps its trajectory in the safe set of that mode.
- Lemma 5 shows that a trajectory first reaches a guard at a state of the entry set, where the switch condition is asked.
- Lemma 6 shows that a trajectory that starts a mode in an entry set needs a fixed positive time to reach a guard that leaves that mode. A bounded time interval therefore contains finitely many switches.

A.4.1 Preliminaries

Speed bound. Recall that we suppose the set Q is finite, X and every U_q are compact by (A2), and f_q and g_q are continuous by (A1). So some $M > 0$ satisfies $\|f_q(x) + g_q(x)u\| \leq M$ for every mode q, every $x \in X$, and every $u \in U_q$. A path γ *moves at speed at most* M when $\|\gamma(t) - \gamma(s)\| \leq M|t - s|$ for all times s and t. A continuous path that is differentiable with $\|\dot\gamma\| \leq M$ at all but finitely many times moves at speed at most M, by the mean value inequality applied between those times.

Contingent cones. Nagumo's theorem, used in Lemma 4, is stated using the notion of *contingent cones*. For a set $E \subseteq \mathbb{R}^n$, write d_E for the distance to E. The contingent (Bouligand) cone of E at a point $x \in E$ is the set (Aubin, 2009, Def. 1.1.3)

$$T_E(x) = \big\{v : \liminf_{s \downarrow 0} d_E(x + sv)/s = 0\big\}.$$

Informally, $T_E(x)$ holds the directions v for which the point $x + sv$ stays within distance $o(s)$ of E along some sequence $s \downarrow 0$. Two facts follow from the definition.

- *Interior points.* If $x \in \operatorname{int} E$, then $T_E(x) = \mathbb{R}^n$ (Aubin, 2009, eq. (1.2)). A condition on the cone therefore says nothing at interior points.
- *Locality.* If $x \in E \cap E'$ and $E' \cap B(x, \rho) = E \cap B(x, \rho)$ for some $\rho > 0$, then $T_{E'}(x) = T_E(x)$. To see this, take v and $s > 0$ with $s\|v\| < \rho/2$. Since $x \in E$, $d_E(x + sv) \leq s\|v\| < \rho/2$. So $d_E(x + sv)$ is the infimum of the distances to the points of E within $\rho/2$ of $x + sv$. These points lie in $B(x, \rho)$. Hence $d_E(x + sv)$ equals the distance from $x + sv$ to $E \cap B(x, \rho)$. The argument for E' gives the distance to $E' \cap B(x, \rho)$, which is the same set. So $d_E(x + sv) = d_{E'}(x + sv)$ for all small s, and the two cones agree.

A.4.2 LEMMAS

Lemma 3 (cone at a regular boundary point)**.** *Let $x \in W_q$ with $h_q(x) = 0$ and $\nabla h_q(x) \neq 0$. Then $T_{\Omega_q}(x) = \{z : \nabla h_q(x) \cdot z \geq 0\}$.*

Proof. We use the theorem of Aubin (2009, Thm. 5.1.10) for one inequality constraint and no equality constraint. In that case the theorem states the following. Let $L \subseteq \mathbb{R}^n$ be a closed set, let c be a continuously differentiable function on an open neighborhood of L, and let $E = \{y \in L : c(y) \geq 0\}$. Let $x \in E$ with $c(x) = 0$. Suppose that some v_0 in the Clarke tangent cone $\hat{T}_L(x)$ (Aubin, 2009, Def. 5.1.3) satisfies $\nabla c(x) \cdot v_0 > 0$. Then $T_E(x) = \{z \in T_L(x) : \nabla c(x) \cdot z \geq 0\}$. We apply the theorem with $c = h_q$ on a small ball around x as follows.

1. Choose $\rho > 0$ such that $\bar{B}(x, 2\rho) \subset W_q$, where $\bar{B}(x, 2\rho)$ denotes a closed ball of radius 2ρ centered at x. Let $L := \bar{B}(x, \rho)$ and $E_\rho := \{y \in L : h_q(y) \geq 0\}$ for later use.
2. Recall that from (A1), h_q is C^1 on the open ball $B(x, 2\rho)$, an open neighborhood of L.
3. The point x is the center of L, so x is interior to L. The Clarke tangent cone at an interior point is the whole space (Aubin, 2009, p. 160, the remark after Def. 5.1.3), so $\hat{T}_L(x) = \mathbb{R}^n$. By the interior-point fact, $T_L(x) = \mathbb{R}^n$ as well.
4. Take $v_0 = \nabla h_q(x)$. By step 3, $v_0 \in \hat{T}_L(x)$. Because $\nabla h_q(x) \neq 0$, $\nabla h_q(x) \cdot v_0 = \|\nabla h_q(x)\|^2 > 0$. With these points we see the assumption of the theorem of Aubin (2009, Thm. 5.1.10) holds.
5. The theorem gives $T_{E_\rho}(x) = \{z \in T_L(x) : \nabla h_q(x) \cdot z \geq 0\} = \{z : \nabla h_q(x) \cdot z \geq 0\}$.
6. The ball $B(x, \rho)$ lies in $W_q \subseteq X$, so E_ρ and Ω_q agree on $B(x, \rho)$. Their cones at x agree by locality. □

Lemma 4 (a mode keeps its trajectory in its safe set)**.** *Fix a mode q and assume (A1), (A4), (A5) and (A7). Let $y(\cdot)$ solve $\dot{y} = F_q^k(y)$ with $y(0) \in \Omega_q$, and let $J_q = [0, t_q)$ be the longest interval on which y exists and stays in W_q. Then $y(t) \in \Omega_q$ for every $t \in J_q$.*

Proof. Nagumo's theorem (Aubin, 2009, Thm. 1.2.1 with Defs. 1.1.2 and 1.1.5) states the following. Let φ be a map from an open set $V \subseteq \mathbb{R}^n$ to $\mathbb{R}^n$, and let $E \subseteq V$ be locally compact. Suppose that φ is continuous on E and that $\varphi(x) \in T_E(x)$ at every $x \in E$. Then from every point of E, some solution of $\dot{x} = \varphi(x)$ stays in E on an interval $[0, s]$ with $s > 0$. We take $V := W_q$, $E := \Omega_q$ and $\varphi := F_q^k$. We state the remaining discussion by the following steps. Steps 1–3 are to check the hypotheses of Nagumo's theorem. Steps 4 and 5 extend its conclusion to all of J_q.

- *Step 1: Ω_q is locally compact.* Let $z_0 \in \Omega_q$. The set W_q is open, so some $\rho > 0$ gives $\bar{B}(z_0, 2\rho) \subset W_q$. The set $\Omega_q \cap \bar{B}(z_0, \rho) = \{z \in \bar{B}(z_0, \rho) : h_q(z) \geq 0\}$ is closed and bounded, so it is compact. This compact set is a neighborhood of z_0 in Ω_q.
- *Step 2: F_q^k is defined on W_q and locally Lipschitz there, hence continuous on Ω_q.* By (A1), f_q and g_q are locally Lipschitz. By (A5), k_q is locally Lipschitz on W_q. So F_q^k is locally Lipschitz on the open set W_q, and W_q contains Ω_q.
- *Step 3: $F_q^k(x) \in T_{\Omega_q}(x)$ at every $x \in \Omega_q$.* Let us see by cases.
  - $h_q(x) > 0$: the set $\{z \in W_q : h_q(z) > 0\}$ is open, lies in Ω_q and contains x. So x is interior to Ω_q, and the cone is all of $\mathbb{R}^n$.
  - $h_q(x) = 0$: by (A4), $\nabla h_q(x) \neq 0$. By Lemma 3, the cone is $\{z : \nabla h_q(x) \cdot z \geq 0\}$. The point x lies in $C_q \cap X_q$, so (A5) gives $k_q(x) \in K_q(x)$. By (3) and (A7), $\nabla h_q(x) \cdot F_q^k(x) \geq -\alpha(h_q(x)) = -\alpha(0) = 0$. So $F_q^k(x)$ is in the cone.
- *Step 4: the solution that stays in Ω_q is $y(\cdot)$.* Nagumo's theorem now gives, from every $z_0 \in \Omega_q$, a solution that stays in Ω_q on some $[0, s]$ with $s > 0$. The field F_q^k is locally Lipschitz on the

open set W_q (Step 2), so the solution through z_0 is unique. For $z_0 = y(t_0)$ with $t_0 \in J_q$, that solution is $y(t_0 + \cdot)$.

- *Step 5: the whole interval.* Let s^* be the supremum of the $s \in J_q$ such that $y(t) \in \Omega_q$ on $[0, s]$. Step 4 with $t_0 = 0$ gives $s^* > 0$. Suppose that $s^* < t_q$. Then $s^* \in J_q$, so $y(s^*) \in W_q$. $y(t) \in \Omega_q$ for $t < s^*$. The set $\Omega_q = \{z \in W_q : h_q(z) \geq 0\}$ is closed relative to W_q, because h_q is continuous by (A1). So $y(s^*) \in \Omega_q$. Step 4 with $t_0 = s^*$ keeps the trajectory in Ω_q beyond s^*. This contradicts the choice of s^*. Hence $s^* = t_q$.

□

Lemma 5 (a trajectory first reaches a guard in its entry set)**.** *Assume (A1) and (A3). Let $\sigma > 0$, and let $x(\cdot)$ be absolutely continuous on $[0, \sigma]$, with $x(t) \in X$ and $\dot{x}(t) = f_q(x(t)) + g_q(x(t))u(t)$ for almost every t, where $u(\cdot)$ is measurable and $u(t) \in U_q$. Suppose that $a_{q,q'}(x(t)) < 0$ for every $t \in [0, \sigma)$ and $a_{q,q'}(x(\sigma)) \geq 0$. Then $x(\sigma) \in \mathrm{Entry}(q, q')$.*

Proof. Write $a = a_{q,q'}$ for simplicity. By (A3), a is continuous, and $x(\cdot)$ is continuous. The interval $[0, \sigma)$ is non-empty because $\sigma > 0$. The value $a(x(t))$ is negative on $[0, \sigma)$ and non-negative at σ. So $a(x(\sigma)) = 0$, and the state $x(\sigma)$ lies in the strip $0 \leq a \leq \varepsilon$. It remains to check the direction condition. The entry set requires it only in the second case.

- *Case $\nabla a(x) \cdot g_q(x) \neq 0$ at some $x \in X$.* The entry set is the whole strip, and nothing remains to check.
- *Case $\nabla a(x) \cdot g_q(x) = 0$ at every $x \in X$.* The function a is C^1 by (A3), so $t \mapsto a(x(t))$ is absolutely continuous. For almost every t its derivative is $\nabla a(x(t)) \cdot f_q(x(t))$, whatever the input. For $0 < s \leq \sigma$,

$$0 \;\leq\; \frac{-a(x(\sigma - s))}{s} \;=\; \frac{1}{s} \int_{\sigma - s}^{\sigma} \nabla a(x(r)) \cdot f_q(x(r))\, dr.$$

  The integrand is continuous in r by (A1) and (A3). Letting $s \downarrow 0$ gives $\nabla a(x(\sigma)) \cdot f_q(x(\sigma)) \geq 0$. This inequality is the direction condition. □

Lemma 6 (minimum dwell time)**.** *Assume (A1)–(A3) and (A8), and let M be the speed bound of this section. Then some $\tau_d > 0$ has the following property. Take a transition $q \to q'$ and a path in X that starts in $\mathrm{Entry}(q, q')$ and moves at speed at most M. The path needs time at least τ_d to reach a guard that leaves q'.*

Proof. Fix a transition $q \to q'$ and a guard that leaves q'. By (A8), the entry set of $q \to q'$ and this guard are disjoint. If either set is empty, the property holds for this pair. Otherwise, both sets are closed: X is closed by (A2), and the guard is closed because $a_{q',q''}$ is continuous by (A3); and the strip and the set $\{x \in X : \nabla a_{q,q'}(x) \cdot f_q(x) \geq 0\}$ are closed by (A1) and (A3), so the entry set is closed. Both sets lie in the compact X. Two disjoint compact sets lie a positive distance Δ apart.

A path at speed at most M covers a distance of at most M per unit time. So the path needs at least Δ/M to go from the entry set to the guard. The set Q of modes is finite, so the pairs are finitely many. The minimum of Δ/M over them is τ_d. □

A.5 Proof of main results

A.5.1 Proof of Proposition 1

Proof. We argue by induction on the switches, and we show along the way that the switching times do not accumulate inside $J(x_0)$. Let S_i denote the statement at the i-th step of the induction. S_i comprises two parts:

$$(S_i) \qquad x(\tau_i) \in \Omega_{q_i}; \quad \text{and} \\ \text{if } i > 0, \;\; x(\tau_i) \in \mathrm{Entry}(q_{i-1}, q_i).$$

Base case ($i = 0$). Take a guard that leaves q_0. Condition (6) puts x_0 outside this guard. The guards are closed by (A3) and finite in number, and $x_0 \in \operatorname{int} X$. So x_0 has a neighborhood inside $\operatorname{int} X$

that meets no guard leaving q_0, and $x_0 \in W_{q_0}$. Condition (5) gives $h_{q_0}(x_0) \geq 0$. Hence, S_0 (i.e., $x_0 \in \Omega_{q_0}$) holds.

Inductive step ($i \geq 0$). Suppose that S_i holds. We discuss the inductive step with the substeps below. Step 1 covers the time range before the next switch. Steps 2–4 assume that $\tau_{i+1} \in J(x_0)$; if it is not the case, $J(x_0)$ ends before the switch, and Step 1 covers the rest of $J(x_0)$. Let $x^* := x(\tau_{i+1})$ be the state at which the trajectory switches from q_i to q_{i+1}.

- *Step 1: the trajectory stays in C_{q_i} until the switch.* Take $t \in [\tau_i, \tau_{i+1}) \cap J(x_0)$.
  - The state $x(\tau_i)$ lies in no guard that leaves q_i: for $i = 0$ by the base case, and for $i > 0$ by (A8) and the second part of S_i. By (A3), mode q_i ends at the instant the state enters such a guard. So $x(t)$ lies in no guard that leaves q_i.
  - The state $x(t)$ lies in $\operatorname{int} X$, because $t \in J(x_0)$. The guards are closed by (A3). As in the base case, $x(t) \in W_{q_i}$.
  - By (A3) and (A6), the trajectory on $[\tau_i, \tau_{i+1})$ solves the closed loop of q_i. Let y be the solution of $\dot{y} = F_{q_i}^k(y)$ with $y(0) = x(\tau_i)$. The field is locally Lipschitz on W_{q_i}, and the second item keeps the trajectory in W_{q_i}. So $x(\tau_i + s) = y(s)$ for every $s \geq 0$ with $\tau_i + s \in [\tau_i, \tau_{i+1}) \cap J(x_0)$.

  The first part of S_i gives $y(0) \in \Omega_{q_i}$. Because $t < \tau_{i+1}$ and $t < t_J$, the second item also holds on $[\tau_i, t']$ for some $t' > t$. So y stays in W_{q_i} on $[0, t' - \tau_i]$, and $t - \tau_i \in J_{q_i}$. Lemma 4 then gives $x(t) \in \Omega_{q_i} \subseteq C_{q_i}$. By (A5) and (3), the input $k_{q_i}(x(t))$ lies in U_{q_i}, so $\|\dot{x}(t)\| \leq M$. If $\tau_{i+1} \in J(x_0)$, then $x^* \in C_{q_i}$ as well, because C_{q_i} is closed and the trajectory is continuous.
- *Step 2: the switching state lies in the entry set.* By Step 1, $a_{q_i,q_{i+1}}(x(t)) < 0$ for $t \in [\tau_i, \tau_{i+1})$. By (A3), x^* lies in $\operatorname{Guard}(q_i, q_{i+1})$. Lemma 5, applied from time τ_i with $\sigma = \tau_{i+1} - \tau_i > 0$ and the input $k_{q_i}(x(t)) \in U_{q_i}$, gives $x^* \in \operatorname{Entry}(q_i, q_{i+1})$.
- *Step 3: the next barrier is non-negative at the switching state.* By Steps 1 and 2, $x^* \in \operatorname{Entry}(q_i, q_{i+1}) \cap C_{q_i}$. Condition (10) gives $h_{q_{i+1}}(x^*) \geq 0$.
- *Step 4: the switching state lies in the interior region of the next mode.* By step 2 and (A8), x^* lies in no guard that leaves q_{i+1}. These guards are closed by (A3), and $x^* \in \operatorname{int} X$ because $\tau_{i+1} \in J(x_0)$. So $x^* \in W_{q_{i+1}}$.

Hence, S_{i+1} holds because Steps 3 and 4 give $x^* \in \Omega_{q_{i+1}}$, and Step 2 gives $x^* \in \operatorname{Entry}(q_i, q_{i+1})$.

The switching times do not accumulate. Take $i \geq 1$ with $\tau_{i+1} \in J(x_0)$. By S_i, the trajectory starts mode q_i in $\operatorname{Entry}(q_{i-1}, q_i)$. On $[\tau_i, \tau_{i+1}]$ the trajectory stays in X, because $\tau_{i+1} \in J(x_0)$. By step 1, $\|\dot{x}\| \leq M$ on $[\tau_i, \tau_{i+1})$, so the trajectory moves at speed at most M on $[\tau_i, \tau_{i+1}]$. It reaches a guard that leaves q_i at τ_{i+1}. Lemma 6 gives $\tau_{i+1} - \tau_i \geq \tau_d$.

The induction reaches every time in $J(x_0)$. Take $t \in J(x_0)$. By the previous paragraph, the interval $[0, t]$ contains at most $1 + t/\tau_d$ switches. The induction therefore reaches the mode q_i that is active at t. The discussion in Step 1 above gives $x(t) \in C_{q_i}$ for $t \leq \tau_{i+1}$. This proves the statement (1) of Proposition 1.

Statement (2). At time t, the state lies in X_{q_i} by the discussion in Step 1 and because X_{q_i} is closed. When (8) is proved, $h_{q_i} < 0$ at every state of $X_U \cap X_{q_i}$. By (1), $h_{q_i}(x(t)) \geq 0$, so $x(t) \notin X_U$. □

A.5.2 Proof of Proposition 2

Proof. The set $S_{q_i,q'}$ lies in $\operatorname{Guard}(q_i, q')$, so only states in that guard need checking.

- *The state is in the guard only at the switching time.* Following the discussion in Step 1 in the proof of Proposition 1, the state lies in no guard that leaves q_i before τ_{i+1}. So if $x(t) \in \operatorname{Guard}(q_i, q')$, then $t = \tau_{i+1}$, and $x(t)$ is the first contact of the trajectory with $\operatorname{Guard}(q_i, q')$. The argument holds even when the state enters two guards at once.
- *At that state the next barrier is non-negative.* Lemma 5 puts $x(t)$ in $\operatorname{Entry}(q_i, q')$, and Proposition 1 puts $x(t)$ in C_{q_i}. Condition (10) then gives $h_{q'}(x(t)) \geq 0$. So $x(t) \in C_{q'}$, and $x(t) \notin S_{q_i,q'}$. □

A.6 LIMITATIONS OF THE ANALYSIS

Scope. Three restrictions apply to Proposition 1.

1. *Nothing outside X.* The conclusion holds only while the trajectory stays in $\operatorname{int} X$. Ames et al. (2017, Rem. 9) likewise state their conclusion only on an interval, the maximal interval of existence of the closed loop.
2. *Trajectories from $X_I \cap \operatorname{int} X$ only.* The proposition does not cover initial states elsewhere in C_{q_0}. Yang et al. (2024a, Def. 6) call a hybrid system (q, q')-safe when every trajectory that starts in mode q at a state of C_q and switches to q' stays in C_q up to that switch. After the switch, the trajectory must stay in $C_{q'}$ until the next switch, or for all later times when no next switch occurs. We prove this conclusion only along trajectories from X_I. The proposition does not cover the trajectory starting on the boundary of X, for example, as with the case in the `ACC` task (see also Appendix D).
3. *Mode domains are required.* The proposition assumes that every condition on mode q is asked on X_q and that (6) is proved. A certificate whose conditions are asked on all of X lies outside the proposition, because (6) is not asked for it.

Weaker than Yang et al. (2024a). The Theorem 2 of Yang et al. (2024a) gives (q, q')-safety from every state of a refined safe set $C_{q,q'}$. It allows any locally Lipschitz controller (Yang et al., 2024a, Def. 4) whose input lies in the admissible set of the refined barrier in mode q and in the admissible set of $h_{q'}$ in mode q'. Our Proposition 1 differs from it in two ways.

- Proposition 1 covers only trajectories from X_I, under a controller that satisfies (A5).
- The refined safe set of Yang et al. (2024a, Rem. 1) excludes the states from which the trajectory must reach $S_{q,q'}$. Our safe set C_q may contain states of $S_{q,q'}$. Proposition 1 says only that a trajectory from X_I does not reach them while mode q is active.

Continuous time only. Lemma 4 concerns the closed loop F_q^k, in which the input follows the state at every time, as (A6) requires. The lemma does not cover a zero-order hold. Suppose that the input is held at u_j over a sample interval. Invariance would then need $\nabla h_q \cdot (f_q + g_q u_j) \geq 0$ at every state with $h_q = 0$ that the trajectory visits during that interval. This requirement concerns the times between samples, and we do not certify it.

A.7 WHY THE STANDARD CBF RESULTS DO NOT APPLY HERE

Condition (7) is asked on $C_q \cap X_q$ only. The standard results ask for more:

- Ames et al. (2017, Def. 5), Ames et al. (2019, Def. 2), and Yang et al. (2024a, Def. 2) require the CBF inequality at every point of a set $D \supseteq C$, where C is the safe set.
- Ames et al. (2017, eq. (2)) define C as a subset of $\mathbb{R}^n$. With $D = C$, the inequality is still asked on all of $\{h_q \geq 0\}$, a larger set than $C_q \cap X_q$.
- Ames et al. (2019, Thm. 2) and Yang et al. (2024a, Prop. 1) therefore do not apply as stated. Yang's Prop. 1 covers one switch with the argument of our inductive step. Its proof takes the invariance of the next safe set as known, from Yang's Assumption 1. That assumption gives a CBF on all of D.

At a point $x \in W_q$ with $h_q(x) = 0$, the CBF inequality becomes

$$\sup_{u \in U_q} \nabla h_q(x) \cdot \big(f_q(x) + g_q(x)u\big) \;\geq\; 0.$$

Under (A4), Lemma 3 makes this inequality the tangency condition of Nagumo's theorem: some input puts $F_q(x, u)$ in $T_{\Omega_q}(x)$. Blanchini (1999, Thm. 3.1) states that theorem for an autonomous system $\dot{x} = f(x)$ and a closed convex set. We call the implication from the CBF inequality to forward invariance the *forward direction*, and the implication from forward invariance to the CBF inequality the *converse direction*. Three cited statements restrict the inequality to the safe set. None of them gives the forward direction under that restriction.

- Ames et al. (2019, Thm. 3) prove the converse direction and conclude that "$h|_C$ is a control barrier function on C".
- In a converse result for barrier functions of a system without input, Ames et al. (2017, proof of Prop. 3) write "we take $D = C$".
- Tonkens & Herbert (2022, Rem. 1), the source of Yang's refinement, call h *valid* when, among other conditions, h satisfies the inequality "$\forall x \in C$". Their Corollary 1 still asks for a CBF on all of their state space X.

Lemma 4 proves the forward direction with the inequality asked only on $C_q \cap X_q$, for a trajectory that stays in W_q. It is a straightforward application of Nagumo's theorem.

Remark 7 (what the solver proves, and why (A4) is needed)**.** Proposition 1 assumes five conditions that the solver proves: (5), (6), (7), (8) and (10). The proofs cite four of them directly. Condition (7) enters only through (A5), because it is what makes (A5) satisfiable.

- Assumption (A5) asks the controller to choose its input from $K_q(x)$ on $C_q \cap X_q$.
- Condition (7) states that $K_q(x)$ is non-empty on the same set.
- Without (7), $K_q(x)$ could be empty at some state, and no controller could satisfy (A5) there.
- Condition (7) does not give the Lipschitz half of (A5). The certificate does not construct the controller.

The two halves of (A5) enter the proofs in different places. The proof of Lemma 4 uses the membership $k_q(x) \in K_q(x)$ only on $\{h_q = 0\} \cap W_q$, where the cone is a half-space. It uses the Lipschitz half on all of W_q. Beyond Lemma 4, the proof of Proposition 1 uses only $k_q(x) \in U_q$, to bound the speed of the trajectory by M.

Lemma 3 describes the cone by the gradient of h_q, so it needs (A4). This assumption cannot be dropped. Take $\dot{x} = 1$ and $h(x) = -x^2$. The safe set is $C = \{0\}$. The boundary inequality holds at the only point of C, yet C is not forward invariant. The only assumption that fails is $\nabla h(0) \neq 0$. Ames et al. (2019, Rem. 5) state the same requirement for their own use of Nagumo's theorem.

A.8 Assumptions in case studies

We note how the assumptions introduced in §A.2 are met in our two case studies: `Rover` and `ACC`.

- (A1): the dynamics of the `Rover` and `ACC` tasks are polynomial, and so is every reported design. When a run chooses the design to report, it also checks with dReal that every archived design is C^1 on $C_q \cap X_q$ (see Appendix B.3).
- (A2): in both tasks, X and every U_q are products of closed bounded intervals.
- (A3): the guards of both tasks are half-spaces with affine guard functions. The other two items of (A3) are modeling assumptions.
- (A4): when a run chooses the design to report from the archive, it checks every archived design with dReal and discards those for which (A4) is not proved (see Appendix B.3). The proposal and the inner loop do not enforce it.
- (A5): a requirement on the controller, with two halves. The Lipschitz half asks k_q to be locally Lipschitz on W_q, and the membership half asks $k_q(x) \in K_q(x)$ on $C_q \cap X_q$. With the certificate's function α, the CBF-QP controller meets the membership half. The CBF-QP controller is not Lipschitz in general.
- (A6): a modeling assumption. The rollouts hold the input over each step.
- (A7): the certificates are proved with $\alpha(s) = 0.4s$.
- (A8): the entry set of $q \to q'$ lies in $\mathrm{Guard}(q, q')$, so (A8) holds when $\mathrm{Guard}(q, q')$ meets no guard that leaves q'. For the `Rover` task (see Appendix D), the switching band shown in Figure 1b separates the two guards. The entry set $5 \leq p_x \leq 5.25$ lies 0.5 away from $\mathrm{Guard}(\mathrm{loose}, \mathrm{firm}) = \{p_x \leq 4.5\}$, and the entry set $4.25 \leq p_x \leq 4.5$ lies 0.5 away from $\mathrm{Guard}(\mathrm{firm}, \mathrm{loose}) = \{p_x \geq 5\}$. The `ACC` task has a single transition, so (A8) holds trivially.

Algorithm 1 Proposed framework

Require: hybrid automaton H, problem specification $\langle \alpha, X_I, q_0, X_{\text{av}}, \{X_{\text{op},q}\} \rangle$, task description in plain language, budgets $B_{\text{out}}, B_{\text{in}}$, population size N

population $P \leftarrow \emptyset$, archive $\mathcal{A} \leftarrow \emptyset$
for $t = 0, \ldots, B_{\text{out}} - 1$ **do**
  op $\leftarrow$ first operator from position $t \bmod 4$ of (E1, E2, M1, M2) whose precondition holds
  parents $\leftarrow$ drawn from P by rank selection
  $\{h_q(\cdot; \theta_q)\} \leftarrow \text{LLM}(\text{op},$ spec, conditions (5)–(11) but (6), evidence, ledger$)$
  for $i = 0, \ldots, B_{\text{in}} - 1$, and up to 3 more once a δ-close round opens **do**
    $\theta \leftarrow$ the starting values if $i = 0$ and every coefficient has one, else CMA-ES
    $\mathcal{V} \leftarrow$ dReal's verdict on each condition, decided separately
    if every condition is proved **then**
      add (h, θ, t) to $\mathcal{A}$ if the safe set is non-empty on a grid; **break**
    if a condition is undecided and dReal returned no counterexample **then break**
    consolidate each counterexample of a violated condition into that condition's sample set
    if (6) or (11) is violated **then break**
    if every counterexample is held within tolerance **then** open a δ-close round (Appendix B.1.1), or **break** if none is left
  admit (h, θ) to P, ranked by the key shown in §3.3.2, $|P| \leq N$
return one design selected post hoc over the archive $\mathcal{A}$

B Implementation details

Algorithm 1 states the whole loop.

B.1 Inner loop

B.1.1 Fit details

The loss. The loss has a term for every condition that the fit samples.

- *Sample sets.* Each sampled condition has its own sample set, drawn over the set that the condition is asked on (Table 1). Conditions (6) and (11) do not depend on the parameters of h_q and have no sample set. The invariance and switch sample sets are drawn differently.
  - The invariance sample set is drawn over all of X and then restricted to the mode domain. The invariance term includes the sample points with $h_q \geq -0.05 \max(m, 1)$, where m is the median of $|h_q|$ over the sample set. These points lie in $\{h_q \geq 0\}$ or in a band below it. A violating point that the fit pushes just below zero therefore stays in the loss. The solver still asks the condition on $\{h_q \geq 0\}$.
  - The switch sample set is drawn over the entry set, without the restriction to C_q.
- *Aggregation.* A violation confined to a small region changes the mean over a sample set little. The same violation is enough for the solver to refute the condition. Each term therefore adds the maximum over its sample set to the mean.
- *Margin.* The loss applies a softplus to each violation plus a margin of 0.05, fifty times δ, so the optimizer aims past bare satisfaction. The solver can still refute a condition that holds by less than δ. The margin and δ are set independently.
- *The switch condition.* The switch condition from q to q' couples two barriers. The fit therefore runs coordinate descent over the modes and holds the other mode's barrier fixed. The configuration fixes the mode order: loose ground before firm ground on the `Rover`, and ice before dry on `ACC`. The loss term of the condition has a source part and a destination part. The source part lowers h_q at the entry states where $h_{q'} < 0$. The destination part raises $h_{q'}$ at the entry states where $h_q \geq 0$. On `Rover`, the source part carries ten times the weight of the destination part. On `ACC`, the two weights are equal.

δ-close rounds. When every counterexample of an inner iteration is held within tolerance, nothing is added to the sample sets and the next fit would see what the last one saw. The inner loop then opens a *δ-close round*: it adds those counterexamples, and those of earlier iterations of the proposal

that were held within tolerance, to the fit's samples and, for each of them, raises the margin of the sample set it lands in by a further 0.05. The inner loop runs four iterations per proposal, a proposal that has opened a round gets three more, and at most three rounds open per proposal.

B.1.2 Comparison to FOSSIL

The proposed method's inner loop basically follows the approach of FOSSIL (Abate et al., 2021a; Edwards et al., 2024) but notably differs from it in the following points:

1. *Fitter.* FOSSIL trains the network with gradient-based optimization, whereas our framework fits the undetermined parameters with CMA-ES because our loss is not smooth in the undetermined parameters as each term adds a maximum over its sample set (see also Appendix B.1.1).
2. *Margin term of the fit loss.* FOSSIL uses a leaky ReLU, whereas our margin term is a softplus. Moreover FOSSIL evaluates its derivative term only at sample points in a band on either side of the zero level set (Abate et al., 2021a, §4.1), whereas our loss has a band on one side only: the invariance term includes the sample points in a band below $\{h_q = 0\}$ (Appendix B.1.1).
3. *Classification of counterexamples.* We classify each counterexample by the larger of two violations, one found by gradient ascent from it and one found by sampling its condition's set. FOSSIL has no such classification. A counterexample held within tolerance stays out of the sample set until a δ-close round opens (Appendix B.1.1). The prompt reports the size of that counterexample's violation (note 2 of Appendix B.6). The solver always decides the original condition, so the classification cannot produce a certificate.

As FOSSIL does, we also accept false counterexamples from dReal. Abate et al. (2021a) state that dReal may return counterexamples whose error is bounded by δ, that such counterexamples cost CEGIS iterations, and that the final result is still correct if the procedure terminates.

In dReal's implementation, a δ-sat answer names a box rather than a point, and the counterexample we use is that box's centre. Since dReal δ-weakens the condition's region as well, the centre can lie outside the set the condition is asked on; the classification above then prefers a reading taken inside that set, and keeps the centre's reading when the sampling half reached fewer than 32 points there, because a smaller reading from too few points is not evidence that no violation is present.

B.2 Outer loop

B.2.1 Motivation of the population ranking key

Each component of the key in §3.3.2 prevents one failure that a ranking without it would have.

1. *Violated plus undecided conditions.* A count of violated conditions alone would give the best possible count to a design whose only unproved condition timed out. A design whose safe set is empty on the grid, although dReal proved every condition, counts as having one violated condition.
2. *Conditions held within tolerance.* Without this count, a design whose remaining counterexamples are all held within tolerance would receive the best possible key. This count includes a condition held within tolerance only when the parameters of the final fit satisfy the condition at its counterexample. The first component counts the condition otherwise.
3. *Persistently refuted conditions.* Without this count, two designs tied on the first two components would be ordered by their largest violation alone, even when no fit has resolved the refuted conditions of one of them. This count ignores the check of the initial guess before any fit, unless that check is the only verification.
4. *Largest violation.* Among designs tied on the three counts, the design with the smaller largest violation ranks higher. This component is the largest violation among the conditions that the last verification classified as violated. Each violation is evaluated with the parameters of the final fit at the counterexample that the last verification returned. This component is infinite when the violation of some such condition is unknown. A violation is unknown in two cases:
   - it cannot be computed, including when its evaluation is not finite;

- it is zero or negative, although the last verification returned a counterexample for the condition.

Among designs tied on the three counts, a design with an unknown violation therefore ranks below every design whose violations are known.

Every certified design has the best possible key, because no condition of it is unproved. The population therefore orders designs with equal keys in two further steps: a certified design ranks above an uncertified one, and of two certified designs, the one with the larger safe-set fraction of X, averaged over the modes, ranks higher. Once the population holds certified designs, this order sets how likely each of them is to be drawn as a parent.

B.3 Final output selection

The loop measures a certified design in three ways, and the prompt that revises such a design carries the measurements. The *safe-set fraction* of a mode is the share of a grid over X at which $h_q \geq 0$. A closed-loop rollout runs the CBF-QP controller with $\alpha(s) = \lambda s$ at the *certified gain*, the λ with which the certificate was proved. It reports the violation rate, the smallest barrier value reached, the goal-reached rate, the mean closest distance to the center of the goal region, and the *intervention rate*. For each trajectory, the loop counts the share of samples at which the *nominal input* of Table 2, clipped to U_q, violates (1). The intervention rate is the mean of these shares over the trajectories. The same design is rolled out again from the same initial states at the *diagnostic gains* $\lambda = 1$ and $\lambda = 2$, so that a shortfall in reaching the goal that a larger gain removes is attributed to the gain rather than to the barrier's shape.

A `Rover` or `ACC` run archives many certificates and reports one, chosen in three steps.

1. *Regularity.* Selection discards every archived design for which dReal does not prove two properties in every mode. The first is (A4). The second is that h_q is defined on $X \cap X_q$ and C^1 on $C_q \cap X_q$. If every archived design is discarded, the run reports no design.
2. *Admission.* Selection admits every remaining design whose safe-set fraction of X, averaged over the modes, is at least 97% of the largest such fraction among the remaining designs.
3. *Rank among the admitted designs.* Each admitted design is rolled out from 32 initial states sampled from X_I, at the certified gain $\lambda = 0.4$ and at the diagnostic gains $\lambda = 1$ and $\lambda = 2$. A gain is valid if its rollout does not violate the barrier, or if halving the rollout's time step removes the violation or reduces the magnitude of the smallest barrier value by a factor of at least 1.5. Each valid gain gives a *rollout score*: higher goal-reached rate first, then smaller mean closest distance to the center of the goal region, then lower intervention rate. Selection represents each design by the best rollout score among its valid gains and ranks the designs by that score. The safe-set fraction breaks the remaining ties.

B.4 CBF-QP controller

A CBF-QP controller returns the input of $K_q(x)$ closest to the nominal input; the name is for the quadratic program that computes it. When $K_q(x)$ is empty, it returns the input of U_q that maximizes $\nabla h_q(x) \cdot F_q(x, u)$. $K_q(x)$ is the intersection of a product of intervals with a half-space, so it is closed and convex and the closest input is unique. On $C_q \cap X_q$, (7) makes $K_q(x)$ non-empty, so the controller returns an input of $K_q(x)$. With the certificate's function α, the controller therefore meets the membership half of (A5). For example, in the `ACC` task (Figure 5), the controller is run with a larger gain $\alpha(s) = 2s$ than the one used in the synthesis; its input still lies in U_q. Where $h_q = 0$, both gains give $\alpha(h_q) = 0$, so the same argument puts its input in $K_q(x)$ there. The proofs use the membership only where $h_q = 0$, and elsewhere only $k_q(x) \in U_q$ (see also Remark 7). The controller is not Lipschitz in general, so the Lipschitz half of (A5) is not assured for either gain.

B.5 Checks of (A4) and C^1 at selection

Result. When a run chooses the design to report, it checks two assumptions of Appendix A.2 on every archived design with dReal (Appendix B.3): (A4), a non-vanishing gradient on $\{h_q = 0\}$, and the C^1 part of (A1). In the runs of §4, no archived design was discarded: all 53 archived designs of the six `Rover` runs that certified and all 32 of the `ACC` run pass both.

- *(A4).* The check divides h_q by the median of $\|\nabla h_q\|$ over the mode domain and calls the result $\tilde{h}_q$. Over the 14 (reported design, mode) pairs, dReal proves $\|\nabla \tilde{h}_q\| \geq 0.2$ on the band $|\tilde{h}_q| \leq 0.1$ in 14 of 14 cases. On firm ground of `Rover` runs 1 and 6 and on both modes of `ACC`, dReal also proves the bound 0.4. The band contains $\{h_q = 0\}$, so (A4) holds on every pair.
- C^1. All seven reported designs are polynomial, so each is C^1.

Why the gradient check is not a condition in the loop. With a linear α, h_q and $100\,h_q$ prove and refute the same conditions, so they are one design, while a bound on $\|\nabla h_q\|$ separates them. Removing that freedom needs a normalizer, and ours is the median of $\|\nabla h_q\|$ over samples. A condition built on it would let a sample decide whether a design is certified, and no sampled quantity grants a certificate. A sampled quantity can only withhold one: a proved design does not enter the archive when some mode's safe set is empty on the grid.

B.6 Detailed configuration

Constants. Each run's configuration sets most constants of Table 2, and each run saves the full configuration it ran with alongside its output. The code fixes the rest: the shells and directions of the points close to the counterexample, the floor of 32 points, the 8 ledger entries, and the initial step and later starts of CMA-ES.

Proposal. The items below give what the loop does with a proposal before the fit and what the prompt discloses.

- When every free coefficient of every mode has a starting value, dReal checks the design at those values before any fit. The starting values come from the model. A coefficient the model leaves open starts from a random draw in $(-1, 1)$, and a design with such a coefficient goes straight to the fitter.
- The loop evaluates the proposal as written. The prompts ask for two kinds of retention: M1 is asked to preserve every term the evidence does not implicate, and M2 is asked to leave every mode's structure unchanged. The loop checks M2's and records a violation in the run log. It evaluates the design either way.
- The prompt does not include condition (6), because no barrier can change its verdict.
- The prompt states that the input drops out of the invariance expression where $\nabla h_q \cdot g_q = 0$. At those states, $\nabla h_q \cdot f_q$ alone must satisfy the condition. We have not established what this disclosed passage does to certification.

Evidence notes. Besides the counterexamples and the fitted values, the evidence block on an uncertified parent can carry the lines below. The loop computes each line from the fit and the verification.

1. *Residual of the fit.* For each mode, the conditions that the fit could not satisfy on its samples, with the largest violation of each. When two or more such conditions are listed, the prompt adds that the fit could not satisfy them together. A switch condition is left out of this list when the other mode's barrier has changed since the fit.
2. For each condition held within tolerance, its value near the counterexample.
3. For a failing invariance condition, a statement that the input cannot act at its counterexample. The loop computes this note only when every unproved condition of the design is held within tolerance. The loop compares $\|\nabla h_q \cdot g_q\|$ at the counterexample with its median over the condition's set, estimated from 2048 sampled points. Below 0.05 of that median, the note marks the condition as out of the input's reach and says that changing the coefficients cannot satisfy the condition there. Note 2 then omits the condition.
4. For a mode whose fitted barrier rounds to the constant zero, a statement that its safe set is the whole domain.

Rounding. Coefficients are rounded to six decimal places before the solver sees them. A certificate is therefore a statement about the rounded barrier.

Table 2: Constants of the reported runs. The entry-set width ε is in the units of the guard's state variable, so each benchmark sets its own.

constant	value
population size	6
selection pressure c	0.5
fitted designs regarded as the same	coefficients of the expanded h_q divided by the largest in magnitude, those below 10^{-3} dropped, the rest rounded to 3 decimals; a non-polynomial h_q is compared as written
inner iterations per proposal	4, and 3 more after a δ-close round opens
solver time per condition	10 s of CPU for the solver process, including its start-up; a timeout is retried once at 180 s, then the condition is undecided
fit time	180 s of CPU for the fit, and 180 s for each of its parallel evaluation workers
fit samples	at most 100000 per set, drawn over a box that contains the set and kept only inside it, 40% of the draws on the box's faces
CMA-ES	4 starts, at most 200 iterations, initial step 0.5; starts after the first drawn from $(-1, 1)$
counterexample cloud	128 Gaussian points with standard deviation 0.1 around the counterexample, clipped to X and kept only inside the condition's set
ascent	step 1% of the widest interval of X, at most 50 steps; stops at the first non-improving step or on leaving the condition's set, and adds no point when no step improved on the counterexample
search inside the condition's set	400000 points drawn uniformly over X, of which those inside the condition's set are kept; 16 ascents from random kept points
points close to the counterexample	512 directions on each of four shells, of radius 10^{-6}, 10^{-5}, 10^{-4} and 10^{-3} times each axis's width of X; points outside the condition's set or outside X are dropped
replacement of the first magnitude	needs at least 32 kept points inside the condition's set
margin of condition (11)	disk radius grown by 0.05 on the `Rover`; $d - 1.8v \le 0.05$ on `ACC`
safe-set fraction	grid of 15 points per axis over X, averaged over the modes
input reach (evidence note 3)	out of reach below 0.05 of the median $\|\nabla h_q \cdot g_q\|$ over the condition's set
outer iterations without improvement before E1 is told	3
structure ledger in the prompt	the 8 highest-ranked structures, shown to E1, M1 and M2 once two structures exist
class-$\mathcal{K}$ function	$\alpha(s) = \lambda s$ with the certified gain $\lambda = 0.4$
weight of the destination part of the switch term	0.1 on the `Rover`; 1.0 on `ACC`
rollout for quality and selection	step 0.01 s; horizon 60 s on the `Rover`, 18 s on `ACC`, 10 s on BarrierBench
nominal input	`Rover` $u = \big(0.5(9.5 - p_x) - 1.2v_x,\ 0.5(5 - p_y) - 1.2v_y\big)$; `ACC` $u = 0.12(30 - v)$
language-model temperature	0 for gpt-oss-120b, including the naive-prompting baseline of §4.2; the API's default for gpt-5.5 and gpt-4o
seed of the fitter and the parent draws	1–9 on the `Rover` runs; 1 on `ACC`; 0 on BarrierBench
solver tolerance δ	10^{-3}
threshold between violated and held within tolerance	20δ
admission to selection	at least 97% of the largest safe-set fraction
entry-set width ε	0.25 on both `Rover` guards; 1.0 on `ACC`

What decides certification. The items below separate what the solver decides from what the loop only measures.

- Sampled points never reach the solver. Only the fitter uses the consolidator's points.
- The solver decides the conditions as Appendix A.1 states them. Condition (11) is asked on the region grown by the margin in Table 2, so the proved condition implies the stated one.
- The solver proves every archived design individually.

- A grid check can keep a proved design out of the archive. No sampled quantity puts a design into it.
- Selection ranks certificates against each other and cannot promote an unproved design.
- The model's account of its own design carries no weight. At step ❼ of Figure 4, the rationale says that the added cross terms vanish inside the crater, which $(p_x - 7)v_x$ does not; the solver decided the design on its form, not on the account.

C EVALUATION ON BARRIERBENCH

We applied the proposed framework to the BarrierBench benchmark suite (Taheri et al., 2026).

- *Problems.* Of the 100 problems, 14 have discrete-time dynamics. We run the other 86, which all have the form $\dot{x} = f(x) + g(x)u$. Of these 86, 58 have an input.
- *Sets.* X_I and X_U are fixed as each problem states them.
- *Input.* BarrierBench synthesizes a controller and checks the closed loop under it. We synthesize no controller. We instead prove that an input in U satisfies the conditions. BarrierBench's problems give no input bounds, so we take $|u_i| \leq 1000$ for every input.
- *State space.* BarrierBench's problems declare no bounded state space. dReal decides a condition only over a bounded set, so we choose X for each problem as a product of intervals. The choice depends on the shape of X_U:
  - if X_U is the complement of a ball, X is the smallest product of intervals that contains that ball;
  - if X_U is the complement of a product of intervals, X is that product with every interval widened by 1 at both ends;
  - otherwise, X is the smallest product of intervals that contains X_I and X_U, with every interval widened by 1 at both ends.

  The first two rules apply to 72 of the 86 problems: 71 ball complements and one complement of a product of intervals. On these 72 problems, every state on the boundary of X belongs to X_U, except the points where X touches the ball. A trajectory from X_I that leaves X elsewhere therefore passes through X_U first. The certificate rules out every trajectory from X_I that reaches X_U. On these 72 problems, safety therefore holds on $\mathbb{R}^n$ for every trajectory that does not leave X through one of the touching points. On the other 14, the certificate covers a trajectory only while it stays in X.
- *Operating set.* The problems pose no task, so no problem has an operating set or a coverage condition.
- *Run.* Each problem gets one run with gpt-4o at the API's default temperature. The class-$\mathcal{K}$ function is $\alpha(s) = 0.4s$, as on the `Rover`. The run stops at its first certificate that passes the regularity step of Appendix B.3, or after 30 outer iterations. Each outer iteration alternates fitting and verifying as in §3.3.1.
- *Budget.* BarrierBench's framework allows 5 main iterations with up to 4 refinements each. We allow more outer iterations because we use no database of solved problems.
- *Model.* BarrierBench does not state which gpt-4o snapshot it used.

We obtained the following results for BarrierBench.

- The 14 discrete-time problems count as failures, so the framework certifies 83 of the 100 problems.
- Of these 83, 72 are certified at outer iteration 0.
- The framework certifies 27 of the 28 continuous-time problems without an input. On the one problem that it does not certify, the run ended during a solver call without a result.
- The framework certifies 56 of the 58 problems with an input. On the two problems that it does not certify, the runs used all 30 outer iterations. On the certified ones it proves only that some input with $|u_i| \leq 1000$ satisfies the conditions, while BarrierBench synthesizes a controller.
- Taheri et al. (2026) report that their framework certifies 46 of the 100 problems with gpt-4o.

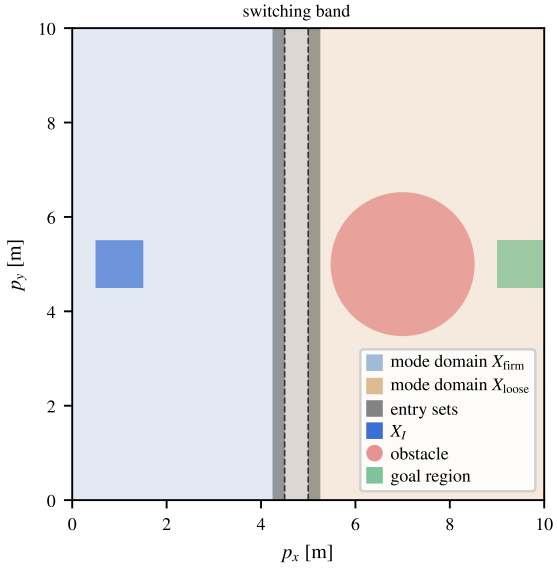


Figure 6: `Rover`'s setting: the two mode domains, overlapping in the switching band $4.5 \le p_x \le 5$, the entry sets of the two guards (drawn at $v_x = \pm 0.1$ toward the destination mode, $v_y = 0$), the initial set, the obstacle and the goal region.

D HYBRID CASE STUDIES

D.1 SPECIFICATION OF TWO-TERRAIN ROVER (ROVER)

The specification of the `Rover` task is as follows and depicted in Figure 6.

- *State and input.* $x = (p_x, p_y, v_x, v_y)$ and $u = (a_x, a_y) \in [-1, 1]^2$ in both modes.
- *Dynamics.* $\dot{p}_x = v_x$, $\dot{p}_y = v_y$, $\dot{v}_x = \kappa_q a_x - \mu_q v_x$, $\dot{v}_y = \kappa_q a_y - \mu_q v_y$, with traction $\kappa_{\text{firm}} = 1.0$, $\kappa_{\text{loose}} = 0.6$ and drag $\mu_{\text{firm}} = 0.05$, $\mu_{\text{loose}} = 0.3$.
- *State space.* $p_x, p_y \in [0, 10]$ and $v_x, v_y \in [-2, 2]$.
- *Guards.* Firm to loose at $p_x \ge 5$, loose to firm at $p_x \le 4.5$. The initial mode is firm. The entry-set width is 0.25 for both guards.
- *Initial set.* $p_x \in [0.5, 1.5]$, $p_y \in [4.5, 5.5]$, $v_x, v_y \in [-0.1, 0.1]$.
- *Obstacle.* A disk of radius 1.5 centered at $(7, 5)$. The model writes X_U, and condition (11) requires X_U to contain the disk with its radius grown by 0.05.
- *Operating sets.* Firm: $p_x \in [0.5, 5]$, $v_x, v_y \in [-0.3, 0.3]$. Loose: $p_x \in [4.5, 9.5]$, $v_x, v_y \in [-0.3, 0.3]$, outside the disk of radius 2.5 centered at $(7, 5)$.
- *Goal region.* $p_x \in [9, 10]$, $p_y \in [4.5, 5.5]$. Only the rollouts use the goal region. No solver condition contains it.

D.2 SPECIFICATION OF ADAPTIVE CRUISE CONTROL (ACC)

The `ACC` task is from Yang et al. (2024a).

- *State and input.* $x = (p, v, d)$: ego position, ego speed and headway to a lead car moving at 16 m/s. The input u is the commanded acceleration, $|u| \le c_q g_0$ with the gravitational acceleration $g_0 = 9.81$, $c_{\text{dry}} = 0.741$ and $c_{\text{ice}} = 0.113$.
- *Dynamics.* $\dot{p} = v$, $\dot{v} = -\text{Fr}_q(v) + u$, $\dot{d} = 16 - v$, with the friction terms $\text{Fr}_{\text{dry}}(v) = 0.05 + 0.005v + 0.0005v^2$ and $\text{Fr}_{\text{ice}}(v) = 0.02 + 0.002v + 0.0005v^2$.

- *State space.* $p \in [0, 400]$, $v \in [0, 40]$, $d \in [0, 200]$.
- *Guard.* Dry to ice at $p \geq 100$, with no guard back. The initial mode is dry. The entry-set width is 1.0.
- *Initial set.* $p \in [0, 5]$, $v \in [29, 30]$, $d \in [88, 92]$.
- *Unsafe region.* $d \leq T_h v$, with the time headway $T_h = 1.8\,\mathrm{s}$. The model writes X_U, and condition (11) requires X_U to contain the region $d - T_h v \leq 0.05$.
- *Parameters.* Yang et al. (2024a) give the form of the dynamics, the unsafe region and the barriers, but no numerical values other than the guard's position. We obtained T_h, c_{dry}, c_{ice} and the lead car's speed by fitting them to Figure 3 of Yang et al. (2024a). The initial set contains the start of the trajectories of that figure. The friction coefficients are our choice.
- *Operating sets.* Dry: $p \in [0, 100]$, $v \in [0, 22]$, $d \geq 2.2v + 0.068783(16 - v)^2$. Ice: $p \in [99, 400]$, $v \in [0, 22]$, $d \geq 2.2v + 0.451048(16 - v)^2$.

D.3 Certified example for Rover

Here we describe the design corresponding to Figures 3 and 4 in full, discuss what its structure shows, and show closed-loop rollouts of the reported design of every `Rover` run that certified.

The run found its first certificate at outer-loop iteration 41 as shown in Figure 4. The safe sets of the design reported in Figure 3, from outer-loop iteration 50, cover 62.8% of X (53.3% on firm ground and 72.2% on loose ground; see also Appendix B.3).

The problem specification.

- The state is $x = (p_x, p_y, v_x, v_y)$, i.e., the horizontal and vertical position and velocity of the rover.
- The guard into loose ground is $p_x \geq 5$. The guard back into firm ground is $p_x \leq 4.5$.
- The configured obstacle (i.e., the avoid set) is the disk of radius 1.5 centered at $(7, 5)$.
- $X_I = [0.5, 1.5] \times [4.5, 5.5] \times [-0.1, 0.1]^2$.

The unsafe set. The method reported X_U as the obstacle grown to radius 1.6:

$$(p_x - 7)^2 + (p_y - 5)^2 \leq 2.56.$$

The prompt supplied the obstacle's center and radius and asked for a growth of at least 0.1 (see Appendix E.1); condition (11) checks that the obstacle grown by 0.05 lies in X_U.

The design. The reported design comprises (the coefficients are not normalized):

$$\begin{aligned}
h_{\mathrm{firm}} = {} & 1589.316599\,(p_x - 7)^2 - 615.936085\,(p_x - 7)(p_x - 5) + 1536.090882\,(p_y - 5)^2 \\
& + 1709.723916\,(5 - p_x)(p_x - 0.5) \\
& + 3833.435898\,(0.3 - v_x)(v_x + 0.3) + 3001.415727\,(0.3 - v_y)(v_y + 0.3) \\
& + 1678.847812\,v_x(p_x - 7) + 2081.317984\,v_y(p_y - 5) \\
& - 616.436703\,p_x + 369.928292\,v_x - 258.409955\,v_y - 1554.950126, \\
h_{\mathrm{loose}} = {} & 735.572896\,(p_x - 7)^2 + 463.296494\,(p_x - 7)(p_x - 5) + 1387.203442\,(p_y - 5)^2 \\
& - 488.933133\,(9.5 - p_x)(p_x - 4.5) \\
& + 3613.272474\,(0.3 - v_x)(v_x + 0.3) + 2577.502051\,(0.3 - v_y)(v_y + 0.3) \\
& + 1381.050326\,v_x(p_x - 7) + 1775.422987\,v_y(p_y - 5) \\
& - 1392.037778\,p_x + 1834.49274\,v_x - 176.029794\,v_y + 6737.175956.
\end{aligned}$$

The two barriers. Both barriers have total degree two and consist of the same monomials. Each has a negative coefficient on v_x^2 and v_y^2, so h_q becomes negative at any position if the speed is large enough. The two barriers are not a rescaling of each other: the ratio of the firm-ground coefficient to the loose-ground one is between 1.06 and 1.22 for the quadratic terms other than p_x^2, but 0.25 for p_x, and the coefficient of p_x^2 is negative on firm ground and positive on loose ground.

The closed loop. Five of the six reported designs reach the goal from all 32 initial states; the remaining one reaches it from 27 of the 32. No trajectory enters X_U, leaves X or reaches a state with $h_q < 0$ in its active mode. All 192 trajectories (six runs, 32 each) switch to loose ground. The table below covers the reported designs of the six runs of §4.2 that certified. The other three of the nine runs archived no certificate.

The configuration is as follows:

- *Run (outer iteration).* The run and the outer iteration of its reported design.
- *Initial states.* The same 32 states for every run, drawn uniformly from X_I.
- *Controller.* The CBF-QP controller of Appendix B.4 with $\alpha(s) = \lambda s$ at $\lambda = 0.4$, the gain with which dReal proved the certificate, and the nominal input of Appendix B.6.
- *Simulation.* Fourth-order Runge–Kutta with a step of 0.01 s over 60 s. The input is held constant over each step.
- *Goal.* A trajectory reaches the goal when it enters $[9, 10] \times [4.5, 5.5]$ in (p_x, p_y).
- *Arrival.* The first time a trajectory enters the goal, averaged over the trajectories that reach it.
- *Clearance.* The smallest distance from any of the run's 32 trajectories to the configured obstacle.

We summarize the results in the table below:

run (outer iteration)	safe-set fraction	goal reached	mean arrival	clearance
1 (50)	0.628	32 of 32	25.2 s	0.18 m
4 (24)	0.743	32 of 32	22.6 s	0.24 m
5 (59)	0.686	32 of 32	37.7 s	0.19 m
6 (59)	0.682	32 of 32	26.6 s	0.13 m
8 (59)	0.737	32 of 32	28.5 s	0.15 m
9 (58)	0.760	27 of 32	43.4 s	0.20 m

Reading Figure 3. Figure 3 draws h_{firm} left of the guard $p_x = 5$ and h_{loose} right of it, so the switching band $4.5 \le p_x \le 5$ shows the firm-ground barrier. Gray marks positions at which $h_q(p_x, p_y, v_x, 0) < 0$ for every $v_x \in [0, 2]$, and the dashed lines are the guards $p_x = 5$ and $p_x = 4.5$. The trajectory lasts 60 s, with a dot every 5 s. The trajectory has $v_y \neq 0$, so the speeds drawn at $v_y = 0$ do not bound its v_x.

D.4 Different gains for ACC

Figure 5 reports the result with gain $\alpha(s) = 2s$ because at this gain, the two trajectories show the order of speeds similar to what Yang et al. (2024a) reported. The table below repeats the comparison at gains 0.4, 1 and 2, with one trajectory per row, started at $(p, v, d) = (2.5, 30, 90)$, under the nominal input $u = 0.12(30 - v)$ and with time step being 0.01 s. At the certified gain 0.4, the car reaches $p = 100$ m sooner with h_{ice} in both modes than with the reported pair. Between the two barrier choices, the time to reach $p = 100$ m differs by at most 0.21 s at every gain.

gain	barriers	v at $p = 10$ m	v at $p = 25$ m	v at $p = 50$ m	time to $p = 100$ m
0.4	reported pair	29.13	27.63	25.54	3.83 s
0.4	h_{ice} in both modes	29.56	28.70	27.20	3.62 s
1	reported pair	29.69	28.75	26.77	3.69 s
1	h_{ice} in both modes	29.34	28.18	26.48	3.70 s
2	reported pair	29.83	29.38	27.25	3.65 s
2	h_{ice} in both modes	29.03	27.67	26.13	3.74 s

E Prompts

The loop of the proposed framework builds each prompt from a template and fills it from the benchmark and from the run. This section prints two prompts that a run for the `Rover` task sent: the proposer prompt of the outer-loop iteration 0 and the text that the M1 operator adds at the outer-loop

iteration 1. It also prints the templates of the E1, E2 and M2 operator blocks and the two premises that the loop can send with M2. The `ACC` run sent the same templates. The BarrierBench runs sent them with added instructions for a fixed X_U. The colors mark three sources of text.

- Black text comes from the template files, including the operator blocks.
- Blue text is what the loop writes from the benchmark's configuration: the dynamics, the obstacles, the task description, the initial and operating sets and the guards. Blue also covers the sentences that the loop writes around these values, such as the instruction on how far to grow the obstacle in X_U.
- Orange text is what the loop writes from the conditions it checks and from the run: the list of conditions, the evidence on earlier attempts, the iteration counter and M2's premise.

How the listings differ from the text sent.

- The loop removes the comments in the template files before it sends a prompt, so the listings contain none.
- The listings replace non-ASCII characters with ASCII, because the listing font lacks them.
- The listings omit the line that marks the start of the model's input in the run's trace.
- In the M2 block, `$premise` marks where the loop inserts one of the two premises printed after the block.
- The premises of §E.3 are printed with line breaks. The loop sends each premise as one line.
- Three objects of the body have other names in the prompts. The *flow domain* is the body's mode domain, the *intent* is the body's task description, and a *form* is the body's structure. The listings keep these names, because the model received them.

Where to look.

- §E.1 is the proposer prompt at outer iteration 0.
  - Lines 139–145 ask the model to make h_q depend on the state variables that the input reaches.
  - Lines 147–155 are the Form budget block. The block asks for polynomial terms and forbids poles. The Syntax block allows sin, cos, tan, exp, log and sqrt, but the Form budget block allows exp, log, sqrt and tan only where no polynomial term can express the design.
  - The Form budget block gives its reason: dReal decides each condition under a time limit, and a condition it cannot decide in time stays undecided. The ranking key of §3.3.2 counts undecided conditions with the violated ones.
  - Neither the instruction on the variables that the input reaches nor the Form budget block names a term of h_q. We did not measure how these instructions change certification.
- §E.2 is what the M1 operator adds at outer iteration 1.
  - The listing holds the M1 operator block, the evidence block on the design that M1 revises, and the output schema.
  - The evidence block carries the fitted values and the solver's counterexamples. The prompt labels these values as what the solver checked, but they are the values before rounding. The block on a certified design, which the listings do not show, labels its values in the same way ("parameter values dReal certified"), and they are also the values before rounding. In this example, the block also carries a note that a counterexample no longer violates its condition under the latest fit.
  - When the evidence shows a counterexample to a switch condition, the prompt adds that the crossing point "currently escapes" the next mode's safe set. Under the latest fit, the values on the same line can contradict this.
- §E.3 gives the E1, E2 and M2 operator blocks. M2 also receives a *premise*, a short text about its parent. The loop picks one of two premises by whether the parent is certified, and the listing prints both. When M2 revises a certified parent, its premise states that there is no violation to close, while its block still asks it to analyse each violation shown; the evidence of a certified parent shows none.

E.1 PROPOSER PROMPT (ROVER, OUTER-LOOP ITERATION 0)

Role

```
You design a **Control Barrier Function (CBF)** for a known hybrid control-affine system, whose
dynamics switch by discrete mode. Propose one local barrier `h_q(x)` per mode; the modes share a
state space, an initial set and an unsafe set. Leave parameters as free symbols -- a numeric
fitter tunes them afterwards (a complete `param_init` may first be verified as written, below).
```

System

```
System (shared state/domain; per-mode dynamics and input bounds):
```
rover_terrain_hybrid: hybrid control-affine system, 2 mode(s), 4 state(s), 2 input(s).
Shared state variables (with verification domain):
* px in [0.0, 10.0]
* py in [0.0, 10.0]
* vx in [-2.0, 2.0]
* vy in [-2.0, 2.0]
Per-mode dynamics xdot = f_q(x) + g_q(x) u and input bound U_q:
- mode 'firm':
  f_firm(x) = [vx, vy, -0.05*vx, -0.05*vy]
  g_firm(x) = [0, 0, 0, 0, 1.00000000000000, 0, 0, 1.00000000000000]
  * ax in [-1.0, 1.0] (mode 'firm')
  * ay in [-1.0, 1.0] (mode 'firm')
- mode 'loose':
  f_loose(x) = [vx, vy, -0.3*vx, -0.3*vy]
  g_loose(x) = [0, 0, 0, 0, 0.600000000000000, 0, 0, 0.600000000000000]
  * ax in [-1.0, 1.0] (mode 'loose')
  * ay in [-1.0, 1.0] (mode 'loose')
```
```

Static obstacles

```
Static obstacles -- the danger geometry of this problem, listed exactly. Your shared `x_unsafe`
must COVER it, and where the block states a clearance, cover it with that clearance:
```
- obstacle 0: the disk (px - 7.0)**2 + (py - 5.0)**2 <= 2.25 (center (7.0, 5.0), radius 1.5)

IMPORTANT -- X_U must cover each obstacle WITH CLEARANCE, not exactly: the verifier checks that every
    ↪ obstacle inflated by 0.05 lies inside X_U. Write each obstacle's clause in X_U grown by at
    ↪ least 0.1 (e.g. a disk of radius r becomes radius r + 0.1). An X_U that reproduces the
    ↪ obstacle exactly WILL be reported as a violation.
```
```

Design intent

```
Design intent (the informal goal your design must realize -- your ground truth):
```
Two-terrain slip rover (crater avoidance) -- design intent

A planar rover drives across open ground toward a goal beyond a crater. You command drive
acceleration; the terrain under the wheels decides how much of that command becomes motion.

The two modes are terrains: **firm ground** near the start, **loose soil** past `px = 5`, where
the wheels slip. The rover crosses into loose soil and can come back. The two modes' flow
domains overlap on `[4.5, 5.0]` -- inside that band the mode is whichever terrain the trajectory
came from, so the two guards cannot chatter at a single boundary.

A family of local barriers, one per mode, is in general less conservative than a single barrier
that must hold under every mode's dynamics at once. Producing that family -- one `h_q` per mode,
each certified against its own mode's dynamics, with both crossings certified between them -- is
the job here.

What makes a certificate mean something

The goal lies beyond the crater, on the loose side. It is fixed by the experiment and you do not
choose it. Reaching it is a task objective, never a safety condition -- no certificate here claims
the rover arrives.

Safety on its own is trivially satisfiable by a useless rover: "never approach the crater, never
build up speed" meets every pointwise condition and means nothing. Two requirements rule that
answer out.

- The **operating set** `X_op`, listed per mode below, is ordinary operation. Your safe set must
  contain it, and the verifier checks that like any other condition.
- Containment is necessary but not sufficient: the safe set must also be **usable**. At run time
  a CBF-QP filters a fixed nominal law and can only pick inputs your `h_q` admits. A barrier
  whose shape makes that filter override the task at almost every step gives a rover that is safe
  because it barely moves -- certified and useless, and a failed design by this intent.

```
  Closed-loop rollouts measure this and report it back to you as quality evidence, never as
  proof.
```

Initial set and operating set

```
The initial set X_I is FIXED by the experiment (you do NOT choose it -- it is the task's
start region, and your barrier must contain it):
```
X_I = (px >= 0.5) & (py >= 4.5) & (vx >= -0.1) & (vy >= -0.1) & (px <= 1.5) & (py <= 5.5) & (vx <=
    ↪ 0.1) & (vy <= 0.1)
```

The operating set X_op is FIXED by the experiment -- you do NOT choose it. Every mode's safe set
must CONTAIN the X_op shown for that mode, and that containment is the verified condition
`coverage[q]` below, not a preference:
```
- X_op[firm] = (px >= 0.5) & (vx >= -0.3) & (vy >= -0.3) & (px <= 5.0) & (vx <= 0.3) & (vy <= 0.3)
- X_op[loose] = (px >= 4.5) & (vx >= -0.3) & (vy >= -0.3) & (px <= 9.5) & (vx <= 0.3) & (vy <= 0.3) &
    ↪ (49.0*(0.142857142857143*px - 1)**2 + 25.0*(0.2*py - 1)**2 >= 6.25)
```
```

Task

```
Task -- propose, consistent with the intent:
- `x_unsafe`: a predicate for the unsafe set X_U -- states outside every mode's safe set,
              shared across modes.
- for EACH mode listed above, an `h`:
  - `h`     : a closed-form local barrier `h_q`; the intended per-mode safe set is
              `C_q = {x : h_q(x) >= 0}`.
  (You do NOT propose a controller: your `h_q` must guarantee that an admissible input
   EXISTS everywhere on `C_q`. The conditions below state this exactly.)
  - `params`: the free parameter names used in this mode's `h`. A name may be reused across
              modes; each mode is fitted independently either way.
  - `param_init` (optional): initial values as `{"<name>": <number>}`. A COMPLETE set (every
              parameter of every mode) may be verified AS AUTHORED before any fitting, costing
              one inner iteration if it fails. Complete means EVERY mode: one missing value
              anywhere turns that check off for all modes at once, not just its own. Whether the
              set is complete or partial, the values you give also seed the fit -- they become the
              optimizer's FIRST start, and its remaining starts are random draws either way.
              Give values you would defend or none, and on a revision carry over the fitted
              values that worked. This does not relax the no-hard-coding rule below.
```

Mode transitions

```
**Mode transitions -- these shape each `h_q`:**
- 'firm' -> 'loose': the guard region is px >= 5.0 (the system runs under 'loose''s dynamics there),
    ↪ and the transition FIRES as the state enters it, i.e. at (px >= 5.0) & (px <= 5.25) & (vx >=
    ↪ 0). The switch condition below is checked at those entry states only -- a state deep inside
    ↪ the guard region is never a switching state, because the trajectory would already have
    ↪ switched on entry.
- 'loose' -> 'firm': the guard region is px <= 4.5 (the system runs under 'firm''s dynamics there),
    ↪ and the transition FIRES as the state enters it, i.e. at (px >= 4.25) & (px <= 4.5) & (vx <=
    ↪ 0). The switch condition below is checked at those entry states only -- a state deep inside
    ↪ the guard region is never a switching state, because the trajectory would already have
    ↪ switched on entry.
```

Certified conditions

```
**Certified conditions -- the exact checks your proposal must pass.** dReal decides them
mechanically; you never author them, but design each `h_q` against this specification:
IMPORTANT -- WHERE each mode is checked: every PER-MODE condition below is quantified over the states
    ↪ that mode can BE at, not the whole domain: a mode leaves as soon as it enters one of its
    ↪ outgoing guards. Flow domains: mode 'firm': px <= 5.0; mode 'loose': px >= 4.5. A mode with
    ↪ no outgoing transition is absorbing and IS checked on the whole domain; `obstacle_coverage[i]`
    ↪ is not per-mode and is quantified over the whole domain.
For EVERY mode q (each checked by dReal as a forall-x condition; a single violating state anywhere in
    ↪ ITS flow domain rejects the candidate) -- with ONE exception, `init`, whose scope is stated on
    ↪ its own line:
- init[firm] : h_q(x) >= 0 for ALL x in X_I -- checked for the INITIAL mode `firm` ONLY, NOT for every
    ↪ mode: a non-initial mode is reached only through a transition. This is a statement about
    ↪ `init` alone -- if a non-initial mode's own X_op happens to contain X_I, its coverage[q] below
    ↪ still obliges h_q there.
- unsafe[q]    : h_q(x) < 0 for ALL x in X_U.
- invariance[q]: sup over u in mode q's OWN input box U_q of [ Lf(h_q) + Lg(h_q)*u + alpha(h_q) ] >=
    ↪ 0, for ALL x in (domain AND the safe set {h_q >= 0}). In words: at every state your safe set
    ↪ contains, SOME admissible input must hold the state in it. A QP picks that input at run time;
    ↪ your h_q must make one EXIST. Here alpha(h_q) = 0.4*h_q (FIXED by config; you cannot change
    ↪ or fit it).
```

```
- switch[q->q']: every domain state at which the q->q' transition FIRES (the guard's entry states -- see the transitions section above, which gives the exact set) with h_q(x) >= 0 must also have h_q'(x) >= 0.
- obstacle_coverage[i]: X_U contains every declared obstacle region (so unsafe[q] actually excludes the obstacles).
- coverage[q]  : h_q(x) >= 0 for ALL x in mode q's OWN operating set X_op[q] (listed above) -- each mode's safe set must CONTAIN every state that mode operates at. The operating sets DIFFER per mode and are a fixed requirement, not something you may shrink: a barrier admitting only one extreme velocity is rejected here even though it satisfies every other condition.
```

Hard constraints

```
Hard constraints (do not violate) -- PER MODE:
- Symbolic, closed-form only: NO neural networks or weight blobs. Use only the state
  variables, that mode's parameters, numeric constants and sin, cos, tan, exp, log, sqrt.
- `x_unsafe` uses only state variables, constants, comparisons and And/Or/Not. It must have
  positive volume inside the verification domain. It must cover every obstacle listed above
  (this is checked: `obstacle_coverage[i]`).
- LEAVE UNKNOWN MAGNITUDES AS FREE PARAMETERS: do NOT hard-code the safe region's size, the
  margin or the scale. A symbol in `h` that is not a state variable IS a free parameter -- that
  is what makes it one -- and listing it in `params` keeps your own declaration readable. EXCEPTION -- the constants this
  problem STATEMENT lists above (the danger regions in the obstacle block, the guard thresholds,
  the domain bounds) SHOULD be literal numbers where your structure needs them. A disclosed fact is
  not a fitted quantity: declare one as a parameter and the fitter will move it whenever another
  condition's loss profits, so your safe set's boundary silently stops following the actual unsafe
  set. Reuse those NUMBERS, not the listed expression as a whole. This EXCEPTION covers them and
  nothing else: the SCALE of a term, its weights and its margins stay free parameters.
- WRITE EACH COEFFICIENT AS A SINGLE FREE-SIGNED PARAMETER: write `a*term`, not `a**2*term`,
  `-b**2*(term)` or "a positive combination of ...". A squared coefficient is always >= 0, so it
  cannot take the negative value some conditions need; `init` and `unsafe` above fix the
  orientation.
- THE INPUT MUST BE ABLE TO MOVE `h_q`. Where `L_g h_q(x) = 0` the input drops
  out of the invariance condition entirely: what is left is `L_f h_q + alpha(h_q) >= 0`, which
  the drift and the barrier's own value must meet with no help from `u`. Fitting cannot add
  what the form lacks: the fitter moves the free symbols your `h_q` already contains, it cannot
  introduce a dependence on a variable your form does not mention. Read the per-mode `f_q`/`g_q` above to
  see which variables the input reaches directly and which it reaches only through others, and
  shape `h_q` so it depends on the ones it reaches.
```

Form budget

```
Form budget -- dReal decides your candidate under a per-condition time budget, and a form it
cannot decide in time is rejected exactly like a wrong one. This applies to every proposal,
including a design built from scratch:
- Be very careful - don't make it more complex than needed.
- Prefer POLYNOMIAL terms in the state variables; use angle variables only through the
  `sin`/`cos` forms already in the dynamics.
- Do NOT use `exp`, `log`, `sqrt` or `tan` unless no polynomial shape can express what you need;
  NEVER nest them or multiply them into a polynomial factor.
- NO denominators that can reach zero anywhere in the verification domain (no poles).
```

Syntax

```
Syntax for the JSON string values -- SymPy parses them strictly and rejects invalid ones:
- Use Python-style operators: `+ - * / **` and comparisons `<= >= < >`.
- Build predicates with `And(...)`, `Or(...)`, `Not(...)` -- NOT the words `and`/`or` -- and
  write each bound as its own comparison, e.g.
  `"And(theta <= 0.1, theta >= -0.1, theta_dot <= 0.5, theta_dot >= -0.5)"`.
- Do NOT use `Abs`, `min`, `max` or piecewise terms; write a magnitude bound as two
  comparisons, e.g. `"Or(theta >= 1.5, theta <= -1.5)"` for `Abs(theta) >= 1.5`.
- Use only these functions anywhere in your expressions: `sin, cos, tan, exp, log, sqrt`.
- Refer to state variables and parameters by exactly the names given above.
```

Operator block and feedback

```
**Revision operator for THIS iteration: E1 -- explore a fundamentally different design.**
No design has been evaluated yet in this run: there is no previous design to patch and no family already tried. Synthesize an initial design whose per-mode barrier structure follows from the intent, the dynamics and the certified conditions above, and say in `design_rationale` what hypothesis it is testing.


Feedback: none -- this is the first attempt.
```

Output

````
Output -- think first, then emit ONE JSON object as the last thing in your reply, with no other
JSON object before it: the extractor takes the first one it can parse, so a draft would be read
as your proposal. Use exactly these keys and SymPy-parseable strings (iteration 0 of
60, numbered from 0 like every `iteration N` and `outer N` label elsewhere in this
prompt, so the first is 0 and the last is one less than the total). `modes` has one entry per mode
    ↪ above, keyed by its name:
```
{
  "design_rationale": "<= 3 sentences: what geometric idea each mode's h encodes and why it should
      ↪ close the currently-failing conditions",
  "x_unsafe": "<predicate over state vars>",
  "modes": {
    "<mode name>": {
      "params": ["<name>", "..."],
      "param_init": {"<name>": 1.0},
      "h": "<expression in state vars and this mode's params>"
    }
  }
}
```
(`param_init` may cover a subset of `params`. `design_rationale` is recorded with the attempt,
so later revisions can see what each tried structure was FOR.)
````

E.2 What M1 appends (Rover, outer-loop iteration 1)

```
**Revision operator for THIS iteration: M1 -- analyze the evidence, then revise the STRUCTURE.**
The evidence in the feedback below (verifier counterexamples and violation margins) describes how
the retained design you are revising fails. FIRST, in your step-by-step thinking, write a short
    ↪ analysis:
for each failing condition, state WHY that design's structure cannot satisfy it -- which term is
implicated, or which coupling is missing. THEN revise the structure accordingly: change, add, or
remove the implicated term(s) while PRESERVING every term the evidence does not implicate. Base
your revision on the design whose evidence is shown; if a more recent attempt is noted there as
"not retained", do not build on it.


Feedback on the design you are revising (do not repeat it):
Previous X_I: (px >= 0.5) & (py >= 4.5) & (vx >= -0.1) & (vy >= -0.1) & (px <= 1.5) & (py <= 5.5) &
    ↪ (vx <= 0.1) & (vy <= 0.1)
Previous X_U: 49.0*(0.142857142857143*px - 1)**2 + 25.0*(0.2*py - 1)**2 <= 2.56
Previous stated intent: "Each mode uses a barrier that is positive on its operating rectangle and on
    ↪ low-speed states, negative inside the crater (obstacle) and decays with velocity to give Lg h
    ↪ != 0 so the QP can use the acceleration inputs. The barrier is a sum of an inflated-obstacle
    ↪ distance term and quadratic confinement terms in px, vx, vy; the confinement terms are
    ↪ weighted so that the obstacle region is always driven negative while the operating set stays
    ↪ positive. The same functional form is used for both terrains, allowing the switch condition
    ↪ to hold automatically at the guard."
Previous per-mode forms:
- mode 'firm': h = -r0**2 + w1*(5.0 - px)*(px - 0.5) + w2*(0.3 - vx)*(vx + 0.3) + w3*(0.3 - vy)*(vy +
    ↪ 0.3) + 49.0*(0.142857142857143*px - 1)**2 + 25.0*(0.2*py - 1)**2 (fit loss 2.026)
  parameter values (what dReal actually checked; carry working values forward via param_init):
    ↪ r0=-2.036786143, w1=0.3567568915, w2=4.374213496, w3=4.335395423
  fit-side residual at the optimizer's OWN optimum (worst violation per condition on the fit's
    ↪ samples; positive = the fit finished with this condition still violated): coverage +0.148,
    ↪ switch[firm->loose]:src +0.384 -- the optimizer converged with these conditions in TENSION:
    ↪ no parameter values it found satisfy them simultaneously within THIS structure, so a
    ↪ parameter-level edit is unlikely to close them; change the structure where the evidence
    ↪ points. (Sampled fit diagnosis, not a certificate.)
- mode 'loose': h = -r0**2 + w1*(9.5 - px)*(px - 4.5) + w2*(0.3 - vx)*(vx + 0.3) + w3*(0.3 - vy)*(vy +
    ↪ 0.3) + 49.0*(0.142857142857143*px - 1)**2 + 25.0*(0.2*py - 1)**2 (fit loss 1.233)
  parameter values (what dReal actually checked; carry working values forward via param_init):
    ↪ r0=-2.134073631, w1=-9.69154313e-05, w2=4.32953531, w3=4.3096002
- (signal: SMT verifier) dReal found a counterexample to the 'coverage[firm]' condition at the state
    ↪ (px=4.996, py=4.862, vx=0.2926, vy=-0.2926). At that state the 'coverage[firm]' requirement
    ↪ is violated. Under the LATEST fitted parameters, h_firm = -0.06789 there (the coverage
    ↪ condition requires h_firm >= 0 on the whole operating set X_op -- your safe set must CONTAIN
    ↪ this state).
- (signal: SMT verifier) dReal found a counterexample to the 'invariance[loose]' condition at the
    ↪ state (px=8.496, py=3.759, vx=-8.315e-05, vy=2.036e-05). At that state the
    ↪ 'invariance[loose]' requirement is violated. Under the LATEST fitted parameters, sup_u
    ↪ [hdot(u) + alpha(h)] = +0.0002042 and h_loose = -8.465e-05 there (invariance requires sup_u
    ↪ [hdot(u) + alpha(h)] >= 0 at EVERY state of the safe set {h >= 0} that lies in loose's flow
    ↪ domain). NOTE: clean at THIS POINT under the latest fit -- see the note below.
- (signal: SMT verifier) dReal found a counterexample to the 'switch[firm->loose]' condition at the
    ↪ state (px=5.028, py=3.818, vx=0.5, vy=-0.4246). At that state the 'switch[firm->loose]'
    ↪ requirement is violated. Under the LATEST fitted parameters, h_firm = -0.0001433 and h_loose
    ↪ = -0.3513 there (a crossing state safe in 'firm' must also have h_loose >= 0). NOTE: clean at
    ↪ THIS POINT under the latest fit -- see the note below.
- (signal: SMT verifier) the final verify DECIDED 'invariance[firm]', 'switch[loose->firm]',
    ↪ 'unsafe[loose]' and none of them is violated under this attempt's final fit. An earlier inner
    ↪ iteration of this attempt had found a counterexample to each.
Revise the per-mode h_q (and X_U if it misrepresents the stated goal) so that EVERY counterexample
    ↪ listed above is resolved while staying faithful to the stated goal. If a condition name has
    ↪ the form 'switch[q->q_prime]', the crossing point from mode q into mode q_prime at that state
```

```
      ↪ currently escapes q_prime's safe set. IMPORTANT -- preserve what is not implicated: the
      ↪ signals above identify SPECIFIC defects. Revise the specific structure they implicate; do NOT
      ↪ delete or rewrite terms the signals do not name.
NOTES ON THE EVIDENCE ABOVE -- stated once, and they apply to every line that carries them:
- Points marked clean under the latest fit. NOTE: at THIS POINT the requirement is NO LONGER violated
      ↪ under the LATEST fitted parameters (the sentence above reports the state as dReal found it).
      ↪ That is a statement about one state, not about the condition: the witness may pre-date the
      ↪ latest fit within this iteration, and dReal returns SOME state satisfying the delta-weakened
      ↪ negation rather than the worst one, so a point that reads clean here can still sit within
      ↪ delta of a live violation. Do not drop this condition from your revision on the strength of
      ↪ this note.
- (signal: SMT verifier, delta-margin) 1 condition(s) of THIS design are delta-BLOCKED against dReal's
      ↪ delta = 0.001 (dReal returned a delta-satisfying witness for each, and the artifact screen
      ↪ found the worst violation it could reach (near the witness, or inside the condition's own
      ↪ region when the witness lies outside it) below 20.delta = 0.02, which is the screen's test
      ↪ and not a bound on the values below) -- enough on its own to block certification,
      ↪ independently of any counterexample listed above -- namely invariance[loose] (holds by
      ↪ 0.000156). A repaired NOTE on a counterexample line, where present, is a statement about that
      ↪ single point, not about these conditions.

Output -- think first, then emit ONE JSON object as the last thing in your reply, with no other
JSON object before it: the extractor takes the first one it can parse, so a draft would be read
as your proposal. Use exactly these keys and SymPy-parseable strings (iteration 1 of
60, numbered from 0 like every `iteration N` and `outer N` label elsewhere in this
prompt, so the first is 0 and the last is one less than the total). `modes` has one entry per mode
      ↪ above, keyed by its name:
```
```
{
  "design_rationale": "<= 3 sentences: what geometric idea each mode's h encodes and why it should
     ↪ close the currently-failing conditions",
  "x_unsafe": "<predicate over state vars>",
  "modes": {
    "<mode name>": {
      "params": ["<name>", "..."],
      "param_init": {"<name>": 1.0},
      "h": "<expression in state vars and this mode's params>"
    }
  }
}
```
(`param_init` may cover a subset of `params`. `design_rationale` is recorded with the attempt,
so later revisions can see what each tried structure was FOR.)
```

### E.3 Operator blocks E1, E2, M2 and the M2 premises

**E1**

```
**Revision operator for THIS iteration: E1 -- explore a fundamentally different design.**
This iteration was chosen by the operator rotation, not by a diagnosis of the run. Your task is
NOT to patch the previous design.

FIRST, read whatever record of the run is below this block -- a RETAINED design, a STRUCTURE
LEDGER, both, or neither. Where a STRUCTURE LEDGER is present it ranks this run's distinct
structures best-first and states its own scope on its first line: read that line, because where
it says the list was cut, the structures below the cut are not listed anywhere in this prompt.
Work out WHY those families stop where they stop before you write anything: which condition the
strongest uncertified entries fail on, and whether they fail it the same way every time or from both
      ↪ sides.

THEN synthesize a NEW design whose per-mode barrier STRUCTURE differs fundamentally from every
family that record shows, and choose that difference so that it targets the blocking condition
you identified, instead of being different for its own sake. Do not carry over a failing structure
with cosmetic changes, and do not merely re-tune coefficients. Name in `design_rationale` which
entry of that record and which blocking condition your hypothesis is aimed at.
```

E2

```
**Revision operator for THIS iteration: E2 -- generalize from multiple parent designs.**
The designs listed in the feedback below are the parents for this crossover. The block that
lists them states in its own header where they came from and what is known about each one -- read
that header and the rows under it before you assume anything about their status.
Identify the structural patterns they share -- which state variables each mode's `h_q` couples,
and how -- and construct a new design that generalizes those shared patterns: keep what the
parents agree on, and where they differ, choose the variant that gives LARGER (less
conservative) safe sets while remaining certifiable. This is a crossover: the offspring should
be recognizably built from the parents' common structure, not a fresh invention.

Agreement is evidence, not proof. Where the parents share a term that the verifier has
repeatedly faulted, the shared term is a shared DEFECT -- say so and replace it rather than
preserving it for the sake of the crossover.
```
```

M2

```
**Revision operator for THIS iteration: M2 -- scoped numeric edit. STRUCTURE IS FROZEN.**
$premise You must NOT change any mode's `h_q` STRUCTURE in
this revision: re-emit the SAME expressions -- same terms, same variable couplings, same
functional forms.

FIRST, in your step-by-step thinking, write a short PARAMETER ANALYSIS before you change
anything:
- name the numeric quantities in the retained design that actually shape where its boundary
  sits, and say what each one does;
- for each violation shown in the evidence, say WHICH of those quantities its margin is a
  function of, and in which direction that quantity must move to close it;
- say what moving it costs -- which other condition's margin shrinks as a result.
THEN make the edit. Keep that analysis in your thinking; `design_rationale` gets ONE sentence
naming the quantity you moved and the margin you aimed it at, so the next iteration is revising
against a stated reason without spending the rationale budget on the derivation.

IMPORTANT -- which numbers you can actually move. A numeric fitter re-optimizes every FREE
parameter after your proposal, so changing a free parameter's `param_init` is usually absorbed:
the fitter re-converges to the same basin. It is still worth stating, for a reason that holds
on this run. If `param_init` is COMPLETE -- every declared parameter of EVERY mode, not just one --
your values are verified AS AUTHORED, with no fit at all, and if they hold that IS the
certificate.
What the fitter CANNOT undo is a number that is not a declared parameter. Your two levers are:
1. **Change a numeric literal that is BAKED into an expression** (a fixed constant, weight, or
   offset that is not a declared free parameter) -- the fitter cannot move these; you can.
2. **Promote a baked numeric literal to a NEW free parameter** so the fitter can move it --
   name it, add it to that mode's parameter list, and optionally seed it via `param_init`.
EXCEPTION -- literals that encode DISCLOSED SPEC GEOMETRY (the numbers that define the unsafe
set, the guard threshold, domain bounds) are FACTS of the problem, not tuning knobs: do not change
them and do not promote them to free parameters -- changing one, or making it fittable, lets your
safe set drift away from the real unsafe set with no error raised anywhere. The levers apply to the
    ↪ SHAPING constants (weights, margins, offsets)
only.
Choose the literal(s) whose movement closes each violation margin shown in the feedback
without reopening another. If every relevant quantity is already a free parameter, use the
    ↪ `param_init` lever
below -- do NOT fall back to a structural rewrite, which belongs to another operator and
fires on its own turn.
If the design contains NO movable baked literal AT ALL -- every shaping constant is already a
free parameter and the only literals left are the protected spec-geometry facts -- your lever is
`param_init` itself. Re-emit the same expressions with a COMPLETE `param_init` (every declared
parameter of every mode): start from the values the evidence shows for the design you are
revising, where it shows them, and move the parameter(s) your analysis found the blocking margins
depend on, in the direction it found. Complete means every declared parameter of EVERY mode --
leave one value out anywhere and the as-authored check is skipped for all modes, not just that
one. A complete set is verified AS AUTHORED before any fit, so the fitter cannot undo it on that
check; if it fails, your values still become the fit's FIRST start, though its remaining starts
are random draws. Re-emit the
design with no change at all only if your analysis finds no parameter that moves a blocking
margin, and then SAY SO in `design_rationale` (e.g. "M2 no-op: no parameter moves a blocking
margin") -- it is the only record of WHY this iteration produced nothing.
```

M2 premise — certified parent

```
This iteration is a scoped NUMERIC pass over a design that is already CERTIFIED. It misses by
nothing: there is no violation to close. What this pass is for is QUALITY -- a less conservative
parameterization of the SAME structure.
```

M2 premise — uncertified parent

```
M2's turn came up in the operator cycle over a retained design that is not certified. Its
structure is frozen this iteration. What it still fails is in the evidence below. If no numeric
edit can close it, say so in `design_rationale`.
```