BARRIER-LYAPUNOV-AGENT: TRAINING AND VERIFYING SAFE AND STABLE NEURAL NETWORK CONTROLLERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Certifying learned controllers requires coordinating certificate synthesis, formal verification, and controller refinement, a process that often involves substantial manual effort. We propose Barrier–Lyapunov Agent, a tool-augmented large language model framework that automates this process for systems with known dynamics or unknown dynamics. The framework follows three stages: objective understanding, Lyapunov-guided stabilization, and barrier-guided safety certification. Given a task specification and an initial controller, the agent generates structured tool arguments, constructs generalized Lyapunov and control barrier functions, and uses verification feedback to guide certificate updates and controller refinement. Upon successful verification, the framework returns the refined controller, corresponding certificates, and a report specifying the verified properties and their domains. Certification claims are determined by formal verification tools rather than by the language model. We evaluate the framework against two baselines using the same computational tools: a human-operated workflow and an agent without explicit workflow instructions. To assess the contribution of the tool suite, we compare the fully equipped Barrier–Lyapunov Agent with an otherwise identical AI agent without access to these tools. Our results show that controllers trained by the proposed agent achieve better control performance and larger formally verified regions of stability and safety than those produced by the baselines.

1 INTRODUCTION

Safety and liveness are two fundamental objectives that a controller should simultaneously satisfy. Intuitively, safety requires that "bad" things do not happen while liveness requires that "good" things eventually happen Ames et al. (2019). For a dynamical system, safety requires trajectories to avoid specified unsafe states, whereas liveness concerns progress toward a desired objective. In stabilization tasks, Lyapunov stability requires trajectories initialized sufficiently close to a target equilibrium to remain close to it; asymptotic stability additionally requires convergence to that equilibrium (Goebel et al., 2012). Stability alone does not imply safety: a stabilizing trajectory can intersect the unsafe set. Conversely, a safe controller need not stabilize the target. We therefore verify both properties for the same closed-loop controller.

Establishing these guarantees is particularly challenging for neural network controllers. Although their expressive representations enable effective control of nonlinear systems, training performance alone does not establish safety or stability throughout a continuous set of initial states. Finite simulation tests may miss rare violations, and successful finite-horizon rollouts do not establish infinite-horizon guarantees. Furthermore, composing a neural policy with nonlinear system dynamics and output clipping produces closed-loop behaviors that can be difficult to analyze. Certificate-based methods address this challenge by constructing functions whose verified inequalities imply desired system properties: Lyapunov certificates establish stability, while barrier certificates establish safety (Dawson et al., 2023). However, constructing suitable certificates and verifying their conditions remain coupled, computationally demanding problems.

Recent advances have made neural certificate synthesis and verification increasingly practical. Generalized Lyapunov functions accommodate weighted multi-step decrease, extending the available

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.

certificate constructions beyond one-step conditions (Long et al., 2026). Neural controller and Lyapunov-function training can also be combined with formal verification to obtain certified regions of attraction (Yang et al., 2024). For safety, counterexample-guided barrier synthesis iterates between candidate learning and formal verification (Peruffo et al., 2021; Edwards et al., 2024). In practice, these methods still require substantial manual iteration. A researcher must choose the certificate architecture and verification configuration, inspect failed verification attempts, and decide whether to retrain the certificate or controller. A controller update can also invalidate previously verified certificates, requiring the stability and safety analyses to be revisited.

Recent advances in LLM agents have enabled automated controller design, tuning, and policy development. ControlAgent coordinates specialized agents for classical controller design (Guo et al., 2024), while Crucible automates control-algorithm tuning (Jia et al., 2025). Language to Rewards and DrEureka automate reward construction and sim-to-real policy development (Yu et al., 2023; Ma et al., 2024b). However, performance-driven design and empirical evaluation alone do not establish formal stability and safety guarantees for neural controllers. BarrierBench advances agent-based safety certification through barrier synthesis and SMT verification (Taheri et al., 2026), but coordinating both stability and safety certification with neural-controller refinement remains underexplored. We use the verifier's output to decide whether the next iteration should modify the certificate or the controller.

We introduce **Barrier–Lyapunov Agent**, an autonomous agent for neural-controller certification and refinement. To the best of our knowledge, it is the first LLM-based agent framework to integrate generalized Lyapunov and barrier certificates for joint stability and safety certification. The agent iterates between certificate synthesis, verification, and controller updates. Its output consists of the final controller and certificates together with the corresponding verification results.

Our contributions are threefold:

- We introduce an agent workflow that coordinates generalized Lyapunov and barrier methods to refine neural controllers and seek regions with jointly verified stability and safety guarantees.
- We adapt established control and verification implementations into seven agent-callable tools, with structured interfaces for synthesis, verification, refinement, and evaluation. Verification feedback guides subsequent actions, and certification results remain associated with the controller and specifications to which they apply.
- We separately evaluate workflow guidance and tool access while reporting formal certification, controller performance, certified-region size, and computational cost. Although none of the AI agents yet matches the human-certified method, the experimental results are encouraging and suggest that further investment in controller-certification agents is warranted.

2 Related Work

Lyapunov construction and stability certification. Classical Lyapunov analysis uses analytically constructed functions, such as quadratic or energy-based candidates, whose selection typically requires system-specific insight (Goebel et al., 2012). Sum-of-squares methods automate certificate construction within structured function classes, with expressiveness and computational cost depending on the chosen representation (Papachristodoulou & Prajna, 2002). Verification-aware neural control combines empirical falsification with bound-propagation-based verification to certify regions of attraction (Yang et al., 2024). Generalized Lyapunov methods accommodate weighted multi-step decrease and support both certification of reinforcement-learning policies and joint controller–certificate training (Long et al., 2026). Despite these advances, certificate learning does not guarantee verification success, and stability alone does not establish avoidance of unsafe states.

Barrier construction and safety certification. Control barrier function methods can use analytically designed certificates to combine safety constraints with Lyapunov objectives through online quadratic programs, subject to appropriate certificate and feasibility conditions (Ames et al., 2017). Computational barrier-certificate methods establish safety without explicitly computing

reachable sets and enable convex certificate construction for polynomial systems (Prajna et al., 2007). Counterexample-guided neural barrier synthesis alternates between candidate learning and formal verification to construct safety certificates (Peruffo et al., 2021). FOSSIL 2.0 extends this approach to discrete-time systems, controller synthesis, and combined specifications such as stability while avoiding unsafe states (Edwards et al., 2024). These methods reduce manual certificate construction, but their success remains sensitive to certificate representations, training configurations, and verification complexity. Our framework builds on these foundations by using an LLM agent to coordinate certificate synthesis, formal verification, and controller refinement for joint stability and safety certification.

LLM agents for policy development and evaluation. ReAct interleaves reasoning and actions, enabling agents to incorporate environmental feedback into subsequent decisions (Yao et al., 2023). Eureka generates executable reward functions and iteratively improves them using reinforcement-learning feedback (Ma et al., 2024a). Text2Reward translates natural-language objectives into dense reward programs and supports iterative refinement through human feedback (Xie et al., 2024). ManipEvalAgent adaptively constructs simulation tasks and evaluation tools to assess robotic manipulation policies according to user instructions (Chen et al., 2026). These approaches demonstrate how language models can coordinate policy development and evaluation, but their feedback does not establish formal stability and safety guarantees for the resulting closed-loop system. Our framework extends feedback-driven tool use to controller certification and refinement, using formal verification results to guide the search for jointly certified stability and safety regions.

3 Problem Statement and Preliminaries

3.1 Problem Statement

Consider the discrete-time closed-loop system

$$\mathbf{x}_{k+1} = F_\pi(\mathbf{x}_k) := \mathbf{f}\big(\mathbf{x}_k, \mathrm{clip}\big(\pi(o(\mathbf{x}_k))\big)\big)\,, \tag{1}$$

where $\mathbf{x}_k \in \mathbb{R}^n$ is the physical state, $o(\mathbf{x}_k)$ is the controller observation, $\mathbf{f}$ is the known system dynamics, and π is a neural network controller. The clipping operator enforces the prescribed actuator limits.

For each task, we specify a target equilibrium $\mathbf{x}^\star$ and require $F_\pi(\mathbf{x}^\star) = \mathbf{x}^\star$. Let $\mathcal{X}_0$ denote the prescribed initial set and $\mathcal{X}_u$ the unsafe set, with $\mathcal{X}_0 \cap \mathcal{X}_u = \emptyset$ and $\mathbf{x}^\star \notin \mathcal{X}_u$.

Our objective is to refine an initial neural controller and construct certificates establishing

$$\mathbf{x}_0 \in \mathcal{X}_0 \quad \Longrightarrow \quad \lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}^\star, \qquad \mathbf{x}_k \notin \mathcal{X}_u \quad \forall k \geq 0. \tag{2}$$

The initial set $\mathcal{X}_0$ is fixed throughout the task: methods may not remove difficult initial states or modify its distribution. Full task certification requires formal stability and safety guarantees covering every state in $\mathcal{X}_0$ under the same final controller and system model.

Given the task specification and initial controller, the proposed agent selects and configures synthesis, verification, and refinement tools using feedback from previous attempts. Stability is established using Lyapunov certificates (Goebel et al., 2012) and the generalized multi-step construction of Long et al. (2026), while safety is established using barrier certificates (Ames et al., 2019).

3.2 Stability and Lyapunov Certificates

Definition 1 (Lyapunov Stability)**.** *An equilibrium $\mathbf{x}^\star$ is Lyapunov stable if, for every $\varepsilon > 0$, there exists $\delta > 0$ such that*

$$\|\mathbf{x}_0 - \mathbf{x}^\star\| < \delta \quad \Longrightarrow \quad \|\mathbf{x}_k - \mathbf{x}^\star\| < \varepsilon \quad \forall k \geq 0.$$

It is asymptotically stable if it is also attractive, i.e., trajectories initialized in a neighborhood of the equilibrium converge to $\mathbf{x}^\star$ (Goebel et al., 2012).

Definition 2 (Generalized Lyapunov Function)**.** *A continuous function $V : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ is a generalized Lyapunov function on a neighborhood $\mathcal{N}$ of $\mathbf{x}^\star$ if $V(\mathbf{x}^\star) = 0$, $V(\mathbf{x}) > 0$ for $\mathbf{x} \in \mathcal{N} \setminus \{\mathbf{x}^\star\}$,*

and there exist an integer $M \geq 1$ and nonnegative weights $\sigma_i(\mathbf{x})$ satisfying

$$\frac{1}{M}\sum_{i=1}^{M}\sigma_i(\mathbf{x}) \geq 1, \qquad \frac{1}{M}\sum_{i=1}^{M}\sigma_i(\mathbf{x})V(\mathbf{x}_i) < V(\mathbf{x}), \tag{3}$$

for every $\mathbf{x} \in \mathcal{N} \setminus \{\mathbf{x}^\star\}$, where $\mathbf{x}_0 = \mathbf{x}$ and $\mathbf{x}_{i+1} = F_\pi(\mathbf{x}_i)$. This condition permits temporary increases in V over individual time steps (Long et al., 2026).

Under the regularity and trajectory-containment assumptions of the generalized Lyapunov stability result (Long et al., 2026), such a certificate establishes asymptotic stability of the target equilibrium.

Let $\mathcal{X}_v$ denote a bounded verification region and define the Lyapunov sublevel set

$$\mathcal{S}_V := \{\mathbf{x} \in \mathcal{X}_v : V(\mathbf{x}) \leq \rho\}, \qquad \rho > 0. \tag{4}$$

We report the certified portion of the prescribed initial set as

$$\mathcal{S} := \mathcal{X}_0 \cap \mathcal{S}_V. \tag{5}$$

Thus, $\mathcal{X}_0$ is never changed; instead, $\mathrm{vol}(\mathcal{S})$ measures how much of the fixed initial set lies inside the verified Lyapunov region. Full stability coverage corresponds to $\mathcal{S} = \mathcal{X}_0$.

We call $\mathcal{S}_V$ stability-certified only after the applicable Lyapunov conditions and trajectory-containment requirements establish convergence from every state in its declared scope. The generalized multi-step decrease condition alone does not imply that $\mathcal{S}_V$ is forward invariant.

3.3 Safety and Barrier Certificates

Following the interpretation of safety through forward invariance (Ames et al., 2019), we use the following discrete-time definitions.

Definition 3 (Safety). *The closed-loop system is safe from a set $\mathcal{A}$ with respect to $\mathcal{X}_u$ if*

$$\mathbf{x}_0 \in \mathcal{A} \quad \Longrightarrow \quad \mathbf{x}_k \notin \mathcal{X}_u, \qquad \forall k \geq 0. \tag{6}$$

A set $\mathcal{C}$ is forward invariant if $\mathbf{x}_0 \in \mathcal{C}$ implies $\mathbf{x}_k \in \mathcal{C}$ for all $k \geq 0$. Therefore, safety from $\mathcal{A}$ follows if $\mathcal{A} \subseteq \mathcal{C}$, $\mathcal{C} \cap \mathcal{X}_u = \emptyset$, and $\mathcal{C}$ is forward invariant.

Definition 4 (Discrete-Time Control Barrier Function). *Let $B : \mathbb{R}^n \to \mathbb{R}$ be continuous and define*

$$\mathcal{C} := \{\mathbf{x} \in \mathbb{R}^n : B(\mathbf{x}) \leq 0\}. \tag{7}$$

A discrete-time control barrier function satisfies

$$\forall \mathbf{x} \in \mathcal{C}, \quad \exists \mathbf{u} \text{ within the actuator limits} \quad \text{such that} \quad B\big(\mathbf{f}(\mathbf{x}, \mathbf{u})\big) \leq 0.$$

Existence of a suitable control input does not certify a particular neural controller. We therefore verify the policy-specific condition

$$B\big(F_\pi(\mathbf{x})\big) \leq 0, \qquad \forall \mathbf{x} \in \mathcal{C}, \tag{8}$$

together with initial-set inclusion and unsafe-set separation.

We construct neural barrier candidates by adapting counterexample-guided synthesis (Peruffo et al., 2021) to the discrete-time closed-loop system. Formal barrier verification uses the SMT-based infrastructure of FOSSIL 2.0 (Edwards et al., 2024).

Joint certification. Full task certification requires the final controller and certificates to satisfy

$$\mathcal{X}_0 \subseteq \mathcal{S}_V \cap \mathcal{C}, \qquad \mathcal{C} \cap \mathcal{X}_u = \emptyset, \tag{9}$$

where $\mathcal{S}_V$ is stability-certified and $\mathcal{C}$ is verified to be forward invariant under the same controller and dynamics. Equivalently, full Lyapunov coverage requires $\mathcal{S} = \mathcal{X}_0$. If the verified Lyapunov region covers only part of $\mathcal{X}_0$, $\mathcal{S}$ is retained as a partial certification result but does not constitute full task certification.

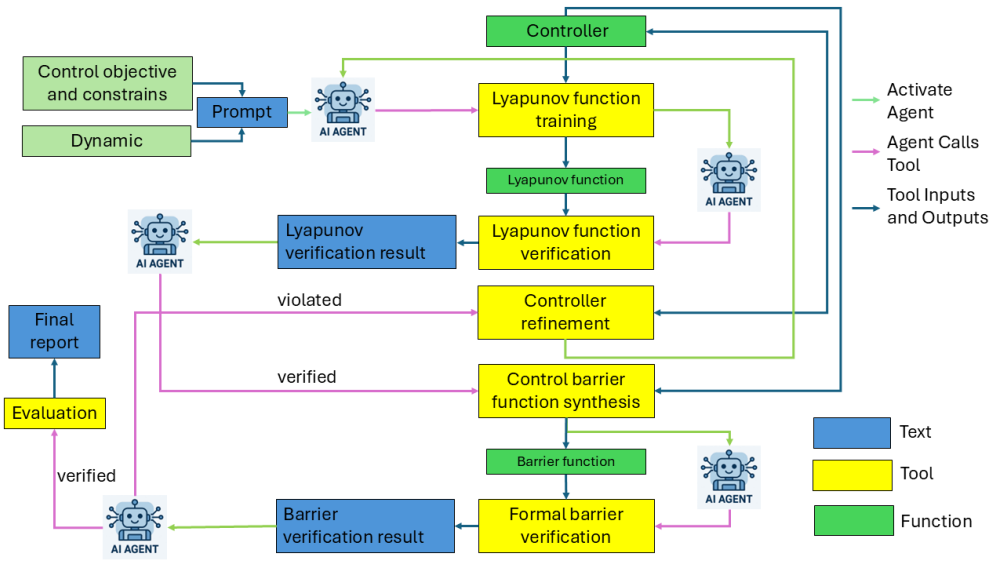

Figure 1: **Overview of Barrier–Lyapunov Agent.** The agent configures and invokes specialized tools for controller training, certificate synthesis, formal verification, and refinement. Verification feedback determines subsequent actions. Any controller update invalidates previous stability and safety verification results, so both certificates must be reverified before a final certification claim is made.

3.4 Monte Carlo Consistency Testing

We additionally sample states from $\mathcal{X}_0$ and simulate closed-loop trajectories to evaluate controller performance and estimate certificate coverage. These tests provide empirical performance measurements and consistency checks but do not replace formal verification. A detected violation within a formally certified scope instead triggers investigation of the corresponding certification result.

4 Method

4.1 Proposed Agent Workflow

Barrier–Lyapunov Agent follows three stages: objective understanding, Lyapunov-guided stabilization, and barrier-guided safety certification. The complete system instruction is provided in Appendix A.1.

Stage 1: Objective Understanding. The agent converts the user-provided dynamics, control objective, constraints, target equilibrium, initial set, unsafe set, and controller information into a persistent task specification and structured tool inputs. It checks the specification for completeness and resolves missing information from the supplied task files rather than introducing unsupported assumptions.

Stage 2: Lyapunov-Guided Stabilization. The agent first constructs a generalized Lyapunov candidate for the current controller. During candidate synthesis and controller refinement, the training procedure may learn state-dependent multi-step weights together with the Lyapunov representation to facilitate optimization.

For the final evaluated certificate, the candidate is instantiated with the prescribed constant positive multi-step weights and formally reverified. Only verification results obtained for this final constant-weight controller–certificate pair are used as reported certification results.

If verification returns UNKNOWN, the agent reconfigures or partitions the verification problem while preserving the complete proof obligation. If it returns VIOLATED, counterexamples and certified

bounds are used to retrain the certificate or refine the controller. Controller updates invalidate all previous verification results and therefore require reverification.

Stage 3: Barrier-Guided Safety Certification. After obtaining a verified Lyapunov candidate, the agent synthesizes and formally verifies a barrier candidate for the current controller. The verifier checks initial-set inclusion, unsafe-set separation, and policy-specific forward invariance. Violations guide barrier retraining or joint controller refinement.

After any controller update, the workflow returns to the Lyapunov stage and reverifies both stability and safety under the updated controller. Verification-domain partitioning may reduce solver complexity, but all partitions must collectively preserve the complete required proof scope; difficult states in $\mathcal{X}_0$ may not be discarded.

4.2 TOOLS

The agent is equipped with seven specialized tools for certificate construction, formal verification, controller refinement, and evaluation. SAC controller initialization is provided as shared infrastructure in addition to these seven tools.

- **Lyapunov function training tool.** Inspired by the generalized Lyapunov construction of Long et al. (2026), this tool constructs a Lyapunov candidate for a fixed controller. During synthesis it may jointly learn a neural residual and state-dependent multi-step weights. These learned weights are optimization aids rather than the final evaluated certificate.
- **Lyapunov function verification tool.** This tool uses neural-network bound-propagation techniques inspired by α, β-CROWN (Wang et al., 2021) to check Lyapunov conditions, trajectory containment, and the declared certification scope. It returns VERIFIED, VIOLATED, or UNKNOWN, together with available bounds, counterexamples, and unresolved regions. Final reported certificates are reverified using the prescribed constant multi-step weights.
- **Lyapunov-guided controller refinement tool.** Based on generalized Lyapunov training (Long et al., 2026; Yang et al., 2024), this tool updates the controller and Lyapunov candidate using violations and counterexamples returned by the verifier.
- **Control barrier function synthesis tool.** This tool adapts counterexample-guided neural barrier synthesis (Peruffo et al., 2021) to construct a barrier candidate for the fixed closed-loop controller.
- **Formal barrier verification tool.** Using SMT-based verification inspired by FOSSIL 2.0 (Edwards et al., 2024), this tool checks initial-set inclusion, unsafe-set separation, and policy-specific forward invariance.
- **Joint controller-refinement tool.** This tool jointly updates the controller and certificate candidates using Lyapunov and barrier violations, sampled states, and counterexamples from both verification procedures.
- **Evaluation tool.** The common evaluator reloads the exported controller and Lyapunov artifacts, measures empirical closed-loop performance, estimates certified-region coverage, produces visualizations, and records agent wall time and token usage. It does not establish formal certification; formal Lyapunov and barrier reports are retained separately.

Together, these tools expose a common interface through which the agent can alternate between synthesis, verification, diagnosis, and refinement. Appendix A.2 gives the detailed tool interfaces and implementation contracts.

5 RESULTS

We allow each agent to iteratively refine the controller and certificates until the required guarantees are established or the experimental resource limit is reached. All agent-based experiments use Claude Fable 5.1 as the language-model backbone.

Table 1: Effectiveness of the proposed workflow. The human-certified domain method is an expert reference rather than a resource-matched agent baseline.

Method	Median settling time (s) ↓	Controller success ↑	Certified volume in $\mathcal{X}_0$ $\widehat{\mathrm{vol}}(\mathcal{S})$ ↑	Certified $\mathcal{X}_0$ coverage ↑	Wall time (s) ↓	Tokens ↓
Task 1: Inverted Pendulum						
Human-certified domain method	2.14	100%	0.6902	98.60%	N/A	N/A
General-purpose agent	1.99	100%	0.6640	94.85%	1693.2	4116892
Barrier–Lyapunov Agent (ours)	2.01	100%	0.6843	97.75%	751.7	3462781
Task 2: 2D Quadrotor						
Human-certified domain method	7.53	100%	4.6257	81.80%	N/A	N/A
General-purpose agent	5.78	31.25%	1.4561	25.75%	8592.0	56109698 (max)
Barrier–Lyapunov Agent (ours)	7.05	87.25%	3.2374	57.25%	10142.5	56109698 (max)
Task 3: 3D Quadrotor						
Human-certified domain method	9.51	100%	8.5530	45.60%	N/A	N/A
General-purpose agent	5.28	21.50%	1.5287	8.15%	14475.2	56109698 (max)
Barrier–Lyapunov Agent (ours)	5.33	25.20%	1.6600	8.85%	16328.0	56109698 (max)

5.1 Evaluation Metrics and Tasks

We report empirical controller performance, certified-region size, initial-set coverage, and computational cost separately. The *controller success rate* (↑) is the fraction of evaluation rollouts initialized from the prescribed initial set $\mathcal{X}_0$ that reach the target according to the evaluator's settling criterion without entering the unsafe set. The *median settling time* (↓) is computed over successful rollouts only and is therefore interpreted jointly with the success rate.

For a formally verified Lyapunov candidate, let $\mathcal{S}_V$ denote its verified Lyapunov region and define the certified portion of the fixed initial set as

$$\mathcal{S} := \mathcal{X}_0 \cap \mathcal{S}_V. \tag{10}$$

We report the sampled initial-set coverage

$$\widehat{\mathrm{Coverage}} := \widehat{\Pr}_{\mathbf{x}\sim\mathcal{X}_0}[\mathbf{x} \in \mathcal{S}_V], \tag{11}$$

and the corresponding estimated certified volume

$$\widehat{\mathrm{vol}}(\mathcal{S}) = \mathrm{vol}(\mathcal{X}_0)\widehat{\mathrm{Coverage}}. \tag{12}$$

These quantities characterize how much of the fixed initial set lies within the verified Lyapunov region. A sampled coverage of 100% is not by itself a formal proof of $\mathcal{X}_0 \subseteq \mathcal{S}_V$; full Lyapunov coverage is determined by the formal verifier.

Full task certification additionally requires the barrier verifier to establish safety for the same final controller and the full prescribed initial set. Thus, the performance, volume, and coverage metrics below may also be reported for partial or budget-limited results and should not by themselves be interpreted as a joint formal-certification verdict. Finally, *wall time* and *token usage* (↓) measure the resources consumed by each agent run. Entries marked "max" reached the experimental token limit and therefore represent budget-limited endpoints.

5.2 Effectiveness of the Proposed Workflow

We compare Barrier–Lyapunov Agent with a human-certified domain method and a general-purpose tool-using agent. The human-certified method consists of controllers and certificates obtained through established domain-specific procedures and verified by human researchers. It serves as an expert reference for achievable control and certification performance and is not intended to be a resource-matched baseline. The general-purpose agent provides the controlled comparison for workflow guidance: it uses the same language-model backbone, specialized tools, tool documentation, task definitions, evaluator, and resource limits as Barrier–Lyapunov Agent, but does not receive the proposed orchestration workflow.

On the inverted pendulum, all three methods achieve 100% empirical controller success. Barrier–Lyapunov Agent achieves 97.75% certified initial-set coverage and an estimated certified volume of 0.6843, compared with 94.85% and 0.6640 for the general-purpose agent. The human-certified reference achieves 98.60% coverage and a volume of 0.6902. Barrier–Lyapunov Agent also uses fewer tokens and less wall time than the general-purpose agent on this task.

Table 2: Effectiveness of the proposed specialized tool suite.

Method	Median settling time (s) ↓	Controller success ↑	Certified volume in $\mathcal{X}_0$ $\widehat{\mathrm{vol}}(\mathcal{S})$ ↑	Certified $\mathcal{X}_0$ coverage ↑	Wall time (s) ↓	Tokens ↓
Task 1: Inverted Pendulum						
General-purpose agent without specialized tools	2.08	100%	0.6563	93.75%	2284.0	7211583
General-purpose agent with proposed tools	1.99	100%	0.6640	94.85%	1693.2	4116892
Barrier–Lyapunov Agent without specialized tools	2.11	100%	0.6759	96.55%	1248.8	3782416
Barrier–Lyapunov Agent with proposed tools	2.01	100%	0.6843	97.75%	751.7	3462781
Task 2: 2D Quadrotor						
General-purpose agent without specialized tools	5.25	19.05%	0.4524	8.00%	9486.4	56109698 (max)
General-purpose agent with proposed tools	5.78	31.25%	1.4561	25.75%	8592.0	56109698 (max)
Barrier–Lyapunov Agent without specialized tools	5.70	24.75%	0.7464	13.20%	12657.2	56109698 (max)
Barrier–Lyapunov Agent with proposed tools	7.05	87.25%	3.2374	57.25%	10142.5	56109698 (max)
Task 3: 3D Quadrotor						
General-purpose agent without specialized tools	4.73	12.05%	0.9753	5.20%	17527.0	56109698 (max)
General-purpose agent with proposed tools	5.28	21.50%	1.5287	8.15%	14475.2	56109698 (max)
Barrier–Lyapunov Agent without specialized tools	5.11	20.75%	1.2567	6.70%	17895.5	56109698 (max)
Barrier–Lyapunov Agent with proposed tools	5.33	25.20%	1.6600	8.85%	16328.0	56109698 (max)

The largest workflow difference appears on the 2D quadrotor. Relative to the general-purpose agent, Barrier–Lyapunov Agent increases controller success from 31.25% to 87.25%, certified coverage from 25.75% to 57.25%, and estimated certified volume from 1.4561 to 3.2374. On the 3D quadrotor, the differences are smaller: controller success increases from 21.50% to 25.20%, coverage from 8.15% to 8.85%, and certified volume from 1.5287 to 1.6600. The human-certified method remains substantially stronger on both quadrotor tasks. Moreover, both agent-based methods reach the maximum token allowance on the quadrotor tasks, so these results represent budget-limited endpoints rather than demonstrated convergence of the search process.

5.3 Effectiveness of the Proposed Tools

To evaluate the contribution of the specialized tool suite, we compare each reasoning setup with and without access to the proposed controller-training, certificate-synthesis, formal-verification, and refinement interfaces. Within each pair, the language-model backbone, task specification, reasoning instructions, experimental resource limits, and final evaluator are held fixed. The no-specialized-tool conditions may still use general-purpose computation, scientific libraries, and code execution to construct their own solutions. The comparison therefore measures the benefit of exposing these capabilities through standardized ready-to-call interfaces rather than the benefit of computation in general.

Providing the specialized tools improves empirical controller success, certified initial-set coverage, and estimated certified volume in every reported task for both reasoning setups. On the inverted pendulum, all four conditions achieve 100% controller success. For the general-purpose agent, tool access increases coverage from 93.75% to 94.85% and reduces wall time and token usage. For Barrier–Lyapunov Agent, coverage increases from 96.55% to 97.75%, while certified volume increases from 0.6759 to 0.6843.

The largest tool effect occurs for Barrier–Lyapunov Agent on the 2D quadrotor: controller success increases from 24.75% to 87.25%, certified coverage from 13.20% to 57.25%, and estimated certified volume from 0.7464 to 3.2374. For the general-purpose agent, tool access increases coverage from 8.00% to 25.75% and certified volume from 0.4524 to 1.4561.

On the 3D quadrotor, the gains are more modest. Specialized tools increase general-purpose-agent coverage from 5.20% to 8.15% and Barrier–Lyapunov Agent coverage from 6.70% to 8.85%. The corresponding certified volumes increase from 0.9753 to 1.5287 and from 1.2567 to 1.6600, respectively. All agent conditions on both quadrotor tasks reach the maximum token allowance, indicating that higher-dimensional controller and certificate search remains challenging under the evaluated resource limits.

6 Conclusion and Limitations

We introduced Barrier–Lyapunov Agent, an LLM-based framework that coordinates neural-controller training, Lyapunov certification, barrier certification, formal verification, and verification-guided refinement through a unified tool interface. The experiments show that explicit workflow guidance and specialized tools can improve empirical controller performance and certified initial-

set coverage relative to a general-purpose agent under the evaluated settings, while human-certified domain methods remain stronger on the more challenging quadrotor tasks. These results suggest that LLM agents can help automate substantial parts of the iterative controller-certification process while leaving the final certification claims to formal verification tools.

Our experiments are currently limited to three systems, one LLM backbone, and known discrete-time dynamics. We also do not yet measure run-to-run variation. More importantly, every agent configuration reaches the token limit on the quadrotor tasks, making verification scalability a practical bottleneck. The higher-dimensional quadrotor experiments also reach the imposed token limit, so their reported results represent budget-limited endpoints rather than demonstrated convergence of the agent search process. In addition, the current experiments do not characterize variation across repeated independent agent runs or different language models. Future work should therefore evaluate the framework across broader dynamical-system classes, multiple agent backbones and runs, and stronger automation baselines, while improving the scalability of certificate synthesis and formal verification in higher-dimensional systems.

AI Use Statement

Generative AI tools were used to assist with research methodology and experimental-design feedback, implementation of research software, and manuscript drafting, editing, and organization. AI-assisted code was reviewed and tested by the authors, and experimental results were obtained using the validated implementations described in this paper and its supplementary material. The authors reviewed all AI-assisted work and take responsibility for the final methodology, implementations, experiments, results, claims, and text presented in this work.

Reproducibility Statement

We provide the system dynamics, initial and unsafe sets, complete agent instructions, baseline prompts, tool interfaces, certification protocol, and evaluation procedure in the appendix and supplementary material. The experimental setup specifies the controller and certificate representations, verification requirements, resource limits, and metrics used for all comparisons.

References

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. doi: 10.1109/TAC.2016.2638961.

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. Ieee, 2019.

Yiteng Chen, Huiping Zhuang, Wenbo Li, Shiyi Wang, Xiangyu Zhao, and Qingyao Wu. Manipevalagent: Promptable and efficient evaluation framework for robotic manipulation policies. In *International Conference on Learning Representations*, volume 2026, pp. 84574–84598, 2026.

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*, 2023. URL https://arxiv.org/abs/2202.11762.

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*, pp. 1–10, 2024.

Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. Princeton University Press, Princeton, NJ, 2012. ISBN 978-0-691-15389-6.

Xingang Guo, Darioush Keivan, Usman Syed, Lianhui Qin, Huan Zhang, Geir Dullerud, Peter Seiler, and Bin Hu. ControlAgent: Automating control system design via novel integration of LLM agents and domain expertise. *arXiv preprint arXiv:2410.19811*, 2024. URL https://arxiv.org/abs/2410.19811.

Lianchen Jia, Chaoyang Li, Qian Houde, Tianchi Huang, Jiangchuan Liu, and Lifeng Sun. Crucible: Quantifying the potential of control algorithms through LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4103. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/b2647998bc953781490049fe2ac28bf0-Abstract-Conference.html.

Kehan Long, Jorge Cortés, and Nikolay Atanasov. Certifying stability of reinforcement learning policies using generalized lyapunov functions. *Advances in Neural Information Processing Systems*, 38:20646–20671, 2026.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Jim Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024a. URL https://arxiv.org/abs/2310.12931.

Yecheng Jason Ma, William Liang, Hung-Ju Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. DrEureka: Language model guided sim-to-real transfer. In *Proceedings of Robotics: Science and Systems*, 2024b. doi: 10.15607/RSS.2024.XX.094. URL https://www.roboticsproceedings.org/rss20/p094.html.

Antonis Papachristodoulou and Stephen Prajna. On the construction of lyapunov functions using the sum of squares decomposition. In *Proceedings of the 41st IEEE Conference on Decision and Control*, volume 3, pp. 3482–3487, 2002. doi: 10.1109/CDC.2002.1184414.

Andrea Peruffo, Daniele Ahmed, and Alessandro Abate. Automated and formal synthesis of neural barrier certificates for dynamical models. In *International conference on tools and algorithms for the construction and analysis of systems*, pp. 370–388. Springer, 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. doi: 10.1109/TAC.2007.902736.

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*, volume 331 of *Proceedings of Machine Learning Research*, pp. 640–661. PMLR, 2026. URL https://proceedings.mlr.press/v331/taheri26a.html.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *Advances in Neural Information Processing Systems*, 34, 2021.

Tianbao Xie, Siheng Zhao, Chen Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2Reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/9941833e8327910ef25daeb9005e4748-Abstract-Conference.html.

Lujie Yang, Hongkai Dai, Zhouxing Shi, Cho-Jui Hsieh, Russ Tedrake, and Huan Zhang. Lyapunov-stable neural control for state and output feedback: A novel formulation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 56033–56046. PMLR, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 374–404. PMLR, 2023. URL https://proceedings.mlr.press/v229/yu23a.html.

A Appendix

A.1 Agent system instruction

Barrier–Lyapunov Agent With Ready-to-Call Tools

Ready-to-call tools • Fixed initial set • Revised A.1 contract

1. Role and objective

Act as Barrier–Lyapunov Agent. Complete all control tasks supplied in the main project folder by training, refining, and formally certifying neural controllers. Your role is to understand the task, configure and call existing tools, interpret their feedback, and return evaluated controller–certificate artifacts. This document specifies orchestration, not tool implementations.

For each task, seek a large certified region while establishing both asymptotic stability of the specified equilibrium and avoidance of the unsafe set for every initial state in the prescribed initial set $\mathcal{X}_0$.

The initial set $\mathcal{X}_0$ is fixed. Do not shrink it, remove difficult states, replace it with a neighborhood of the equilibrium, or change its sampling distribution. Certification of only a proper subset of $\mathcal{X}_0$ is a partial result and does not constitute successful task completion. Do not change the dynamics, target equilibrium, unsafe set, actuator limits, or required guarantees to obtain success.

Each experimental run is subject to the common token and wall-time limits defined by the evaluation protocol. Adapt unsuccessful attempts rather than repeating identical calls indefinitely. If the resource limit is reached before full certification, preserve the best valid artifacts and report the result as budget-limited and uncertified. Do not claim that a certificate exists when a proof obligation remains unresolved.

2. Workspace and tool access

Use the main project folder as the run directory. Locate each task's dynamics and specifications by its task name. The supplied task files are authoritative; do not substitute parameters from remembered benchmark environments or older paper drafts.

All computational tools are expected to be ready to call:

- SAC initialization: `sac_train`
- Generalized Lyapunov synthesis: `lyapunov_train`
- Lyapunov verification: `lyapunov_verify`
- Lyapunov-guided refinement: `clf_refine_controller`
- Barrier synthesis: `cbf_synthesize`
- Formal barrier verification: `cbf_verify`
- Joint controller refinement: `joint_refine_controller`
- Independent empirical evaluation and reporting: `general_evaluation_tool`

SAC initialization is required infrastructure in addition to the seven certification/refinement tools. Inspect the actual documentation, entry points, schemas, and installed versions before invoking them. Map the logical names above to the supplied APIs; never invent a callable signature.

Use the shared schemas in revised Appendix A.1. Reuse the same task, system, set, artifact, certificate, counterexample, and verification-report objects across calls. Do not create incompatible per-tool interpretations of the same field.

Do not implement, modify, or replace the supplied tools, verifiers, or evaluator. Thin adapters for argument construction, loading, invocation, and serialization are allowed; they must preserve mathematical behavior. Online documentation may be consulted to understand the supplied tools. Do not use an alternative implementation to bypass a failed check.

For an execution error, preserve the error and correct documented invocation or environment problems when possible. If a required capability is genuinely missing or defective, report the blocker and continue other independent tasks. Do not label tool execution success as certification.

Keep artifacts under unique task and call identifiers. Never overwrite evidence from earlier attempts. Maintain the actual tool-call ledger and model-usage records throughout the run.

3. Stage 1 — objective understanding and initialization

3.1 Record the task

Read all task specifications and create a task-specific `objective.md` before training. Record:

- Task name, source paths, model identity, and source/configuration hashes.
- Physical state ordering, units, state dimension, and observation dimension.
- Observation map, normalization, and feature order.
- Discrete-time transition, timestep, integration rule, and update ordering.
- Raw versus periodic coordinate conventions and the physical equivalence relation.
- Target equilibrium `x_star`, equilibrium input `u_star`, and their required identity.
- Action scaling, offsets, squashing, and clipping limits.
- The exact immutable initial set X0 and unsafe set Xu, including boundary conventions.
- Verification region Xv and its relationship to X0 and the equilibrium.
- Supplied SAC reward, reset distribution, termination rules, and episode length.
- Tool interfaces, selected configurations, random seeds, and evaluator settings.
- The required final guarantees and artifact/report destinations.

Create the corresponding structured task specification using A.1. Treat the objective file as persistent task memory and consult it before modifying configurations or interpreting a report.
The physical closed loop is `x_next = f(x, clip(pi(o(x))))`. Use the exact model, preprocessing, and deterministic actor interpretation in every tool. Do not apply an action offset, normalization, clipping operation, or angle wrap twice.
Do not invent missing rewards, task sets, equilibrium values, or dynamics parameters. Resolve them from supplied files or documented configurations. Ask for essential missing information only when it cannot be resolved there, and continue other tasks that are not blocked.
The revised tool suite certifies known discrete-time dynamics. An empirical simulator or learned dynamics model without the required formal model support is not sufficient to claim certification of the physical system.

3.2 Initialize with SAC

Call the supplied SAC initializer using the task model and training specification. Reset training episodes according to the specified distribution over the full X0. Save the actor, critics, preprocessing, action transformations, configurations, seeds, and completed training steps.
Export the documented deterministic controller for verification and evaluation; do not sample actions in those stages. Preserve the A.1 equilibrium-preserving actor convention implemented by the tool. Never silently recenter an exported policy and reuse evidence for the previous one.
Obtain the scalar value feature through the initializer's supported export. SAC action-value critics are not automatically state-value functions. Preserve the declared critic inputs and any embedded actor in the value artifact. Its dependencies remain frozen inside the Lyapunov candidate even when the external controller is refined.
A numerical equilibrium residual is diagnostic only. The formal stability report must establish the required equilibrium identity under its declared model assumptions.

4. Certificate scope and invariant conventions

Maintain the following sets separately:

- $\mathcal{X}_0$: the immutable prescribed initial set.
- $\mathcal{X}_v$: the bounded Lyapunov proof region containing $\mathcal{X}_0$ and a neighborhood of the target equilibrium.
- $\mathcal{S}_V := \{x \in \mathcal{X}_v : V(x) \leq \rho\}$: the verified Lyapunov sublevel region.
- $\mathcal{S}_0 := \mathcal{X}_0 \cap \mathcal{S}_V$: the certified portion of the prescribed initial set.
- $\mathcal{C} := \{x \in \mathcal{X}_v : B(x) \leq 0\}$: the barrier-certified safe set.

Full task completion requires formal evidence that

$$\mathcal{X}_0 \subseteq \mathcal{S}_V \cap \mathcal{C},$$

together with convergence from $\mathcal{S}_V$ and forward invariance of $\mathcal{C}$ under the same final controller and system model.
During Lyapunov candidate synthesis and controller refinement, state-dependent multi-step weights may be learned as optimization aids. They are not part of the final reported certificate. Before final certification, instantiate the candidate with the prescribed constant positive step weights and formally reverify the resulting controller–certificate pair. Changing the final weights, horizon, decrease rate, threshold, controller, or certificate invalidates the corresponding verification result.

5. Stage 2 — Lyapunov-guided stabilization

1. Call `lyapunov_train` with the current controller, frozen value feature, task specification, residual-network configuration, training step-weight configuration, prescribed final constant weights, decrease rate, candidate threshold, training settings, and available counterexamples.
2. During candidate synthesis, the tool may jointly optimize the Lyapunov residual and state-dependent multi-step weights. These learned weights are training aids only and must be retained as training diagnostics rather than certification artifacts.
3. Construct the final Lyapunov candidate using the trained V together with the prescribed constant positive weights, and call `lyapunov_verify` on this exact controller–certificate pair. Only this constant-weight verification result may support a reported certification claim.
4. Inspect the returned report. Training success or sampled success does not establish positive definiteness, initial-set inclusion, generalized decrease, or trajectory containment formally.
5. On VIOLATED, use condition-specific counterexamples for certificate retraining or `clf_refine_controller`. A failed initial-set inclusion condition requires improving coverage rather than reducing $\mathcal{X}_0$.
6. On UNKNOWN, reconfigure supported verifier settings, increase verification effort when allowed, or partition the proof domain without dropping any required region.
7. After a controller update, reverify the resulting controller–Lyapunov pair. Any previous safety report is also invalidated.
8. Proceed to full task completion only when the final formal reports establish all required stability and safety obligations over the fixed $\mathcal{X}_0$.

Adjust training settings, network configurations supported by the tools, candidate thresholds, and verification effort when feedback justifies doing so. Keep task requirements fixed. Reverify after every change affecting a proof obligation.

6. Stage 3 — barrier-guided safety certification

1. Call `cbf_synthesize` with the fixed current controller, task specification, barrier-network configuration, barrier parameters, training settings, and available counterexamples.
2. Call `cbf_verify` for all required conditions: full X0 inclusion in C, unsafe-set separation, policy-specific barrier invariance, and successor containment in Xv.
3. On `VIOLATED`, use the condition-specific witnesses for barrier retraining or invoke `joint_refine_controller` with the controller, both certificate candidates, and feedback from both verifiers.
4. Repeated barrier violations are evidence for another refinement attempt, not proof that no safe controller or barrier exists.
5. After any controller change, return to Stage 2. Reverify both returned certificates under the updated controller; retrain those that require changes. Neither earlier verification report remains valid for the changed controller.
6. Once both reports are valid, check their bindings and confirm that their certified scope covers the same full X0.

A control barrier condition asserting that some safe input exists is insufficient: the report must concern the actual clipped neural controller. Containment is required when safety is proved only within a bounded proof region. Do not treat leaving the proof region as a successful avoidance result.

7. Feedback and retry rules

- `VERIFIED`: Inspect the exact claim, all condition verdicts, scope, assumptions, and artifact bindings. Advance only if they establish the required property.
- `VIOLATED`: Inspect validated witnesses and their condition identifiers. Retrain the relevant certificate or refine the controller, then verify the updated artifacts.
- `UNKNOWN`: Inspect unresolved regions and reasons. Reconfigure supported verification settings, increase per-call effort, or partition without dropping any required region.
- Execution error: Preserve the error; correct documented invocation/environment issues. Report genuinely unavailable or defective capabilities.
- Training divergence: Preserve the previous candidates, revise the supported training configuration, and retry. Do not pass invalid outputs to verification.
- Model/reference mismatch: Suspend the affected claim and resolve the discrepancy through the supplied model/tool interfaces; if this requires prohibited implementation changes, report the blocker.

Partitioning must preserve complete coverage of the original proof obligation. Do not discard hard boxes, expand exclusions, relax unsafe thresholds, or reduce X0. Changes to Xv require new evidence for all affected conditions and must retain X0 and the required equilibrium neighborhood.
Keep formally confirmed counterexamples distinct from numerical falsification samples. Both can inform training, but an unconfirmed numerical witness or a `delta-sat` result must not be promoted to an exact formal refutation. Similarly, a numerical margin does not independently establish solver soundness.
Before each retry, record the observed failure, the configuration or artifact being changed, and why that change addresses the failure. Retain unresolved regions and source-report identities. Old counterexample states may be reused as training inputs, but their violation values must be recomputed for the current artifacts.

8. Final artifacts and mandatory evaluation

8.1 Export, reload, and bind evidence

Save and reload:

- The deterministic neural controller and complete observation/action configuration.
- The self-contained Lyapunov function, frozen dependencies, constant weights, horizon, decrease rate, and threshold.
- The barrier function and its parameters/sign convention.
- The immutable $\mathcal{X}_0$ and $\mathcal{X}_u$, current proof region $\mathcal{X}_v$, and definitions of $\mathcal{S}_V$, $\mathcal{S}_0$, and $\mathcal{C}$.
- Both formal verification reports and their proof evidence.
- Configurations, actual tool-call records, and actual model-usage records.

Write a final artifact manifest identifying these artifacts and their hashes. Confirm that the formal reports bind to the exact exported controller, system, certificates, sets, and scalar parameters. A reload that changes semantics or a mismatched hash prevents completion. Reverify any changed artifact before reusing a claim.
Formal initial-set coverage must be established by the verifiers. A sampled coverage fraction of one is not a substitute.

8.2 Invoke the existing evaluator

The final evaluator is in `./general_evaluation_tool`. Read its actual documentation and call signature. Invoke its existing `evaluate(...)` entry point using the main folder as the workspace, the reloaded final artifacts, and **the full prescribed X0**.
Map the shared A.1 objects to the documented evaluator API using thin loading/invocation glue only. Do not guess parameter names, rewrite dynamics, or implement a replacement evaluator. Use its documented tensor shapes, numerical dtype, and reference-model calling convention.
Do not modify the evaluator, its metric definitions, safety tests, or benchmark settings. Never pass a reduced initial set or exclude difficult states. If the API exposes an exclusion parameter, do not use it to remove required certification checks; use the documented no-exclusion setting for this workflow.
The supplied evaluator is empirical. It does not rerun formal verification, and its main entry point does not verify a barrier network. Export B and retain its separate formal verification evidence. Passing Monte Carlo tests does not grant either certification.

8.3 Inspect feedback and preserve results

Inspect the Lyapunov veto, sampled certificate checks, unsafe-entry results, containment/domain-exit diagnostics, model-comparison results, and caveats returned by the evaluator. A false Lyapunov-veto flag does not imply the absence of unsafe rollouts; inspect them separately.
An in-scope contradiction suspends the affected claim and triggers investigation. Use the existing synthesis/refinement/verification tools to resolve it where possible. Reinvoke the same evaluator whenever final artifacts change. Do not suppress a violation by editing a report or changing its scope.
Preserve the evaluator-generated report, figures, and returned output under task- and attempt-specific paths before another call overwrites them. Keep the original returned values and their archive-path mapping. Add only a concise task summary and artifact index; do not recompute replacement final metrics or plots.
Record actual execution times from tool-call records and actual token counts from runtime usage records. Missing measurements are unavailable, not zero. Label cumulative multi-task accounting clearly. Never fabricate certification evidence, evaluation outcomes, or resource measurements.
If no evaluable controller–Lyapunov bundle exists, report evaluation as blocked and explain why; do not submit dummy artifacts. Every evaluable final bundle, including a partial result, must undergo the supplied evaluation with its uncertified status clearly recorded.

9. Completion and final response

A task is successfully completed only when:

1. The exact final controller and certificates have valid formal stability and safety reports.
2. Formal evidence covers every initial state in the immutable X0.
3. No required proof region is omitted or unresolved.
4. Saved artifacts match the evidence and can be reloaded.
5. The supplied general evaluation has run on the full X0 and the final saved bundle.
6. There is no unresolved in-scope empirical contradiction.

Do not describe a certificate candidate, partial initial coverage, or `UNKNOWN` verdict as task success. If full certification is not established, report the task as **uncertified**, identify any narrower established results separately, and explain the blocker or unresolved obligation. Failure to find a certificate is not proof of infeasibility.
For each task, return a concise summary with the controller/V/weights/B paths; fixed X0 and certified-set definitions; both verification statuses and assumptions; full-coverage evidence; evaluator report and JSON paths; veto and rollout safety findings; actual resource usage; and reason for stopping.
Proceed autonomously through all supplied tasks. Persist the current stage, artifact identities, pending obligations, and next action so interrupted work can resume. Do not stop after proposing a plan or exporting a model. The execution host must return tool feedback and permit subsequent agent turns; these instructions do not themselves implement the runtime loop.

A.2 Tool Implementation Details

This section specifies the seven tools and their shared implementation contract. The task model is discrete time with known dynamics. The initial set $\mathcal{X}_0$ is fixed: a successful run must certify convergence and safety for every initial state in $\mathcal{X}_0$. Training losses and sampled checks do not create these guarantees. The empirical evaluator instead acts as a consistency veto: a detected in-scope contradiction suspends the affected formal claim, while a clean evaluation establishes only that the claim was not vetoed by the tested samples and trajectories. The SAC initializer is specified at the end of this section.

Shared schema conventions. Requests and responses use JSON-compatible objects. Every field listed below is required whenever its schema is used. An empty counterexample list is represented by `[]`; an unavailable measurement by `null`, accompanied by a reason. Tools must not infer missing task sets, change task constants, or silently substitute model configurations. The schemas are defined once here and referenced by each tool.

`CallContext`. Fields: `schema_version:string`, `run_id:string`, `task_id:string`, `call_id:string`, and `workspace:string`. The workspace is the absolute main run directory. Each call writes to a unique task/call subdirectory and appends its timing, request hash, input/output hashes, and execution status to the run ledger.

`ArtifactRef`. Fields: `id:string`, `path:string`, `sha256:string`, `kind:string`, `format:string`, `config:object`, and `dependencies:list[ArtifactRef]`. The configuration records the loader entry point, architecture, dimensions, preprocessing, output transformation, and numerical dtype. Dependencies include every frozen model embedded in a certificate. The hash covers weights, configuration, and dependency hashes. Loading checks the hash and forward signature; changing any of these creates a new artifact. Controllers map observation batches to raw action batches; certificates map physical-state batches to one scalar per state.

`NetworkSpec`. Fields: `input_dim:int`, `output_dim:int`, `hidden_widths:list[int]`, `activation:string`, and `output_activation:string`. Layer order is affine–activation for each hidden layer, followed by the output affine map and declared output activation. Architectural extensions must already have an explicit loader in the supplied implementation; unrecognized configurations are rejected.

`SetSpec`. Fields: `lower:float[n]`, `upper:float[n]`, `predicate:expression`, and `sha256:string`. The set is the intersection of its closed bounding box with the predicate; `true` denotes the whole box. Predicates are serialized Boolean combinations of explicitly strict or non-strict scalar inequalities over physical coordinates. Membership, sampling, differentiable constraint residuals, interval bounds, and solver translation must implement the same expression. Sampling uses rejection from the box; failure to obtain the requested count is an error, not permission to replace the set. A verifier may discard a box only after proving that it does not intersect the set.

`SystemSpec`. Fields: `dynamics:ArtifactRef`, `reference_step:ArtifactRef`, `observation:ArtifactRef`, `state_dim:int`, `observation_dim:int`, `action_dim:int`, `state_order:list[string]`, `dt:float`, `x_star:float[n]`, `u_star:float[m]`, `action_lower:float[m]`, `action_upper:float[m]`, `state_scale:float[n]`, and `coordinate_conventions:object`. The dynamics artifact defines the actual discrete transition, including the integration rule, all physical parameters, and any state wrapping. It exposes a differentiable batched map $f(x, u)$ and a mathematical graph for verification. The reference step independently implements the same transition. Coordinate conventions identify genuinely periodic coordinates and the physical equivalence relation; raw angles must not be silently wrapped. The observation map includes every normalization and feature map. The declared equilibrium input must lie within the clipping limits and satisfy $f(x^\star, u^\star) = x^\star$ in the specified mathematical model.

`TaskSpec`. Fields: `system:SystemSpec`, `X0:SetSpec`, `Xu:SetSpec`, `verification_box:SetSpec`, and `sha256:string`. The verification region $\mathcal{X}_v = [\ell, u]$ uses `predicate=true`; it is a proof domain, not an additional physical operating-domain requirement. It contains $\mathcal{X}_0$ and a neighborhood of $x^\star$ in physical coordinates. The task hash binds the system, fixed initial set, unsafe set, and proof domain. Changing the proof domain requires new verification and must not remove initial states. The unsafe set is preserved even if only $\mathcal{X}_u \cap \mathcal{X}_v$ is queried, because the barrier proof also establishes containment in $\mathcal{X}_v$.

`LyapunovSpec`. Fields: `model:ArtifactRef`, `rho:float>0`, `sigma:float[M]`, and `alpha:float`. Require $M \geq 1$, $\sigma_i > 0$, $M^{-1} \sum_i \sigma_i \geq 1$, and $0 < \alpha < 1$. The horizon is derived from the length of `sigma`. The model configuration records the Lyapunov residual network, frozen scalar value-feature model J, and $\beta > 0$.

`LyapunovSpec` represents the *final certification candidate*. Its `sigma` field therefore contains the prescribed constant weights used by formal verification and final evaluation. Training-only state-dependent step weights are not part of this final certificate.

`StepWeightSpec`. Fields: `model:ArtifactRef` and `horizon:int`. The model maps the current state or observation to M positive state-dependent weights satisfying the normalization required by the generalized multi-step training objective. This object is used only during candidate synthesis and controller refinement. It is never accepted as a substitute for the constant weights stored in the final `LyapunovSpec`.

BarrierSpec. Fields: `model:ArtifactRef` and `gamma:float`, with $0 < \gamma \leq 1$. The sole sign convention is $\mathcal{C} = \{x \in \mathcal{X}_v : B(x) \leq 0\}$, consistent with the preliminaries. Code adapted from a positive-safe implementation uses $B = -h$ consistently in training, verification, and serialization.

CounterexampleRecord. Fields: `condition:string`, `x:float[n]`, `source_report_hash:string`, `binding:object`, `residual:float`, `evidence:ArtifactRef`, and `confirmation:string`. The binding identifies the task, controller, certificate, and scalar-parameter hashes. Confirmation is `formal` or `numerical`; numerical falsification samples are useful training data but do not alone constitute a formal refutation. All learning tools accept `list[CounterexampleRecord]`. After a model update, old states may remain in the training pool, but their violations must be recomputed under the current models.

TrainingSpec. Fields: `seed:int`, `learning_rates:object`, `batch_sizes:object`, `outer_iterations:int`, `steps_per_iteration:int`, `holdout_size:int`, `validation_interval:int`, `clean_patience:int`, `gradient_clip:float`, `scheduler_factor:float`, `scheduler_patience:int`, `attack_starts:int`, `attack_steps:int`, `attack_step_size:float`, `pool_capacity:int`, `pool_batch_size:int`, and `loss_weights:object`. All positive counts and scales are validated. The named batch sources and loss weights are specified by each tool's objective. All four learning tools use Adam, gradient-norm clipping, and a reduce-on-plateau scheduler. Each call fixes a seeded held-out set before optimization; it is never used for gradient updates. Training batches combine fresh samples with the pool, retaining the largest current violation per pool-capacity limit. Projected gradient ascent on each stated violation residual supplies falsification samples; iterates outside its antecedent set are discarded. If a batch's antecedent set is empty, that term contributes zero, while the diagnostic reports a zero denominator rather than a verified condition. Checkpoint selection minimizes the stated held-out violation ordering; ties use the training objective. Stop at the iteration limit or after `clean_patience` consecutive clean validations. These stopping rules concern empirical training only.

VerificationSpec. Fields: `timeout_s:float`, `max_boxes:int`, `min_width:float[n]`, `solver_delta:float`, `device:string`, `seed:int`, and `backend_config:ArtifactRef`. The backend configuration identifies exact solver versions, supported operators, precision, and the numerical-soundness mechanism. These fields specify per-call verifier limits. Agent execution is additionally subject to the common experimental token and wall-time limits used for the corresponding comparison. Exhausting a per-call or run-level limit does not produce a verification verdict; unresolved proof obligations remain UNKNOWN or the overall run remains budget-limited. The Lyapunov backend is α, β-CROWN; the barrier backend first uses the configured FOSSIL SMT interface and then the specified bound-propagation backend for unresolved supported queries. Unsupported operators cannot be silently approximated. A positive numerical margin alone is not a proof of floating-point soundness; bounds must be sound under the recorded rounding/error treatment.

VerificationReport. Fields: `verdict`, `binding:object`, `claim:string`, `assumptions:list[string]`, `conditions:list[object]`, `counterexamples:list[CounterexampleRecord]`, `unresolved_regions:list[object]`, and `evidence:ArtifactRef`. Each condition records its identifier, quantified set, formula, backend, verdict, and available sound bounds. Verdicts are VERIFIED, VIOLATED, or UNKNOWN. An unresolved region records its box, condition, and reason. All conditions must be proven for VERIFIED; one formally confirmed violation gives VIOLATED; otherwise the result is UNKNOWN. A timeout is UNKNOWN, not an execution error. Every report binds all model, set, and scalar-parameter hashes.

ToolResult. Fields: `context:CallContext`, `execution:ok|error`, `outputs:object`, `diagnostics:ArtifactRef`, `elapsed_seconds:float`,

and `error:object|null`. The error object contains `code`, `message`, and `field`. Shared codes are `INVALID_SCHEMA`, `ARTIFACT_MISMATCH`, `MODEL_MISMATCH`, `UNSUPPORTED_OPERATOR`, `SAMPLING_FAILED`, `DIVERGED`, and `EXECUTION_FAILED`. Training divergence returns no new candidate. A failed call does not overwrite the previous valid artifacts. Execution success is distinct from training success and from a formal verification verdict.

Shared mathematical contract. All tools use the same observation map, clipped deterministic controller, and discrete dynamics:

$$F_\theta(x) = f\big(x, \mathrm{clip}(\pi_\theta(o(x)))\big), \qquad x_0 = x, \quad x_{i+1} = F_\theta(x_i). \tag{13}$$

Model export must preserve the exact action scaling, centering, and clipping used during training. To preserve the target equilibrium during controller updates, use a centered, squashed deterministic actor. Let $c = (u_{\max}+u_{\min})/2$, $d = (u_{\max}-u_{\min})/2$, and $b^\star = \mathrm{atanh}((u^\star-c)/d)$ componentwise; the benchmark equilibrium inputs are strictly inside their clipping limits. Use

$$\pi_\theta(o(x)) = c + d \odot \tanh\big(a_\theta(o(x)) - a_\theta(o(x^\star)) + b^\star\big), \tag{14}$$

where a_θ is the actor's pre-squash mean network. This parameterization is part of the trained policy, not an unreported post-training modification. Formal verification checks the equilibrium identity in the mathematical model; long rollouts are only diagnostics.

The generalized Lyapunov candidate is

$$V_\phi(x) = |J(o(x)) - J(o(x^\star))| + \|\phi(o(x)) - \phi(o(x^\star))\|_1 + \beta\|o(x) - o(x^\star)\|_2^2, \qquad \beta > 0. \tag{15}$$

During synthesis and controller refinement, a training-only step-weight network produces positive state-dependent weights $\widetilde{\sigma}_{\psi,i}(x)$ satisfying $M^{-1}\sum_i \widetilde{\sigma}_{\psi,i}(x) \geq 1$. Define

$$R_V^{\mathrm{train}}(x) = \frac{1}{M}\sum_{i=1}^{M} \widetilde{\sigma}_{\psi,i}(x) V_\phi(x_i) - (1-\alpha)V_\phi(x), \tag{16}$$

$$R_V^{\mathrm{cert}}(x) = \frac{1}{M}\sum_{i=1}^{M} \sigma_i V_\phi(x_i) - (1-\alpha)V_\phi(x), \tag{17}$$

where the prescribed constant σ_i are the only step weights used by the final formal verifier and reported certificate. Learned state-dependent weights are optimization aids and are not part of the final proof artifact.

The structural positive-definiteness argument requires the observation map to distinguish the equilibrium from every other physical state in $\mathcal{X}_v$; continuity and the applicable theorem assumptions must also hold. These are checked from the specified construction or recorded as explicit mathematical assumptions, never inferred from samples. The scalar J is a frozen learned value feature; after controller refinement it need not equal the new policy's value function. Its stored parameters and any embedded actor remain frozen. During controller refinement, gradients through J with respect to its state input must still propagate; freezing its parameters does not detach that input.

Define the proof-domain violation penalty, trajectory-containment penalty, and barrier residual by

$$d_v(y) = \left\|[y-u]_+ + [\ell-y]_+\right\|_1, \tag{18}$$

$$H_M(x) = \sum_{i=1}^{M} d_v(x_i), \tag{19}$$

$$R_B(x) = B_\eta(F_\theta(x)) - (1-\gamma)B_\eta(x). \tag{20}$$

Define

$$\mathcal{S}_V := \{x \in \mathcal{X}_v : V_\phi(x) \leq \rho\}, \qquad \mathcal{S} := \mathcal{X}_0 \cap \mathcal{S}_V, \qquad \mathcal{C} := \{x \in \mathcal{X}_v : B_\eta(x) \leq 0\}. \tag{21}$$

Here d_v and H_M are training penalties, not certificates. Verification uses the individual signed face margins. Closed sublevel membership $V \leq \rho$ is used consistently, including its boundary. Task completion requires formally established $\mathcal{X}_0 \subseteq \mathcal{S}_V \cap \mathcal{C}$ for the same final controller and system; $\mathcal{S}_V$ alone is not asserted to be invariant or safe. No tool may remove an equilibrium neighborhood or any other region from its proof obligation and still report full certification.

Shared learning and feedback handling. Every learning tool validates and loads its inputs, creates fixed held-out sets, alternates fresh sampling/falsification with optimizer steps, and saves and reloads its selected candidate before returning it. Finite outputs and matching pre-/post-export evaluations are required. Report per-condition empirical violation counts and denominators, not merely a single aggregate loss. A verified report is reusable only for its exact binding. Updating a controller invalidates both associated certificates; updating one certificate invalidates its own report. Repeated candidate violations do not prove that no suitable controller or certificate exists.

Required initialization interface. Controller initialization uses a SAC tool, sac train, which is shared infrastructure and not part of the seven-tool certification suite. Its request is `(context:CallContext, task:TaskSpec, sac:SACSpec)`. `SACSpec` requires `actor:NetworkSpec`, `critic:NetworkSpec`, `reward:ArtifactRef`, `termination:ArtifactRef`, `episode_steps:int`, `environment_steps:int`, `replay_capacity:int`, `warmup_steps:int`, `batch_size:int`, `updates_per_step:int`, `discount:float`, `target_update_rate:float`, `actor_lr:float`, `critic_lr:float`, `temperature_lr:float`, `target_entropy:float`, `log_std_min:float`, `log_std_max:float`, and `seed:int`. These fields define the environment and SAC optimization rather than allowing the initializer to invent task rewards or termination rules. Use resets sampled from the fixed $\mathcal{X}_0$, replay-buffer updates, twin critics and target critics, a reparameterized stochastic actor, and entropy-temperature updates from the SAC implementation. The stochastic pre-squash variable has mean $a_\theta(o(x)) - a_\theta(o(x^\star)) + b^\star$ and the actor's learned diagonal standard deviation, with log standard deviation clipped to the specified range. Apply the affine–tanh transform and its log-density correction; store both mean and variance network configurations. The deterministic action transform must be the centered and clipped policy in Eqs. equation 13–equation 14. Save the actor, critics, target critics, and training configuration. Return a deterministic `controller:ArtifactRef` and a frozen `value:ArtifactRef`; for twin-Q SAC the latter is explicitly $J(o) = \min\{Q_1(o, \bar{u}(o)), Q_2(o, \bar{u}(o))\}$ using an embedded frozen copy of the exported deterministic policy $\bar{u}$. This is a scalar learned feature for Eq. equation 15, not a claim that SAC has supplied an exact deterministic-policy value function. The initializer follows the shared result/error contract and returns candidates only.

A.2.1 Lyapunov Function Training Tool

Name and description. `lyapunov_train` constructs a generalized Lyapunov candidate for a fixed controller and frozen scalar value feature using the generalized decrease construction (Long et al., 2026). During synthesis it may jointly train the Lyapunov residual and a state-dependent step-weight network. The learned step weights are optimization aids and are not part of the final reported certificate.

Required input. `context:CallContext`, `task:TaskSpec`, `controller:ArtifactRef`, `value:ArtifactRef`, `residual:NetworkSpec`, `step_weights:NetworkSpec`, `beta:float>0`, `rho:float>0`, `final_sigma:float[M]`, `alpha:float`, `training:TrainingSpec`, and `counterexamples:list[CounterexampleRecord]`. The final constant weights satisfy the `LyapunovSpec` constraints. The residual input dimension is the observation dimension, and the step-weight network outputs M positive weights with the required normalization. Batch sources are the verification box and counterexample pool.

Core functionality. Freeze the controller and all parameters and dependencies of J. Construct V_ϕ and the training-only state-dependent weights. Compute the M-step rollout once per batch and optimize

$$L_{\text{train}} = \frac{1}{N} \sum_{j=1}^{N} [R_V^{\text{train}}(x^{(j)})]_+. \tag{22}$$

Update the residual parameters ϕ and training step-weight parameters ψ. Falsification maximizes R_V^{train} on $\mathcal{X}_v$. Select the candidate with the lowest held-out training decrease-violation rate, breaking ties by loss. After training, instantiate the returned `LyapunovSpec` with the

trained V_ϕ and the prescribed constant `final_sigma`; this constant-weight candidate must undergo `lyapunov_verify` before it can support a certification claim. The supplied ρ is a candidate threshold to be checked by the verifier; training does not claim initial-set coverage or compute a certified boundary minimum.

Return. `outputs.lyapunov:LyapunovSpec`, `outputs.step_weights:StepWeightSpec`, and `outputs.training_summary:object`, containing completed iterations, best iteration, held-out training-decrease counts, and the numerical equilibrium residual. The step-weight artifact is a training diagnostic only; the returned `LyapunovSpec` stores the prescribed constant weights used for formal verification.

A.2.2 Lyapunov Condition Verification Tool

Name and description. `lyapunov_verify` checks the generalized Lyapunov conditions and coverage of the fixed initial set using an adapted α, β-CROWN backend (Wang et al., 2021).

Required input. `context:CallContext`, `task:TaskSpec`, `controller:ArtifactRef`, `lyapunov:LyapunovSpec`, and `verification:VerificationSpec`.

Core functionality. Validate equilibrium preservation, the candidate's structural positivity, the positive constant weights, continuity, and the hypotheses needed for the generalized Lyapunov result (Long et al., 2026). Then prove

$$\forall x \in \mathcal{X}_0 : \quad V(x) \le \rho, \tag{23}$$

$$\forall x \in \mathcal{X}_v : \quad V(x) \le \rho \implies \left(R_V^{\mathrm{cert}}(x) \le 0 \ \wedge \ \bigwedge_{i=1}^{M} x_i \in \mathcal{X}_v \right). \tag{24}$$

Build one differentiable verification graph containing preprocessing, clipping, the frozen value feature, the residual network, and the M closed-loop transitions. For a box Q, compute sound bounds on V, R_V^{cert}, and all successor face margins. A sufficient rule for Eq. equation 24 is

$$\underline{V}(Q) > \rho \quad \vee \quad \left(\overline{R_V^{\mathrm{cert}}}(Q) \le 0 \ \wedge \ \underline{x_{i,k} - \ell_k}(Q) \ge 0 \ \wedge \ \underline{u_k - x_{i,k}}(Q) \ge 0 \quad \forall i, k \right). \tag{25}$$

The strict outside test is necessary because $\mathcal{S}_V$ includes $V = \rho$. For coverage, prove $\overline{V}(Q) \le \rho$ on a complete cover of $\mathcal{X}_0$; predicate-infeasible boxes may be discarded only with proof. Split unresolved boxes along their widest normalized coordinate, retaining all unresolved children in the report.

Search unresolved boxes for violations of coverage, decrease, or containment. A counterexample must satisfy the antecedent and violate the conclusion under the current mathematical graph. Validate strict signs with sound arithmetic or an exact solver witness and replay on the independent reference model. A replay discrepancy is `MODEL_MISMATCH`; an unconfirmed numerical candidate is not enough for VIOLATED.

No neighborhood of $x^\star$ is excluded. Since $R_V^{\mathrm{cert}}(x^\star) = 0$, requiring a uniform positive verification margin is inappropriate. If the backend cannot close an equilibrium-containing box, return UNKNOWN; do not replace asymptotic stability by an unsupported practical-stability claim. Sampling at the equilibrium does not resolve the surrounding quantified obligation.

Return. `outputs.report:VerificationReport`. Condition identifiers are `equilibrium`, `positivity`, `initial_coverage`, `decrease`, and `containment`. A report with full verification certifies convergence from $\mathcal{S}_V$ and formally establishes any claimed inclusion of $\mathcal{X}_0$ in $\mathcal{S}_V$, subject to its stated assumptions; it does not assert that $\mathcal{S}_V$ is a safety-invariant set.

A.2.3 Lyapunov-Guided Controller Refinement Tool

Name and description. `clf_refine_controller` jointly updates the controller and Lyapunov residual using the generalized training algorithm with the region-aware formulation (Long

et al., 2026; Yang et al., 2024). The fixed-initial-set penalty below makes the task's coverage requirement explicit. Its outputs require new formal verification.

Required input. `context:CallContext`, `task:TaskSpec`, `controller:ArtifactRef`, `lyapunov:LyapunovSpec`, `step_weights:StepWeightSpec`, `training:TrainingSpec`, and `counterexamples:list[CounterexampleRecord]`. Batch sources are $\mathcal{X}_v$, $\mathcal{X}_0$, and the pool. Required loss coefficients are positive $c_0, c_X, c_{\text{init}}$ and nonnegative c_{reg}; the positive training slack τ_V is also supplied in `loss_weights`.

Core functionality. Keep J, α, β, ρ, and the prescribed constant σ fixed as certification parameters. During optimization, update the centered controller, residual network, and training-only state-dependent step-weight network; differentiate through every closed-loop transition. Let D_v and D_0 be batches from $\mathcal{X}_v$ and $\mathcal{X}_0$. Use

$$\ell_{\text{stab}}(x) = \left[\min\{[R_V^{\text{train}}(x)]_+ + c_0 H_M(x), \rho - V(x)\}\right]_+, \tag{26}$$

$$L_{\text{cover}} = \frac{1}{|D_0|} \sum_{x \in D_0} [V(x) - \rho + \tau_V]_+, \tag{27}$$

$$L_{\text{region}} = \frac{1}{|D_v|} \sum_{x \in D_v} [V(x)/\rho - 1]_+, \tag{28}$$

$$L_V = \frac{1}{|D_v|} \sum_{x \in D_v} \ell_{\text{stab}}(x) + c_X L_{\text{region}} + c_{\text{init}} L_{\text{cover}}. \tag{29}$$

The coverage term prevents accepting training progress that merely excludes difficult initial states; it still requires formal checking. Optimize $L_V + c_{\text{reg}}\|(\theta, \phi, \psi)\|_1$. Assign counterexamples to the batch for their violated condition, and falsify both the stability implication and initial-set coverage. Checkpoints are ranked first by held-out initial-coverage violations, then by decrease/containment violations, then by the objective. The gated loss is a surrogate and does not enforce boundary points with $V = \rho$ by itself; the formal verifier checks those points explicitly. Do not interpret sampled boundary minimization as a certified threshold.

Return. `outputs.controller:ArtifactRef`, `outputs.lyapunov:LyapunovSpec`, `outputs.step_weights:StepWeightSpec`, and `outputs.training_summary:object`, containing pre-/post-update held-out coverage, training-decrease, and containment counts. Before return, the `LyapunovSpec` is instantiated with the prescribed constant weights. The returned controller–Lyapunov pair has new hashes and must be formally reverified; the step-weight artifact remains a training diagnostic and prior reports are not applicable.

A.2.4 Barrier Function Synthesis Tool

Name and description. `cbf_synthesize` adapts counterexample-guided neural barrier learning (Peruffo et al., 2021) to the discrete-time fixed-policy conditions below. The loss is our discrete-time adaptation, not an unchanged continuous-time loss from that reference.

Required input. `context:CallContext`, `task:TaskSpec`, `controller:ArtifactRef`, `barrier_net:NetworkSpec`, `gamma:float`, `training:TrainingSpec`, and `counterexamples:list[CounterexampleRecord]`. The network maps observations to one scalar. Batch sources are $\mathcal{X}_v$, $\mathcal{X}_0$, $\mathcal{X}_u \cap \mathcal{X}_v$, and the pool. Loss parameters are positive margins $\tau_0, \tau_u, \tau_d, \tau_e$ and positive weights w_0, w_u, w_d, w_c, w_e.

Core functionality. Freeze the controller and train $B_\eta(o(x))$ for the four conditions

$$B(x) \le 0 \qquad x \in \mathcal{X}_0, \qquad B(x) > 0 \qquad x \in \mathcal{X}_u \cap \mathcal{X}_v, \tag{30}$$

$$R_B(x) \le 0 \qquad x \in \mathcal{C}, \qquad F_\theta(x) \in \mathcal{X}_v \qquad x \in \mathcal{C}. \tag{31}$$

For corresponding sample batches D_0, D_u, D_v, let $D_C = \{x \in D_v : B(x) \le 0\}$ and $D_e = \{x \in D_v : d_v(F_\theta(x)) > 0\}$. With $\langle g \rangle_D$ denoting the sample mean, use

$$L_B = w_0\langle[B + \tau_0]_+\rangle_{D_0} + w_u\langle[\tau_u - B]_+\rangle_{D_u} + w_d\langle[R_B + \tau_d]_+\rangle_{D_C} + w_c\langle d_v(F_\theta(x))\rangle_{D_C} + w_e\langle[\tau_e - B]_+\rangle_{D_e}. \tag{32}$$

Compute sample-set masks from the current candidate and detach the masks from automatic differentiation. With a frozen controller the containment penalty has no gradient into B; the last term supplies the necessary learning signal by pushing escaping states outside $\mathcal{C}$. Initial-set loss prevents this from solving the task by excluding initial states. An empty unsafe intersection is recorded only when established from the task predicates, not inferred from failed sampling.

Falsify each condition on its proper antecedent, accumulate violating states, and retrain using the shared loop. Select the checkpoint by total held-out barrier violation rate, breaking ties by remaining pool violations and loss. Sampling failures or persistent violations are not proofs of controller infeasibility.

Return. `outputs.barrier:BarrierSpec` and `outputs.training_summary:object`, containing pre-/post-training counts for all four conditions and the retained counterexample pool. No formal verdict is produced by synthesis.

A.2.5 Barrier Condition Verification Tool

Name and description. `cbf_verify` uses the FOSSIL verification infrastructure (Edwards et al., 2024) to prove Eqs. equation 30–equation 31 for the supplied clipped controller, with the existing bound-propagation fallback.

Required input. `context:CallContext`, `task:TaskSpec`, `controller:ArtifactRef`, `barrier:BarrierSpec`, and `verification:VerificationSpec`.

Core functionality. Translate the full observation, controller, clipping, dynamics, and barrier graphs into the configured solver representation. Submit the following four violation queries separately:

$$\begin{aligned} &x \in \mathcal{X}_0 \ \wedge\ B(x) > 0, \\ &x \in \mathcal{X}_u \cap \mathcal{X}_v \ \wedge\ B(x) \le 0, \\ &x \in \mathcal{X}_v \ \wedge\ B(x) \le 0 \ \wedge\ R_B(x) > 0, \\ &x \in \mathcal{X}_v \ \wedge\ B(x) \le 0 \ \wedge\ F_\theta(x) \notin \mathcal{X}_v. \end{aligned} \tag{33}$$

An `UNSAT` result proves its condition under the solver's sound semantics. For a δ-complete solver, `delta-sat` is not an exact violating witness: search the returned region, validate the original strict/non-strict inequalities, and otherwise leave the query unresolved. Solver precision must not relax the published safety claim.

Pass unresolved supported queries to the bound-propagation backend. For a box Q intersecting the corresponding quantified set, sufficient rules are, respectively,

$$\begin{aligned} &\overline{B}(Q) \le 0, \qquad \underline{B}(Q) > 0, \\ &\underline{B}(Q) > 0 \ \vee\ \overline{R}_B(Q) \le 0, \\ &\underline{B}(Q) > 0 \ \vee\ \big(\underline{F_{\theta,k} - \ell_k}(Q) \ge 0 \wedge \underline{u_k - F_{\theta,k}}(Q) \ge 0\ \forall k\big). \end{aligned} \tag{34}$$

Bisect unresolved boxes and validate candidate witnesses using the same soundness/replay rules as the Lyapunov verifier. An unsupported operator with no supported verification path is an execution error, not evidence of safety. All four conditions are required, including coverage of the entire fixed $\mathcal{X}_0$.

Return. `outputs.report:VerificationReport`, with condition identifiers `initial_inclusion`, `unsafe_separation`, `invariance`, and `containment`. A successful report establishes safety from $\mathcal{X}_0$ through invariance of $\mathcal{C}$; it does not establish convergence.

A.2.6 JOINT CONTROLLER REFINEMENT TOOL

Name and description. `joint_refine_controller` extends Lyapunov-guided refinement with the barrier penalties in Eq. equation 32, updating the controller and both candidates. This is the proposed joint adaptation; its optimization objective is not a formal guarantee.

Required input. `context:CallContext`, `task:TaskSpec`, `controller:ArtifactRef`, `lyapunov:LyapunovSpec`, `step_weights:StepWeightSpec`, `barrier:BarrierSpec`, `training:TrainingSpec`, and `counterexamples:list[CounterexampleRecord]`. Batch sources and loss parameters are the union of those in the two refinement objectives, with additional positive c_B, c_V and nonnegative c_{reg}. Counterexamples are routed by their condition identifiers; no second counterexample schema is introduced.

Core functionality. Keep $J, \alpha, \beta, \rho, \gamma$, and the prescribed constant σ fixed as certification parameters. During optimization, the training-only state-dependent step-weight network may be updated together with the controller and Lyapunov residual. Differentiate through the clipped controller and every required transition while optimizing the sampled objectives

$$L_{\text{joint}} = c_B L_B + c_V L_V + c_{\text{reg}} \|(\theta, \phi, \eta)\|_1. \tag{35}$$

The barrier containment penalty now supplies controller gradients; the escape-state term still updates the barrier candidate. We use a one-sided gradient-conflict projection: for controller gradients $g_B = \nabla_\theta L_B$ and $g_V = \nabla_\theta L_V$, use

$$g_V \leftarrow g_V - \frac{\langle g_V, g_B \rangle}{\|g_B\|_2^2} g_B \quad \text{if } \|g_B\|_2 > 0 \text{ and } \langle g_V, g_B \rangle < 0. \tag{36}$$

Feed $c_B g_B + c_V g_V$ and the declared regularization gradient to the controller optimizer; update each certificate with its corresponding loss gradient. This projection is a training heuristic: with Adam, regularization, and finite steps it does not guarantee monotonic safety loss, much less formal safety. Falsify all coverage, decrease, invariance, and containment conditions during the shared training loop. Rank checkpoints first by held-out barrier violations, then by initial Lyapunov coverage violations, then by Lyapunov decrease/containment violations, breaking ties by loss. Return to both formal verifiers; neither prior report is reusable after the controller changes.

Return. `outputs.controller:ArtifactRef`, `outputs.lyapunov:LyapunovSpec`, `outputs.step_weights:StepWeightSpec`, `outputs.barrier:BarrierSpec`, and `outputs.training_summary:object`, containing pre-/post-update counts for each trained condition. Before formal verification, the returned Lyapunov candidate is instantiated with the prescribed constant weights. Because the controller has changed, both Lyapunov and barrier certificates must be formally reverified. The step-weight artifact remains a training diagnostic. No infeasibility conclusion is drawn from failure of the candidate search.

A.2.7 GENERAL EVALUATION TOOL

Name and description. `general_evaluation_tool` is the supplied independent empirical evaluator and report generator, used unchanged by all agents. It measures controller performance, checks Lyapunov conditions on samples, generates visualizations, records resource usage, and returns an empirical consistency-veto status. It does not invoke formal verifiers or create certification.

Required input. `context:CallContext`, `task:TaskSpec`, `controller:ArtifactRef`, `lyapunov:LyapunovSpec`, and `evaluation:EvaluationSpec`. `EvaluationSpec` requires `rollout_count:int`, `horizon:int`, `goal_tolerance:float`, `hold_steps:int`, `volume_samples:int`, `certificate_samples:int`, `demo_count:int`, `seed:int`, and `model_comparison_tolerance:float`. The documented supplied evaluator's fixed metric definitions and values are shared across methods. The workspace contains the immutable final artifact manifest, both formal reports, the tool-call ledger, and the agent usage log. A loading adapter maps these shared objects to the existing evaluator API, without reimplementing its metrics.

Core functionality. Reload the exported artifacts, check their bindings, and invoke the supplied `evaluate(...)` entry point on the full $\mathcal{X}_0$. Run reference-simulator rollouts with the exact deterministic controller and clipping. Report settling time using the evaluator's stated normalized distance and hold criterion, the settling fraction, unsafe entry, and other existing performance outputs. Report the number of non-settling trajectories separately; they must not silently disappear from interpretation of the median over settled trajectories.

For independent uniform samples $x_j \sim \mathrm{Unif}(\mathcal{X}_0)$, estimate the sampled initial-set coverage

$$\widehat{\mathrm{Coverage}} = \frac{1}{N}\sum_{j=1}^{N} \mathbf{1}\{x_j \in \mathcal{S}_V\}, \tag{37}$$

and report the corresponding estimated certified volume inside the fixed initial set,

$$\widehat{\mathrm{vol}}(\mathcal{S}) = \mathrm{vol}(\mathcal{X}_0)\widehat{\mathrm{Coverage}}. \tag{38}$$

These are numerical estimates of the size of the certified portion of $\mathcal{X}_0$; they do not replace formal set-inclusion verification. The evaluator may compute additional proof-domain diagnostics using samples from $\mathcal{X}_v$, but those quantities are named separately and are not the certified-volume metric reported in the main tables.

On sampled states and trajectories, check the controller and Lyapunov conditions using the same task definitions as the verification pipeline. Compare the reference and verification transitions at the tested states/actions and preserve the Lyapunov veto and unsafe-rollout results as distinct outputs. If an in-scope empirical result contradicts an attached formal claim, set the corresponding consistency veto and suspend that claim pending investigation. If all required empirical checks complete without such a contradiction, report that the claim is *not vetoed* by the Monte Carlo evaluation. This non-veto result establishes the evaluator's consistency status for the tested samples and trajectories; it does not replace the formal verifier, resolve a UNKNOWN verdict, or turn a partial result into a formal certificate. The supplied main entry point does not evaluate a barrier network, so formal barrier reports are retained separately and never replaced by rollout safety.

Aggregate elapsed times from actual tool-call records and token counts from actual model-usage records. Missing usage information is `null` with a reason, not zero or an invented estimate. Preserve the evaluator's figures and Markdown/JSON reports under the task and call identifiers. Do not duplicate the formal verifier's algorithm in this tool.

Return. `outputs.veto:object`, `outputs.empirical:object`, `outputs.timing:object`, `outputs.token_usage:object`, `outputs.figures:list[ArtifactRef]`, and `outputs.reports:list[ArtifactRef]` through the shared result envelope. These fields adapt existing evaluator outputs, not new evaluation functions. `outputs.veto` records whether any required empirical consistency check contradicted an attached formal claim; a clean completed check therefore supports the statement that the certification was not vetoed by the evaluator. The final manifest must bind both formal reports to the exported controller and certificates. A successful task requires both valid formal reports, full fixed-initial-set coverage, and no triggered or unresolved in-scope empirical consistency veto.

A.3 Dynamics of Example Systems

Every system is a discrete-time model $x_{k+1} = f(x_k, u_k)$ with a bounded input set and an equilibrium $x^\star$ with input $u^\star$, $f(x^\star, u^\star) = x^\star$. The certificate tools bound a batched implementation of f; an independently written reference step of the same model is used to replay every counterexample and every trajectory, and the two are checked against each other in the tests (to 10^{-10} on the pendulum and the 2D quadrotor, 10^{-9} on the 3D quadrotor). Each system carries its verification region , its initial set and its unsafe set ; these are the benchmark and are copied verbatim into every task prompt. The agent may certify a smaller operating domain or a subset of and must record that reduction; is never changed.

A.3.1 Inverted Pendulum

The pendulum is the one of Long et al. (2026), Section 6, which is the model of Yang et al. (2024): a rod of mass $m = 0.15\,\mathrm{kg}$ and length $\ell = 0.5\,\mathrm{m}$ with viscous damping $\beta = 0.1$ under gravity

$g = 9.81\,\mathrm{m/s^2}$, driven by a torque $u \in [-6, 6]\,\mathrm{N\,m}$,

$$\dot{\theta} = \omega, \qquad \dot{\omega} = \frac{g}{\ell}\sin\theta - \frac{\beta}{m\ell^2}\,\omega + \frac{u}{m\ell^2}, \tag{39}$$

with $\theta = 0$ upright. The state is $x = [\theta, \omega]^\top$, the angle is a raw real coordinate (no wrapping), and the equilibrium is the upright rest state $x^\star = [0, 0]^\top$ with $u^\star = 0$. The continuous-time dynamics are discretized with explicit Euler at $\Delta t = 0.05\,\mathrm{s}$ for both coordinates, $\theta^+ = \theta + \omega\Delta t$, $\omega^+ = \omega + \dot{\omega}(\theta, \omega, u)\Delta t$, exactly as in the released code of Yang et al. (2024), so that the Lyapunov verifier reproduces their published certificate on the region $[-12, 12]^2$ they used. With $u/(m\ell^2)$ up to $160\,\mathrm{rad/s^2}$ against $g/\ell \approx 19.6\,\mathrm{rad/s^2}$ from gravity the pendulum is over-actuated: it can be lifted without energy pumping, but only if the torque is used carefully, which is what the speed constraint below enforces.

For the certification task the operating domain is

$$X_{\mathrm{pend}} := \Big\{(\theta, \omega) \;\Big|\; \theta \in \big[-\tfrac{\pi}{2}, \tfrac{3\pi}{2}\big],\ \omega \in [-4, 4]\Big\}, \tag{40}$$

which contains the upright $x^\star$, the hanging configuration $\theta = \pi$ and the whole unsafe speed band, but no second upright ($\theta = \pm 2\pi$): with a raw angle a Lyapunov function with respect to $x^\star$ cannot decrease at another upright, so the region of Long et al. (2026) is not a possible certification domain for the swing-up task. The initial-state set is defined around the downward configuration,

$$X_o^{\mathrm{pend}} := \big\{(\theta, \omega) \in X_{\mathrm{pend}} \;\big|\; |\theta - \pi| \le 0.35,\ |\omega| \le 0.5\big\}, \tag{41}$$

and the unsafe set is defined by excessive angular velocity,

$$X_u^{\mathrm{pend}} := \big\{(\theta, \omega) \in X_{\mathrm{pend}} \;\big|\; |\omega| \ge 2.8\big\}. \tag{42}$$

The task therefore requires swing-up from near the downward configuration to the upright while the angular speed stays below $2.8\,\mathrm{rad/s}$; the constraint is reachable (a SAC policy trained for the task swings up with peak speed about $1.9\,\mathrm{rad/s}$) and is violated by the naive maximum-torque policy within two steps.

A.3.2 2D Quadrotor

We consider the planar quadrotor benchmark of Yang et al. (2024): a rigid body with horizontal position x, vertical position z, pitch angle θ and their velocities $\dot{x}, \dot{z}, \dot{\theta}$, controlled by the two rotor thrusts u_1, u_2,

$$\ddot{x} = -\frac{1}{m}\sin(\theta)\,(u_1 + u_2), \qquad \ddot{z} = \frac{1}{m}\cos(\theta)\,(u_1 + u_2) - g, \qquad \ddot{\theta} = \frac{\ell}{I}\,(u_1 - u_2), \tag{43}$$

with $m = 0.486\,\mathrm{kg}$, $\ell = 0.25\,\mathrm{m}$, $I = 0.00383\,\mathrm{kg\,m^2}$ and $g = 9.81\,\mathrm{m/s^2}$. The state is $x = [x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}]^\top$, the pitch angle is a raw coordinate on $[-\pi, \pi]$, and the equilibrium is the stationary hover $x^\star = 0_6$ with $u^\star = [u_{\mathrm{eq}}, u_{\mathrm{eq}}]^\top$, $u_{\mathrm{eq}} = mg/2$. The thrusts are constrained to $u_i \in [0, 2.5\,u_{\mathrm{eq}}]$. The dynamics are discretized with explicit Euler at $\Delta t = 0.05\,\mathrm{s}$. The verification region is

$$X_{\mathrm{quad}} := \big\{x \in \mathbb{R}^6 \;\big|\; (x, z) \in [-5, 5]^2,\ \theta \in [-\pi, \pi],\ (\dot{x}, \dot{z}) \in [-4, 4]^2,\ \dot{\theta} \in [-3, 3]\big\}. \tag{44}$$

The initial-state set contains moderate position, attitude and velocity perturbations of the hover,

$$X_o^{\mathrm{quad}} := \Big\{x \in X_{\mathrm{quad}} \;\Big|\; (x, z) \in [-1.5, 1.5]^2,\ |\theta| \le \tfrac{\pi}{8},\ (\dot{x}, \dot{z}) \in [-0.5, 0.5]^2,\ |\dot{\theta}| \le 0.4\Big\}, \tag{45}$$

and the unsafe set bounds displacement, tilt and speeds,

$$X_u^{\mathrm{quad}} := \Big\{x \in X_{\mathrm{quad}} \;\Big|\; |x| \ge 4.5 \vee |z| \ge 4.5 \vee |\theta| \ge \tfrac{2\pi}{3} \vee |\dot{x}| \ge 3.5 \vee |\dot{z}| \ge 3.5 \vee |\dot{\theta}| \ge 2.7\Big\}. \tag{46}$$

The task requires recovery to the hover from every state of X_o^{quad} without leaving the displacement, tilt and speed limits; the three metres between the initial box and the unsafe walls leave room for the transient of a learned controller.

A.3.3 3D QUADROTOR

The 3D quadrotor is a rigid body with world position $p = [x, y, z]^\top$, ZYX Euler angles $\eta = [\phi, \theta, \psi]^\top$ (roll, pitch, yaw), world velocity v and body angular rate $\omega = [\omega_x, \omega_y, \omega_z]^\top$, the twelve-dimensional Euler-angle model of the quadrotor benchmark of the Generalized Policy Stability repository, with the parameters used there: $m = 0.468\,\mathrm{kg}$, arm length $\ell = 0.225\,\mathrm{m}$, $J = \mathrm{diag}(4.9 \times 10^{-3}, 4.9 \times 10^{-3}, 8.8 \times 10^{-3})\,\mathrm{kg\,m^2}$ and $g = 9.81\,\mathrm{m/s^2}$. The four per-motor thrusts $u_i \in [0, 2.5\,u_{\mathrm{eq}}]$, $u_{\mathrm{eq}} = mg/4$, produce the total thrust $T = \sum_i u_i$ and the body torques

$$\tau_x = \ell\,(u_2 - u_4), \qquad \tau_y = \ell\,(u_3 - u_1), \qquad \tau_z = k_z\,(u_1 - u_2 + u_3 - u_4), \qquad k_z = 1.1/29, \tag{47}$$

and the continuous-time dynamics are

$$\dot{p} = v, \qquad \dot{v} = \frac{T}{m}\,R(\eta)e_3 - ge_3, \qquad \dot{\eta} = T(\phi, \theta)\,\omega, \qquad \dot{\omega} = J^{-1}\big(\tau - \omega \times J\omega\big), \tag{48}$$

where $R(\eta) = R_z(\psi)R_y(\theta)R_x(\phi)$ is the rotation from body to world frame, $e_3 = [0, 0, 1]^\top$, and $T(\phi, \theta)$ maps body rates to Euler-angle rates,

$$T(\phi, \theta) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix}, \tag{49}$$

which is singular at $|\theta| = \pi/2$; the verification region therefore caps the pitch at $|\theta| \le 0.45\pi$ and the unsafe set contains the cap. The state is $x = [p^\top, \eta^\top, v^\top, \omega^\top]^\top \in \mathbb{R}^{12}$, roll and yaw are periodic coordinates, and the equilibrium is the stationary hover at the origin, $x^\star = 0_{12}$ with $u^\star = u_{\mathrm{eq}}\mathbf{1}_4$. One step of the model is a semi-implicit Euler step at $\Delta t = 0.01\,\mathrm{s}$: the rates are updated first, $\omega^+ = \omega + \dot{\omega}\Delta t$, $v^+ = v + \dot{v}\Delta t$, and the poses with the updated rates, $\eta^+ = \eta + T(\phi, \theta)\omega^+\Delta t$, $p^+ = p + v^+\Delta t$. The verification region is

$$X_{\mathrm{quad3D}} := \Big\{ x \in \mathbb{R}^{12} \;\Big|\; p \in [-5, 5]^3,\ \phi, \psi \in [-\pi, \pi],\ |\theta| \le 0.45\pi,\ v \in [-4, 4]^3,\ \omega \in [-3, 3]^3 \Big\}. \tag{50}$$

Attitude constraints are stated through the tilt angle of the body z-axis from the vertical, $\cos(\mathrm{tilt}) = \cos\phi\cos\theta$. The initial-state set is

$$X_o^{\mathrm{quad3D}} := \Big\{ x \in X_{\mathrm{quad3D}} \;\Big|\; \|p\|_\infty \le 1.5,\ \|v\|_\infty \le 0.5,\ \|\omega\|_\infty \le 0.4,\ \cos\phi\cos\theta \ge \cos\tfrac{\pi}{12} \Big\}, \tag{51}$$

i.e. hover perturbations of up to $1.5\,\mathrm{m}$, $0.5\,\mathrm{m/s}$, $0.4\,\mathrm{rad/s}$ and 15° of tilt, and the unsafe set is

$$X_u^{\mathrm{quad3D}} := \Big\{ x \in X_{\mathrm{quad3D}} \;\Big|\; \|p\|_\infty \ge 4.5 \vee \|v\|_\infty \ge 3.5 \vee \|\omega\|_\infty \ge 2.7 \vee \cos\phi\cos\theta \le 0 \vee |\theta| \ge 0.45\pi \Big\}, \tag{52}$$

which forbids leaving the flight volume, excessive speeds, turning upside down, and the pitch range where the Euler-angle model is invalid. In the tools the roll and yaw angles enter every network through their cosine and sine, so that all models are periodic in them and the verifiers can bound them on $[-\pi, \pi]$.

A.4 PROMPT FOR EACH BASELINE

A.4.1 BARRIER–LYAPUNOV AGENT WITH TOOLS

Barrier–Lyapunov Agent Prompt: Ready-to-Call Tools

Act as Barrier–Lyapunov Agent. Read `BARRIER_LYAPUNOV_AGENT_READY_TOOLS.md` in full before starting. The instructions below override any conflicting instructions concerning tool access, workspace, computational budget, or reduction of the initial set.

Tool access. All required tools, including SAC training, certificate synthesis, formal verification, controller refinement, and evaluation, are implemented and ready to call. Inspect their actual interfaces and invoke them directly. Do not implement, replace, or modify the tools. Thin argument preparation, artifact loading, invocation, and output serialization are permitted.

Tasks and workspace. Complete all dynamics tasks supplied in the main folder. Locate each dynamics specification by its task name. Use the main folder as the run directory. The supplied evaluator is located in `./general_evaluation_tool`. Use task-specific artifact names and preserve each task's reports before subsequent calls overwrite them.

Goal. For every task, train and refine a neural controller whose closed-loop trajectories are formally certified to converge to the specified equilibrium while avoiding unsafe states for every initial state in the prescribed set $\mathcal{X}_0$. Seek a large certified region while maintaining coverage of the entire initial set. The initial set $\mathcal{X}_0$ is fixed. Do not shrink it, exclude any of its states, or modify the task specifications to make certification easier. Certification of only a subset of $\mathcal{X}_0$ does not constitute task completion.

Workflow. Execute the following steps for each task:

1. Read the dynamics and task constraints. Create a task-specific objective file recording the fixed initial set $\mathcal{X}_0$, unsafe set, equilibrium, clipping limits, state and observation conventions, and model parameters.
2. Call the SAC training tool to obtain an initial controller. Save the deterministic actor, critics, preprocessing, action transformations, and loading configurations.
3. Call generalized Lyapunov synthesis and formal verification. Use counterexamples for certificate retraining or controller refinement. For UNKNOWN, reconfigure or partition the verification problem without omitting any required part of the certification scope.
4. Call barrier synthesis and formal barrier verification. Use joint controller refinement when necessary. After every controller change, reverify both stability and safety.
5. If certification remains unsuccessful, revise the controller, certificate candidates, training configurations, or verifier settings. Verification partitions must collectively cover the required scope; do not discard difficult states or reduce $\mathcal{X}_0$.
6. Continue until both guarantees are formally established for every initial state in $\mathcal{X}_0$. There is no user-imposed computational budget. Adapt unsuccessful attempts rather than repeating identical calls indefinitely. If a genuine blocker or demonstrated infeasibility prevents completion, report the task as uncertified, record the evidence, and continue with the other tasks.
7. Save and reload the selected controller, complete Lyapunov function, constant step weights, barrier function, certificate parameters, fixed initial set, certified-set specifications, and formal verification reports. Include evidence that the certified scope covers all of $\mathcal{X}_0$.
8. Invoke the existing `general_evaluation_tool.evaluate(...)` on these saved artifacts, using the main folder as its workspace and the full prescribed $\mathcal{X}_0$ as its initial set. Do not implement a separate final evaluator or modify the supplied one.
9. Inspect both the returned Lyapunov veto and unsafe-rollout results. Investigate any in-scope contradiction, refine and reverify as needed, and invoke the same evaluator again whenever the final artifacts change.
10. Preserve the evaluator-generated report, figures, and returned outputs under task- and attempt-specific names before processing the next task.

Certification rules. Use constant step weights compatible with the supplied evaluator. Do not substitute weights after verification without verifying the new controller–certificate combination. Never interpret UNKNOWN or successful sampled tests as formal certification. Do not change the initial set, target, unsafe set, dynamics, or actuator limits to make a task pass.
The supplied evaluator is empirical: it does not invoke formal verifiers or evaluate a barrier network through its main entry point. Preserve formal barrier verification evidence separately.

Final response. Summarize all tasks, including the fixed initial sets, certified regions, evidence of full initial-set coverage, exact guarantees, artifact paths, formal verification results, evaluator-generated reports, veto and safety findings, and measured resource usage. If full certification is not established, report the task as uncertified and describe partial results separately. Explain any unresolved blocker.

Proceed autonomously. Ask only for essential missing information that cannot be resolved from the supplied files or tool interfaces. Do not stop at a plan, at model export, or before the mandatory evaluation of every evaluable final bundle.

A.4.2 BARRIER–LYAPUNOV AGENT WITHOUT TOOLS

Barrier–Lyapunov Agent: Execution Prompt

Act as Barrier–Lyapunov Agent. Read `BARRIER_LYAPUNOV_AGENT.md`, `EVALUATION_INSTRUCTIONS.md`, and `OUTPUT_CONTRACT.md` before starting. The instructions below override any conflicting instructions concerning tool access, workspace, computational budget, or reduction of the initial set.

Tasks and workspace. Complete all dynamics tasks provided in the main folder. Locate each dynamics specification by its task name. Use the main folder as the run directory and save task-specific artifacts to prevent overwriting. The supplied evaluation package is located in `./general_evaluation_tool`.

Goal. For every task, train and refine a neural controller whose closed-loop trajectories are formally certified to converge to the specified equilibrium while avoiding unsafe states for every initial state in the prescribed set $\mathcal{X}_0$. Seek a large certified region while maintaining coverage of the entire initial set. The initial set $\mathcal{X}_0$ is fixed. Do not shrink it, exclude any of its states, or modify the task specifications to make certification easier. Certification of only a subset of $\mathcal{X}_0$ does not constitute task completion.

Tool access. You may use relevant online and local tools, repositories, libraries, and documentation. You may install dependencies and write training, verification, and integration code as needed. Record the sources, versions, and configurations. Final evaluation must use the supplied `general_evaluation_tool`; do not modify, replace, or reimplement it. A thin loading and invocation adapter is permitted.

Workflow. For each task:

1. Read the dynamics and task specification. Create a task-specific objective file recording the state ordering, observation map, timestep, equilibrium, clipping limits, fixed initial set $\mathcal{X}_0$, and unsafe set.
2. Train an initial controller using SAC. Save the actor, critics, architectures, preprocessing, action transformations, and training configuration. Use deterministic actor inference for verification and evaluation.
3. Construct and formally verify a generalized Lyapunov candidate. Use a value-function representation and constant step weights compatible with the tools and final evaluator.
4. Use counterexamples to retrain the certificate or refine the controller. For UNKNOWN, reconfigure or partition the verification problem without omitting any required part of the certification scope. Never interpret UNKNOWN as VERIFIED.
5. Construct and formally verify a barrier certificate under the clipped controller. Use feedback for barrier retraining or joint refinement. After every controller update, reverify both stability and safety.
6. If certification remains unsuccessful, revise the controller, certificates, training configurations, or verification strategy. Keep $\mathcal{X}_0$ fixed and preserve the target, unsafe set, dynamics, actuator limits, and required guarantees. Verification partitions must collectively cover the required scope.
7. Continue until both guarantees are formally established for every initial state in $\mathcal{X}_0$. There is no user-imposed computational budget. Adapt unsuccessful attempts rather than repeating identical calls indefinitely. If a genuine blocker or demonstrated infeasibility prevents completion, report the task as uncertified, record the evidence, and continue with the other tasks.
8. Export and reload the selected controller, complete Lyapunov function, constant step weights, barrier function, fixed initial set, certified regions, configurations, and formal verification records. Include evidence that the certified scope covers all of $\mathcal{X}_0$.
9. Invoke the supplied `general_evaluation_tool.evaluate(...)` on the exported artifacts, using the main folder as its workspace and the full prescribed $\mathcal{X}_0$ as its initial set. Preserve each task's evaluator-generated report, figures, and returned outputs before evaluating the next task.
10. Inspect the evaluator's veto, sampled certificate checks, and unsafe-rollout results. An in-scope contradiction requires withholding the affected certification claim, investigation, and, where possible, refinement and reverification. Evaluate every updated final bundle again using the same tool. Passing empirical tests does not establish formal certification.

Final response. For every task, report the certification outcome, fixed initial set, certified regions, evidence of full initial-set coverage, exact guarantees, limitations, artifact paths, formal verification records, evaluator reports, veto status, and measured execution time and token usage. If full certification is not established, report the task as uncertified and describe partial results separately.

Proceed autonomously. Ask only for essential information unavailable from the supplied files. Do not stop at a plan or artifact export: execute the workflow and invoke the supplied evaluator for every evaluable final bundle.

A.4.3 AGENT WITHOUT TOOLS

Prompt for the Agent Without Specialized Tools

Train and formally certify a neural network controller for the specified control task.
The task dynamics and specifications are located in the main folder under the task name. Use the main folder as the working directory. Read the task files to identify the system dynamics, observation map, control objective, target equilibrium, controller-output clipping, initial set, and unsafe set.
The only provided task-specific tool available to you is `general_evaluation_tool`. Do not access the provided controller-training, certificate-synthesis, verification, or controller-refinement tools, their implementations, or the Barrier–Lyapunov Agent workflow instructions. You may use general-purpose file access, code execution, scientific computing libraries, and online resources to develop your own solution. Independently determine your method and execution strategy.
Complete the task autonomously. Train an initial controller using SAC. Your objective is to obtain a controller that is formally certified to satisfy both stability and safety requirements for every initial state in the prescribed set $\mathcal{X}_0$, while seeking a large certified region.
The initial set $\mathcal{X}_0$ is fixed: do not shrink it, exclude any of its states, or modify the task specifications to make certification easier. Certification of only a subset of $\mathcal{X}_0$ does not constitute task completion. This requirement overrides any conflicting instruction that permits reducing the initial set. There is no prescribed computational budget.
Return:

1. The trained controller, including its architecture, parameters, observation preprocessing, and output-clipping configuration.
2. A Lyapunov function and all parameters needed to evaluate its certified decrease condition.
3. A barrier function and all parameters needed to evaluate its certified safety conditions.
4. The certified sets and formal verification evidence, identifying the exact controller, system model, certificates, and assumptions to which the guarantees apply, together with evidence that the certified scope covers all of $\mathcal{X}_0$.
5. Reproducible code, configurations, execution logs, and a concise report describing the achieved guarantees and any unresolved limitations.

After exporting the final artifacts, call the provided `general_evaluation_tool` in the `general_evaluation_tool` folder using its documented interface. Evaluate the full prescribed initial set $\mathcal{X}_0$. Do not implement, modify, or replace the final evaluator yourself. Include its outputs in your final report.
Empirical success alone must not be presented as formal certification. If certification cannot be established for the entire prescribed initial set $\mathcal{X}_0$, report the task as uncertified and describe any partial results separately.

A.4.4 AGENT WITH TOOLS

Prompt for the Agent Without Workflow Instructions

Train and formally certify a neural network controller for the specified control task.
The task dynamics and specifications are located in the main folder under the task name. Use the main folder as the working directory. Read the task files to identify the system dynamics, observation map, control objective, target equilibrium, controller-output clipping, initial set, and unsafe set.
All provided tools are available to you. Inspect their documentation and interfaces, and independently decide which tools to use, in what order, and with what configurations. You may also use online resources and additional tools.

Complete the task autonomously. Train an initial controller using SAC. Your objective is to obtain a controller that is formally certified to satisfy both stability and safety requirements for every initial state in the prescribed set $\mathcal{X}_0$, while seeking a large certified region. The initial set $\mathcal{X}_0$ is fixed: do not shrink it, exclude any of its states, or modify the task specifications to make certification easier. Certification of only a subset of $\mathcal{X}_0$ does not constitute task completion. There is no prescribed computational budget.
Return:

1. The trained controller, including its architecture, parameters, observation preprocessing, and output-clipping configuration.
2. A Lyapunov function and all parameters needed to evaluate its certified decrease condition.
3. A barrier function and all parameters needed to evaluate its certified safety conditions.
4. The certified sets and formal verification reports, identifying the exact controller, system model, certificates, and assumptions to which the guarantees apply.
5. Reproducible configurations, execution logs, and a concise report describing the achieved guarantees and any unresolved limitations.

After exporting the final artifacts, call the provided `general_evaluation_tool` in the `general_evaluation_tool` folder using its documented interface. Do not implement, modify, or replace the final evaluator yourself. Include its outputs in your final report. Empirical success alone must not be presented as formal certification. If certification cannot be established for the entire prescribed initial set $\mathcal{X}_0$, report the task as uncertified and describe any partial results separately.