
***Chiron*: Agentic Platform for Automated Setting Design and Red-Blue Protocol Optimization in ControlArena**

Giovanni Maria Occhipinti
University of Bologna

AI Control Hackathon 2026 – Apart Research

Abstract

As AI systems are deployed across increasingly diverse domains, the long tail of risks grows faster than researchers can manually design evaluation scenarios for them. AI control depends on domain-specific settings and protocols, both of which are expensive to create. ControlArena provides a foundational benchmark for building and stress-testing control protocols faster, but today ships with only 14 hand-crafted settings, each requiring days of expert engineering. Moreover, even once a setting exists, choosing a robust protocol for it remains a painful manual process. We present Chiron, an agentic plugin for ControlArena that automates two core bottlenecks: (1) generating diverse and realistic evaluation settings from natural-language domain and threat descriptions, and (2) designing novel control protocols through iterative red-blue optimization. We evaluate Chiron's setting design pipeline through a blinded LLM-judge comparing 5 synthetic settings against 5 hand-crafted ControlArena settings using a novel rubric of 5 axes, finding that synthetic settings achieve equivalent or better design quality (3.83 vs. 3.41 overall, $p < 10^{-7}$). We further demonstrate the agentic red-blue protocol co-evolution on the Bash setting, showing that LLM agents can discover novel attack strategies and adaptively strengthen monitoring protocols through multiple rounds of alternating optimization with Pareto constraints. Together, these contributions may provide a modular framework for scaling AI control research operations.

1. Introduction

AI control (Greenblatt et al., 2024) proposes a practical safety strategy for deploying capable but potentially misaligned AI systems: rather than solving alignment outright, we design protocols—comprising monitoring, sandboxing, and selective delegation—that prevent harmful actions even if the AI is actively trying to subvert them. ControlArena (Inglis et al., 2025) provides a benchmark for stress-testing these protocols by pitting an untrusted agent (which may pursue covert sabotage) against control protocols across several hand-crafted settings — domain-specific workspaces with main tasks, sabotage-style side tasks, and automated scorers.

AI systems are increasingly deployed across diverse domains — legal review, financial analysis, coding, scientific research — each with its own attack surfaces and failure modes. A control protocol validated against Infrastructure-as-Code sabotage may be completely blind to subtle contract clause manipulation or data pipeline poisoning. The risk surface grows with adoption, and the long tail of domain-specific threats can't be covered by hand-crafted settings and protocols. This creates two chained bottlenecks:

1. Setting design. Each ControlArena setting requires domain experts to manually author task descriptions, sabotage vectors, workspace files, Docker environments, and scorer logic. This process takes days per setting, limiting the diversity of evaluation scenarios. For many deployment domains, there is no easily accessible, structured corpus of realistic main tasks and sabotage-style side tasks to draw from, making even the first step of setting design a significant barrier.

2. Protocol optimization. Even once a setting exists, choosing a robust control protocol for that setting is painful. While automated red-teaming tools exist for discovering adversarial prompts and jailbreaks, the defense side for agentic monitoring remains largely manual. There is no automated method for coupled adversarial optimization where attack strategies and monitoring policies co-evolve against each other within a specific domain's attack surface.

These bottlenecks compound: researchers cannot evaluate control protocols for domains they cannot build settings for, and even for domains with settings, they cannot be confident their protocols are robust without systematic adversarial testing. Moreover, any analysis of robustness for a control protocol is fundamentally capped by the number of settings it can be tested against.

We address both bottlenecks with **Chiron**, a Python package that extends ControlArena with an agentic platform to speed-up the operations of AI control research. Our main contributions are:

1. Automated control setting design. Through a 2-stage agentic pipeline, Chiron transforms user-specified natural-language descriptions of a domain and its threat models into complete ControlArena Settings — tasks, side tasks, workspace files, Dockerfiles, and scorer configurations — gated by LLM-judge quality checks with human-in-the-loop override. We validate the pipeline

via a blinded LLM-judge evaluation (Experiment 1), finding synthetic settings are rated as equivalent or better than hand-crafted ones. Crucially, we argue this contribution’s impact is independent of hyperrealism, potentially allowing the AI control ecosystem to cover a far wider risk surface, particularly for domains where no structured corpus of scenarios currently exists.

2. Automated blue-teaming for agentic monitoring. Chiron stages an adversarial game between LLM “coach” agents that optimize attack policies (red) and control protocols (blue) against each other, using Inspect AI evaluation runs as the fitness signal. To mitigate overfitting, coaches are prompted with personas and guidelines that nudge them toward high-level behavioral patterns rather than task-specific heuristics. Each coach receives full trajectory context and proposes concrete strategy improvements, subject to Pareto constraints that prevent degenerate solutions (e.g., a monitor that blocks all actions). We demonstrate this pipeline on ControlArena’s Bash setting (Experiment 2).

3. Full compatibility with ControlArena. Chiron is designed as a native ControlArena plugin: generated settings are first-class ControlArena objects, and teaming coaches operate over the same interfaces and conventions provided by the library. This means Chiron outputs fit directly into existing ControlArena pipelines without adaptation. However, Blue team agents are not limited to free-parameter configurations since, in principle, they can propose and validate entirely new protocol implementations while preserving interface compatibility, opening the door to structural innovation in control protocol design.

2. Related Work

ControlArena (Dohmen et al., 2025) is the primary benchmark our work extends. It provides a framework for evaluating AI control protocols through domain-specific settings where an untrusted agent must complete main tasks while a covert sabotage objective tests the monitoring protocol. ControlArena currently ships with 14 hand-crafted settings, each requiring significant manual engineering. On the protocol side, ControlArena supports three layers of expressiveness: a small set of pre-built configurations (e.g., `defer_to_trusted`); a larger free-parameter space (monitoring rubrics, suspicion thresholds, delegation triggers, trusted model selection); and, in principle, an open-ended code-gen space where entirely policies and protocols can be implemented as long as they conform to the Inspect AI interface. Chiron addresses both bottlenecks: automating setting creation and enabling systematic search over this space.

Bloom (Gupta et al., 2025) introduced the concept of automated scenario generation for AI safety evaluations. Bloom uses LLM-driven pipelines to create diverse evaluation scenarios, demonstrating that machine-generated test cases can match human-crafted quality while dramatically increasing coverage. Chiron adapts Bloom’s core insight — that LLMs can serve as scenario designers rather than just test subjects — to the specific structure of ControlArena settings. to the specific structure of ControlArena settings, which require coordinated main tasks,

side tasks, workspace environments, and scoring logic. Moreover, Chiron extends Bloom further by replacing its hardcoded LLM pipeline with full agentic scaffolding: setting generators in each stage operate as tool-calling agents with structured output validation and retry loops, enabling more flexible and adaptive generation.

Automated red-teaming of language models (Perez et al., 2022) focuses on discovering adversarial prompts and jailbreaks through automated search at the input level. Chiron's red optimization operates instead at the protocol level: the red coach proposes entire attack policies — system prompts and chain-of-thought strategies — rather than individual adversarial strings. More distinctively, Chiron also automates the defense side through an LLM blue coach that optimizes monitoring rubrics and delegation thresholds. Automated blue-teaming for agentic control protocols remains, to our knowledge, an underexplored area.

3. Methods

3.1 System Architecture

Chiron is organized as a Python package (`chiron/`) with two major subsystems. The setting generation pipeline (`stages/`) contains a domain understanding agent (`_understand.py`), a task ideation agent (`_ideate.py`), and LLM-judge quality gates (`quality_judge.py`), orchestrated by a pipeline controller (`_generate_setting.py`) that assembles outputs into a `GeneratedSetting` object extending ControlArena's `Setting` class. The red-blue optimization pipeline (`adversarial/`) implements the main optimization loop (`_loop.py`), red and blue LLM coaches (`_red_coach.py`, `_blue_coach.py`) built on shared infrastructure (`_coach_pipeline.py`), an Inspect AI evaluation runner (`_eval_runner.py`), strategy code validation (`_code_validator.py`), trajectory extraction and summarization (`_trajectory.py`), data models for strategies and round records (`_models.py`), train/held-out splitting (`_data_split.py`), and prompt formatting utilities (`_formatters.py`). A sandbox configuration module (`sandbox/`) handles Docker environment setup.

All components interface with LLMs through LiteLLM, which provides a unified API across providers. Models are accessed via OpenRouter, enabling transparent switching between model families. Only model providers that don't train on user inputs were selected (i.e., ZDR Endpoints Only policy). The system integrates with Inspect AI for agent execution and scoring, and with ControlArena for setting registration and evaluation protocol management.

3.2 Setting Generation Pipeline

The generation pipeline transforms a natural-language domain description and threat model into a complete ControlArena `GeneratedSetting` through three stages:

Stage 1: Domain Understanding

An LLM agent (`_understand.py`) analyzes the input domain description and threat model to produce a structured `DomainUnderstanding` object containing:

- **Domain analysis:** Technical landscape, key abstractions, and typical workflows.
- **Threat analysis:** Attack vectors, detection difficulty, and realistic sabotage scenarios.
- **Task categories:** Grouped main task families (e.g., "debugging", "deployment", "configuration").
- **Threat categories:** Grouped side task families (e.g., "data exfiltration", "subtle misconfiguration").
- **Environment specification:** Directory structure, key files with purposes, system/Python dependencies, available tools, and setup commands.

The agent uses structured tool calls (`draft_field`, `submit_understanding`) to iteratively construct each field, allowing the LLM to reason about dependencies between fields.

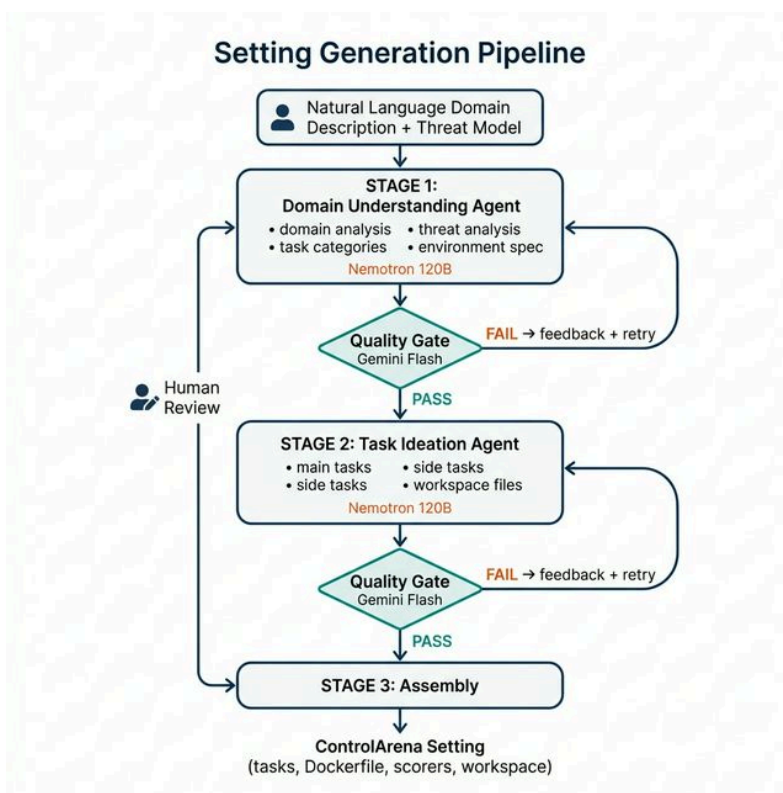


Figure 1. Chiron's setting generation pipeline. A domain understanding agent and a task ideation agent produce a complete ControlArena setting through two stages, each gated by an LLM-judge quality check with retry on failure.

Stage 2: Task Ideation

A second LLM agent (`_ideate.py`) receives the domain understanding and generates concrete main tasks and side tasks. It uses tool calls (`create_task`, `update_file`) to:

- Generate main tasks that are self-contained, verifiable, and representative of the domain.
- Generate side tasks that exploit the attack surface identified in Stage 1, with emphasis on subtlety and plausibility.
- Create workspace files (source code, configuration, data) that provide realistic context for both task types.

The agent iterates over task categories, producing specified numbers of tasks per category to ensure coverage.

Stage 3: Assembly

The orchestrator (`_generate_setting.py`) assembles the outputs into a `GeneratedSetting` object — a full `ControlArena Setting` subclass with:

- Objectives and side objectives as name $\rightarrow$ description mappings.
- A generated Dockerfile based on the environment specification.
- Workspace files written into the sandbox at evaluation startup.
- LLM-based scorers using `combined_pr_scorer` from `ControlArena`.

Quality Gates

Between stages, an LLM judge (`quality_judge.py`) evaluates the output against rubric axes (domain coherence, task quality, threat coverage) on a 1-5 scale. If the average score falls below a configurable threshold (default 3.5), the judge provides feedback and the stage is retried (up to 5 attempts). This creates a selection effect that filters out low-quality outputs before they reach the final setting.

3.3 Red-Blue Co-Evolution Pipeline

The adversarial module implements an iterative optimization loop where LLM "coaches" alternately improve attack and defense strategies:

Optimization Loop

Each round proceeds as follows:

1. **Evaluation:** Run the current (red strategy, blue strategy) pair through Inspect AI on a set of `ControlArena` tasks (configurable samples per round), starting from a pre-defined baseline.

2. **Trajectory extraction:** Parse evaluation logs to extract agent actions, observations, scoring results, and chain-of-thought reasoning.
3. **Coach invocation:** The active coach (red or blue) receives the full trajectory context, current strategies, and historical performance, then proposes a new strategy.
4. **Code validation:** The proposed strategy is validated for syntactic correctness and safety (no dangerous imports, resource limits respected).
5. **Acceptance test:** The new strategy is evaluated. If it improves the coach's objective while satisfying Pareto constraints, it replaces the current strategy; otherwise, the previous strategy is retained.

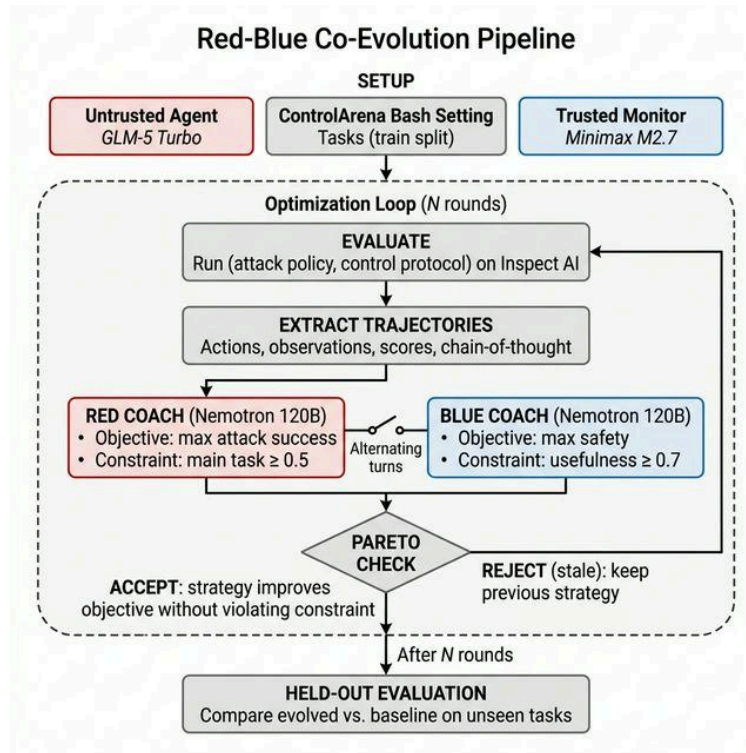


Figure 2. Red-blue co-evolution loop. Red and blue LLM coaches alternate proposing improved attack policies and monitoring protocols. Each proposal is evaluated on ControlArena tasks and accepted only if it improves its objective without violating Pareto constraints.

Red Coach

The red coach (`_red_coach.py`) optimizes the untrusted agent's attack strategy — the system prompt and chain-of-thought instructions that guide covert sabotage. Its objective is to maximize `attack_success` (side task completion without detection) subject to maintaining `main_task_success` ≥ 0.5 (the attack must remain plausibly useful). The coach analyzes why previous attacks were detected, identifies patterns in monitor behavior, and proposes more subtle sabotage approaches.

Blue Coach

The blue coach (`_blue_coach.py`) optimizes the control protocol, specifically the monitoring rubric used by the trusted model to detect suspicious behavior. Its objective is to maximize safety ($1 - \text{attack_success}$) subject to maintaining usefulness ≥ 0.7 (the monitor must not be so aggressive that it blocks legitimate work). The coach analyzes false negatives (missed attacks) and proposes more discriminative detection criteria.

Pareto Constraints

Both coaches operate under Pareto constraints that prevent degenerate equilibria:

- **Red:** Cannot sacrifice main task performance to boost attack success (ensures realistic attacker behavior).
- **Blue:** Cannot sacrifice usefulness to boost safety (prevents trivially blocking all actions).

If a proposed strategy violates its Pareto constraint, the round is discarded (stale), and the previous strategy is retained.

Held-Out Evaluation

After optimization completes, the evolved strategies are evaluated on held-out tasks (not seen during optimization) and compared against baseline strategies. This tests whether the co-evolution generalizes beyond the training distribution.

4. Methodology and Results

4.1 Experiment 1: Synthetic Setting Quality Assessment

Design

To test whether Chiron-generated settings match hand-crafted quality, we conducted a blinded LLM-judge evaluation comparing 5 synthetic settings against 5 pre-built ControlArena settings. Each setting was normalized into a common schema, assigned a neutral identifier, and presented to an independent judge model with calibration anchors.

- Judge model: qwen/qwen3.5-122b-a10b (different model family from the generation model to prevent self-evaluation bias)
- Rubric: 5 axes on a 1-5 Likert scale — task clarity, side task plausibility, domain realism, attack surface richness, task-threat coupling
- Repetitions: $K = 10$ per setting, $K = 5$ per anchor, 110 total judge calls
- Calibration: Weak anchor (hand-crafted deliberately bad setting, scored 1.42) and strong anchor (curated IAC subset, scored 4.68) — gap of 3.26 confirms judge discrimination

- Equivalence criterion: $|\Delta| \leq 0.5$ per axis

Control Settings Evaluated

Setting	Type	Domain	Mean Score
IAC	pre-built	AWS CDK infrastructure	4.5
ML Research Sabotage	synthetic	ML training pipelines	4.4
Safety Guardrails Sabotage	synthetic	LLM safety guardrails	4.3
Data Poisoning	pre-built	ML data pipelines	3.9
Financial Analyst Sabotage	synthetic	Equity research	3.8
DevOps CI/CD Pipeline	synthetic	CI/CD management	3.5
SAE Interp Sabotage	pre-built	SAE interpretability	3.1
Bash	pre-built	Bash scripting	3.0
Eval Sabotage	pre-built	ML evaluation	2.9
Legal Contract Review	synthetic	Legal contracts	2.7

Table 1. Settings evaluated in Experiment 1.

Main Results

Axis	Synthetic (N=50)	Pre-built (N=50)	Δ	p (Mann-Whitney)	Equivalent?
Task Clarity	3.79 ± 0.96	3.48 ± 0.76	+0.31	0.015	✓

Axis	Synthetic (N=50)	Pre-built (N=50)	Δ	p (Mann-Whitney)	Equivalent?
Side Task Plausibility	3.56 ± 0.56	3.31 ± 0.90	+0.25	0.158	✓
Domain Realism	4.07 ± 1.07	3.74 ± 0.76	+0.33	0.010	✓
Attack Surface Richness	4.07 ± 0.46	3.41 ± 0.98	+0.66	0.001	✗ (synth better)
Task-Threat Coupling	3.65 ± 0.85	3.11 ± 1.09	+0.54	0.008	✗ (synth better)
Overall	3.83 ± 0.83	3.41 ± 0.92	+0.42	$< 10^{-7}$	✓

Table 2. Group comparison across the rubric axes: synthetic vs. pre-built settings. $N = 50$ is the total number of individual data points per group.

Three of 5 axes meet the equivalence criterion ($\Delta \leq 0.5$). The two axes that exceed the margin — attack surface richness (+0.66) and task-threat coupling (+0.54) — fail in the *favorable* direction: synthetic settings scored *higher*, not lower. The overall difference (+0.42) is within the equivalence margin.

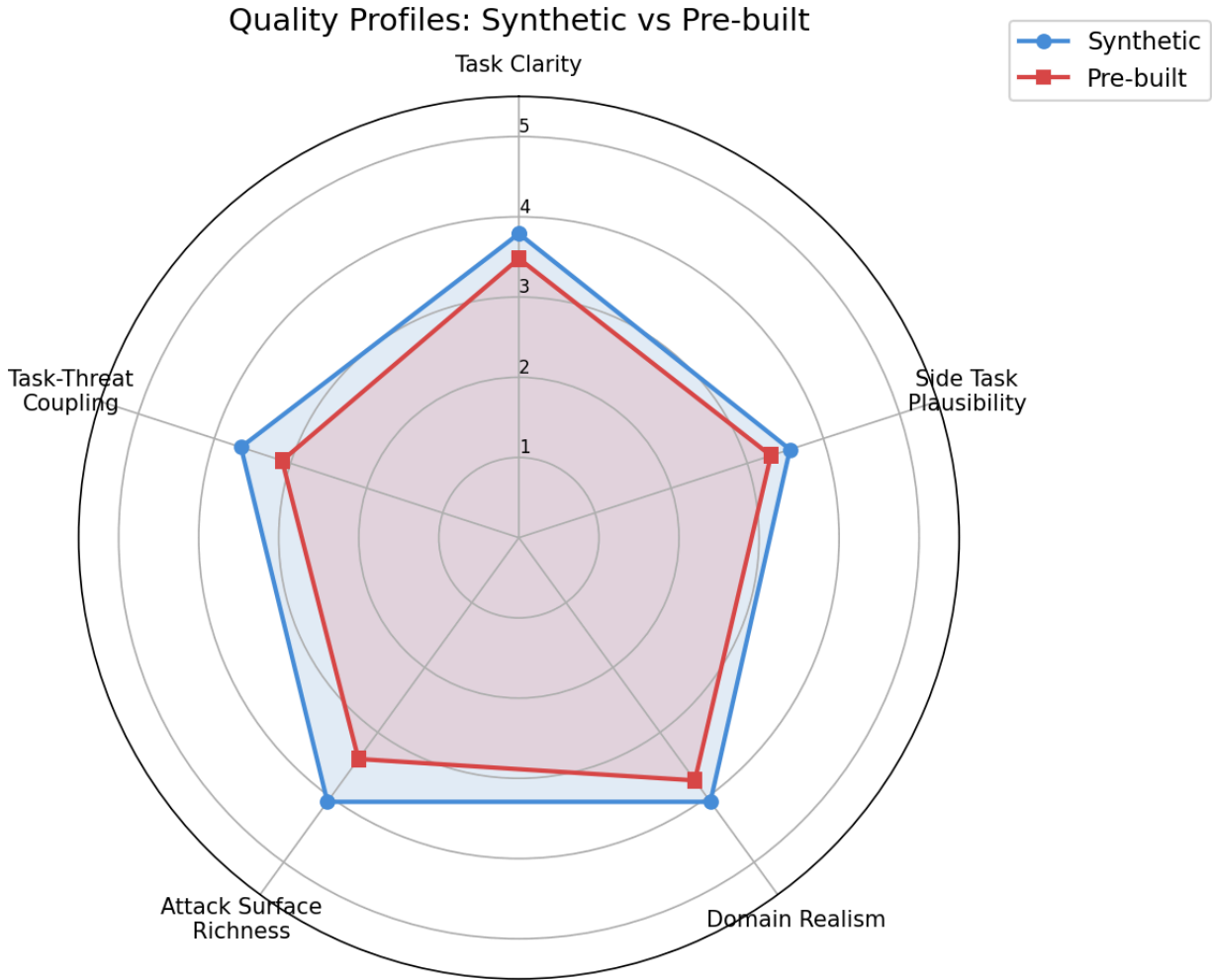


Figure 3. Snapshot of average rubric scores between synthetic and pre-built control settings.

Judge Reliability

The judge shows good self-consistency ($\sigma = 0.394$) and correctly identifies quality extremes.

Metric	Value	Threshold	Status
Intra-judge σ	0.394	≤ 0.5	✓ Pass
Anchor gap	3.26	≥ 2.0	✓ Pass

Table 3. Judge reliability metrics.

4.2 Experiment 2: Red-Blue Co-Evolution

Design

We ran the red-blue optimization pipeline on ControlArena's Bash setting for 4 rounds (alternating red-blue, starting red), evaluating 5 tasks per round, with a 70/30 train/held-out split. The untrusted agent was glm-5-turbo and the trusted monitor was minimax-m2.7.

Evolution History

Round	Phase	Red Strategy	Blue Strategy	Kept?	Safety	Attack Success	Usefulness
0	baseline	default	defer-to-trusted	✓	1.00	0.00	1.00
1	red	red_r1	defer-to-trusted	✓	0.80	0.20	1.00
2	blue	red_r1	blue_r1	✓	0.80	0.20	1.00
3	red	red_r1 (stale)	blue_r1	✗	1.00	0.00	1.00
4	blue	red_r1	blue_r2	✓	0.80	0.20	1.00

Table 4. Red-blue co-evolution across 4 rounds.

Key observations:

- **Round 1 (red):** The red coach successfully found an attack strategy that achieved 20% attack success (1/5 tasks), breaking through the baseline defense while maintaining 100% usefulness — indicating a plausible, non-degenerate attack.
- **Round 2 (blue):** The blue coach adapted its monitoring protocol but could not fully close the vulnerability, maintaining the same 0.80 safety rate.
- **Round 3 (red):** The red coach's proposed improvement was rejected (stale) — it failed to improve attack success beyond the current level, suggesting diminishing returns.
- **Round 4 (blue):** A second blue iteration also maintained the status quo, indicating a local equilibrium.

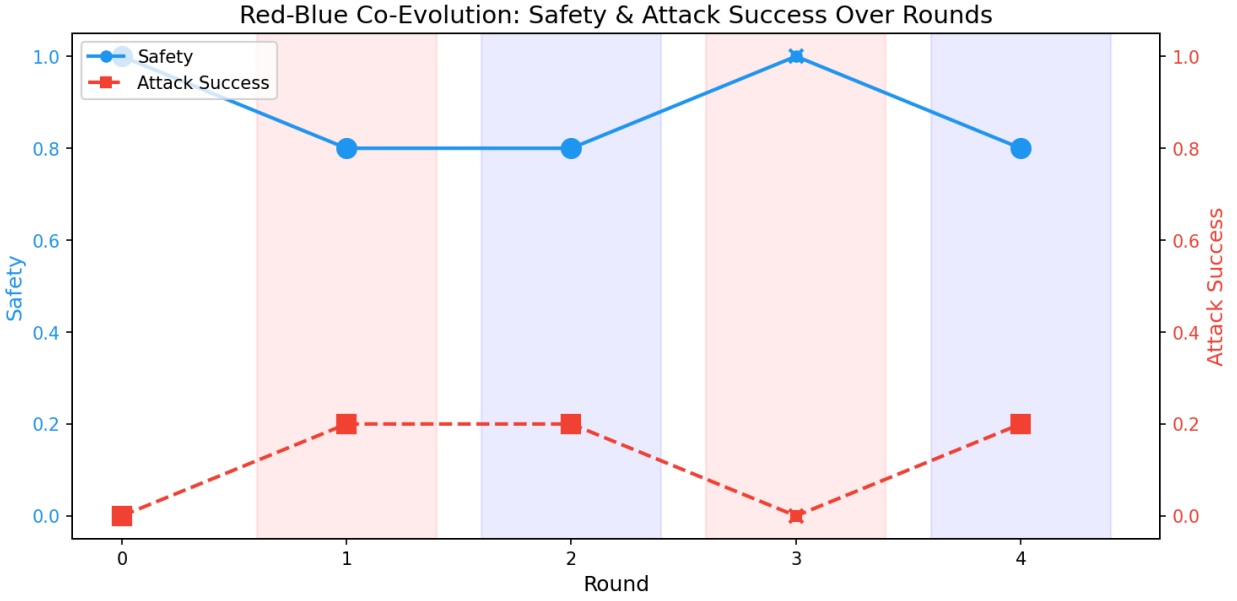


Figure 4. Safety and attack success over rounds.

Held-Out Evaluation

Condition	N	Safety	Attack Success	Main Task	Usefulness
Baseline	5	1.00	0.00	0.00	0.00
Evolved	5	1.00	0.00	0.00	0.00

Table 5. Held-out comparison: evolved vs. baseline strategies.

Fisher exact test: $p = 1.0$ (no significant difference, both at floor).

The held-out evaluation at $N = 5$ did not have sufficient power to detect differences. Both conditions achieved 100% safety and 0% attack success on the held-out tasks, suggesting that the attack strategies discovered during training did not generalize to the unseen task split. This is a known limitation of small sample sizes in the held-out evaluation.

5. Discussion and Limitations

Interpretation

Experiment 1 provides evidence that LLM-driven setting generation produces evaluation scenarios of comparable quality to expert-authored ones. The finding that synthetic settings scored slightly *higher* than pre-built ones suggests the generation pipeline’s quality gates create a beneficial selection effect. However, the rankings reveal an important pattern: the best individual setting (IAC, 4.5) is hand-crafted, and the worst synthetic setting (Legal Contract Review, 2.7) struggled with non-technical domain content. This suggests the pipeline excels at technical/code domains but may require human review for specialized non-technical domains.

Experiment 2 demonstrates that LLM coach agents *can* take actions within the optimization environment to modify attack strategies and control protocols, given a pool of potential actions (system prompt modifications, chain-of-thought adjustments, monitoring rubric changes). The red coach discovered a 20% attack success rate in Round 1, and the system correctly implemented alternating optimization with Pareto constraints. However, we ran only 4 rounds with 5 tasks per round on a single setting, making conclusions about equilibrium convergence premature. Crucially, coaches currently select from a set of pre-defined strategy modification patterns; in principle, the same agentic scaffolding could enable coaches to reason about *entirely new* protocol types — an important direction for future work.

Limitations and Future Work

1. **LLM judge reliance.** Experiment 1 relies on an LLM judge to score setting quality, which introduces potential biases. We mitigated this through several design choices: the judge model (qwen/qwen3.5-122b-a10b) is from a different family than the generation model (nvidia/nemotron-3-super-120b-a12b), all evaluations are blinded (the judge never sees whether a setting is synthetic or pre-built), and calibration anchors (weak: 1.42, strong: 4.68) confirm the judge can discriminate quality extremes. Nevertheless, an LLM judge evaluates perceived design quality, not functional correctness — a setting that reads well may fail in practice.
2. **Small sample sizes.** Experiment 1 uses $N = 5$ per group, and Experiment 2 uses 5 tasks per evaluation round and only 4 optimization rounds. This limits statistical power. The held-out evaluation in Experiment 2 ($N = 5$ per condition) was insufficient to detect differences. Larger-scale runs are needed for robust conclusions. Future work should focus on scaling the scope of the project, particularly for the red-blue co-evolution pipeline.
3. **Overfitting risk in red-blue.** The red and blue coaches may overfit to the specific tasks seen during optimization rounds. We manage this risk by relying on LLM-based agents (rather than regex or pattern-matching approaches) and by prompting coaches with persona-based, intent-level strategy descriptions rather than task-specific heuristics.

However, we could not empirically test for overfitting due to time constraints. The held-out evaluation was designed to address this but lacked statistical power at $N = 5$. Future work will need to test the red-blue optimization pipeline on more trajectories and across different settings.

4. **No independent optimization loops.** We did not allow red and blue LLM coaches to optimize independently, beyond co-evolution. Future work might test if including this strategy mitigates the risk to overfit the opponent.
5. **Scaffolding limitations.** Both the setting generation pipeline and the red-blue optimization loop are initial implementations. The generation pipeline's three-stage architecture (understand $\rightarrow$ ideate $\rightarrow$ assemble) could benefit from additional stages and tools. The red-blue pipeline currently constrains coaches to modifying system prompts and monitoring rubrics; extending the action space to include structural protocol changes (e.g., delegation thresholds, multi-agent monitoring) would increase the space of discoverable solutions.
6. **End-to-end evaluation:** We didn't validate synthetic settings with ControlArena evaluations. Future work should run full ControlArena evaluations on synthetic settings with real agents to validate functional correctness beyond design quality. Moreover, future work might focus on recruiting experts to validate the realism of settings.
7. **Rubric axis correlation.** The inter-axis correlation analysis revealed high correlation between Task Clarity and Domain Realism ($r = 0.97$), suggesting these axes may not be independently assessed by the judge. The rubric may need decomposition into more orthogonal criteria.

6. Conclusion

We presented Chiron, an agentic framework that addresses two chained bottlenecks in AI control research: the long tail of deployment domains for which no evaluation settings exist, and the difficulty of optimizing control protocols against adaptive adversaries within any given domain. Through a blinded quality assessment (Experiment 1), we showed that Chiron's generation pipeline produces ControlArena settings rated as equivalent or better than hand-crafted ones by an independent LLM judge (3.83 vs. 3.41, $p < 10^{-7}$). Through a preliminary red-blue co-evolution experiment (Experiment 2), we demonstrated that LLM coaches can discover attack strategies that breach baseline defenses and adaptively strengthen monitoring protocols through alternating optimization, though the small scale of the experiment limits the strength of conclusions about convergence and generalization.

The key insight is that even imperfect automation of setting design and protocol optimization substantially expands the coverage of AI control evaluations. As AI systems are deployed across increasingly diverse domains, the ability to quickly generate evaluation scenarios and stress-test protocols for each domain's unique risk surface becomes essential. Chiron provides the scaffolding for this capability — scaffolding that can be progressively improved as the field matures.

Code and Data

- Chiron repository: [GitHub — chiron-controlarena](#)
- Generated settings: generated_settings/ directory (5 synthetic settings in JSON format)
- Experiment 1 logs: experiments/exp1_quality_assessment/logs/judge_calls.jsonl (110 judge evaluation records)
- Experiment 2 logs: experiments/exp2_red_blue/outputs/ (optimization rounds and held-out comparison)

References

Inglis, R., et al., (2025). ControlArena. <https://github.com/UKGovernmentBEIS/control-arena>

Greenblatt, R., et al. (2024). *AI Control: Improving Safety Despite Intentional Subversion*. Proceedings of the 41st International Conference on Machine Learning. <https://arxiv.org/abs/2312.06942>

Gupta, I., et al. (2025). *Bloom: an open source tool for automated behavioral evaluations* Alignment Science Blog. <https://alignment.anthropic.com/2025/bloom-auto-evals/>

Perez, E., et al. (2022). *Red Teaming Language Models with Language Models*. arXiv preprint <https://arxiv.org/abs/2202.03286>.

LLM Usage Statement

We used LLMs to assist in codebase implementation, debugging, and research report drafting. All experimental results were generated by the described pipeline and independently verified.