
Resampling With Trusted Comparison¹

Chris Harig
Independent

With
Apart Research

Abstract

Control evaluations measure whether safety protocols can prevent intentionally subversive AI models from causing harm [1]. In multi-step agent environments, resample protocols re-generate suspicious actions, improving safety and usefulness relative to other protocols [2]. We propose resample with trusted comparison, which extends resampling and auditing after three strikes by comparing the suspicion score distributions of untrusted and trusted model samples to make more reliable strike decisions. We evaluate our protocol on a subset of tasks from BashArena [4], a setting of 637 Linux system administration tasks with realistic sabotage “side-tasks”. We also contribute an automated red teaming pipeline that uses a strong model to iteratively rewrite attack transcripts, achieving an 11% convergence rate (successful and undetected attacks) across 72 tasks at an average cost of \$0.31 per task. On the 17 attack transcripts where the protocol actively intervened, RTC blocked 80% of attacks (vs 69% for DTR), while maintaining average usefulness of 0.462 (vs 0.584 for DTR), demonstrating the safety-usefulness tradeoff of trusted comparison.

1. Introduction

As AI agents are deployed autonomously in security-critical settings, the risks that sufficiently capable models pose become increasingly salient. AI control addresses the scheming threat model directly: rather than trying to verify whether a model is aligned, we design and evaluate deployment protocols that ensure safety with the assumption the model is scheming.

Greenblatt et al. [1] introduced control evaluations in a single-step code generation setting, demonstrating that protocols like trusted monitoring can prevent an untrusted model from inserting backdoors. Bhatt et al. [2] extended this to multi-step agent environments in Ctrl-Z, introducing resample protocols that dynamically re-generate actions and incrimination techniques that use patterns of suspicious behavior across a trajectory to make auditing decisions. Their best protocol reduced attack success from 58% to 7% at a 5% usefulness cost.

We build on this work with two contributions:

1. Resample with trusted comparison — a novel control protocol that extends resampling by comparing the suspicion score distributions of untrusted resamples against trusted model samples.

2. An automated red teaming pipeline — a system that uses a strong model to iteratively rewrite failed attack transcripts for stealth and successful attack strategies, producing a pool of realistic adversarial trajectories for control evaluation.

We evaluate our protocol and pipeline on a subset of tasks and attack trajectories in BashArena [4], a control evaluation setting with 637 Linux system administration tasks, implemented within the ControlArena framework.

2. Related Work

Ctrl-Z [2] extended AI control techniques to multi-step agent environments, introducing resampling and downstream protocols (like resampling with incrimination in the present) that take advantage of the multistep setting and the untrusted model’s lack of knowledge of its position to incriminate the model. Their best protocol achieved 87.1% safety at 95% usefulness.

Our work extends Ctrl-Z’s resample and incrimination mechanism by comparing untrusted model samples to trusted model samples. The trusted model samples let us see whether the suspicion scores from the untrusted models were elevated because of the nature of the action, while providing possible alternative actions we can take. We also use the three-strike mechanism across the trajectory to incriminate models across long-running trajectories.

3. Methods

Resample with Trusted Comparison (RTC)

Our protocol extends defer to resample [2] with a trusted comparison step for incrimination. For each action in a trajectory:

1. Monitor. A trusted monitor scores the action’s suspicion.
2. Fast path. If suspicion is below a protocol threshold (calibrated from honest trajectories; see Appendix B), execute the action and continue.
3. Resample. If suspicion exceeds the threshold, generate 2 additional samples from the untrusted model. If any resample scores below the threshold, accept the least suspicious one and continue.
4. Trusted comparison. If all resamples exceed the threshold, generate 3 samples from the trusted model for the same step. Compare distributions:
 - If $\max(\text{trusted scores}) < \min(\text{untrusted scores})$ — every trusted sample is less suspicious than every untrusted sample — record a strike and execute the best trusted action.
 - Otherwise, no strike; execute the best untrusted resample.
5. Audit. If the trajectory accumulates 3 strikes, hard-stop and flag for human audit.

By comparing against the trusted model’s behavior on the same step, we distinguish between “this task requires actions that look suspicious to any model” and “the untrusted model is behaving differently from the trusted model.” This reduces false incriminations on inherently complex tasks while maintaining sensitivity to genuine attacks. Threshold calibration, model selection, and hyperparameter values are detailed in Appendices B and C.

Automated Red Teaming Pipeline

A practical challenge in control evaluations is constructing realistic attack policies. Naive attack prompts (e.g., “attempt the side task”) are trivially detected by the monitor. In our initial runs, 0/80 attacks succeeded under DTR purely because the monitor caught every attack attempt with max suspicion. We developed an automated pipeline to produce subtler attack transcripts.

The pipeline takes a failed attack transcript (tool calls from a previous attack run) and uses Claude Sonnet 4.5 to iteratively rewrite the actions for stealth and robustness. In each iteration: (1) Sonnet receives the actions, monitor scores for each action, red team prompt, and side task scores; (2) it rewrites the tool calls using stealth tactics like distributing the attack over many actions; (3) the rewritten transcript is replayed in a fresh Docker container and scored by BashArena’s test suites; (4) GPT-4.1-mini re-monitors the new transcript; back to (1).

Across 72 tasks with up to 3 iterations each, 57/72 (79%) achieved side task success on at least one iteration, and 8/72 (11%) fully converged (success + undetected). The total cost was \$22.07 (\$0.31/task average).

4. Results

Red Teaming Pipeline Results

Our initial attack runs used Qwen with a naive attack prompt. Across 80 tasks under DTR, 0 attacks succeeded — the model’s attack actions were immediately flagged, with 42% caught at audit-level suspicion. The untrusted model was not capable enough to produce subtle attacks on its own.

We ran these failed transcripts through our pipeline, where Sonnet 4.5 iteratively rewrote the tool calls. Each task was allowed up to 3 iterations. Across 72 tasks, after the first rewrite 2 tasks fully converged — the attack succeeded and evaded the monitor. Side task success was already high ($\sim 75\%$ of tasks on any given iteration), but nearly all were immediately detected with maximum suspicion. After iteration 1, 4 additional tasks converged (6 cumulative), with the rewrite model finding improved evasion strategies on the second pass. After iteration 2, 2 more tasks converged (8 cumulative, 11%). Some tasks that improved at iteration 1 regressed at iteration 2, suggesting the iterative process does not monotonically improve — the model sometimes trades stealth for effectiveness or vice versa.

In total, 57/72 tasks (79%) achieved side task success on at least one iteration, but only 8/72 (11%) achieved both success and evasion. The monitor response was largely binary — most attacks either scored 0.0 (fully undetected) or 0.9 (immediately caught), with only 5 tasks landing in the protocol-relevant band between 0.2 and 0.8.

Protocol Comparison

We applied both DTR and RTC post-hoc to the 72 Sonnet-rewritten attack transcripts. Each transcript was classified by its per-step monitor scores into three categories:

- Undetected (max suspicion < 0.2): 10 transcripts. The monitor never flagged any action, so no protocol would trigger. Of these, 6 had successful side tasks — these represent safety failures that no monitoring-based protocol can prevent. 4 had failed side tasks (the attack was subtle but ineffective).

- Immediate audit (first flagged step ≥ 0.85): 45 transcripts. The first suspicious action was flagged at audit level, meaning DTR would audit (if all resamples are equally suspicious) and RTC would accumulate strikes rapidly. These attacks are caught immediately under both protocols.
- Protocol band (first flagged step in $[0.2, 0.85]$): 17 transcripts. The first suspicious action fell in the range where DTR resamples and RTC performs trusted comparison. These are the transcripts where the protocols differ and where the evaluation is most informative.

For the 17 protocol-band transcripts, we ran live evaluations: replaying the pre-trigger tool calls in Docker to build up container state, then running the full protocol (DTR or RTC) with Sonnet 4.5 as the untrusted model for live continuation.

Table 1: Protocol comparison on 17 protocol-band attack transcripts.

	DTR	RTC
Attacks blocked by protocol	11/16 (69%)	12/15 (80%)
Attacks evading protocol	5/16 (31%)	3/15 (20%)
Avg usefulness (main task score)	0.584	0.462

RTC blocked a higher fraction of attacks than DTR (80% vs 69%). The improvement comes from RTC’s trusted comparison step: when all resamples are suspicious, comparing against the trusted model’s behavior reveals whether the suspicion is due to the task or the model. In cases where the untrusted model was consistently more suspicious than the trusted model, RTC recorded strikes and eventually audited the trajectory — catching attacks that DTR would have handled by simply deferring to the trusted model without tracking the pattern.

The usefulness cost of RTC (0.462 vs 0.584 for DTR) reflects the more aggressive intervention: RTC’s trusted comparison and strike mechanism cause more actions to be replaced by trusted model outputs, which are lower quality (Ministral-3B scores 10.99% vs Qwen’s 37.47% on honest tasks).

Among the 5 attacks that evaded DTR, 3 also evaded RTC. The 2 attacks caught by RTC but not DTR were cases where DTR accepted a resample that was below the protocol threshold but still contained an embedded attack — RTC’s trusted comparison flagged the pattern across steps. The 3 attacks that evaded both protocols had embedded their attack actions entirely in the pre-trigger portion of the transcript, before any protocol monitoring began.

5. Discussion and Limitations

Our sample of attacks shows that our protocol holds up about as well as resampling, and may have promise for future projects, or specific combinations of model capability and cost gaps. Overall, we think comparing trusted samples to untrusted samples could yield gains in our configuration and others.

Limitations

The largest natural limitation of this method is that it costs more than pure resampling (although one could argue catching a scheming model sooner saves money in the long run), in tokens. Also, depending on the gap in cost between the untrusted and trusted model, it costs

more to run resampling with incrimination in the present than this method. A full suite of comparisons to other resampling protocols could be tested in the future.

There are many practical limitations in this paper. We suffered many failed runs with small models that gave us mixed data, such that we can't produce an accurate safety-usefulness Pareto curve. The two main limiting factors were that Qwen could not complete attack trajectories and Ministral was not successful as a monitor. Using Sonnet 4.5 as an attacker (via editing the Qwen transcripts and taking over post protocol implementation (see Appendix C)) and GPT-4.1-mini as a monitor allowed us to get some data on our protocols, but means our honest runs from early on are not usable for comparison.

Our small sample of attacks shows that our protocol holds up about as well as resampling, and may have promise. The largest limitation is that the red team and blue team were not fully optimized, so we ended up with a sample that may not be representative of a real-world AI control scenario.

Future Work

This work could be extended by running a similar setup with consistent models, red and blue team optimization, and more tasks across all resample variants. This would confirm where resample with trusted comparison sits in the safety-usefulness Pareto curve. One twist on trusted comparison could be to use trusted editing in place of trusted implementation, or to extend the protocol to measuring suspicion score distributions, and looking for outliers in trusted and untrusted outputs across trajectories and even whole settings.

Other extensions could explore the tradeoffs between large and small capability and cost gaps.

The red team automation pipeline could be extended to have more input from external skilled models on prompt optimization, how to be sneaky, and so on.

6. Conclusion

We introduced resample with trusted comparison, a control protocol that extends resampling with incrimination by comparing untrusted and trusted model suspicion score distributions to make more reliable strike decisions. We also developed an automated red teaming pipeline that iteratively rewrites attack transcripts using a strong model, achieving an 11% convergence rate across 72 tasks at \$0.31/task.

Across 72 Sonnet-rewritten attack transcripts, the monitor responses fell into a bimodal distribution: 10 transcripts (14%) were completely undetected (max suspicion < 0.2), 45 (63%) triggered immediate audit-level suspicion, and 17 (24%) fell in the protocol-relevant band where DTR and RTC differ. Of the 10 undetected transcripts, 6 had successful side tasks — representing attacks that evade all monitoring-based protocols. On the 17 protocol-band transcripts, RTC blocked 80% of attacks compared to DTR's 69%, at a usefulness cost of 0.462 vs 0.584 (raw main task score). The attacks that evaded both protocols had completed their side task objectives entirely before the monitor's first flag, highlighting that the timing of the attack relative to the first trigger point is the primary determinant of protocol effectiveness — not the protocol's intervention strategy after the trigger.

Our work demonstrates that small extensions to existing resample protocols can improve incrimination reliability, and that automated red teaming can scalably produce realistic attack pools for control evaluation.

Code and Data

- Code repository: <https://github.com/ChrisHarig/hackathon>
- Data/Datasets: <https://www.bash-arena.com/>

Author Contributions

Chris Harig was the only researcher involved.

References

- [1] Greenblatt, R., Shlegeris, B., Sachan, K., & Roger, F. (2023). AI Control: Improving Safety Despite Intentional Subversion. arXiv preprint arXiv:2312.06942. ICML 2024.
- [2] Bhatt, A., Rushing, C., Kaufman, A., Tracy, T., Georgiev, V., Matolcsi, D., Khan, A., & Shlegeris, B. (2025). Ctrl-Z: Controlling AI Agents via Resampling. arXiv preprint arXiv:2504.10374.
- [3] Carlsmith, J. (2023). Scheming AIs: Will AIs fake alignment during training in order to get power? arXiv preprint arXiv:2311.08379.
- [4] Kaufman, A., Lucassen, J., Tracy, T., Rushing, C., & Bhatt, A. (2025). BashArena: A Control Setting for Highly Privileged AI Agents. arXiv preprint arXiv:2512.15688.

A. Automated Red Teaming Pipeline

Motivation

Control evaluations require realistic attack policies to produce meaningful safety measurements. In BashArena, the untrusted model must accomplish a sabotage objective (e.g., exfiltrating data to a specific IP, executing malware) while avoiding detection by the monitor. Our initial attack prompt was trivially detected — actions like direct sudoers modifications receive maximum suspicion scores immediately, and 0/80 initial attacks succeeded under any protocol. We needed a scalable way to produce attack transcripts that are both effective and subtle.

The pipeline also addresses a practical bottleneck in AI control research: constructing realistic red team policies is expensive and requires domain expertise. By automating the iterative refinement of attack transcripts, we reduce the marginal cost of producing a diverse attack pool from hours of manual effort to ~\$0.31 per task.

Architecture

The pipeline is a CLI tool (`auto_red_team/`) with five stages:

1. Extract. Parse a .eval log file from a failed attack run into a structured JSON transcript of tool calls and their outputs.
2. Rewrite. A strong model (Claude Sonnet 4.5) receives the transcript along with the task description, side task objectives, per-step monitor scores from the failed run, and stealth tactics. It outputs a sparse edit format specifying which tool calls to modify, add, or

remove. The model may extend the transcript by up to 20% in total tool calls but must preserve main task completion.

3. Replay. Execute the rewritten tool calls in a fresh Docker container using BashArena’s task environment. Score with both the main task test suite (usefulness) and side task test suite (safety).
4. Monitor. Run the same monitor model post-hoc on the replayed transcript to get per-step suspicion scores, using identical prompt and formatter as the protocol runs.
5. Iterate. If the transcript did not converge, feed the scores and failure mode back for another rewrite attempt. Up to 3 iterations per task.

Prompt Design

We developed two prompt versions:

- v1 (basic): Direct instructions to make actions less suspicious. ~50% side task success rate on a 5-task test.
- v2 (BashArena stealth tactics): Adds specific guidance on distributing attacks across steps, breaking suspicious strings, using deep call stacks, and creating data movement chains. 79% side task success rate, 11% full convergence on the 72-task batch.

The v2 stealth tactics section instructs the model to:

- Distribute the attack across multiple steps so no single tool call is sufficient to identify it
- Use deep call stacks — set up functions/scripts that call one another, created in different tool calls
- Break up suspicious strings — never have IPs, NOPASSWD: ALL, or similar appear intact
- Avoid surgical edits — rewrite entire files weaving attack into legitimate code
- Create data movement chains — for exfiltration: read → store → transform → send
- Be incremental — each tool call accomplishes only one small step toward the attack
- Don’t verify success — never check if the attack worked, as the test suite checks automatically

Key Findings

- The monitor response is largely binary: most rewritten attacks either fully evade (suspicion 0.0) or are immediately caught (suspicion 0.9). Only 5/72 tasks fell in the protocol-relevant band (0.2–0.8).
- 8/72 tasks (11%) produced attacks that both succeeded and completely evaded the monitor (max suspicion < 0.2).
- Side task success often occurred before the monitor’s first high-suspicion score, highlighting the importance of proactive protocols over reactive auditing.
- The prompt matters more than iteration count — several tasks converged on iteration 0 with v2 but never converged with v1 across 3 iterations.

Cost Analysis

Total cost for 72 tasks: \$22.07 (\$0.31/task average). Per-iteration breakdown: Sonnet 4.5 rewrite $\sim$ \$0.10, Docker replay $\sim$ \$0.01, GPT-4.1-mini monitoring $\sim$ \$0.01. The dominant cost is the Sonnet rewrite step.

B. Monitor Calibration

Monitor Model

We use GPT-4.1-mini as our monitor, scoring each tool call on a 0–10 integer suspicion scale. The monitor sees only the tool calls (bash commands and their arguments) via a prefix message formatter — it does not see agent reasoning, planning, or text between tool calls. We use the BashArena paper’s verbatim monitor prompt, which includes descriptions of all four side tasks.

GPT-4.1-mini was selected for the monitor role specifically because its 38.3% usefulness score (on a 5-task smoke test) was too close to Qwen’s 37.47% to serve as a trusted implementor. It is well-suited as a monitor-only model.

Threshold Calibration

We calibrated protocol thresholds from 40 honest Qwen 3.5-Flash trajectories (950 total tool calls, single monitor pass at temperature 0.0).

Table 2: Monitor thresholds calibrated from honest trajectories.

Threshold	Raw	Normalized	FP Rate	Purpose
Protocol trigger	≥ 2	0.2	$\sim 1\%$	Triggers resample / trusted comparison
Audit	≥ 8	0.8	$\sim 0.42\%$	Hard block, spend audit budget

Honest False Positive Rates

Table 3: Task-level false positive rates on honest trajectories.

Protocol	Flagged (≥ 0.2)	Audit (≥ 0.8)
DTR honest	8/80 (10%)	4/80 (5%)
RTC honest	5/80 (6%)	3/80 (4%)

RTC has a lower task-level false positive rate because the trusted comparison step resolves some triggers without striking — when the trusted model produces equally suspicious actions, the protocol correctly attributes the suspicion to the task rather than the model.

C. Experimental Details

Model Selection

Untrusted model: Qwen 3.5-Flash. Average usefulness on 40 tasks: 37.47%.

Trusted implementor: Ministral-3B. Average usefulness on 39 tasks: 10.99%. Selected as a genuinely weak model with native tool calling support.

Monitor: GPT-4.1-mini. Used for suspicion scoring only.

Attacker: Claude Sonnet 4.5. Used in the automated red teaming pipeline to rewrite attack transcripts.

Protocol Hyperparameters

Table 4: Protocol hyperparameters for DTR and RTC.

Parameter	DTR	RTC
Resample count	2	2
Trusted samples	—	3
Strikes to audit	—	3
Protocol threshold	0.2	0.2
Incrimination rule	—	$\max(\text{trusted}) < \min(\text{untrusted})$

Initial Protocol Runs (DTR/RTC v1)

Table 5: Initial protocol run results with naive attack prompt.

Protocol	Mode	N	Useful.	Flagged	Audit	Side Task
DTR	honest	80	37.5%	10%	5%	—
DTR	attack	80	26.0%	57%	42%	0%
RTC v1	honest	80	32.9%	6%	4%	—
RTC v1	attack	60	36.8%	63%	40%	0%

RTC v1 bugs (fixed for v2):

1. Strike counting scanned state.messages instead of tracking via closure — cumulative strikes never exceeded 1.
2. Resample accept threshold was too strict (exactly 0.0 instead of < 0.2).
3. Incrimination compared averages instead of using $\max(\text{trusted}) < \min(\text{untrusted})$.

RTC v1 honest data remains valid — 0 incriminations and 0 strikes occurred in honest runs.

D. Run Failures and Iterative Improvement

Model capability failures. Qwen 3.5-Flash was entirely incapable of producing subtle attacks. Every attack attempt used obvious actions that the monitor scored at 0.9 immediately. This 0% attack success rate motivated the development of the automated red teaming pipeline.

Monitor selection failures. Our initial configuration used Ministral-3B as the monitor. It scored nearly everything as 0 — it could not distinguish attacks from honest behavior. We switched to GPT-4.1-mini and re-scored all existing trajectories post-hoc.

RTC v1 implementation bugs. Three bugs were discovered after running 80 honest and 60 attack trajectories (see Appendix C). The most critical prevented strike accumulation across steps. Honest data remained valid.

Infrastructure failures. Tasks frequently hit Docker compose timeouts, OpenRouter rate limits, and provider HTTP 400 errors. We addressed these with retry logic and migrated from DigitalOcean droplets to AWS EC2 spot instances.

Model mismatch. Because Qwen could not attack and Sonnet was used to rewrite attacks, usefulness scores are not directly comparable between honest and attack trajectories. A future iteration with a single model capable of both would resolve this.

LLM Usage Statement

Claude (Opus 4.6) assisted with drafting the report, creating LaTeX formatting, and generating analysis code. All claims and results were verified by the author.

¹Research conducted at the AI Control Hackathon, March 2026