LEARNING CYBER DEFENSE WITH LIMITED INTERACTION: AN EMPIRICAL STUDY OF REFLECTIVE PROMPT OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Cyber defense resembles a game of Whack-A-Mole: restoring a compromised host removes attacker access, but incurs an immediate cost and may provide only temporary protection. Successful remediation can be verified, but whether it was worth the cost depends on subsequent events. We investigate whether reflective prompt optimization can learn effective defense policies with fewer environment interactions than proximal policy optimization (PPO) and the critic-free group relative policy optimization (GRPO). We also identify classes of rewards for which it works well and where it falls short. Using CybORG CAGE-2, a simulated network-defense environment, we modify the original objective and design when-to-whack scenarios emphasizing verified remediation, sustained defense, and a hybrid that strikes a balance between the two. We also design which-to-whack scenarios, in which the defender chooses which host to restore under distinct or uniform damage costs for pairs of compromised hosts. Our evaluation compares GEPA, which optimizes a pretrained language-model agent's instructions using return-only or enriched trajectory feedback, with PPO and GRPO tuned per scenario.

On the when-to-whack scenarios, after about one hundred episodes of interaction, every GEPA run exceeds the mean return of PPO and GRPO. Moreover, PPO needs 10–50$\times$ more interactions to catch up while GRPO matches GEPA after 200K steps in mixed and sustained object, but fall short after five million steps in verified remediation and original settings. On the which-to-whack scenarios, both PPO and GRPO outperform GEPA. Overall, we find that GEPA is best where reward follows a locally verifiable event or restoration is cheap, where it closes 85–98% of the gap between an untrained PPO and PPO trained for five million steps. We also find that enriched feedback can improve GEPA's performance under verified remediation and the original objective, while return-only feedback suffices under the mixed and sustained objectives. Separate matched-trajectory experiments show that changing feedback while holding the instruction and experience fixed changes the corrective strategies proposed during reflective prompt optimization.

1 INTRODUCTION

Defending a computer network is like a game of Whack-A-Mole. An attacker who has broken into one machine uses it to reach the next, and works toward the machines that an organization depends on. The defender receives alerts about intrusions, but they are imperfect. At each step, they may wipe a suspicious machine to restore it back to a clean image (the *Restore* action). The whack works, in that it removes the attacker's access to the machine, but it also has a price, in that the machine cannot serve normal users while it is being restored. The whack may also not be lasting, as the attacker can usually repeat the attack used previously to break in again. However, holding back has a price too, since damage may accrue for as long as the attacker retains a presence (Figure 1a). What has to be learned is therefore not only how to whack but *when a whack is worth it* and, when several moles are up, *which one to hit*. What counts as "worth it" is set by the operator, who may value confirmed cleanups, damage prevented over a whole incident, or both. Autonomous cyber defense asks an agent to learn these decisions from partial observations, against an adversary, with actions that cost now

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.

and pay only if their effect lasts (Standen et al., 2021; Kiely et al., 2023). This paper investigates how to learn policies from limited interactions.

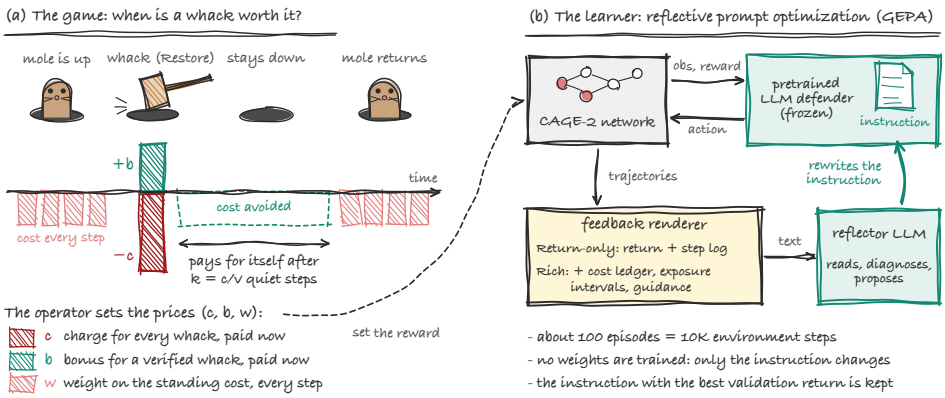


Figure 1: **Overview.** (a) Cyber defense as Whack-A-Mole: a whack (Restore) is charged c at once and earns b if the cleanup is verified, while the standing cost it avoids arrives later and only until the mole returns. A reward configuration (c, b, w) sets these prices. (b) GEPA learns when to whack without training weights: a frozen pretrained defender plays about one hundred episodes, a renderer turns the trajectories into text, and a reflector rewrites the instruction. We ask where this is more sample-efficient than training an RL policy, and where not.

The standard way to obtain such an agent is to train a reinforcement-learning (RL) policy. The strongest learned defenders in the CAGE challenges use proximal policy optimization (PPO; Schulman et al., 2017; Foley et al., 2022; Kiely et al., 2023; Le & Stadler, 2025), trained from scratch: a small network must discover from reward alone what each of its actions does, and only then when a Restore pays off. However, RL requires interactions that real-world cyber defense typically does not support. In our experiments PPO needs 60K–500K environment steps to reach the policies that reflective optimization reaches in 10K. Realistic training environments are usually based on physical or virtual machines (Standen et al., 2021) and emulated networks, making each step of training slow.

Reflective prompt optimization offers an attractive alternative: rather than training weights, it rewrites a pretrained language-model agent's instructions, using a reflection model that reads a few trajectories, diagnoses what went wrong, and proposes a revision (Pryzant et al., 2023; Yuksekgonul et al., 2024; Agrawal et al., 2026) (Figure 1b). The agent does not need to start from scratch, since a pretrained model already knows what restoring a server means, and each episode is used as a whole trajectory with a diagnosis in natural language rather than as a scalar meaningful only in the context of a specific model. GEPA, a representative of this family, reportedly beats RL fine-tuning with up to 35× fewer rollouts on reasoning and question-answering benchmarks (Agrawal et al., 2026).

However, whether this advantage generalizes to other scenarios is not obvious. On some benchmarks the reward for a given action can be assessed immediately, while in cyber defense, whether a whack was worth its price becomes apparent only much later. More basically, a reflector can only write down what it can read off from a few trajectories; it does not have a way to accumulate statistics across hundreds of samples like an RL network does. We therefore expect the reward to decide the winner at a small budget: a reward whose optimal policy can be expressed in a few sentences should favor reflection, whereas one that requires a table should favor RL.

Our question is thus: *where is reflective prompt optimization more sample-efficient than RL, and where is it not?* We study GEPA as a representative of the family; the argument of Section 3.4 rests on the policy being text written from a few trajectories, which holds across the family. We answer the question in the context of CybORG CAGE-2, a public network-defense simulator, holding the attack and defense dynamics fixed, varying what the reward asks the defender to know, and giving both learners the same small budget of environment steps. Our answer: what decides the winner at a small budget is how much of a decision's consequence a single trajectory reveals, a property readable off the reward before training. Our work makes three primary contributions:

1. **Where GEPA has an advantage.** In when-to-whack scenarios, in which a whack is paid in around 100 steps or fewer, every GEPA run beats PPO, tuned for the same budget of 10K steps, and GRPO at that budget. Further, PPO needs 10–50× more interactions to catch up, although the

margin shrinks as the payback is delayed. A decomposition of the return shows why: GEPA learns to stop wasted whacks under every reward, but to target them or act early only where the payoff is quick, and RL learns only the first. Controls show this is not the pretrained prior alone: the unoptimized prompt and hand-prompted frontier models fall far short of the optimized policies, and a 14B open-weight actor matches a frontier-scale one.

2. **Where GEPA falls short.** Under which-to-whack scenarios, where the rewrard depends on a hidden set of compromised machines, both RL baselines are ahead at the same budget, and GEPA learns only the few costliest pairs it can readily identify. When individual trajectories do not reveal the consequences of an action then RL is generally superior because it estimates the hidden information. In contrast, reflection tries to compress the hidden information using a few trajectories, which is hopeless.
3. **How enhanced feedback design shifts the boundary.** When the reward is sparse, enhanced feedback can increase the value of the reflector's proposals from 14% to 72%, improving every policy learned for the original reward function. This effect is absent in scenarios where a whack takes 50–100 steps to pay back and where the damage is concentrated and visible at once.

Overall, we offer empirical insights grounded in cyber defense relating prompt optimization and RL.

2 Background and related work

Cyber defense and its learners. Cyber defense can be modeled as a partially observed game between an attacker (Red) and a defender (Blue). Examples include NASim and CyberBattleSim (Schwartz & Kurniawati, 2019; Microsoft Defender Research Team, 2021), Yawning Titan (Andrew et al., 2022), and CybORG (Standen et al., 2021), the basis of the CAGE challenges (Kiely et al., 2023; 2025a). In CAGE Challenge 2, Red spreads from a user host across a 13-host network toward the operational server while Blue, from noisy alerts, monitors, analyses, removes, decoys, or restores one host per step; Restore re-images the host, always evicts Red, and costs 1, Remove is free but fails once Red has full control, and a compromised host is charged per step about 0.1 (user), 1 (enterprise or operational), or 10 (operational impact). RL defenders range from value-based to actor-critic and planning methods (Sutton & Barto, 2018; Mnih et al., 2015; 2016; Silver & Veness, 2010; Hammar et al., 2024); competitive approaches (Foley et al., 2022; Le & Stadler, 2025; Wolk et al., 2022; King et al., 2025; Kiely et al., 2025b) mostly use PPO (Schulman et al., 2017), whose critic credits actions with estimated future return. Language-model defenders with manually engineered prompts approach the best RL agents in CAGE-2 (Mohammadi et al., 2025), which plain hand-written prompts do not (Section 4.3), and language models also serve as attackers or help RL defenders (Rigaki et al., 2023; Castro et al., 2025; Loevenich et al., 2025; Chowdhry et al., 2026).

Reflective prompt optimization. Prompt optimizers rewrite a system's prompt from execution feedback with frozen weights, by search or evolution (Zhou et al., 2023; Yang et al., 2024; Fernando et al., 2023; Guo et al., 2024), at the program level (Khattab et al., 2024; Opsahl-Ong et al., 2024), or by reflection with natural-language diagnosis (Pryzant et al., 2023; Yuksekgonul et al., 2024; Cheng et al., 2024; Madaan et al., 2023; Shinn et al., 2023; Zhang et al., 2025). We use GEPA (Agrawal et al., 2026), which evolves along a Pareto frontier, proposing revisions from a minibatch of trajectories with validated return. Analyses of such optimizers (Liu et al., 2026; Abu Hara et al., 2026; Huang et al., 2024; Gao et al., 2026; Vassilyev et al., 2026) ask where a language model can replace an RL algorithm (Hatgis-Kessell & Brunskill, 2026) and show the fragility of scoring each candidate once on a small minibatch (Ren et al., 2026); they vary the environment, whereas we hold it fixed and vary the reward. Credit assignment links outcomes to earlier decisions (Minsky, 1961; Harutyunyan et al., 2019; Arjona-Medina et al., 2019; Ni et al., 2023); for language agents, outcome-only policy gradients degrade with the horizon while critics learn slowly (Jiang et al., 2026), and natural-language critics (Hong et al., 2025; Park et al., 2025) are the nearest relatives of our Rich feedback.

3 Cyber defense as Whack-A-Mole

A Whack-A-Mole player decides *when* a whack is worth its price and, with several moles up, *which* one to hit; the reward decides which decision matters and what kind of knowledge it takes (Table 1). This section makes both precise and describes how the two learners acquire that knowledge.

3.1 SETTINGS

Moles and whacks. A *mole* is a host compromised by the attacker (Red), and a *whack* is a Restore attempt to remove its access. Restore incurs a charge whether or not the host is compromised or the attempt succeeds. Remediation is *verified* when Red has root/SYSTEM access on the target before the step and no such access after the complete environment step. The attacker may subsequently return, so verified remediation does not establish whether restoration was worth its cost. The defender chooses one action per step from partial observations; ground-truth root states are used for scoring but are not supplied directly as policy inputs.

Rewards and payback. For the four *when-to-whack* scenarios, a reward configuration is a triple (c, b, w): the charge per Restore attempt, the bonus for the first verified remediation of each eligible host per episode, and the weight on standing cost. Eleven hosts are eligible for the bonus: all hosts except User0 and Defender. The reward at step t is:

$$r_t = b\, q_t - c\, \mathbf{1}[a_t = \text{Restore}] - w\, g_t, \tag{1}$$

where q_t indicates a bonus-eligible remediation and g_t is the original standing cost. These settings change how outcomes are scored while preserving the simulator's attack and defense dynamics.

Table 1: **Settings.** Costs, bonuses, and damage penalties for the when-to-whack and which-to-whack scenarios.

When to whack	Description	Restore cost c	Bonus b	Standing cost weight w
Verified Remediation	Bonus for verified remediation.	5	10	0
Original CAGE-2	Restoration costs plus ongoing standing cost.	1	0	1
Sustained Defense	Higher Restore cost; unchanged standing cost.	5	0	1
Mixed	Remediation bonus; halved standing cost.	5	5	0.5
Which mole to whack	**Description**			
Distinct pair costs	Best target depend on hidden pair costs among compromised hosts.			
Uniform pair costs	Best target depend on effective Restores;ineffective ones cost.			

Note: Every Restore attempt incurs its charge. Bonus b is paid for the first verified remediation per eligible host per episode, excluding certain special hosts (e.g., User0 and Defender). Verification requires root/system access before Restore and none after the complete environment step. Weight w scales the original standing costs. Both targeting scenarios use Restore cost 1 and no bonus.

3.2 WHEN IS A WHACK WORTH IT?

Restoration can offset its cost through an immediate remediation bonus or through damage prevented afterward. Consider a successful Restore that earns no bonus and whose only benefit is to prevent a standing cost of v per step for L steps. Its net benefit is

$$\Delta J = vL - c, \tag{2}$$

so it breaks even when $L = c/v$. We call this the *payback horizon*. For example, preventing a standing cost of 0.1 per step offsets the Restore charge after ten steps under Original CAGE-2 and fifty under Sustained Defense.

Bonuses change this calculation. Under Verified Remediation, an eligible bonus of ten exceeds the Restore charge of five. Mixed combines a one-time remediation bonus with ongoing damage penalties. The bonus of five covers the Restore charge for the first verified remediation of an eligible host. Further Restores of that host receive no bonus, so their value depends on the damage prevented.

Table 1 summarizes these different incentives. The payback calculation is conditional on successful restoration and the assumed duration of protection, not a measured action advantage. In the simulator, ineffective attempts, reinfection, and the remaining episode length affect whether restoration pays off.

The top block of Table 1 lists the four configurations in order of payback horizon ($k \approx 0$, 10, 50, and 100 steps); *Original*, the stock CAGE-2 objective, is a reference point: it runs under its own protocol with different actor-input features and held-out seeds. In all four, hosts of a class are alike, so what must be learned fits in a sentence: whether to whack, how promptly, and which class of hosts first.

3.3 WHICH MOLE TO WHACK?

To study target selection, we introduce two scenarios in the same simulator. Each ten-step episode starts from a saved incident with five to seven of eight protected hosts compromised. The defender restores one flagged host per step, or Monitors if none is flagged. Both scenarios impose a cost of one per Restore attempt and provide no remediation bonus.

Damage depends on pairs of hosts: a pair incurs a penalty only while both hosts remain root-compromised. The reward is

$$r_t = -c\,\mathbf{1}[a_t = \text{Restore}] - \beta \sum_{i<j} C_{ij}\,\mathbf{1}[i, j \in S_{t+1}], \tag{3}$$

where $c = 1$, $\beta = 2$, and S_{t+1} is the set of root-compromised hosts after the complete environment step. The pair rates C_{ij} are fixed throughout training and evaluation and are not supplied to the learners.

Distinct pair costs. Here $C_{ij} = W_{ij}$, where W is a symmetric matrix with unit row sums. Every host therefore has the same total pair weight, but its value as a restoration target depends on which other hosts are compromised. This scenario tests whether learners can infer useful targeting rules from aggregate rewards.

Uniform pair costs. Here $C_{ij} = 1/7$ for every pair. This removes differences in pair rates while retaining the need to avoid ineffective restoration. A flagged host need not be root-compromised, and restoring User0 does not remove the attacker's original access. Such attempts still incur a charge without clearing a root compromise. This control tests whether learners can identify ineffective actions even without a table of distinct pair costs.

3.4 LEARNING METHODS AND FEEDBACK

PPO and GRPO train neural defenders from scratch. PPO estimates action advantages using a learned value function; GRPO instead uses group-normalized episode returns as advantages for the actions in each trajectory. GEPA updates a pretrained agent's instruction: a reflector reads the current instruction and feedback from a minibatch of trajectories, proposes a revision, and candidate instructions are selected using validation return.

For the four when-to-whack settings, we compare two feedback renderings of the same objective. *Return-only* supplies observations, actions, per-step rewards, and episode return without additional accounting. *Rich* adds a decomposition into action charges, weighted standing costs, and cleanup bonuses; accumulated costs over intervals without restoration; and fixed guidance for the objective (Appendix D.4). The interval summaries associate observed costs with preceding actions without simulating alternative actions. For the which-to-whack scenarios, the reflector receives observable action and per-step reward traces.

Hypothesis. We hypothesize that GEPA learns efficiently when a useful decision rule can be inferred from a few trajectories and expressed in an instruction. Its advantage should diminish when effective decisions require estimating action consequences across many episodes, such as learning hidden pair costs or identifying consistently ineffective Restores. We test this hypothesis by comparing GEPA with both PPO and GRPO across the six reward settings.

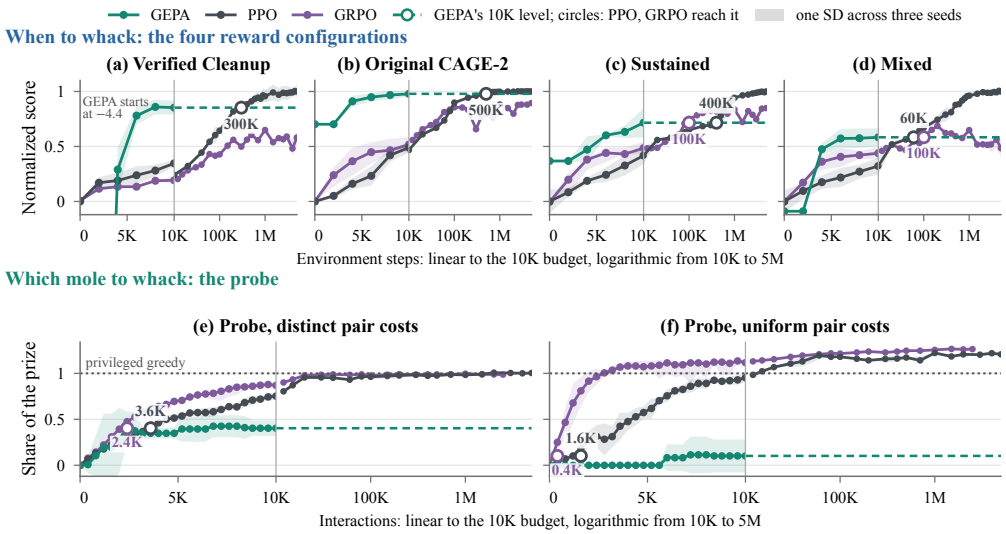


Figure 2: **When to whack, and which mole to whack.** (a–d) Normalized score (0: untrained PPO; 1: PPO after five million steps) under the four configurations, linear to the 10K budget and logarithmic beyond, where PPO and GRPO continue; circles mark where each first reaches GEPA's 10K level (dashed). (e–f) Share of the prize (0: uniform policy; 1: privileged greedy) in the scenario, with the same axes and markers; GRPO, a critic-free RL baseline, is continued to 2.5M there. GEPA uses Rich feedback in (a–d) and per-step feedback in (e–f)

4 Experiments

4.1 Experimental setup

Our main comparisons use CybORG CAGE-2 with Meander, a sensor true-positive rate of 0.95, and no false alarms. When-to-whack settings use 100-step episodes and 50 held-out evaluation episodes; which-to-whack scenarios use ten-step episodes and 200 held-out incident starts, with evaluation cohorts shared across methods within each setting. GEPA optimizes DeepSeek-V3.1 instructions using Claude Opus 4.6 as the reflector. PPO and GRPO train matched neural actors from scratch, with hyperparameters tuned for each when-to-whack setting. The 10K-step budget counts GEPA's search and validation episodes and all RL training transitions, including every GRPO group member, but excludes held-out evaluation, both RL hyperparameter searches, and pretraining. For when-to-whack, both RL methods continue to five million steps: GRPO retains its selected configuration, while PPO uses shared defaults that differ from its 10K selection under Verified Remediation and Original CAGE-2. We report mean return and sample SD across three run means. Normalization uses untrained PPO and five-million-step PPO for when-to-whack, and uniform and privileged one-step greedy policies for the scenarios; the greedy reference is not an upper bound. PPO's interaction multiplier uses its first saved checkpoint reaching GEPA's corresponding 10K mean, divided by 10K, without requiring a sustained crossing. Full protocols appear in Appendices B and C.

4.2 Experimental results

Does the sample efficiency reflective prompt optmization advantage carry over to when-to-whack decisions?

Yes, under all four rewards: every GEPA run exceeds the mean return of budget-tuned PPO and of GRPO at 10K steps, and PPO needs 10–50× more interaction to catch up.

Figure 2a–d plots the learning curves; Appendix Table 7 reports mean 10K returns for the four when-to-whack settings. Under every configuration both GEPA conditions end above PPO at 10K steps, and GRPO ends between PPO and GEPA except under Verified Remediation, where it is below PPO (Table 2), so every GEPA run exceeds both RL baselines. Measured as the first saved checkpoint at which PPO's mean curve reaches GEPA's 10K score, PPO requires 60K–500K steps, a 6–50× speedup over all eight configuration–feedback pairs and 10–50× against the higher-mean GEPA

condition. Shifting GEPA's mean by one SD either way (Appendix Table 7) keeps the speedup at least 8× against the higher-mean condition and 2× over all eight pairs. GRPO, continued to five million steps, reaches GEPA's 10K score after about 100K steps under Sustained Defense and Mixed but remains below it at five million steps under Verified Remediation and Original (Figure 2a–d). All 24 GEPA runs exceed both RL baselines' mean returns at 10K steps (Appendix Tables 8 and 7).

Table 2: **The six scenarios differ by how much of a decision's consequence one trajectory reveals**, in the blocks and order of Table 1. Scores at 10K: normalized (0: untrained PPO; 1: PPO at 5M) for the configurations, share of the prize (0: uniform; 1: one-step greedy) for the scenario; Rich − Return-only: return difference at 10K. GEPA: Rich feedback in the configurations, per-step feedback in the scenario, where GRPO was tuned and PPO was not. Waiting means not whacking; a whack's benefit is the damage it prevents, arriving after the payback horizon.

Setting	What one trajectory shows about a decision	GEPA at 10K	PPO / GRPO at 10K	Rich − Return-only	What GEPA learns
When to whack (Sec. 3.2)					
Payoff in the same step					
Verified Remediation	the whack's payoff: a bonus paid at once	0.85	0.35 / 0.19	+3.3	avoid waste, then target
Payback after about 10 quiet steps					
Original CAGE-2	the damage accrued while waiting, summed	0.98	0.49 / 0.52	+41.7	avoid waste; act early
Payback after 50–100 quiet steps					
Sustained Defense	the damage while waiting, not the benefit	0.72	0.42 / 0.49	−27.4	avoid waste
Mixed	same, with the benefit even later	0.58	0.32 / 0.44	−17.7	avoid waste
Which mole to whack (Sec. 3.3): no compact policy					
Distinct pair costs	a shift in a sum of many hidden pair costs	0.40	0.75 / 0.87	–	three costliest pairs
Uniform pair costs	nothing: a wasted whack leaves no trace	0.10	0.95 / 1.12	–	almost nothing

The pretrained prior does not explain the gap. GEPA adapts a pretrained actor that is told the objective in words, whereas PPO trains a small network from random initialization on the scalar reward alone. The prior by itself does not produce the optimized policy: the zero-shot prompt scores 0.70 under Original, where it alone already exceeds PPO's 0.49 at 10K, but only 0.37 under Sustained Defense, −0.09 under Mixed, and −4.42 under Verified Remediation, where it issues about 57 Restores per episode. Hand-written prompts on stronger models do not close the gap either (Section 4.3). Nor is a frontier-scale actor required: with Return-only feedback, a 14B open-weight actor reaches -73.3 ± 6.3 against Meander, matching DeepSeek-V3.1 (-71.8 ± 4.0; Table 3).

Under which scenarios does GEPA's advantage hold vs. not?

It decreases with delay between action and payoff, and reverses when the optimal policy is a lookup table rather than a compact instruction; there PPO generally outperforms GEPA.

The six settings differ in how much of the consequences of a decision a single trajectory reveals. The payback horizon provides a clear ordering on the when-to-whack scenarios—the shift between scenarios can be modeled in terms of a sum of the hidden values induced by an action. Tables 1 and 2 list the settings and outcomes in this order; GEPA's standing, what it learns, and the value of Rich feedback all increase, but PPO's catch-up time does not.

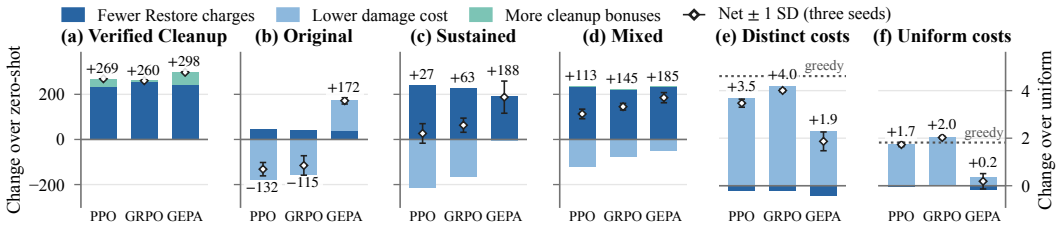


Figure 3: **Relative performance over 10K steps.** Return gains for PPO, GRPO, and GEPA, decomposed into Restore savings, damage savings, and bonus gains. Negative segments: losses; numbers: net gains; whiskers: SD across three seeds. (a–d) Relative to the zero-shot prompt, in payback order; GEPA uses Rich feedback. (e–f) Relative to the uniform policy; GEPA uses per-step feedback; dotted lines: one-step greedy.

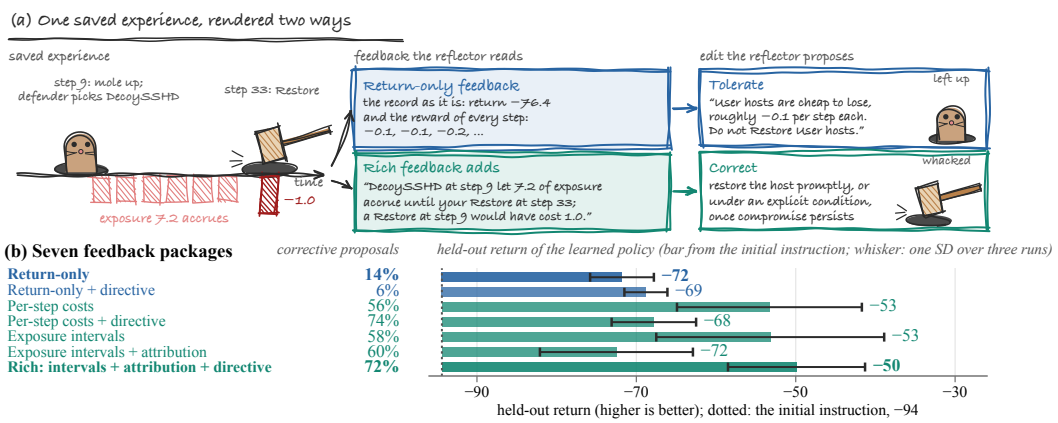


Figure 4: **Feedback design changes what the reflector proposes and what the defender learns.** (a) The same instruction and trajectories rendered two ways, and the edit the reflector proposes in each case (excerpts). (b) Seven feedback packages, without (blue) and with (green) cost accounting: the share of corrective proposals over 50 matched batches (numbers), and the held-out return of the policy each package learns in the feedback study (DeepSeek-V3.1 against Meander; bars from the initial instruction, whiskers one SD over three runs).

Immediate Payoff (Verified Remediation). The first eligible verified remediation earns a bonus of ten points. GEPA's normalized score rises from -4.4 to 0.75 within 6K steps; PPO reaches that level at 200K. Relative to a zero-shot prompt, GEPA reduces Restores from about 57 to 9 per episode, saving 242 points, and increases bonuses earned from 2.6 to 8, adding 55 points (Figure 3a). PPO and GRPO also improve mainly through lower Restore charges relative to the same reference, but their bonus gains are smaller: 35 and 5 points, respectively. Rich exceeds Return-only by 3.3 points.

Payoff within a few steps (Original CAGE-2). For a low-value host, a successful Restore costing one breaks even after ten steps if it prevents a standing cost of 0.1 per step throughout that interval. GEPA reaches a normalized score of 0.90 at 4K steps, compared with 200K for PPO. Relative to the zero-shot prompt, standing cost falls from 146 to 46 with Return-only feedback and to 12 with Rich; these reductions account for 77–78% of the return improvement (Figure 3b). At 10K steps, PPO and GRPO remain 132 and 115 return points below the zero-shot reference: their savings on Restore charges are outweighed by higher standing costs.

Feedback affects reflective proposals and policy returns. Rich feedback summarizes observed costs alongside the Restore charge (Figure 4a). In a matched experiment with 50 batches, holding the instruction and two trajectories fixed while changing the rendering from Return-only to Rich increases the share of proposals classified as corrective from 14% to 72%. Packages with cost accounting reach 56–74% ($p < 10^{-4}$), whereas a directive-only control yields 6% (Appendix Table 17). Separate optimization runs achieve mean returns of -53, -53, and -50 with Per-step, Intervals, and Rich feedback, respectively, compared with -72 for Return-only (Figure 4b; Appendix Table 16). In the main experiment under Original CAGE-2, every Rich run exceeds every Return-only run, and mean accumulated standing cost is approximately one-quarter as large (Appendix Tables 8 and 9). The

Table 3: **Actors, prompts, and optimizers on the original objective.** Separate scaffolding and evaluation episodes; mean ± SD across three optimizer runs. Last column: first saved PPO checkpoint matching the return (Meander / B-line). Initial returns: Appendix Table 20. †Hand-written prompt without optimization.

Actor	Optimizer, feedback	Return, Meander	Return, B-line	PPO steps to match
DeepSeek-V3.1	GEPA, Return-only	-71.8 ± 4.0	-59.7 ± 6.9	372K / 400K
	GEPA, Rich	-49.9 ± 8.6	-55.9 ± 5.7	492K / 400K
Qwen3-14B	GEPA, Return-only	-73.3 ± 6.3	-33.6 ± 3.1	372K / >560K
	GEPA, Rich	-69.0 ± 10.4	-40.8 ± 2.2	372K / >560K
Claude Opus 4.6	hand-written prompt†	-137.7 ± 57.5	-85.9 ± 43.1	212K / 240K
GPT-5.1 Codex Max	hand-written prompt†	-164.1 ± 49.4	-105.5 ± 8.1	156K / 200K

matched experiment isolates feedback's effect on proposed instructions, while separate optimization runs evaluate its effect on policy return.

Restoration costs (Sustained Defense and Mixed). For a low-value host, a successful Restore without a bonus requires 50 steps of avoided standing cost under Sustained Defense and 100 under Mixed, assuming the saving persists. An eligible Mixed bonus instead offsets the five-unit Restore charge immediately. GEPA still leads at 10K steps but remains further from PPO's five-million-step return (Table 2). With Rich feedback, its improvement over the zero-shot reference is dominated by 194–234 points of reduced Restore charges (Figure 3c–d). PPO and GRPO also reduce charges, but much of this saving is offset by higher weighted standing costs relative to the same reference. Rich's mean return is 27 and 18 points below Return-only under Sustained Defense and Mixed, respectively. A Restore-price sweep with one optimizer run per condition shows a similar pattern: Rich performs better relative to Return-only when restoration is cheaper (Appendix D.10).

Target selection (which-to-whack scenarios). The which-mole scenario (Section 3.3) is constructed so that no compact instruction attains the reward: a trajectory records only the total damage, which mixes 10–21 active pairs, so the effect of restoring one host rather than another cannot be attributed to any entry of the table from one episode. We report gains over the uniform policy as a share of the gap to a privileged one-step greedy reference that reads the table and the true state (the prize), which is myopic and not an upper bound. With distinct pair costs ($\lambda = 0$), PPO leads from 4K interactions onward and in every seed at 10K (Figure 2e); GEPA's gain comes from three accepted edits that name the costliest pairs, after which it plateaus. When all pairs cost the same ($\lambda = 1$), what remains is which flagged hosts are not worth restoring, and a wasted Restore leaves no trace except a decrease in damage that did not occur (Figure 2f). GRPO, which needs no critic and, unlike PPO, was tuned for the scenario (Appendix B.4), does better still and passes the one-step greedy reference within the budget under uniform costs (Table 2), staying there when continued to 2.5M. Each learner's gain is pair-damage savings bought with slightly more Restore charges (Figure 3e–f), and PPO continued to 5M reaches the reference under distinct costs and passes it under uniform ones.

4.3 Performance on other attackers and actors

Separate feedback-study experiments evaluate DeepSeek-V3.1 and Qwen3-14B against Meander and B-line under the original objective, using their own controller and evaluation episodes (Table 3; Appendix E). With Return-only feedback, Qwen3-14B achieves a similar mean return to DeepSeek-V3.1 against Meander (-73.3 versus -71.8) and a higher mean against B-line (-33.6 versus -59.7). Rich minus Return-only differences are $+21.8$ and $+4.3$ against Meander, and $+3.8$ and -7.1 against B-line, respectively; the smaller positive differences are modest relative to variation across runs. For Qwen3-14B against B-line, Rich reduces standing cost but increases Restore charges by more, lowering total return (Appendix Table 20). A separate Qwen3-4B study reports -115.3 against Meander under its own protocol (Appendix E.1). Hand-written prompts for Claude Opus 4.6 and GPT-5.1 Codex Max also yield lower observed mean returns than the optimized policies, based on four or five evaluation episodes per condition. The historical PPO crossing times in Table 3 use different evaluation protocols and provide descriptive context for these comparisons.

5 Discussion and Conclusion

At 10K steps, every observed GEPA run exceeds the PPO and GRPO means in all four when-to-whack settings, while both RL baselines lead in the which-to-whack scenarios. This pattern supports the hypothesis that reflection works well when a few trajectories support useful instruction changes. Rich feedback helps under Original CAGE-2, but its benefits vary across settings and attackers. Longer PPO training achieves higher returns, and fewer environment interactions need not imply lower runtime or financial cost. Future work will vary rewards while keeping the rest of the experimental setup fixed, and evaluate additional reflective optimizers and attackers.

Reproducibility statement

Section 3.2 and Table 1 specify the reward configurations; Appendices B and D give seeds, budgets, GEPA, PPO, and GRPO settings, feedback renderers, and the hyperparameter-selection protocol, and

Appendix C the scenario's incident banks, reward structure, and learner settings. We will release the code, initial and optimized instructions, feedback renderers, reflector proposals with judge labels, PPO configurations, the scenario's incident banks and reward table, and per-episode evaluation logs.

ETHICS STATEMENT

This work studies defensive agents in a simulated network. It releases no offensive capability, and the attacker agents are the scripted agents distributed with the public CAGE-2 challenge.

AI USE STATEMENT

Language models are the object of study in this work: they act as the defender, the reflector, and the judge that labels proposals. We also used a language-model assistant to help write analysis scripts, to check numbers against logs, and to reorganize and edit the manuscript. The authors reviewed all content and are responsible for it.

REFERENCES

Rahaf Abu Hara, Vaibbhav Murarri, and Claudio Zito. Reflective prompted policy optimization: Trajectory-grounded revision and salience bias. *arXiv preprint arXiv:2605.08315*, 2026.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026.

Alex Andrew, Sam Spillard, Joshua Collyer, and Neil Dhir. Developing optimal causal cyber-defence agents via cyber security simulation. *arXiv preprint arXiv:2207.12355*, 2022.

Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. RUDDER: Return decomposition for delayed rewards. *Advances in Neural Information Processing Systems*, 2019.

Sebastián R Castro, Roberto Campbell, Nancy Lau, Octavio Villalobos, Jiaqi Duan, and Alvaro A Cardenas. Large language models are autonomous cyber defenders. In *2025 IEEE Conference on Artificial Intelligence (CAI)*, pp. 1125–1132. IEEE, 2025.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next AutoDiff: Generative optimization with rich feedback, execution traces, and LLMs. In *Advances in Neural Information Processing Systems*, 2024.

Hassan Chowdhry, Jaume Manero, and Srinivas Sampalli. Evaluating ai agents for cyber defense: A comparison of deep reinforcement learning and llm approaches. In *Intelligent Data Engineering and Automated Learning – IDEAL 2025*, pp. 423–433. Springer Nature Switzerland, 2026.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Myles Foley, Chris Hicks, Kate Highnam, and Vasilios Mavroudis. Autonomous network defence using reinforcement learning. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security*, pp. 1252–1254, 2022.

Zhaolin Gao, Yu Wang, Bo Liu, Thorsten Joachims, Kianté Brantley, and Wen Sun. p1: Better prompt optimization with fewer prompts. *arXiv preprint arXiv:2604.08801*, 2026.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, 2024.

Kim Hammar, Neil Dhir, and Rolf Stadler. Optimal defender strategies for CAGE-2 using causal modeling and tree search. *arXiv preprint arXiv:2407.11070*, 2024.

Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado van Hasselt, Greg Wayne, Satinder Singh, Doina Precup, and Rémi Munos. Hindsight credit assignment. *Advances in Neural Information Processing Systems*, 2019.

Stephane Hatgis-Kessell and Emma Brunskill. When are LLMs sufficient policy optimizers for sequential RL tasks? *arXiv preprint arXiv:2605.30719*, 2026.

Joey Hong, Kang Liu, Zhan Ling, Jiecao Chen, and Sergey Levine. Natural language actor-critic: Scalable off-policy learning in language space. *arXiv preprint arXiv:2512.04601*, 2025.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024.

Zishang Jiang, Tingyun Li, Jinyi Han, Xinyi Wang, Sihang Jiang, Yizhou Ying, Xiaojun Meng, Jiansheng Wei, Jiaqing Liang, and Yanghua Xiao. From outcomes to actions: Leveraging hindsight for long-horizon language agent training. In *International Conference on Machine Learning*, 2026.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. DSPy: Compiling declarative language model calls into self-improving pipelines. In *International Conference on Learning Representations*, 2024.

Mitchell Kiely, David Bowman, Maxwell Standen, and Christopher Moir. On autonomous agents in a cyber defence environment. *arXiv preprint arXiv:2309.07388*, 2023.

Mitchell Kiely, Metin Ahiskali, Etienne Borde, Benjamin Bowman, David Bowman, Dirk Van Bruggen, KC Cowan, Prithviraj Dasgupta, Erich Devendorf, Ben Edwards, et al. CAGE challenge 4: A scalable multi-agent reinforcement learning gym for autonomous cyber defence. *AI Magazine*, 46(3):e70021, 2025a.

Mitchell Kiely, Metin Ahiskali, Etienne Borde, Benjamin Bowman, David Bowman, Dirk Van Bruggen, KC Cowan, Prithviraj Dasgupta, Erich Devendorf, Ben Edwards, et al. Exploring the efficacy of multi-agent reinforcement learning for autonomous cyber defence: A CAGE challenge 4 perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 28907–28913, 2025b.

Isaiah J King, Benjamin Bowman, and H Howie Huang. Automated cyber defense with generalizable graph-based reinforcement learning agents. *arXiv preprint arXiv:2509.16151*, 2025.

Duc Huy Le and Rolf Stadler. Learning optimal defender strategies for CAGE-2 using a POMDP model. In *2025 21st International Conference on Network and Service Management (CNSM)*, pp. 1–9. IEEE, 2025.

Shiyan Liu, Qifeng Xia, Qiyun Xia, Yisheng Liu, Xinyu Yu, and Rui Qu. Reflection in the dark: Exposing and escaping the black box in reflective prompt optimization. *arXiv preprint arXiv:2603.18388*, 2026.

Johannes Loevenich, Erik Adler, Tobias Huerten, and Roberto Rigolin F Lopes. Design and evaluation of an autonomous cyber defence agent using DRL and an augmented LLM. *Computer Networks*, 262:111162, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Microsoft Defender Research Team. CyberBattleSim. `https://github.com/microsoft/cyberbattlesim`, 2021.

Marvin Minsky. Steps toward artificial intelligence. In *Proceedings of the IRE*, volume 49, pp. 8–30, 1961.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Hamoun Mohammadi, Jonathan J Davis, and Mitchell Kiely. Leveraging large language models for autonomous cyber defense: Insights from CAGE-2 simulations. *IEEE Intelligent Systems*, 2025.

Tianwei Ni, Michel Ma, Benjamin Eysenbach, and Pierre-Luc Bacon. When do transformers shine in RL? decoupling memory from credit assignment. In *Advances in Neural Information Processing Systems*, 2023.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of EMNLP*, 2024.

Chanwoo Park, Xiangyu Liu, Asuman Ozdaglar, and Kaiqing Zhang. Do LLM agents have regret? a case study in online learning and games. In *International Conference on Learning Representations*, 2025.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of EMNLP*, pp. 7957–7968, 2023.

Xuanfei Ren, Allen Nie, Tengyang Xie, and Ching-An Cheng. POLCA: Stochastic generative optimization with LLM. *arXiv preprint arXiv:2603.14769*, 2026.

Maria Rigaki, Ondřej Lukáš, Carlos A Catania, and Sebastian Garcia. Out of the cage: How stochastic parrots win in cyber security environments. *arXiv preprint arXiv:2308.12086*, 2023.

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

Jonathon Schwartz and Hanna Kurniawati. Autonomous penetration testing using reinforcement learning. *arXiv preprint arXiv:1905.05965*, 2019.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, 2010.

Maxwell Standen, Martin Lucas, David Bowman, Toby J Richer, Junae Kim, and Damian Marriott. CybORG: A gym for the development of autonomous cyber agents. *arXiv preprint arXiv:2108.09118*, 2021.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Nikita Vassilyev, William Berrios, Ruowang Zhang, Bo Han, Douwe Kiela, and Shikib Mehri. Reflective context learning: Studying the optimization primitives of context space. *arXiv preprint arXiv:2604.03189*, 2026.

Melody Wolk, Andy Applebaum, Camron Dennler, Patrick Dwyer, Marina Moskowitz, Harold Nguyen, Nicole Nichols, Nicole Park, Paul Rachwalski, Frank Rau, et al. Beyond CAGE: Investigating generalization of learned autonomous network defense policies. *arXiv preprint arXiv:2211.15557*, 2022.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations*, 2023.