POPULATION-BASED SELF-PLAY FOR LLM SAFETY TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

While self-play promises unbounded synthetic data for LLM post-training, standard single-policy methods quickly plateau. Training against only the latest opponent causes policies to cycle between behaviors and collapse onto narrow strategies. We address this by applying population-based self-play via Policy-Space Response Oracles to attacker–defender safety training. In each generation, a new policy is initialized from the base model and trained against the Nash equilibrium of the entire opposing population. To make population payoff matrices scalable at 8B parameters, we introduce a racing evaluation budget and a rollout cache. On standard safety benchmarks, our defender achieves 96.2% average safety, outperforming baseline self-play methods (at most 86.4%) at helpfulness comparable to the other self-play methods and with little loss of general capability. Ablations confirm that each element is necessary: training against single opponents induces cycling, uniform weighting degrades safety, and warm-starting triggers policy collapse. Finally, we show that strong populations expose reward specification flaws, enabling targeted fixes to the reward function.

1 INTRODUCTION

RL post-training improves language models by learning from feedback and environment interaction rather than static supervision. Yet the standard training environments (e.g., math (Guo et al., 2025) and code (Wei et al., 2025)) are hitting a ceiling: as with SFT scaling before them (Yuan et al., 2023), additional compute now yields diminishing returns on held-out tasks. In response, recent efforts have gravitated toward self-play and self-evolving frameworks, where the environment continuously adapts to the model itself to yield an unbounded stream of novel tasks (Zhao et al., 2025). Here, we show empirically that a population of policies holds substantial room for improvement that single-model self-play leaves untapped, and we capture it by applying population-based self-play, an established game-theoretic method, to multi-agent LLM training for jailbreak safety.

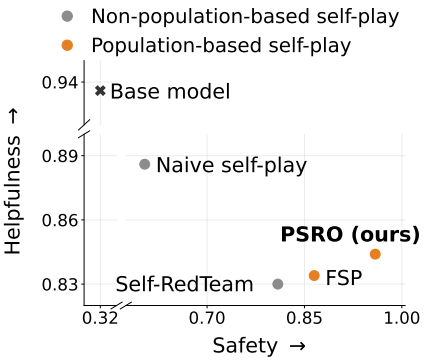

Figure 1: **Safety vs. helpfulness** of defenders trained from the same abliterated Llama-3.1-8B, each at its fifth generation; safety averages all ten harmful-prompt measures of Tables 1 and 3, helpfulness the two benign benchmarks (both axes are broken to show the base model). Population-based self-play (orange) is safer than single-policy self-play (gray); PSRO is the safest of all methods.

Despite its theoretical appeal, self-play for LLMs plateaus quickly in practice. Most existing approaches, which we term *naive self-play*, repeatedly fine-tune a single model against only the latest version of its opponent and discard all earlier versions (Ge et al., 2023). The resulting model collapses onto a narrow set of strategies, forgets how to beat past opponents, and can cycle among strategies that each beat the last, rather than making steady progress (Figure 2 (c)).

Population-based self-play addresses these failures. We adopt its standard game-theoretic form, Policy-Space Response Oracles (PSRO), as developed in prior work (Lanctot et al., 2017; Ma et al.,

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.

2024). Each generation trains a new policy from the base model rather than continuing an old one; every trained policy is retained, so earlier opponents never leave the training distribution; and each new policy trains not against the latest opponent but against a mixture of past opponents, weighted by the Nash equilibrium of their payoff matrix. The idea is simple, but whether it carries over to language models is an empirical question. This paper studies what it takes to make population-based self-play work for LLMs, and what breaks when it is pushed further.

Maintaining a population is not free, however. Each new member must be evaluated against every existing one, so the payoff matrix grows quadratically with the population, and each entry requires thousands of generated and judged conversations. We combine several practical measures to keep training both affordable and stable: each best response is a bounded REINFORCE (Hu et al., 2025) run on a LoRA adapter, and a racing evaluation budget and a rollout cache together cut evaluation cost by roughly an order of magnitude. With these measures, population-based self-play becomes practical for large-scale LLM training.

We evaluate on jailbreak safety, following the attacker–defender setup of prior self-play work (Liu et al., 2026; Malfa et al., 2026). The resulting defender strikes the best balance of safety and helpfulness among the self-play methods we compare (Figure 1): it is clearly safer than single-policy self-play, and it is also safer than fictitious self-play (Heinrich & Silver, 2016), which also keeps a population but weights it uniformly, while staying equally helpful on benign requests. To understand what drives this result, we ablate the two ingredients that distinguish our method from naive self-play. Training each generation against the latest opponent instead of the population (naive self-play) fails to retain its progress across generations, and continuing to train an existing model instead of starting each member from the base model (warm start) causes the defender to collapse into generating unhelpful replies that end in blank output within two generations. Both the population and fresh initialization are therefore needed. The same study exposes what limits the approach: once the population is strong, exploiting flaws in the reward becomes the attacker's best strategy (Gao et al., 2022), a regime naive self-play never reaches, and we harden the reward against these exploits. Together, these results suggest that population-based self-play is a practical route to safer LLMs, provided that every member starts fresh and the reward is hardened against a strong opponent. We release our code to support further work in this direction.

2 MOTIVATION

Before introducing the method, we establish why a population is worth maintaining in our setting, and why a single model trained with naive self-play cannot offer the same benefits. We work in the two-player zero-sum setting of prior self-play work, in which an attacker generates jailbreak prompts and a defender must respond safely. The defender's observation space coincides with the attacker's output space, so it is open-ended and unbounded, and the distribution over it shifts as the attacker learns. No training corpus, evaluation suite, or judge covers more than a small region of this space, so any single run converges to a strategy fitted to the particular region it encountered – a legitimate solution, but only one of many.

A training run is therefore a draw rather than a determinate outcome: the configuration fixes how a policy is trained, but not which of the many valid strategies it arrives at. We verify this with a pilot experiment that motivates the rest of the paper: six copies of the same attacker configuration, differing only in which sample of the training corpus they consume. Their success rates span an order of magnitude on a common held-out set (Figure 2(a), markers at $k{=}1$), and they differ in content as well as strength: the two strongest runs score identically, yet each succeeds on a different set of prompts, sharing fewer than half of their successes. Because the runs fail in different places, together they cover far more than any one alone. We write pool@k for the success rate of k independently trained policies queried once each, counting a test case as solved if any member solves it; pool@k reaches $3.4\times$ pool@1 at $k{=}6$ and is still increasing (Figure 2(a)). Seen from the defender's side, the six attackers together break $3.4\times$ as many prompts as a single one does on average, so the pool is a far stronger adversary to train against than any of its members. This headroom motivates population-based training in multi-agent RL: rather than deploying k models, we aim to capture that coverage in a single policy trained against the population.

The first problem is that a single policy collapses onto a narrow set of strategies. One might expect a policy trained for longer to acquire and retain all of the strategies above by itself, but reinforcement

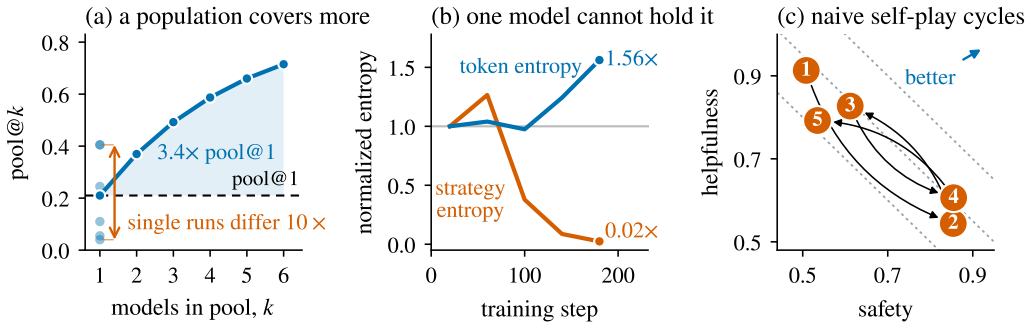

Figure 2: **Why train against a population.** **(a)** pool@k represents the fraction of prompts successfully broken by at least one of k independently trained attackers, with six combined attackers (k=6) breaking 3.4 times as many prompts as a single attacker does on average. **(b)** One policy cannot hold that coverage: as a single attacker trains, the entropy over the attack strategies it uses falls to $0.02\times$ its initial value, even though token-level entropy *rises* to $1.56\times$. **(c)** Naive self-play's defenders, numbered by generation: each generation trades safety for helpfulness or the reverse, and none moves toward the upper right. Protocols and full results are in Section 4.

learning works against this: optimizing expected return concentrates probability mass on whichever strategy currently attains the highest reward, suppressing the alternatives. Prior self-play work has reported that token-level entropy does not collapse during training, which might suggest that the policy remains diverse. We find the opposite at the level of strategies: strategy entropy, the entropy over the attack strategies a policy uses (each attack identified by its opening framing), collapses to $0.02\times$ its initial value while token-level entropy rises to $1.56\times$ (Figure 2b). The policy produces fewer distinct strategies, but expresses them in longer, more varied sentences, so a token-level monitor reads the opposite of what is happening (Section 4.5). Because the limitation is one of retention rather than exposure, training longer does not remedy it.

The second problem is that naive self-play does not keep what it has learned. Strategies in this game beat one another in cycles, so successive opponents do not form a ladder (Figure 2c). First, it trains each new defender against the newest attacker only, so once training moves on, nothing asks the defender to keep resisting the attacks it handled before: attack success repeatedly climbs back up across generations. For example, on WildJailbreak (Jiang et al., 2024), the fourth-generation defender is broken by only 10% of the prompts, yet the fifth, trained against a newer attacker, is broken by 74%. A defense learned against one attacker is lost once training moves on to the next. Second, over five generations its defender switches between refusing broadly, which is safe but unhelpful, and complying broadly, which is helpful but unsafe; the average of safety and helpfulness stays between 0.66 and 0.73, so the defender never finds a policy that does both. A policy trained against only the latest opponent can therefore be pushed around this cycle rather than up a ladder.

These observations define the question this paper studies: *can a single policy trained against an equilibrium-weighted population capture that population's coverage, and what does it take to make this work for LLM self-play?* Section 3 describes how we instantiate Policy-Space Response Oracles for this purpose, and Section 4 examines its performance against single-policy self-play, its scalability with population growth, and the necessity and computational cost of equilibrium weighting.

3 Method

The game. We treat attacker/defender self-play as a two-player game. An attacker policy π^A maps a seed prompt to a jailbreak attempt, a defender policy π^D answers it, and a fixed judge scores the exchange from its labels for prompt harmfulness, response harmfulness and refusal (Appendix B); we write $u(\pi^A, \pi^D)$ for the attacker's expected score and project it to a zero-sum payoff, so the defender's payoff is $-u$. What matters for the training dynamics is that this game is *non-transitive*: as Section 2 showed, training against the current opponent cycles rather than improving, so "the current opponent" is not a reliable proxy for "all opponents".

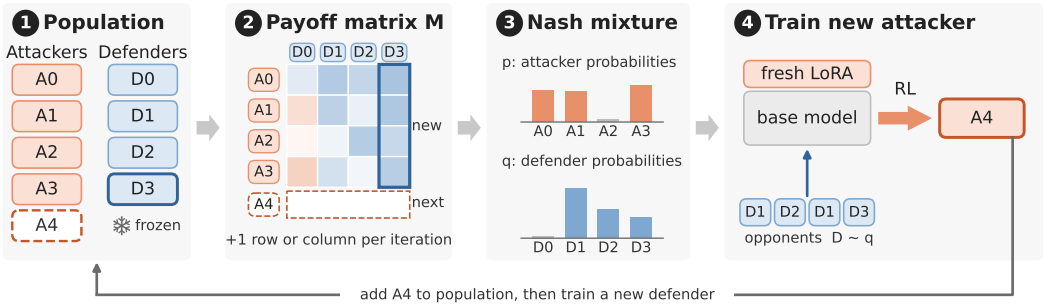

Figure 3: **One iteration of PSRO. (1)** The population holds frozen attackers and defenders; A_0 and D_0 are the base model. **(2)** Each new member adds one row (attacker) or one column (defender) to the payoff matrix M, the only part measured in that iteration. **(3)** Solving M gives the Nash mixtures p and q, probability distributions over the existing members rather than new policies. **(4)** A new attacker is trained with RL from a fresh LoRA on the base model, against defenders sampled from q, and joins the population. The roles then alternate: the next iteration trains a new defender against attackers sampled from p, the one after trains an attacker again, and so on.

PSRO. We build on Policy-Space Response Oracles (Lanctot et al., 2017), which is designed for non-transitive games of exactly this kind. We keep its game-theoretic core and adapt the parts that a language model breaks; we state the algorithm first and the adaptations after it. PSRO replaces the single incumbent opponent of self-play with a *population* of frozen policies per player, $\Pi^A = \{\pi_0^A, \ldots, \pi_m^A\}$ and $\Pi^D = \{\pi_0^D, \ldots, \pi_n^D\}$ (we also write A_i for π_i^A and D_j for π_j^D), together with the *empirical payoff matrix* M, whose entry $M_{ij} = u(\pi_i^A, \pi_j^D)$ is the measured payoff of one attacker against one defender:

$$M = \begin{pmatrix} u(\pi_0^A, \pi_0^D) & \cdots & u(\pi_0^A, \pi_n^D) \\ \vdots & \ddots & \vdots \\ u(\pi_m^A, \pi_0^D) & \cdots & u(\pi_m^A, \pi_n^D) \end{pmatrix} \in \mathbb{R}^{(m+1)\times(n+1)},$$

with $m{+}1 = |\Pi^A|$ rows and $n{+}1 = |\Pi^D|$ columns. M is rectangular in general, since the two populations differ in size whenever one role has been trained and the other has not yet. Each iteration (Figure 3) does two things. It first solves the *meta-game*: treating M as the payoff matrix of a small normal-form game whose pure strategies are the population members, it computes a Nash equilibrium (p, q). Here p and q are probability distributions over the frozen members of Π^A and Π^D; for example, p_i is how often the attacker plays π_i^A. They are not new policies. The meta-game only decides which existing policies to mix, and with what weight. It then calls a *best-response oracle*: a new policy is trained to maximize payoff against the opponent *mixture* q (respectively p), added to the population, and evaluated against every member of the opposing population to extend M. Iterating grows both populations, so each meta-game covers more of the full game than the one before it.

This design resolves the failure modes detailed in Section 2. First, because the population is frozen and monotonically grows, the diverse strategies discovered across distinct members are continuously preserved rather than forcing a single policy to retain them all. Second, because past opponents are never discarded, strategies critical against early-stage attacks remain in the training opponent pool. Finally, by sampling opponents from an equilibrium mixture that concentrates probability mass on opponents that remain hard to beat, a new policy cannot score well by merely exploiting a single outdated vulnerability. Instead, it must systematically improve against the population as a whole, preventing training from getting trapped in non-transitive loops.

Instantiating PSRO for LLMs. Three choices are needed to turn the abstract loop into something that runs on language models. *(i) The oracle is a fixed-budget REINFORCE run.* Since an exact best response is unavailable, each new policy is trained for at most 300 steps with REINFORCE (Hu et al., 2025), normalizing rewards over the whole batch and stopping early once training stops improving (Appendix G), and one opponent is sampled from the mixture for every episode. Appendix G lists the remaining hyperparameters. *(ii) Every oracle starts from the base model.* Each

new population member is a freshly initialized LoRA adapter on the same base model with a fresh optimizer state, so every generation begins with exactly the adaptability the first one had; the base model itself is the first member of each population (π_0^A, π_0^D), which gives the first iteration a well-defined opponent and keeps an untrained reference in the population. We test whether this reset is necessary by comparing it with continuing the previous member instead (Section 4.5). *(iii) One role is trained per iteration* (Figure 3). Each iteration trains one new policy, and the roles alternate: an attacker, then a defender, then an attacker, and so on. After a new policy is added, we measure only its row (attacker) or column (defender) of M and re-solve the equilibrium before training the next policy. Each new defender is therefore trained against a mixture that already includes the attacker added just before it.

The cost of a population. PSRO needs the payoff matrix before every iteration, because the equilibrium that selects the opponents is computed from M. Each new member must therefore be played against the whole opposing population, and a population of K members per role requires K^2 cells. For language models, each cell is expensive: it takes thousands of attacker and defender generations from an 8B model, each scored by the judge. Fictitious self-play avoids this cost, since it weights all past opponents equally and never estimates M, but it spends much of its training on opponents that no longer matter. PSRO instead spends rollouts on the matrix so that training concentrates on the members that remain hard to beat. In practice the trade-off favours PSRO: fictitious self-play's defenders never meet the stopping rule and use 61% more training steps, yet end as helpful as PSRO's and less safe. The matrix costs far less than the training it saves: about 13 minutes per role on average, against 3.8 hours of training (Section 4.6).

Making payoff estimation affordable. The meta-game itself is cheap: it is small and zero-sum, so we solve it exactly as a linear program (Appendix E). The cost lies in filling M, which we reduce in two ways. *(i) A racing evaluation budget.* Only matchups between members that matter to the current equilibrium receive the full episode budget; all other new cells stop after a small pilot batch (Appendix D). *(ii) Reusing frozen rollouts.* The attacker moves first and never sees the defender, and evaluation episodes are seeded, so we cache each attacker's rewrites and reuse them against every defender: a new defender's column needs only defender and judge calls. Cells are keyed on both adapters' hashes, so interrupted runs resume. Training does not use these caches. Together, the racing budget and this reuse cut matrix-estimation cost by roughly an order of magnitude (Figure 5) without changing what is estimated: a cell stopped early is an exact prefix of its full-budget episode sequence, and a reused rollout is identical to the one it would otherwise regenerate.

4 Experiments

4.1 Setup

Models. Both roles start from the same 8B base model, Meta-Llama-3.1-8B-Instruct-abliterated, whose refusal behaviour has been removed by weight editing. Consequently, our defender begins well below standard instruction-tuned baselines in Table 1. Each population member is modeled as a rank-64 LoRA adapter. All game payoffs and training rewards are evaluated by WildGuard (Han et al., 2024), a fixed open-source safety classifier.

Benchmarks. Training draws seed prompts from the WildJailbreak vanilla splits (Jiang et al., 2024): 50,050 harmful and 20,000 benign prompts, sampled in equal proportion during training. The benign half is necessary: a defender scored only on harmful prompts can score perfectly by refusing everything. Evaluation uses four groups of off-the-shelf benchmarks, which we always report together. All of them evaluate a model as the responder to fixed prompts, so every result we report is for the defender; the attacker is used only during training. *Jailbreak attacks* report how often the defender is compromised (attack success rate, ASR): WildGuardTest , WildJailbreak harmful, Do-Anything-Now (Shen et al., 2024), HarmBench (Mazeika et al., 2024), SORRY-Bench (Xie et al., 2025), and StrongREJECT (Souly et al., 2024). *Harmful requests without a jailbreak* report the refusal rate (RTA): the unsafe prompts of XSTest (Röttger et al., 2024). *Over-refusal* reports compliance on benign requests that resemble harmful ones: WildJailbreak benign and the safe prompts of XSTest. *General capability* checks that safety training does not cost the model its usefulness: the mean accuracy over GSM8K (Cobbe et al., 2021), ARC-Challenge (Clark et al., 2018), Hel-

Table 1: **Main comparison of defenders.** Attack success rate (ASR, ↓) on five jailbreak benchmarks, refusal rate (RTA, ↑) on XSTest and StrongREJECT, benign compliance (↑) on WildJailbreak and XSTest, and reasoning accuracy (↑). *Safety* averages 1−ASR and the two refusal rates; *Helpful* averages the two benign columns. Llama rows are trained from the abliterated Llama-3.1-8B-Instruct and reported at their fifth generation; Qwen2.5 rows are transcribed from Malfa et al. (2026). Bold marks the best value per column. More results can be found in Table 3

Model	Harmful refusal							Benign compliance		Reasoning	Average	
	WG:Test adv ASR ↓	WG:Test vani ASR ↓	WJB adv ASR ↓	DAN adv ASR ↓	HarmBench adv ASR ↓	XSTest vani RTA ↑	StrongREJECT adv RTA ↑	WJB adv Comply ↑	XSTest vani Comply ↑	Avg. acc. ↑	Safety ↑	Helpful ↑
Qwen2.5-3B-IT	0.282	0.119	0.949	0.677	0.241	0.610	0.497	0.992	0.960	0.745 ± 0.042	0.548	0.976
Qwen2.5-7B-IT	0.279	0.040	0.897	0.613	0.234	0.915	0.470	0.992	0.912	0.850 ± 0.026	0.617	0.952
+ Self-RedTeam	0.178	0.002	0.726	0.423	0.156	**0.925**	0.371	0.992	0.888	0.818 ± 0.037	0.687	0.940
+ ABS	0.169	0.010	0.756	0.383	0.156	0.915	0.644	0.988	0.884	0.847 ± 0.024	0.726	0.936
Qwen2.5-14B-IT	0.146	0.017	0.845	0.647	0.119	0.735	0.647	**1.000**	**0.984**	$\mathbf{0.874 \pm 0.030}$	0.658	**0.992**
Llama-3.1-8B-IT	0.024	0.007	0.147	**0.003**	0.057	0.920	0.866	0.784	0.932	0.695	0.935	0.858
Llama-3.1-8B-IT-AB (abliterated base)	0.424	0.454	0.938	0.910	0.575	0.415	0.380	0.932	0.940	0.673	0.356	0.936
+ Self-RedTeam (our reproduction)	0.095	0.019	0.159	0.400	0.196	0.835	0.690	0.780	0.880	0.662	0.808	0.830
+ naive self-play	0.234	0.053	0.737	0.630	0.259	0.730	0.647	0.816	0.956	0.650	0.638	0.886
+ fictitious self-play	0.018	0.036	0.301	0.180	0.091	0.775	0.898	0.724	0.944	0.633	0.864	0.834
+ PSRO (ours)	**0.000**	**0.000**	**0.092**	0.040	**0.023**	0.900	**0.989**	0.752	0.936	0.629	**0.962**	0.844

laSwag (Zellers et al., 2019), BoolQ (Clark et al., 2019) and CommonsenseQA (Talmor et al., 2019). Appendix H gives the protocol of every benchmark.

Baselines. We compare against two single-policy self-play methods and one population method. Naive self-play trains a single model per role, continued from one generation to the next, against the latest opponent only. Self-RedTeam (Liu et al., 2026) is an online self-play recipe in which one policy plays both roles; we reproduce it on the same base model. Fictitious self-play (Heinrich & Silver, 2016) keeps a population like ours but weights all past opponents uniformly instead of by the Nash equilibrium. We also ablate warm starting, which continues the previous member instead of starting each new one from the base model (Section 4.5). Every self-play method is reported at its fifth generation. The ABS rows (Malfa et al., 2026) are transcribed from its paper, which evaluates Qwen2.5 only. Full hyperparameters and payoff-estimation settings are given in the appendix.

4.2 RQ1: Does population-based self-play outperform prior self-play methods?

Population-based self-play is clearly safer than single-policy self-play (Table 1). PSRO and fictitious self-play reach an average safety of 0.962 and 0.864, against at most 0.81 for Self-RedTeam and naive self-play, at similar helpfulness. The gap is largest on Do-Anything-Now, where 0.400–0.630 of attacks succeed against the single-policy methods, compared with 0.040–0.180 against the population methods. We evaluate on the benchmarks and protocol of prior self-play work: they are the standard benchmarks in this area and the ones on which published baselines report results. All Llama rows share the base model, harness and judge, and every self-play method is reported at its fifth generation.

Between the two population methods, PSRO is better, and the difference is safety. PSRO has lower attack success than fictitious self-play on every attack benchmark, refuses more unsafe XSTest prompts (0.900 against 0.775) and more StrongREJECT prompts (0.989 against 0.898), and its average safety is 0.10 higher. Their helpfulness is nearly identical (0.844 against 0.834). Section 4.6 traces this to training: fictitious self-play's defenders never meet the stopping rule, while PSRO's converge within 200 steps.

Safety training costs some helpfulness, for every self-play method. All of them comply with fewer benign requests than the base model (0.936): PSRO 0.844, fictitious self-play 0.834, Self-RedTeam 0.830 and naive self-play 0.886, the last because it is also the least safe. General capability drops little: reasoning accuracy stays within 0.044 of the base model for every self-play method. The Qwen rows of Table 1, transcribed from ABS, use a different base model and harness and are indicative only; even so, PSRO, starting from a model whose refusal behaviour has been largely removed, has lower attack success than ABS on Do-Anything-Now (0.040 against 0.383) and HarmBench (0.023 against 0.156).

4.3 RQ2: Does PSRO keep improving as the population grows?

PSRO's defender keeps improving as the population grows, both inside the game and on held-out benchmarks. We first test the defender inside the game. At each iteration, we take PSRO's current defender, the Nash mixture over its defenders, and play it against every attacker in the final population, including attackers trained later, recording the best payoff any of them achieves (Table 2b; lower is better for the defender). This worst-case payoff drops at every iteration, from -0.279 to -0.700: each new defender is harder to beat, even by attackers it never trained against.

The improvement carries over to the held-out benchmarks (Figure 4). Every PSRO defender stays in the high-safety region (0.862–0.959), while helpfulness rises from 0.760 and 0.736 at the first two generations to 0.844–0.882 from the third on, so safety plus helpfulness grows from 1.649 at D_1 to 1.803 at D_5, its highest. Neither baseline improves with more generations (Table 3). Fictitious self-play, which keeps a population but weights it uniformly, ends where it started. Naive self-play trades safety for helpfulness and back (Figure 2c), with safety between 0.603 and 0.825, so its fifth defender falls back to the level of its first.

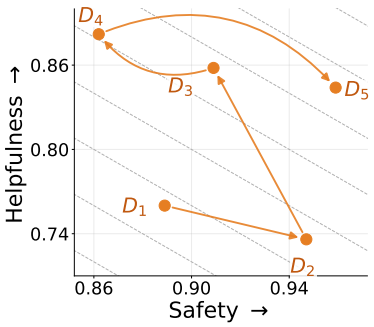

Figure 4: **PSRO's defender improves across generations.** Each point is one generation's defender ($D_1 \rightarrow D_5$), placed by mean safety and mean helpfulness from Table 3. Dashed lines connect points of equal safety + helpfulness; later generations lie on higher lines.

4.4 RQ3: How far can the current reward carry the population?

A strong population turns flaws in the reward into the attacker's best strategy. We found this during development, when running an earlier population for eight generations under the original reward. The first generations improved honestly, but from the third the attacker began to exploit two gaps. In *label drift*, it rewrote benign seeds into harmful requests to collect the harmful-compliance reward; drift rose from 0.7% at the base model to 41.7% by the fifth generation. In *topic substitution*, it replaced a hard harmful request with an easier one it already knew how to execute, which keeps the label harmful and so escapes the drift check. Attacks then collapsed onto templates: in the worst case, 95% shared a single opening clause. Naive self-play, run for the same number of generations, shows no such collapse (at most 0.8% of its attacks share an opening), because it never produces a defender hard enough to make gaming the reward pay off. We fixed both exploits (Appendix C); all other results in this paper use the new reward.

4.5 RQ4: Why does keeping a population help?

A single policy cannot retain strategy diversity (Figure 2 (b)). We train a single attacker against a fixed base defender until convergence, using the same base model, LoRA rank, reward, and judge as in Section 4.1. At each checkpoint we generate attacks on a fixed set of 600 prompts and classify each by its opening clause. The number of distinct strategies falls from 191 to four, and strategy entropy drops to 0.02× its initial value. Token-level entropy rises to 1.56× over the same interval, because the surviving strategies are expressed in longer and more varied sentences. A diversity monitor tracking token entropy would conclude that exploration is increasing at exactly the point it has ceased. A population does not have this problem: reading the attacks and replies of its members shows that each attacker leads with a different tactic, which the next attacker typically abandons, and that the defenders refuse in different styles rather than one (Appendix F). This reading comes from an earlier PSRO run with the same base model but a previous version of the reward function, so it illustrates the trend rather than reproducing the exact numbers of the run reported here.

A population needs members that can still learn. We test a warm-start variant of PSRO that continues the previous same-role adapter instead of training a new one from the base model. Its first defender, trained from the base model as ours are, works as expected; its second, which continues the first, collapses during training. The fraction of its training replies that violate the answer format rises from 0.2 to 0.98, and at its final checkpoint 96–100% of its benchmark answers are empty,

(a) payoff matrix after five iterations

	D_0	D_1	D_2	D_3	D_4	D_5
A_0	−0.47	−0.75	−0.75	−0.74	−0.79	−0.79
A_1	+0.26	−0.73	−0.53	−0.72	−0.73	−0.72
A_2	+0.11	−0.28	−0.75	−0.74	−0.63	−0.74
A_3	+0.11	−0.45	−0.67	−0.51	−0.73	−0.72
A_4	−0.06	−0.67	−0.69	−0.68	−0.70	−0.71
A_5	−0.04	−0.68	−0.67	−0.69	−0.69	−0.70

(b) Nash defender at each iteration

k	Nash after D_k joins	worst case ↓
1	D_1 1.00	−0.28
2	D_1 .33, D_2 .67	−0.59
3	D_2 .60, D_3 .40	−0.60
4	D_2 .26, D_3 .16, D_4 .58	−0.68
5	D_5 1.00	−0.70

Table 2: **Payoff matrix and Nash mixtures. (a)** The payoff matrix after five iterations (attacker's payoff; negative favours the defender). **(b)** The Nash mixture over defenders after each defender D_k joins, which the next attacker A_{k+1} is trained against, and its worst case: the payoff that the strongest attacker of the final population (A_0–A_5, including attackers trained later) obtains against it. Members with zero weight are omitted.

ending after the reasoning block without an answer. The likely cause is entropy collapse: RL has already concentrated the first defender on a narrow set of behaviours, and a successor that starts from it has too little left to explore for further training to succeed. Cold-start PSRO avoids this by drawing every new member afresh from the base model, so each generation starts with the same adaptability. It also tolerates a weak member: a member that trains badly is simply given little weight by the equilibrium, and the next iteration starts again from the base model. In our run, the fourth defender carries most of the weight when it joins (0.58; Table 2b), but once the fifth joins, the equilibrium moves all weight to D_5 and gives D_2–D_4 none.

The payoff matrix exposes non-transitivity, and the equilibrium uses it. No generation dominates while the population grows (Table 2a): the third defender does worse against A_3 (-0.51) than the older D_2 does (-0.67), and the fourth does worse against A_2 (-0.63) than D_2 and D_3 (-0.75, -0.74). The equilibrium therefore mixes across generations instead of following the newest member: from the second iteration on, it spreads its weight over two or three earlier defenders (Table 2b), and it concentrates on a single defender only once one beats every attacker, as D_5 does. Weighting all members uniformly instead would spend up to about 80% of the opponent draws on members the equilibrium gives no weight; Section 4.6 quantifies what this costs.

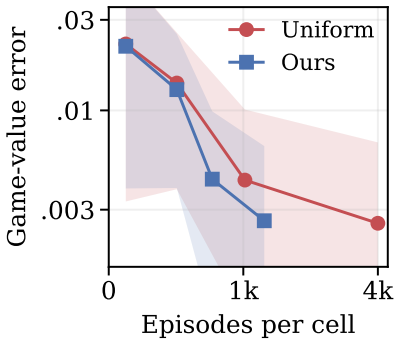

Figure 5: **Racing reaches the same accuracy with fewer episodes.** Median error of the estimated game value on the 8×8 payoff matrix of an earlier eight-generation run vs. the average number of episodes spent per cell, for uniform allocation and our racing scheduler. Both spend the same 256-episode pilot on every cell, after which racing stops sampling settled cells. Bands: 10th–90th bootstrap percentiles.

4.6 RQ5: Is the payoff matrix necessary, and can it be estimated efficiently?

Estimating the payoff matrix costs less than training against outdated opponents. The simpler alternative to Nash weighting is fictitious self-play, which weights all past opponents uniformly and so never estimates the payoff matrix. PSRO pays for the equilibrium by estimating the matrix after every new policy. This cost is small: after each new policy is trained, filling in its row or column of the matrix takes about 13 minutes on average, whereas training the policy takes 3.8 hours. Fictitious self-play pays instead in training: uniform weighting would put up to 83% of an attacker's opponent draws on defenders the equilibrium gives zero weight, and its defenders never meet the stopping rule, running the full 300 steps while PSRO's converge after 172 on average. Over five iterations PSRO uses 1,840 training steps and fictitious self-play 2,960; the extra 1,120 steps take about 17.7 hours at PSRO's rate, against about 2.2 hours of matrix estimation for the ten policies PSRO trains. The extra training does not buy a better defender: fictitious self-play's fifth defender is as helpful as PSRO's but less safe (0.864 against 0.962, Table 1). A likely reason is

that its training signal is diluted: much of each batch comes from outdated opponents the defender already beats, while PSRO concentrates on the opponents that remain hard to beat.

Efficient estimation (Figure 5). A population of K members per role has K^2 cells, and evaluating each with the full budget of $4{,}000$ episodes quickly becomes unaffordable as K grows. We cut this cost in two ways. First, a *racing scheduler* starts every cell with a small pilot of 256 episodes and keeps sampling only the cells whose value is still uncertain enough to change the equilibrium; settled cells stop early. On an earlier eight-generation run with the same method and base model (Appendix D), this lowers the average to $1{,}336$ episodes per cell, 67% fewer than the full budget, while reaching the accuracy that uniform allocation attains only near the full $4{,}000$ (Figure 5). Second, a *rollout cache* saves the result of every matchup. Members never change once they are trained, so a matchup played in one iteration is reused in every later one, and each iteration only plays the matchups that involve its new member. Together, the two mechanisms cut evaluation cost by roughly an order of magnitude without changing the estimated equilibrium.

5 Related Work

Game-theoretic self-play for LLM safety. Closest to us is GRTS (Ma et al., 2024), which casts red-teaming as a two-player game solved with PSRO; we adopt this formulation. GRTS trains 3B models, measures the defender only with the cost model it was trained on, and compares against no self-play baseline. Neither GRTS nor later self-play work separates what the population contributes from what the equilibrium solver contributes. We do, by comparing latest-opponent self-play, fictitious self-play, and PSRO under one setup. We further show at 8B, on external benchmarks, that the method depends on freshly initialised members and a hardened reward, and we make the payoff matrix affordable enough to grow the population. GRTS released no code or checkpoints, so we compare through these ablations rather than directly.

Latest-opponent self-play. Most recent methods train each role against the opponent's current version. Self-RedTeam (Liu et al., 2026) and SSP (Wang et al., 2026) use one policy for both roles. ABS (Malfa et al., 2026), MAGIC (Wen et al., 2026), and AdvGRPO (Bullwinkel et al., 2026) train separate attacker and defender models. MART (Ge et al., 2023) and Active Attacks (Yun et al., 2025) alternate attack generation with defender fine-tuning. Our naive self-play baseline follows this recipe. None of these methods retains past opponents, so none can counter forgetting or cycling. GPT-Red (Wallace et al., 2026) trains against a population of defenders, but does not describe how members are maintained and trained.

Population-based training in games. PSRO (Lanctot et al., 2017) and its scalable variants (McAleer et al., 2021) generalise fictitious self-play (Heinrich & Silver, 2016), which weights all past opponents uniformly; we use fictitious self-play as the baseline for the Nash meta-solver. Populations are needed because real games are non-transitive (Balduzzi et al., 2019; Czarnecki et al., 2020). These methods assume matchups are cheap to evaluate. Where they are not, adaptive sampling allocates evaluations by uncertainty (Rowland et al., 2020), which our racing scheduler applies to LLM rollouts. Continuing to train an existing policy runs into the loss of plasticity (Dohare et al., 2024) and entropy collapse (Cui et al., 2025) documented for continued RL; our warm-start ablation shows the same failure inside PSRO, which is why every member starts from the base model.

6 Conclusion

Self-play methods that train only against the latest opponent cycle and collapse onto narrow strategies. Population-based self-play avoids both by training each new policy from the base model against the Nash equilibrium of the opposing population, and racing evaluation and rollout caching make it affordable at 8B parameters. The resulting defender is the safest among the self-play methods we evaluate while remaining comparably helpful.

The main limitation is that the population is only as good as its reward: a stronger population is also a stronger adversary of the reward function, and it exploited gaps in ours before we fixed them (Section 4.4). Scaling further will require rewards that are robust to exploitation by construction.

AI USE STATEMENT

We used generative AI tools to review experimental methods, interpret results and write research code and configurations. We also used these tools to search and summarize related work, check bibliographic information, draft and edit manuscript text, prepare LaTeX, and generate figures and figure captions. The authors take responsibility for the paper's final content, including AI-assisted text, claims, and artifacts.

REPRODUCIBILITY STATEMENT

Section 3 describes the algorithm and Section 4.1 the experimental setup. Appendices B, G, D, E and H give the reward, the training settings and hyperparameters, payoff-matrix estimation, the meta-game solver, and the evaluation protocol of every benchmark, and Appendix A reports every generation of every run. All models, data and judges we use are publicly available. Each method is trained once, and all reported numbers come from these runs. We release the code, including the training and evaluation configurations, and will release the trained checkpoints upon publication.

REFERENCES

David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech M. Czarnecki, Julien Perolat, Max Jaderberg, and Thore Graepel. Open-ended learning in symmetric zero-sum games, 2019. URL https://arxiv.org/abs/1901.08106.

Blake Bullwinkel, Eugenia Kim, Amanda Minnich, and Mark Russinovich. Learning to attack and defend: Adaptive red teaming of language models via grpo, 2026. URL https://arxiv.org/abs/2606.09701.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions, 2019. URL https://arxiv.org/abs/1905.10044.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng, Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and Ning Ding. The entropy mechanism of reinforcement learning for reasoning language models, 2025. URL https://arxiv.org/abs/2505.22617.

Wojciech Marian Czarnecki, Gauthier Gidel, Brendan Tracey, Karl Tuyls, Shayegan Omidshafiei, David Balduzzi, and Max Jaderberg. Real world games look like spinning tops, 2020. URL https://arxiv.org/abs/2004.09468.

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Maintaining plasticity in deep continual learning, 2024. URL https://arxiv.org/abs/2306.13812.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization, 2022. URL https://arxiv.org/abs/2210.10760.

Suyu Ge, Chunting Zhou, Rui Hou, Madian Khabsa, Yi-Chia Wang, Qifan Wang, Jiawei Han, and Yuning Mao. Mart: Improving llm safety with multi-round automatic red-teaming, 2023. URL https://arxiv.org/abs/2311.07689.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Seungju Han, Kavel Rao, Allyson Ettinger, Liwei Jiang, Bill Yuchen Lin, Nathan Lambert, Yejin Choi, and Nouha Dziri. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms, 2024. URL https://arxiv.org/abs/2406.18495.

Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games, 2016. URL https://arxiv.org/abs/1603.01121.

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. Reinforce++: Stabilizing critic-free policy optimization with global advantage normalization, 2025. URL https://arxiv.org/abs/2501.03262.

Liwei Jiang, Kavel Rao, Seungju Han, Allyson Ettinger, Faeze Brahman, Sachin Kumar, Niloofar Mireshghallah, Ximing Lu, Maarten Sap, Yejin Choi, and Nouha Dziri. Wildteaming at scale: From in-the-wild jailbreaks to (adversarially) safer language models, 2024. URL https://arxiv.org/abs/2406.18510.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Perolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning, 2017. URL https://arxiv.org/abs/1711.00832.

Mickel Liu, Liwei Jiang, Yancheng Liang, Simon Shaolei Du, Yejin Choi, Tim Althoff, and Natasha Jaques. Chasing moving targets with online self-play reinforcement learning for safer language models, 2026. URL https://arxiv.org/abs/2506.07468.

Chengdong Ma, Ziran Yang, Hai Ci, Jun Gao, Minquan Gao, Xuehai Pan, and Yaodong Yang. Evolving diverse red-team language models in multi-round multi-agent games, 2024. URL https://arxiv.org/abs/2310.00322.

Gabriele La Malfa, Emanuele La Malfa, Saar Cohen, Jie M. Zhang, Michael Luck, Michael Wooldridge, and Elizabeth Black. The attacker in the mirror: Breaking self-consistency in safety via anchored bipolicy self-play, 2026. URL https://arxiv.org/abs/2605.08427.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, David Forsyth, and Dan Hendrycks. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal, 2024. URL https://arxiv.org/abs/2402.04249.

Stephen McAleer, John Lanier, Roy Fox, and Pierre Baldi. Pipeline psro: A scalable approach for finding approximate nash equilibria in large games, 2021. URL https://arxiv.org/abs/2006.08555.

Mark Rowland, Shayegan Omidshafiei, Karl Tuyls, Julien Perolat, Michal Valko, Georgios Piliouras, and Remi Munos. Multiagent evaluation under incomplete information, 2020. URL https://arxiv.org/abs/1909.09849.

Paul Röttger, Hannah Rose Kirk, Bertie Vidgen, Giuseppe Attanasio, Federico Bianchi, and Dirk Hovy. Xstest: A test suite for identifying exaggerated safety behaviours in large language models, 2024. URL https://arxiv.org/abs/2308.01263.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models, 2024. URL https://arxiv.org/abs/2308.03825.

Alexandra Souly, Qingyuan Lu, Dillon Bowen, Tu Trinh, Elvis Hsieh, Sana Pandey, Pieter Abbeel, Justin Svegliato, Scott Emmons, Olivia Watkins, and Sam Toyer. A strongreject for empty jailbreaks, 2024. URL https://arxiv.org/abs/2402.10260.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge, 2019. URL https://arxiv.org/abs/1811.00937.

Eric Wallace, Christopher A. Choquette-Choo, Nikhil Kandpal, Sam Toyer, Dylan Hunn, Stephanie Lin, Yuxin Wen, Xiangyu Qi, Christopher Wolff, Zizhao Wang, Milad Nasr, Sicheng Zhu, Chuan Guo, Juan Felipe Cerón Uribe, Kaiwen Wang, Aiden Low, Kai Xiao, and Kai Chen. Gpt-red: Automated red teaming via self-play at scale, 2026. URL https://arxiv.org/abs/2607.26115.

Hao Wang, Yanting Wang, Hao Li, Rui Li, and Lei Sha. Be your own red teamer: Safety alignment via self-play and reflective experience replay, 2026. URL https://arxiv.org/abs/2601.10589.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution, 2025. URL https://arxiv.org/abs/2502.18449.

Xiaoyu Wen, Zhida He, Han Qi, Ziyu Wan, Zhongtian Ma, Ying Wen, Tianhang Zheng, Xingcheng Xu, Chaochao Lu, and Qiaosheng Zhang. Magic: A co-evolving attacker-defender adversarial game for robust llm safety, 2026. URL https://arxiv.org/abs/2602.01539.

Tinghao Xie, Xiangyu Qi, Yi Zeng, Yangsibo Huang, Udari Madhushani Sehwag, Kaixuan Huang, Luxi He, Boyi Wei, Dacheng Li, Ying Sheng, Ruoxi Jia, Bo Li, Kai Li, Danqi Chen, Peter Henderson, and Prateek Mittal. Sorry-bench: Systematically evaluating large language model safety refusal, 2025. URL https://arxiv.org/abs/2406.14598.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models, 2023. URL https://arxiv.org/abs/2308.01825.

Taeyoung Yun, Pierre-Luc St-Charles, Jinkyoo Park, Yoshua Bengio, and Minsu Kim. Active attacks: Red-teaming llms via adaptive environments, 2025. URL https://arxiv.org/abs/2509.21947.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data, 2025. URL https://arxiv.org/abs/2505.03335.

Table 3: **Defender learning dynamics: every generation of every method.** Each row is the defender trained in one generation, evaluated on the same benchmark harness as the main results. Naive self-play, fictitious self-play and PSRO are reported for their first five generations. *Safety* averages all ten harmful columns ($1-$ASR for the eight ASR columns, the rate for the two refusal columns; warm-start D_2 has no StrongREJECT refusal and averages nine); *Helpful* averages the two benign columns. An empty response is scored as a failure on every benchmark: an attack success on the harmful benchmarks and a non-compliance on the benign ones. †Warm-start PSRO collapsed during training: from step 120 of its second defender, most responses stop after the reasoning block without an answer (96–100% empty at the final step 160). To compare it fairly we report that defender's step-100 checkpoint, the last before the collapse, instead of its final one. Its third generation starts from the collapsed defender and is not reported. $\mathrm{SR12}_{\mu}$ and $\mathrm{SR12}_{\max}$ are the mean and maximum StrongREJECT harmfulness score over the 11 wrapping templates. Cells marked – are not yet evaluated.

		Harmful ASR ↓								**Harmful RTA ↑**		**Benign comply ↑**		**Average ↑**	
Method	**Gen.**	$\mathrm{WG}_{\mathrm{adv}}$	$\mathrm{WG}_{\mathrm{van}}$	WJB	DAN	HarmB	$\mathrm{SR12}_{\mu}$	$\mathrm{SR12}_{\max}$	SORRY	XST	StrongREJ.	WJB	XST	Safety	Helpful
Llama-3.1-8B-IT-AB (base)	D_0	0.424	0.454	0.938	0.910	0.575	0.525	0.882	0.884	0.415	0.380	0.932	0.940	0.320	0.936
Naive self-play	D_1	0.249	0.092	0.781	0.677	0.260	0.372	0.813	0.355	0.865	0.762	0.928	0.936	0.603	0.932
	D_2	0.045	0.032	0.095	0.360	0.084	0.189	0.553	0.145	0.905	0.847	0.484	0.824	0.825	0.654
	D_3	0.217	0.044	0.642	0.457	0.237	0.315	0.803	0.184	0.950	0.766	0.904	0.920	0.682	0.912
	D_4	0.024	0.010	0.100	0.367	0.090	0.241	0.747	0.086	0.790	0.793	0.416	0.964	0.792	0.690
	D_5	0.234	0.053	0.737	0.630	0.259	0.391	0.819	0.216	0.730	0.647	0.816	0.956	0.604	0.886
Fictitious self-play	D_1	0.021	0.022	0.381	0.207	0.105	0.057	0.083	0.216	0.830	0.932	0.828	0.864	0.867	0.846
	D_2	0.006	0.007	0.194	0.117	0.029	0.025	0.039	0.114	0.820	0.973	0.636	0.940	0.926	0.788
	D_3	0.012	0.012	0.209	0.113	0.049	0.042	0.118	0.139	0.870	0.919	0.604	0.800	0.910	0.702
	D_4	0.000	0.005	0.067	0.020	0.019	0.022	0.053	0.123	0.810	0.895	0.436	0.872	0.940	0.654
	D_5	0.018	0.036	0.301	0.180	0.091	0.049	0.182	0.161	0.775	0.898	0.724	0.944	0.866	0.834
Cold-start PSRO (invalid-move reward)	D_1	0.018	0.017	0.136	0.103	0.070	0.067	0.170	0.166	0.790	0.852	0.548	0.972	0.889	0.760
	D_2	0.012	0.002	0.049	0.013	0.020	0.022	0.181	0.086	0.895	0.959	0.672	0.800	0.947	0.736
	D_3	0.015	0.007	0.161	0.043	0.045	0.058	0.181	0.120	0.870	0.848	0.812	0.904	0.909	0.858
	D_4	0.021	0.002	0.195	0.167	0.101	0.090	0.299	0.105	0.750	0.849	0.824	0.940	0.862	0.882
	D_5	0.000	0.000	0.092	0.040	0.023	0.019	0.031	0.093	0.900	0.989	0.752	0.936	0.959	0.844
Warm-start PSRO	D_1	0.012	0.005	0.135	0.070	0.051	0.041	0.065	0.205	0.860	0.943	0.692	0.788	0.922	0.740
	D_2†	0.018	0.012	0.300	0.680	0.342	0.029	0.128	0.150	0.865	–	0.504	0.860	0.801	0.682

A Defender generations of the baseline and ablation runs

Table 3 reports every defender generation of four runs that the main text compares against or ablates: naive self-play, fictitious self-play (uniform weighting over the population), cold-start PSRO trained with the invalid-move reward, and warm-start PSRO. Naive self-play does not improve monotonically: its second-generation defender reaches WJB ASR 0.095, and its third gives most of that back (0.642). Across runs, lower ASR is usually bought with benign compliance: the naive second generation complies with only 0.484 of benign WildJailbreak prompts, and fictitious self-play falls to 0.436 by its fourth generation. Low ASR therefore has to be read together with the compliance columns. Warm-start PSRO is a separate case: its second defender collapsed during training, and from step 120 most of its responses end after the reasoning block without an answer. Rather than report a policy that answers almost nothing, we report its step-100 checkpoint, the last before the collapse; its third defender starts from the collapsed one and is omitted.

B REWARD

The oracle reward is computed per role from WildGuard's three labels for the exchange: prompt harmfulness, response harmfulness, and response refusal. The attacker's reward weights these at 0.5, 1.0, and 0.5; the defender's weights response harmfulness and response refusal at 1.0 each. A response that breaks the chain-of-thought format receives -1. Rewards are normalised within each batch and clipped to $[-10, 10]$. Ties are removed before normalisation. Episodes whose judged prompt label differs from the assigned one are handled by the label-drift policy of Appendix C: the attacker's reward is capped at zero, and the episode is dropped when the defender is the role being trained. The oracle reward is general-sum; the zero-sum projection of Section 3 is applied only to the payoff matrix.

C REWARD HACKING IN LONG-HORIZON SELF-PLAY

The mechanism. Each attacker turn is issued a seed prompt together with the task label it is meant to preserve (`generated_harmful` or `generated_benign`), rewrites it, and is scored from the safety judge's labels for the resulting exchange. Two properties of the upstream objective make that score gameable. The refusal component is read from the judge's *observed* label for the rewritten prompt rather than from the label the turn was assigned, and a harmful response is paid the positive attacker reward even when the assignment was benign. Together they let the attacker choose which branch of the reward it is graded under, simply by rewriting a benign seed into a harmful request. On an earlier eight-generation PSRO run with the same base model, trained under this original objective, a drifted benign seed scored $+2.0$ against $+0.007$ for a faithful one, and the drift rate rose from 0.7% at the base model to 41.7% by the fifth generation as successive attackers discovered it.

The fix. We use two measures. The first is an asymmetric label-drift policy: when the judge's label for a rewritten prompt flips relative to the assignment, the attacker's reward is capped at zero, and the episode is dropped entirely when the defender is the optimised role, so a flipped label can never be worth more than playing the assigned task and never contaminates defender training. Payoff cells are handled separately, with benign-to-harmful drift dropped and resampled rather than scored. The second, used for the runs that adopt the intended objective, scores *both* outcome components from the assigned task type rather than the observed label: a harmful response to a benign assignment counts as breaking the task rather than as a win, and any label flip in either direction replaces the whole reward with a fixed floor strictly below the worst faithful outcome, so relabelling is never competitive with playing the task as issued.

What the fix does not cover. Both measures key on the task *label*. They therefore catch a benign assignment rewritten into a harmful one, but not the substitution described in Section 4.4, in which a harmful assignment is replaced by a different harmful request: the label is preserved, so no drift is registered, while the attack no longer bears on the prompt it was issued. Closing this requires a fidelity term relating the rewritten attack to its seed rather than to its label, and we report the fidelity statistics themselves – template share, distinct openings, and content overlap with the assigned prompt – as the instrument that makes the failure visible.

D PAYOFF-MATRIX ESTIMATION

Cell payoff. A cell M_{ij} is the expected zero-sum payoff of attacker π_i^A against defender π_j^D, measured on seed prompts drawn in an exact $50/50$ harmful/benign ratio. Each episode is one attacker rewrite followed by one defender response, scored by WildGuard, and payoffs lie in $[-1, 1]$ with the attacker as the maximiser. Episodes whose judged prompt label drifts from benign to harmful are dropped and resampled rather than scored (Appendix C).

Racing budget. When a new member joins, its row or column of M consists of pending cells, and the scheduler assigns each of them one of two budgets before any is evaluated: a pilot of 256 episodes or the full $4{,}000$. It solves the meta-game on the part of the matrix already measured, filling each unmeasured cell with the mean of its row and column. An attacker is kept if it is one of the two with the highest payoff against the current defender mixture or carries equilibrium weight above 0.01; a defender is kept if it is one of the two that concede the least against the current attacker mixture or carries weight above 0.01. The newest attacker and defender are always kept, since they have no measured cells to be ranked on. A pending cell receives the full budget only if both its attacker and its defender are kept, and the pilot budget otherwise. A reduced budget is an exact prefix of the full-budget episode sequence, so a capped cell estimates the same quantity with fewer samples. The rule is allowed to be wrong: replayed offline on a completed $4{\times}4$ matrix, it used $18.9\times$ less wall-clock time than giving every cell $4{,}000$ episodes and recovered the defender's equilibrium support 62% of the time, against 92% for the full schedule. In the final $8{\times}8$ matrix of an earlier eight-generation run with the same method and base model, 30 cells stopped at 256 episodes, 18 at 768, and 16 ran the full $4{,}000$, for $85{,}504$ episodes in total ($1{,}336$ per cell on average) against $256{,}000$ for uniform full evaluation.

Convergence criterion. A cell is declared converged using bounded empirical-Bernstein confidence intervals, computed simultaneously for three series (all episodes, harmful-only, benign-only) at eight pre-registered episode gates, with a Bonferroni correction over gates and series that keeps the family-wise error at $\alpha = 0.05$ (per-interval $\alpha \approx 0.0021$). Convergence requires a confidence radius of at most 0.1, a change in the running mean of at most 0.05, stability over three consecutive gates, and at least 256 episodes.

Rollout cache. Population members are frozen, so a cell is determined by the identities of its two policies. We key every cell on the SHA-256 hashes of the two adapters and reuse stored episodes whenever a cell is requested again: across iterations, when a run is extended, and when a run resumes after interruption. On the same earlier run, over the 16 iterations that grow the matrix from $1{\times}1$ to $8{\times}8$, it generated $95{,}040$ new episodes; recomputing every requested cell from scratch would have required $492{,}608$.

Figure 5 protocol. The reference equilibrium is the Nash equilibrium of the converged matrix. For a given budget allocation, each cell's value is read at the largest recorded gate not exceeding its budget, and its sampling noise is modelled as Gaussian with the standard deviation implied by that gate's 95% confidence half-width. We draw 500 bootstrap matrices, solve each exactly (Appendix E), and report the median and the 10th–90th percentiles of the absolute game-value error. The uniform curve gives every cell the same budget; the racing curve replays the scheduler, giving each cell the smaller of its final budget and the elapsed per-cell budget.

E EXACT META-GAME SOLVER

The meta-game solved at each iteration is a zero-sum matrix game over the current populations, so its Nash equilibrium is the solution of a linear program rather than a fixed point that has to be approached iteratively. For the row player we solve

$$\max_{p,\,v} \; v \quad \text{s.t.} \quad p^\top M \geq v\mathbf{1}^\top, \;\; p^\top \mathbf{1} = 1, \;\; p \geq 0,$$

and the mirrored program for the column player, using a standard simplex/interior-point solver after shifting M to be positive. At our population sizes this returns the exact equilibrium in milliseconds, and it removes two things that would otherwise have to be tuned and reported: an iteration count and a residual-regret tolerance. We use it in place of the iterative regret matching that PSRO implementations commonly default to; it is a convenience, not a contribution, and either choice recovers the same equilibrium up to solver tolerance.

F WHAT EACH MEMBER OF THE POPULATION LEARNS

The payoff matrix records how well each policy scores, not how it plays. To see what the population contains, we read the stored episodes of the payoff cells of an earlier eight-member PSRO run, trained from the same base model with the method of Section 3 but an earlier version of the reward. The tables therefore illustrate what population members learn, not the exact behaviour of the run reported in the main text. We characterise every attacker by how it rewrites a prompt and every defender by how it answers one. Because every cell is evaluated on the same seed sequence, and a racing-truncated cell is an exact prefix of it (Appendix D), the comparison is paired: all attackers rewrite the same seed prompts, and all defenders answer the same attacks.

Protocol. Tables 4 and 5 are computed from the harmful half of the episodes. For attackers we take each policy's rewrites of the $1{,}642$ harmful seeds that all eight attackers were evaluated on, and label every rewrite with the tactic families it uses (persona or role-play, journalism, fiction, step-by-step or technical detail, authority, urgency, hypothetical, research, covert operation, structured output, security testing) by keyword rules. The *primary* tactic is the family of the earliest match, that is, the framing the rewrite opens with, and a rewrite with no match is a direct request; each column therefore sums to 100%. For defenders we take every defender's replies to the same $1{,}024$ attacks (all eight attackers' rewrites of 128 shared seeds) and assign each reply one style from WildGuard's labels together with two phrase rules (an explicit refusal, and an offer of a safe alternative). The keyword rules are transparent but coarse: they measure which framing a text uses, not whether that framing is what made the attack succeed.

Every attacker attacks from a different aspect. The base attacker π_0^A mostly restates the harmful request (42.6% direct requests, 0.90 tactics per rewrite). Each trained attacker then settles on a different leading tactic (Table 4): π_1^A opens 90.0% of its rewrites with a persona, π_2^A moves to journalism and fiction, π_3^A to urgency (42.1%), π_5^A to journalism (89.6%), and π_6^A to step-by-step technical requests (99.5%). A leading tactic is typically abandoned by the next attacker rather than refined: persona falls from 90.0% at π_1^A to 8.9% at π_2^A, journalism from 89.6% at π_5^A to 0.0% at π_6^A, and step-by-step from 99.5% at π_6^A to 4.9% at π_7^A. On the same seed, consecutive attackers share few tactics: the mean Jaccard overlap of their tactic sets is between 0.04 and 0.24 for every consecutive pair. This is the behaviour the method is built to elicit: each attacker is a best response to a defender that has already been trained against its predecessors, so the attack that still pays is one aimed at a different weakness.

Defenders differ in how they refuse, not only how often. The trained defenders reach similar refusal rates through different styles (Table 5). π_4^D refuses explicitly (66.6% explicit refusals only), π_6^D refuses and offers a safe alternative (56.8%), and π_2^D, π_3^D, π_5^D and π_7^D decline by explaining why the request is a problem without an explicit refusal (78–93%). The styles carry different costs on benign requests: over-refusal ranges from 7.1% (π_4^D) to 37.5% (π_2^D).

Takeaway. No single policy carries the behaviours the population discovered. Journalism, urgency and step-by-step requests each lead an earlier attacker, at 89.6%, 42.1% and 99.5% of its rewrites, yet the final attacker π_7^A opens with them in only 2.5%, 3.4% and 4.9% of its rewrites. A defender trained against the latest attacker alone would therefore be exercised on one aspect of the attack surface at a time, while earlier aspects drop out of its training distribution. Keeping every attacker in the population, and training each new defender against a mixture over it, is what keeps all of these aspects in play; the tables are the direct evidence that the population is not redundant with its most recent member.

Table 4: **Primary attack tactic of every attacker in the population.** Each column rewrites the same 1,642 harmful seed prompts; each rewrite is assigned the single tactic its opening framing uses (the earliest matching tactic keyword), so every column sums to 100%. Bold marks the leading tactic of each attacker. The last row counts all tactics present in a rewrite, not only the opening one. Values are percentages.

Primary tactic	π_0^A	π_1^A	π_2^A	π_3^A	π_4^A	π_5^A	π_6^A	π_7^A
Direct request (no framing)	**42.6**	0.5	14.5	14.2	**51.3**	0.4	0.0	21.1
Persona / role-play	11.0	**90.0**	8.9	6.0	28.8	0.5	0.1	**48.1**
Journalism / documentary	4.4	1.0	**26.1**	4.1	0.1	**89.6**	0.0	2.5
Fiction / creative	8.7	2.6	23.4	12.7	3.7	4.9	0.0	8.4
Step-by-step / technical detail	9.4	1.0	6.5	3.7	5.0	1.5	**99.5**	4.9
Authority / official	6.5	1.4	9.4	10.1	4.9	1.4	0.3	4.6
Urgency / emotion	2.4	0.7	3.3	**42.1**	0.3	0.4	0.0	3.4
Hypothetical	4.8	0.1	0.1	0.1	0.0	0.0	0.0	0.1
Research / academic	3.3	0.7	2.8	2.8	3.2	0.4	0.0	0.4
Covert / espionage	3.8	1.7	1.9	2.5	1.9	0.8	0.1	4.9
Structured output	2.5	0.2	1.2	0.4	0.7	0.1	0.0	1.0
Security testing / simulation	0.5	0.1	1.9	1.4	0.2	0.1	0.0	0.8
Tactics per rewrite (mean)	0.90	2.80	1.81	1.53	0.58	3.05	1.34	1.31

Table 5: **Primary defense style of every defender in the population.** Every defender answers the identical set of 1,024 harmful attacks (all eight attackers' rewrites of the same seeds). Each reply is assigned one style, in order: degenerate if it has fewer than five words; harmful if WildGuard labels the response harmful; otherwise by whether it contains an explicit refusal and whether it offers a safe alternative; a reply WildGuard labels a refusal without an explicit refusal phrase is an implicit refusal, and the remainder are harmless completions. Every column sums to 100%; bold marks the leading style. The last row is the share of each defender's benign requests (all of its cells) that WildGuard labels as refused. Values are percentages.

Primary style (harmful attacks)	π_0^D	π_1^D	π_2^D	π_3^D	π_4^D	π_5^D	π_6^D	π_7^D
Harmful compliance	**90.3**	10.7	0.3	9.1	4.9	3.1	0.6	1.0
Refusal + safe alternative	0.2	**39.6**	0.0	0.1	22.6	0.4	**56.8**	0.5
Explicit refusal only	0.1	10.4	0.0	0.0	**66.6**	0.3	3.4	0.7
Implicit refusal / explanation	2.9	29.7	**92.6**	**77.6**	0.6	**89.9**	31.8	**91.7**
Harmless completion	6.4	9.6	6.9	12.8	1.7	6.0	4.6	5.9
Degenerate / empty (< 5 words)	0.0	0.0	0.2	0.4	3.7	0.3	2.7	0.3
Over-refusal on benign requests	35.5	13.0	37.5	13.0	7.1	9.9	8.1	21.6

G IMPLEMENTATION DETAILS AND HYPERPARAMETERS

Every population member is trained by the same best-response oracle, whose settings are listed in Table 6. Attackers and defenders share all settings except the learning rate and the length of the auxiliary SFT phase. Each oracle starts from a freshly initialised LoRA adapter on the base model, with a fresh optimiser state, and trains against opponents drawn from the current Nash mixture: one frozen opponent is sampled per episode.

Table 6: **Oracle hyperparameters**, shared by every generation of the reported run.

	Attacker	**Defender**
Base model	Llama-3.1-8B-Instruct, abliterated	
Initialisation	fresh LoRA on the base model, fresh optimiser	
LoRA rank / α / dropout	64 / 64 / 0	
LoRA target modules	`q,k,v,o,gate,up,down_proj`	
Policy-gradient estimator	REINFORCE, batch-normalised reward, no critic	
Clip ratio	0.2	
KL penalty to the base model	0 (monitored, not optimised)	
Learning rate	1×10^{-5}	4×10^{-5}
Schedule	constant, 5% linear warm-up	
Optimiser	Adam, $\beta = (0.9, 0.95)$, no weight decay, grad-norm clip 1.0	
Rollout / train batch (prompts)	128 / 32	
Samples per prompt	1	
Maximum oracle steps	300	
Auxiliary SFT phase	first 30 steps	first 10 steps
Episode	2 turns: attacker rewrite, then defender response	
Training prompts	WildJailbreak vanilla, exact 50/50 harmful/benign	
Sampling	temperature 1.0, top-p 1.0, up to 2,048 new tokens	
Judge	WildGuard-7B	
Precision / parallelism	bf16, DeepSpeed ZeRO-3	
Hardware per oracle	4× H200 (policy and vLLM colocated), judge served separately	
Seed	8888	

Auxiliary SFT phase. During the first steps of each oracle (Table 6), a supervised loss on format-demonstration data is added to the policy-gradient loss with weight 1.0, so that a freshly initialised adapter produces well-formed turns before the RL signal dominates. The attacker's data are 1,180 prompt rewrites. The defender's data are 15,000 HelpSteer3 and 15,000 benign WildJailbreak responses, generated by the base model (temperature 0.6, top-p 0.9), filtered by WildGuard, and written in the chain-of-thought response format.

Early stopping. Each oracle trains for at most 300 steps and may stop earlier. Training metrics are averaged over 20-step windows aligned with the checkpoint interval, and no run stops before step 60. An attacker stops once its attack success rate has not improved by 0.01 over its best window for 3 consecutive windows, provided its chain-of-thought format violations stay at or below 0.30; an absolute threshold would never fire for attackers, whose success against a trained defender stays well below 1. A defender stops once the attack success rate against it stays at or below 0.01 for 3 consecutive windows while its benign compliance stays at or above 0.912 and its format violations at or below 0.10; the compliance guard prevents a defender from meeting the rule by refusing everything. Either role is stopped as collapsed if its format violations reach 0.80 on 2 consecutive windows. The checkpoint at the stopping step becomes the new population member.

Checkpoint selection. Checkpoints are saved every 20 steps, and each new population member is the final checkpoint of its oracle run: the checkpoint at the early-stopping step, or at step 300 if the rule never fires. We do not select checkpoints on held-out prompts, because the training prompts leave no disjoint set to select on.

H EVALUATION BENCHMARKS

All benchmarks evaluate the defender as the responder to a fixed set of prompts. Table 7 lists each one; the paragraphs below give the protocol.

Table 7: **Evaluation benchmarks.** Prompts is the number of prompts scored. WildGuard labels each response for harmfulness and refusal.

Group	Benchmark	Prompts	Metric	Judge
Jailbreak	WildGuardTest (adv / vani)	337 / 412	ASR (harmful response)	WildGuard
	WildJailbreak harmful	2,000	ASR (compliance)	WildGuard
	Do-Anything-Now	300	ASR (compliance)	WildGuard
	HarmBench (precomputed)	1,500	ASR (harmful response)	WildGuard
	StrongREJECT (Table 1)	2,294	$1-$ASR	WildGuard
	StrongREJECT, 12 templates	313×12	harmfulness score	StrongREJECT grader
	SORRY-Bench	440	fulfilment rate	SORRY-Bench judge
No jailbreak	XSTest unsafe	200	RTA	WildGuard
Over-refusal	WildJailbreak benign	250	compliance	WildGuard
	XSTest safe	250	compliance	WildGuard
Capability	GSM8K (5-shot)	1,319	exact match	–
	ARC-Challenge	1,172	accuracy	–
	HellaSwag	10,042	accuracy	–
	BoolQ	3,270	accuracy	–
	CommonsenseQA	1,221	accuracy	–

Safety harness. Every benchmark judged by WildGuard runs through the evaluation harness of Han et al. (2024), with WildGuard decoding greedily. Instruction-tuned models are served with their own chat template; our trained defenders are served with the reasoning template they were trained in, and only the text inside the answer tags is judged, truncated to 512 tokens. Defenders decode greedily on WildGuardTest, WildJailbreak, Do-Anything-Now and HarmBench, and sample once at temperature 1.0 on XSTest. WildGuardTest and HarmBench count a response as an attack success when WildGuard labels it harmful; WildJailbreak and Do-Anything-Now count it when WildGuard labels it a compliance, and the benign WildJailbreak split reports the same compliance rate, where higher is better. On WildJailbreak and Do-Anything-Now the judge is shown the plain request rather than the jailbreak that wraps it. Refusal (RTA) and compliance on XSTest use WildGuard's refusal label. In Table 3, an empty response is scored as a failure on every benchmark.

StrongREJECT. We report StrongREJECT (Souly et al., 2024) under two protocols, both with jailbreaks applied. The column in Table 1 follows Malfa et al. (2026): it pairs StrongREJECT's forbidden prompts with its jailbreak variants (the unwrapped prompt and 36 jailbreaks), samples 62 prompts per variant for $2{,}294$ generations at temperature 0.7, and reports one minus the rate at which WildGuard labels the response harmful. The second protocol applies the static wrapping templates released with the benchmark to all 313 prompts: AIM, evil confidant, developer mode v2, refusal suppression, prefix injection, distractors, short style injection, Wikipedia with title, poems, few-shot JSON and combination 2, together with the unwrapped prompt. We exclude the encoding-based templates (Base64, ROT13, disemvowel), because the grader scores the raw response and would mis-score an encoded harmful answer as harmless. Responses are generated greedily (up to $2{,}048$ tokens) and scored with the official fine-tuned StrongREJECT grader, which returns a harmfulness score in $[0, 1]$ (the expected value of its 1–5 rating). We report the mean and the maximum of the per-template scores over the 11 wrapped templates; the unwrapped score is reported separately.

SORRY-Bench. The 440 unsafe requests of SORRY-Bench (Xie et al., 2025) are answered greedily (up to $2{,}048$ tokens) and labelled by the benchmark's fine-tuned judge (Mistral-7B-Instruct-v0.2, released June 2024); we report the fraction of responses that fulfil the request.

General capability. Accuracy is measured with the LM Evaluation Harness on vLLM, without a chat template. GSM8K is 5-shot and scored by exact match on the final answer; ARC-Challenge,

HellaSwag, BoolQ and CommonsenseQA are zero-shot and scored by log-likelihood accuracy. We report the unweighted mean of the five.