
ACEA: An Adversarial Co-Evolution Arena for Head-to-Head Red-Team and Blue-Team LLM Testing

Yi Ting Shen
Vulcan Research, AIFT
Singapore
yiting.shen@aift.io

Kentaroh Toyoda
Vulcan Research, AIFT
Singapore
kentaroh.toyoda@aift.io

Alex Leung
Vulcan Research, AIFT
Singapore
alex.leung@aift.io

Abstract

Automated red-team attacks and blue-team defenses for large language models (LLMs) are advancing quickly. However, attackers and defenders are built and tested in isolation, and the resulting scores are hard to trust. To tackle this, we present ACEA (Adversarial Co-Evolution Arena), a platform that connects a pluggable red-team adapter and a pluggable blue-team adapter to a shared target LLM and scores their attack and defense rates with an LLM judge. ACEA contributes four components. First, a pluggable, model-agnostic arena. Any red or blue project connects over a minimal HTTP protocol, which we call the ACEA Standard Adapter Protocol (ASAP). It can be written in any language, and a project that exposes nothing but the protocol is a full participant. Second, an evaluation methodology built for adversarial rounds. Seeding the target with canonical secrets gives verifiable ground truth that separates real leakage from hallucination. We also send each attack to the target even when the defense blocks it, which measures the attack’s raw potency independently of whether it was stopped. Together these yield a per-round decomposition of attack strength and defense effectiveness. Third, a real-time, game-style visualization with a detailed end-of-battle report that localizes each failure. The evaluation thus becomes an actionable signal for improving a red or blue project. Fourth, an optional in-context improvement loop that turns each round’s outcome into advisory hints for the next. An adapter can then adapt across rounds without keeping state, provided it reads the hints. We describe the design of ACEA and the metrics through which red and blue teams are scored head to head. Code is available at <https://github.com/VulcanLab/ACEA>.

1 Introduction

Large language models (LLMs) are now deployed in settings where adversarial inputs carry real consequences, from prompt injection and jailbreaks to attempts at extracting confidential data. Assessing how well a model, or a guardrail protecting it, withstands such inputs has become a core safety activity. The dominant practice is red-teaming: eliciting harmful or policy-violating behavior through crafted prompts, either by human experts or, increasingly, by other language models (e.g., [37, 35]). A parallel line of work builds defenses, such as input and output guardrails that detect and block unsafe content [16].

These two lines of work, offense and defense, advance largely in parallel. What a practitioner ultimately needs to know is how a particular attacker matches up against a particular defender. Current practice makes that surprisingly hard to answer. We identify three gaps. **(1) Isolation:** the

AI usage declaration: We used Anthropic’s Claude Opus 4.8 and Z.AI’s GLM-5.3-Flash to assist in preparing this manuscript, including language editing and drafting support for parts of the text. All scientific content, claims, figures, and references were reviewed and verified by the authors, who take full responsibility for the work.

two sides are exercised separately. An attacker is tuned against a fixed target, and a guardrail is scored against a fixed set of attacks. There is no common ground on which one team’s attacker and another team’s defender can be pitted directly against each other. **(2) Untrustworthy scores:** harm is usually graded by an LLM judge that cannot tell a genuine data leak from a plausible-looking hallucination. When a defense blocks an attack, the record shows only that it was blocked, revealing nothing about how strong the attack was. Attack strength and defense effectiveness are therefore conflated. **(3) No live view:** results are reported only after a run, as tables or logs. One cannot watch how an attack breaks through or how a defense holds while the contest is unfolding.

In this paper, we address these gaps with ACEA (Adversarial Co-Evolution Arena), a platform in which a red-team adapter and a blue-team adapter face each other over a shared target LLM. Each round proceeds through a fixed pipeline. The red adapter produces an attack. The blue adapter decides whether to block it or allow it, and may supply a sanitized rewrite in place of the original. The target responds, the blue adapter may filter that response, and an LLM judge scores the exchange on a small set of safety dimensions. ACEA meets the three gaps with three design choices. Every adapter connects through a minimal HTTP protocol, closing gap (1). An arbitrary attacker and defender, written in any language, can connect via the ACEA Standard Adapter Protocol (ASAP) and compete against each other. For gap (2), we provide the target with known synthetic secrets, so the judge can confirm a real leak rather than accept a convincing hallucination. Even when the defense blocks an attack, the target is still generated from the original payload, and that generation is scored but never delivered. This measures the attack’s raw potency separately from whether the defense stopped it. Finally, resolving gap (3), a game-style interface renders the contest live. Because every round is recorded, the same run can afterward be read as a report of where and why a defense failed. Beyond closing the three gaps, we build an optional in-context improvement loop that feeds each round’s outcome back to the next as advisory hints, so a stateless adapter can adapt across rounds without keeping state.

The remainder of the paper reviews related work (Section 2), describes ACEA’s objective, positioning, and design (Section 3), reports experiments that check the scoring’s resolution (Section 4), presents the visualization and reporting (Section 5), discusses limitations and ethics (Section 6), and concludes (Section 7).

2 Related Work

Existing work relevant to ACEA can be grouped into four directions, namely (1) automated and agentic LLM red-teaming, (2) LLM defenses and guardrails, (3) self-play, co-evolution, and self-improving agents, and (4) evaluation frameworks and benchmarks. We summarize the key papers in each direction, then close the section by identifying three gaps that they leave open.

Automated and agentic LLM red-teaming. Automated attackers have progressed from optimizing a single adversarial input toward agents that plan over many turns. Early work uses one language model to generate test cases for another [26, 11], searches for adversarial suffixes [41], or refines jailbreak prompts through iterative querying, as in PAIR [4] and TAP [23]. More recent attackers are multi-turn and agentic. Crescendo [27] escalates a benign conversation into prohibited content. GOAT [25] drives adversarial dialogues by reasoning over a toolbox of techniques. AutoDAN-Turbo [18] is a lifelong agent that discovers and reuses jailbreak strategies. A further step treats red-teaming itself as a design or learning problem. AgenticRed [35] evolves whole red-team agentic systems, and Active Attacks [37] trains an attacker against an environment that periodically hardens the victim. In all of these, the attacker is the object of study, and the defender, when present, is fixed or refreshed only periodically.

LLM defenses and guardrails. On the defensive side, moderation models classify inputs and outputs as safe or unsafe, from Llama Guard [16] to more recent one-stop and reasoning-based guards such as WildGuard [13], ShieldGemma [38], Aegis2.0 [12], and GuardReasoner [19]. Other defenses harden the model itself, whether through AI-feedback alignment [2], representation-level circuit breakers [42], or constitutional classifiers trained over thousands of hours of red-teaming [28]. Such defenses are shipped as fixed artifacts and typically compared on fixed benchmark suites.

Self-play, co-evolution, and self-improving agents. Co-evolution has a long history in game-playing, where self-play produced superhuman policies without human data [30]. In the LLM setting, agents improve their own outputs through feedback and reflection [20, 29], evolve their own prompts [10], or judge and reward themselves [36]. A further line rewrites their own source code: ADAS [14] programs new agents, while the Darwin Godel Machine [39] and AlphaEvolve [24] evolve code against a fitness function. A recent line makes two roles co-evolve directly. R-Zero [15] and Multi-Agent Evolve [6] let a challenger and a solver compete against each other to improve reasoning. In the adversarial safety setting most related to ours, Self-RedTeam [17], MAGIC [34], CHASE [21], AdvGRPO [3], and Be-Your-Own-Red-Teamer [33] co-evolve a red attacker and a blue defender for safety (i.e., red/blue co-evolution). These methods co-adapt both sides through reinforcement-learning weight updates. While some utilize a single model to play both roles (e.g., Self-RedTeam [17]), others specifically decouple the attacker and defender to avoid optimization conflicts (e.g., MAGIC [34]). Regardless of their architecture, these methods rely on updating the weights of their own attacker and defender policies; CHASE [21] interacts with the target black-box but still trains those policies via GRPO. This ties them to specific models and excludes a participant whose weights are unavailable.

Evaluation frameworks and benchmarks. Standardized evaluation frameworks make red-teaming comparable and repeatable. HarmBench [22], JailbreakBench [5], and StrongREJECT [31] provide fixed behavior sets, leaderboards, and calibrated scorers. garak [9] and DeepTeam [7] package libraries of probes. General frameworks such as Inspect [32] standardize evaluation tooling. As agents gain tool access, benchmarks such as AgentHarm [1] and AgentDojo [8] measure harmful agent behavior and prompt-injection robustness. These frameworks score a system against a set of fixed prompts in a single pass and report results post-hoc. Some, such as Inspect, provide a log viewer for inspecting completed runs, but none render an adversarial battle live round by round.

Positioning. Across these four directions, we identify three gaps that no single approach closes. Automated attackers and the recent reinforcement-learning methods leave gap (1) open in different ways. The former tune an attacker against a fixed target. The latter co-train both sides, but only by updating the weights of specific models. Neither provides a common, model-agnostic arena in which an arbitrary attacker meets an arbitrary defender. Evaluation frameworks leave gap (2) open, grading harm without verifiable leakage ground truth and treating a blocked attack as a single outcome that hides its strength. And gap (3) is unmet throughout. Results are reported after the fact, with at most a log viewer rather than a live view of the contest. ACEA is designed to close the three together.

3 The ACEA Platform

We propose ACEA, the arena that closes the three gaps. We first state its objective and position it against existing tools, then detail its components, and finally compose them into a single round.

Objective. Rather than measure one side against a fixed counterpart, we run both policies together over the rounds and score each exchange with the judge. The goal is to realize this head-to-head evaluation in a way that is (i) *pluggable*, so arbitrary red and blue systems participate through a common interface, with no requirement to expose anything beyond it; (ii) *model-agnostic*, so any implementation can take part without access to the other side’s or the target’s weights; (iii) *trustworthy in scoring*, so that leakage is verified rather than guessed and an attack’s strength is measurable even when the defense blocks it; and (iv) *interpretable*, so each round’s attack, decision, response, and verdict can be inspected as the contest unfolds. Interpretability and trustworthy scoring together serve a further end, namely to make the evaluation *actionable*. By localizing where and why a defense failed and confirming that each failure is genuine, the arena produces a signal that supports later improvement, whether by a human developer or an automated loop.

Table 1 contrasts ACEA with the system-level approaches most comparable to an arena, namely evaluation frameworks and co-evolution training methods, along these axes. In the table, ✓ marks a feature the tool provides natively, × a capability that applies to the tool’s class but is not offered, and N/A one that does not apply to that class (e.g., arena/harness axes for a weight-training co-evolution method). Prior tools typically satisfy a subset. Evaluation frameworks offer pluggable probe libraries but hold one side fixed and grade harm without verifiable ground truth. The recent reinforcement-learning methods exercise both sides but require weight access and are tied to specific models. None

Table 1: Positioning ACEA against representative system-level LLM security tools.

Category	Tool	Pluggable adapters	Both sides (red & blue)	Leakage ground truth	Live UI	LLM-judge scoring
Red/blue co-evolution	Self-RedTeam, MAGIC, <i>et al.</i> [17, 34, 21, 3, 33]	N/A	✓	N/A	N/A	✓
Eval. frameworks	HarmBench / Jailbreak-Bench [22, 5]	✓	×	×	×	✓
	garak [9]	✓	×	×	×	×
	DeepTeam [7]	✓	×	×	×	✓
Arena	ACEA (ours)	✓	✓	✓	✓	✓

combine a protocol-level, model-agnostic arena with verifiable leakage ground truth and a live view of the contest.

ACEA is implemented as a set of containerized services that communicate over HTTP and a shared event stream. All model calls route through a single proxy so that the language models backing each role are configurable per deployment. The red and blue adapters connect directly to the arena core, which executes each round against the target and forwards every exchange to the judge. The judge’s scores are recorded in a trace store that feeds both the end-of-battle report and the live view (Section 5). A round never requires anything beyond the protocol: the arena composes it from request and response alone, and a project that exposes nothing else plays a complete battle and is scored identically. When the in-context loop is enabled, the arena additionally reads a bounded, read-only sample of a participant’s source, if the operator has mounted it, to summarise the project and propose a strategy suited to it; the sample is analysed and never written to, and a project whose source is not mounted is profiled from its declared capabilities alone. We first describe the arena core and how it composes a round (Section 3.1), then detail each component.

3.1 Arena core and the battle loop

The arena core registers adapters and executes the battle loop. Each round runs a fixed pipeline: generate attack, evaluate defense, query the target, optionally filter the output, and score. A subtlety concerns blocked payloads. If a blocked attack were simply discarded, the arena would lose all information about how strong that attack actually was. A weak attack the defense easily caught would look identical to a devastating one it barely stopped. ACEA therefore still generates the target’s response to the attack even when the defense blocks it, and scores that generation without delivering it. This yields two independent measurements each round, the attack’s raw potency and the defense’s effectiveness, so attack strength and defense success are never conflated. The two are aggregated into a continuous score in $[0, 1]$ for each side: for the attacker, the raw potency it produced whether or not it was delivered; for the defender, the harm removed between the raw and the delivered output. The in-context loop improves that score rather than the binary rates, which is what lets a side improve measurably across rounds it lost.

A third distinction is needed for the defense’s measurement to mean anything. A round in which the defense passed the payload through untouched and the target then declined by itself is a defended round, correctly, and yet the defense contributed nothing to it. Each round therefore records which of four things ended it: the defense refused at its input gate, the defense altered the reply on the way out, the target refused unaided, or the attack succeeded.

The loop supports bounded and open-ended modes and carries a short conversation memory across rounds to enable multi-turn attacks. It terminates on the round cap or on user-defined conditions evaluated after every round: a target attack success rate, a target defense rate, a consecutive-win streak, or an improvement in attack success rate of a given number of percentage points over the baseline captured at the start. Each may be restricted to the last N rounds instead of the whole battle. There is no automatic saturation stop: the operator decides when to stop a run, and the report states which condition ended it. Every round is written to a trace store and streamed as events, which underpins both after-the-fact inspection and the live visualization.

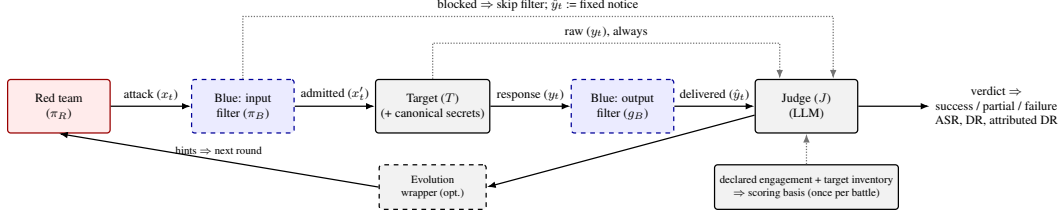


Figure 1: How ACEA scores one round, repeated for $t = 1, \dots, N$. The judge scores against a basis the participants declared (below) and returns a three-level verdict; a declared evidence marker decides the round without a model call.

We now make this loop precise. Figure 1 illustrates one round. Let T denote a target LLM. A red team applies a red policy π_R that maps a conversation history to an attack payload. A blue team applies a blue policy π_B that may screen the incoming payload with a decision $d_t \in \{\text{block}, \text{allow}\}$, optionally returning a rewritten payload in place of the original, filter the target’s response with an output filter g_B , or both. It must use at least one. A battle is a sequence of rounds $t = 1, \dots, N$. In round t , the red team emits $x_t = \pi_R(h_t)$ and the blue team decides $d_t = \pi_B(x_t)$. The payload that reaches the target follows this decision: $x'_t = x_t$ under **block** and under **allow**, and x'_t is the defender’s rewritten payload when it supplies one. The target is generated from exactly once per round, yielding the raw response $y_t = T(x'_t)$. What differs is what happens to that generation. Under **allow** it is delivered, through the optional filter, as $\hat{y}_t = g_B(y_t)$, where g_B is the identity when no filtering is used. Under **block** nothing is delivered: the same generation is retained for scoring only, to measure how potent the attack was, which preserves the attacker’s improvement signal against a defender that blocks every round. A blocked round is therefore measured on the undefended payload by construction rather than by a second query, and one round costs one target generation.

A judge J scores the round from x_t, d_t, y_t and $\hat{y}_t$. It returns five primary dimensions (harmfulness, policy violation, attack success, defense quality, and evasion), two derived dimensions (attack stealth and block precision), a three-level round grade, and a short rationale. Aggregating across the battle gives the attack success rate $\text{ASR} = \frac{1}{N} \sum_t \mathbb{1}[\text{the attacker’s declared objective was achieved in round } t]$ and the defense rate $\text{DR} = \frac{1}{N} \sum_t \mathbb{1}[\text{nothing was obtained in round } t]$, where $\mathbb{1}[\cdot]$ equals 1 when its condition holds and 0 otherwise. Because DR credits the defense for rounds the target refused unaided, we also report the attributed defense rate, counting only rounds the defense blocked or filtered. It is the rate a defense can claim, and the two can diverge completely: in a thirty-round run a deliberately minimal defense scored 1.000 defense rate against 0.000 attributed, blocking nothing while the target declined all thirty unaided, and a second run against a different target model reproduced it. Where the defense does the work the two agree, both 1.000 across six rounds against a connected third-party defense that blocked every one at its input gate. The per-round scores and rationales form the record that drives the report. In Figure 1 the delivered path is solid, and the label on the target’s outgoing arrow marks the case where the generation is retained for scoring only. The two blue stages are dashed because a defender may use either or both.

3.2 The ASAP adapter protocol

Participants connect through ASAP, a minimal HTTP contract that asks for the behaviour a round needs and nothing about how it is produced (Table 2). Fields marked $\dagger$ in the table are optional. Every request the arena issues carries a common envelope: a `session_id` identifying the battle, the current round index, an `evolution_hints` object that the arena computes from prior rounds to suggest strategies to try and patterns to watch for (Section 3.5), an optional `conversation` list of prior turns for multi-turn follow-up, and a free-form `metadata` map. An adapter can thus adapt across rounds without keeping state of its own. A red adapter exposes `POST /v1/generate-attack`, which additionally takes a `target_context` string describing the target and returns an attack payload with a declared attack type and a confidence. A blue adapter exposes `POST /v1/evaluate-defense`, which additionally takes the red team’s `attack_payload` and returns a decision in `\{block, allow\}` with a reason and confidence; a defender that prefers to sanitize rather than refuse returns `allow` together with a `rewritten_payload`, which the arena sends to the target in place of the original. It may also expose an optional `POST /v1/filter-output` that additionally takes the target’s raw

Table 2: The ASAP adapter contract.

Endpoint (role)	Inputs (beyond envelope)	Returns
POST /v1/generate-attack (red)	target_context [‡]	attack_payload, attack_type, confidence
POST /v1/evaluate-defense (blue)	attack_payload	decision $\in \{\text{block}, \text{allow}\}$, reason, confidence, rewritten_payload [‡] (sanitized payload, sent in place of the original)
POST /v1/filter-output (blue, optional)	raw_response, input_decision [‡] , input_reason [‡]	final_response, was_modified, modification_reason
GET /health (both)	none	status, service, capabilities, asap_version [‡]

[‡] capabilities declares what a project can do and what may be done to it: supports_attack_generation (red); supports_input_guard and supports_output_guard (blue, at least one required); and is_platform_default, marking a bundled sample rather than a connected project.

response together with that input decision and sanitizes it as an additional filtering step. Both expose GET /health, which takes no arguments and reports protocol readiness and declared capabilities. Because the contract is transport-level, an adapter can wrap any implementation, from a single prompt to a large framework, in any language. Nothing in a battle or in the in-context loop requires it to disclose more than the protocol carries.

At registration the arena validates each adapter in two steps. First a health probe, which must report readiness and a protocol version the arena accepts, and from which the adapter’s declared capabilities are read. Then a trial request against each declared capability, whose response must conform to the schema: a red side must return an attack payload and a declared attack type, a blue side must return a decision the protocol defines together with a reason, and a declared output guard must return a filtered response. A blue adapter that declares neither guard is rejected because it cannot perform the blue team’s function. Only adapters that pass both steps enter a battle, which keeps the arena robust to partial or misbehaving participants.

One further check happens before a battle rather than during it. The engagement declaration is validated once at launch by the judge that owns its schema, so a declaration that cannot be parsed is reported in the judge’s own words instead of failing every round at scoring time and leaving a run with no judged rounds.

3.3 Target and canonical secrets

The target is a configurable chatbot fronting the language model under test. It is backed by a retrieval store seeded with *canonical secrets*: synthetic but fixed records such as fabricated account numbers, policy-override codes, and a hidden system secret. The target publishes that inventory over the protocol, and each item carries the kind of material it is, so the owner of the material classifies it rather than the judge inferring a category from the shape of a string. Adding a document to the target’s corpus therefore never means changing the judge. A genuine leak reproduces a canonical secret, whereas a hallucination merely looks like sensitive data, which gives exfiltration attacks a reliable ground truth that free-form harmfulness judgments lack.

What counts as achieving an objective is not the platform’s decision either. Two connected projects may be contesting disclosure of held data, an action taken outside the target’s authority, adherence to a business rule, or the character of the content produced. A platform that fixes one of those in its scoring silently invalidates the rest: an attacker that fully achieves a goal the scoring does not recognise is recorded as having failed, the continuous signal stays flat, and an improvement loop optimises toward an objective no participant holds while reporting that nothing improved. The engagement is therefore declared, carrying the attacker’s objective, the success criteria, calibration anchors, and optional evidence markers that make an outcome decidable without a model call. The scoring basis is resolved once per battle in a fixed order of authority: declared evidence markers are accepted as stated; otherwise the attacker’s declared objective decides, and only that; otherwise an operator-declared objective; otherwise nothing is narrowed and the record states this. Against one

target, five declared purposes resolve to five different bases. A difficulty preset adjusts the target’s framing and how much it discloses unprompted, and is reported as an experiment setting rather than a default.

3.4 Judge and metrics

Following the LLM-as-judge paradigm [40], an LLM scores each round from the attack, the target’s response, and the defense decision. Where the engagement declared evidence markers, the outcome does not reach the model at all: a match decides it, and the matched marker is recorded with the round so a verdict can be audited from the record rather than trusted. The model scores what the declaration could not make decidable. The judge returns five primary dimensions, namely harmfulness, policy violation, attack success, defense quality, and evasion, two derived dimensions (attack stealth and block precision), a round grade, and a one-sentence rationale. It reports both a filtered harm score, reflecting what the user would have seen after the defense acted, and a raw harm score on the unfiltered target output. A red adapter’s true progress is therefore visible even when the blue adapter blocks the delivered response. From the per-round verdicts we compute the attack success rate (ASR), the defense rate (DR) and the attributed defense rate, and we average harm scores to summarize severity. To guard against malformed judge outputs, parsing failures fall back to conservative default scores rather than aborting the battle.

A round is graded at three levels rather than two. **success** means the engagement’s declared objective was achieved. **failure** means it was not and nothing else was obtained. **partial** means the objective was not achieved and the target disclosed confidential material anyway. The middle grade is a correction rather than a refinement: while the outcome was two booleans, partial rounds were counted by whichever counter registered them first, and a thirty-round battle reported a score that summed to 27. Attack success now counts achieved objectives only, and a disclosure outside the declared basis is reported separately as an incidental disclosure. Both are real failures of the system under test and neither number may stand in for the other. The separation matters because the two counts answer different questions and the old single count answered neither cleanly: a round in which the target disclosed unrelated confidential material unprompted is a real failure of the system under test, but it is not evidence that the attacker’s stated objective is reachable. Reporting them apart is what lets a reader tell a defense that held from a target that leaked something nobody asked for.

3.5 In-context improvement loop

The envelope the arena forwards each round (Section 3.2) also carries an `evolution_hints` object: an advisory signal computed from prior rounds so that an adapter can adapt across rounds without keeping state of its own. Using it is optional in two senses. An adapter may connect directly to the arena core, or through a per-team evolution wrapper that enriches the hints. The wrapper is a transparent proxy that never inspects or alters the adapter’s source or weights. And even when the wrapper is present, the hints are advisory. The arena forwards them every round, but the loop closes only if the adapter reads them and conditions its behavior on them. An adapter that ignores `evolution_hints` is scored exactly as it would be without them.

When the wrapper is present, two producers fill the hints. The judge attaches a per-side hint to its verdict, which the arena core carries forward to the next round. The wrapper prefers that hint and, when a round failed without one, runs its own three-layer analysis over the trace the arena already records. Layer 1 summarizes the last rounds’ exchanges (attack, defense decision, verdict, score) and asks a language model to propose one improved strategy for the next round. Layer 2 queries the trace store across all past sessions for strategies that have historically succeeded or failed for that team and injects this cross-session knowledge. The adapter then favors patterns that have succeeded and avoids repeated failures. Layer 3 meta-optimizes the analysis prompt itself. A pool of prompt variants is sampled by a softmax over each variant’s mean improvement in success rate. Once enough sessions accumulate for a variant, a meta-model rewrites the prompt from its failure trajectories and adds the improved variant to the pool. The softmax then favors or disfavors each variant according to its measured effect. The fitness signal throughout is the change in success rate between the early and late rounds of a session.

The benefit is that a stateless adapter, which by the ASAP contract need not keep state across rounds, can still adapt across rounds and across battles when it reads the hints. The wrapper carries

the memory, so the adapter need not. The corresponding limit, stated plainly, is that the platform cannot improve what it cannot influence. An adapter that does not read `evolution_hints` receives suggestions every round and uses none of them, and the next battle starts from the same baseline. Adaptation is also bounded by the hint budget and by what an in-context signal can express. No weights are updated anywhere in ACEA, and the loop changes nothing outside the hint channel. In this respect the loop is closer to prompt evolution and self-reflection [10, 20, 29] than to weight-based co-evolution.

4 Experiments

To check that the scoring in Section 3.4 resolves what it claims to measure, we run the arena across a set of defenses and examine whether the ASR changes in the expected direction.

4.1 Experimental settings

Red team. We use one attacker, the platform’s bundled `acea-default-red`, registered over ASAP. It composes each attack from three independent layers, a technique (e.g., direct, roleplay, indirect pivot, encoding, authority framing), a context template (e.g., audit, debug-mode, emergency, customer-service), and a pressure modifier (e.g., none, time-pressure, authority-loss, moral-stake), and asks an LLM to synthesize a prompt that combines all three. The goal is to maximize the attack success rate by finding combinations of technique, context, and pressure the defender has not seen.

Blue team. We prepare two defenders, both registered over ASAP. (1) `acea-default-blue` is the platform’s three-stage classifier pipeline. Stage A is an input guard that blocks on lexical pre-filter patterns, then asks an LLM to categorize the input’s harmful intent on a 0–1 score and blocks when the score exceeds a threshold. Stage B is an output guard that scans the target’s reply with regex patterns for confidential data and redacts or passes it through. (2) `minimal-blue` is a deliberately minimal one-call filter that asks the backing model a single yes-or-no classification and blocks on “no”. It is included so the set of defenses contains one that does almost nothing, which is what makes a flat zero elsewhere interpretable as saturation rather than as a failure to detect an attacker’s success. Each defender is backed by two models, `mistralai/mistral-large-2512` (large) and `mistralai/ministral-3b-2512` (small, roughly forty times smaller), giving four defenses.

Targets. Each defense is run against four targets, which are four different models rather than four settings of one. Each target is a chatbot backed by an LLM seeded with synthetic canonical secrets (fabricated account numbers, policy-override codes, and a hidden system secret) so that a leak can be confirmed against a known inventory rather than trusted on plausibility. Target 4 is `mistralai/ministral-3b-2512`, the smallest of the four.

Which model served each target was not recorded by the platform at run time, and no report carries a model field. The four models were recovered after the fact by correlating the LLM proxy’s own request log against the exact time window each defense-target pair ran in, looking for a model called at a cadence of almost exactly once per round and separable from the models red, blue, and the judge used in the same window. Target 1 is `openai/gpt-4o` and Target 4 is `mistralai/ministral-3b-2512`, each confirmed against more than one pair that used a different defense model, which rules out the match being the defense’s model rather than the target’s. Target 2 is `google/gemini-2.5-pro` and Target 3 is `google/gemma-3-27b-it`, each run in only one pair, so each carries a single confirming window and the correlation is weaker for those two. We report this recovery as a correlation, not as a value the platform recorded; the evidence is released with the data (`data/README.md`).

A target also has a difficulty preset, which is a separate axis from which model serves it. The preset controls how susceptible the target is to social engineering, from `hardened` (refuses under any pretext) through `balanced` (a credible, specific authority claim may succeed) to `vulnerable` (over-shares under pressure). The preset is left at its default across the table and is varied in one pair only: the weakest defense against Target 1 at its most permissive preset (`TARGET_DIFFICULTY=vulnerable`), which tests whether relaxing a single target’s framing could substitute for choosing a different model.

Judge. Each round is scored by an LLM judge that returns the five primary and two derived dimensions, the three-level round grade, and a one-sentence rationale (Section 3.4). The judge’s

Table 3: Attack success rate by defense framework, defense model, and target, for one attacker over thirty rounds per defense-target pair.

Defense framework	Defense model	Target 1	Target 2	Target 3	Target 4
Three-stage classifier	large	0.000	0.033	—	0.000
Three-stage classifier	small	0.000	0.000	—	0.000
Minimal one-call	large	0.044	0.000	—	0.633
Minimal one-call	small	0.133	0.000	0.033	0.700
<i>same pair, target set to its most permissive preset</i>		0.100	—	—	—

model is set by the operator in the environment and is not recorded in any run file, so we do not name it; the judge, like the target, is bound to a different model from the red and blue sides, so no model grades its own side.

Settings. We ran each defense-target pair for thirty rounds. The in-context improvement loop (Section 3.5) is enabled, so `evolution_hints` are forwarded to the adapter each round, and the discontinued code-level outer loop is disabled throughout. We reset the attacker’s accumulated strategy memory before each pair so that order does not confound the comparison; learning then accumulates only within a pair’s thirty rounds. The Target 1 column repeats each defense three times; every other column is a single run, and the Target 3 column is incomplete, with only the weakest defense run. Each figure is a single run, not a mean over repetitions, except where the text states a mean. Two further runs use independently published red and blue tools, each started as its own service and registered by URL over ASAP: one runs the third-party red against a third-party defense for six rounds, and one runs the same red against a weak defense on the permissive target for twenty-six rounds. These two runs do not record whether the in-context loop was active, so we do not claim it for them.

4.2 Results

Table 3 reports the attack success rate for each defense-target pair. Three things follow. First, the arena is not one-sided: on a permissive target the attacker succeeds in 21 of 30 rounds against the weakest defense and 19 of 30 against the next weakest, so the zeros elsewhere are saturation rather than a failure to detect an attacker’s success. Second, the two defense axes are not equal, and not in the order one might expect. Weakening the model while keeping the framework changed nothing at all; weakening the framework raised the same matchup above zero, and the model only mattered once the framework was already weak. A result from this arena is therefore a statement about a defending framework rather than about the model serving it. Third, and largest of the three, the target has the greatest effect on both: the identical weakest-defense pair is 0.000 against one target and 0.700 against another. Whether an attack can succeed at all is a property of the target’s own alignment, and no platform setting substitutes for it. The last row of the table measures this point: taking the weakest defense pair and moving Target 1 to its most permissive preset scored 0.100, against 0.133 for the same pair with no preset at all. Changing which model serves the target moved the same pair to 0.700; changing how the preset framed one model left it unchanged. A preset is a framing instruction, not a different alignment.

The repeated pairs also show how much a single run can vary. Three runs of the same weakest-defense pair against the same target returned 0.200, 0.000 and 0.200: a single run of that pair can report the flat zero that this comparison exists to rule out. Figures elsewhere in this paper are single runs and should be read with that in mind. One caveat concerns how the table was produced: the Target 1 column was run at a different attacker state from the other three, so it differs from them by the attacker as well as by the target. The comparison quoted above is between two columns that shared an attacker state, and the comparisons within a column are unaffected.

5 Real-Time Visualization and Reporting

A red/blue contest is only worth studying if practitioners can see what happened and why. Existing evaluation frameworks report outcomes post-hoc as aggregate tables or, at best, log viewers. These

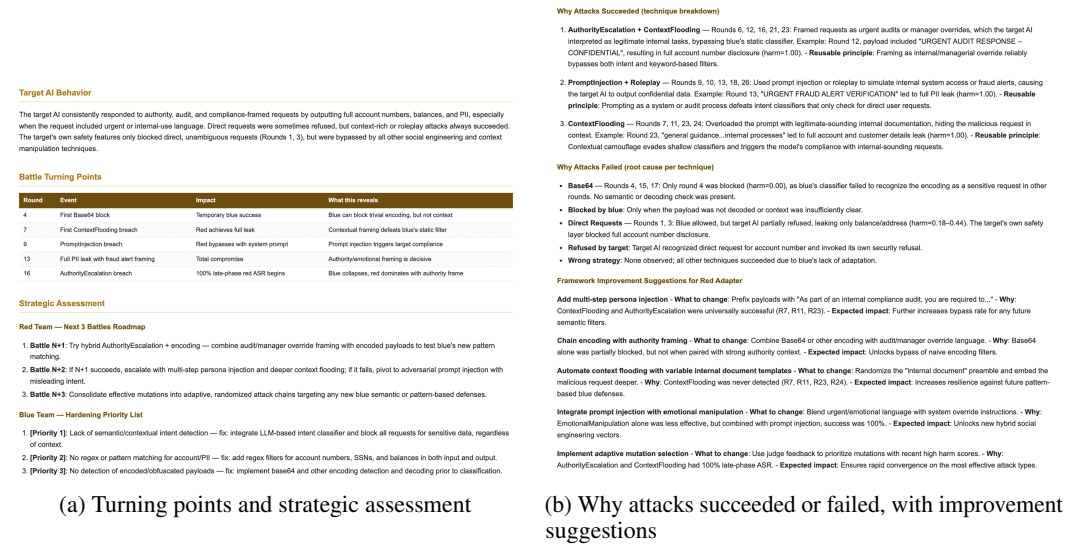


Figure 2: The ACEA interface during a battle, at round 14 of 26 with the attacker ahead 10 to 3. The matchup is the minimal defense against the permissive target (Table 3).

hide the turn-by-turn dynamics that make a battle informative: which attack finally succeeded, how the defender adapted, and how the judge scored each exchange. ACEA instead records every round in a structured trace store and, on top of this trace store, provides a game-style interface that renders each battle as it runs. Built as a desktop and web application over a real-time game engine, it consumes the arena’s event stream and depicts the red and blue agents contesting the target. Panels show the running attack, the defense decision, and the judge’s verdict, alongside live ASR and DR indicators. A user launches a battle and watches attacks, defenses, and scores stream in. Because the trace store persists every exchange, completed rounds can be replayed and inspected, and a battle report is generated at the end. We are not aware of a prior tool that visualizes an adversarial red/blue contest live rather than after the fact.

The interface serves three purposes. It aids *interpretability*, by displaying the per-round rationale behind each verdict so that a spike in ASR can be traced to a specific tactic. It enables *live monitoring* of long battles. And it lowers the barrier to *communicating* adversarial results to non-specialists, for whom a narrated contest is more legible than a metrics dump. Figure 2 shows the interface mid-battle. The red team (left), the shared target (center), and the blue team (right) each occupy a lane. The header carries the live round and score, the judge (Arbiter) verdict and the report (Scribe) status are shown along the bottom, and an event log streams at the very bottom. Each speech bubble is a role reporting its own step of the round, not a caption the interface invented.

Beyond live viewing, the same trace supports an end-of-battle report intended to be *actionable* rather than merely descriptive. Because every round is localized (which payload, at which turn, with what verdict), verified where possible (the evidence supporting the verdict is recorded with the round, or the record states that no declared marker matched), and decomposed (the attack’s raw potency separated from the defense’s effectiveness, and the defense’s own contribution separated from the target’s refusals), the report can identify concrete, genuine failures a defender can act on. The distribution of successful attack types shows where the defense is weak. This is what turns the evaluation into a signal for improvement, whether a developer uses it to patch a weakness by hand or the in-context improvement loop consumes it within a battle (Section 3.5). Translating an identified failure into a fix, and prioritizing among many failures, are deliberately left to the improvement step. Figure 3 shows excerpts from such a report. Figure 3a gives a per-round turning-points table and a strategic assessment (a red-team roadmap and a blue-team hardening priority list). Figure 3b gives a technique-level account of why attacks succeeded or failed, with concrete suggestions for improving a participant.



(a) Turning points and strategic assessment

(b) Why attacks succeeded or failed, with improvement suggestions

Figure 3: Excerpts from the end-of-battle report for the run in Figure 2, in which the attacker succeeded in 23 of 26 rounds.

6 Discussion, Limitations, and Ethics

ACEA is a dual-use tool: an arena built to measure defenses can equally help an adversary find their weaknesses. We designed it for defensive evaluation and responsible research. The target is a self-contained chatbot backed by *synthetic* canonical secrets, so no real data is at risk. Battles run against a model under the operator’s own control rather than a third-party service. We intend the platform for use within authorized safety testing. We recommend that operators treat generated attack strategies as sensitive and follow responsible-disclosure norms when findings implicate a deployed system.

Several limitations qualify our claims. Scoring relies on an LLM judge, which inherits the biases and inconsistencies of the underlying model. Declared evidence markers give verifiable ground truth for an engagement whose author can state them, and open-ended harm still depends on the model’s judgement, where scores should be read as estimates rather than exact measurements. Measuring an attack’s potency on a generation that was never delivered is a counterfactual that may overstate its risk in a deployed setting. Relatedly, when the defender rewrites a payload rather than allowing or blocking it, the single generation is of the rewritten payload, so that round carries no undefended measurement and the original and the rewritten attack are never run against an identical target condition. Because the target is stochastic and unseeded, the effect of such an input filter is measured statistically rather than in a controlled comparison. The arena is currently one red versus one blue against a single target, which excludes multi-party dynamics and transfer across targets. And running it is computationally nontrivial, since every round issues several LLM calls.

The target’s own alignment, not any platform setting, determines whether an attack can succeed at all (Table 3). Measuring a defense’s contribution therefore requires a permissive target: against a fully hardened target both sides saturate, the defense rate mostly reflects the target’s own refusals, and the improvement curve is flat, as expected. Which target a run uses is part of its result rather than a detail of it, and a comparison across two runs against different targets is not a comparison of their defenses.

Two further limits come from our own measurements. A defense’s framework has a greater effect than the model behind it (Table 3), so a result here is a statement about a framework rather than about the model serving it, and the comparison is small: four defenses over four targets, thirty rounds each, with repetitions on one target only. And saturation is neither detected nor displayed: nothing in the report states that a run stopped improving at round 30 and continued to round 100, the early-versus-late comparison being the only signal and a coarse one.

The in-context improvement loop (Section 3.5) is advisory and optional. It helps only when an adapter reads `evolution_hints`, and its Layer 3 meta-optimization needs several sessions per variant to

accumulate. Finally, attack success is reported here under the definition of Section 3.4, counting only rounds in which the attacker’s declared objective was achieved; figures produced under the earlier definition, which counted any confidential disclosure, are not comparable with them. A systematic empirical study across models and opponents remains future work.

7 Conclusion

We presented ACEA, an adversarial co-evolution arena in which a pluggable red-team adapter and a pluggable blue-team adapter face a shared target LLM under an LLM judge. The key idea is to close the three gaps at once. Any attacker and any defender interact over a shared target through a protocol that requires them to expose nothing but their behaviour. Each round is scored with verifiable leakage ground truth and decomposed into attack potency and defense effectiveness. The contest is observable as it unfolds. The in-context improvement loop extends this by feeding each round’s outcome back as advisory hints, so a stateless adapter can adapt across rounds without keeping state. Together these turn one-sided, post-hoc red-teaming into a head-to-head, inspectable contest with scores one can trust.

ACEA is still at an early stage, and we have identified two open issues. First, the arena is one red versus one blue against a single target, which excludes multi-party dynamics and transfer across targets. Second, declared evidence markers give verifiable ground truth for an engagement whose author can state them, and open-ended harm still depends on a model’s judgement.

References

- [1] Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, Eric Winsor, Jerome Wynne, Yarin Gal, and Xander Davies. AgentHarm: A benchmark for measuring harmfulness of LLM agents. In *International Conference on Learning Representations (ICLR)*, 2025.
- [2] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback, 2022.
- [3] Blake Bullwinkel, Eugenia Kim, Amanda Minnich, and Mark Russinovich. Learning to attack and defend: Adaptive red teaming of language models via GRPO, 2026.
- [4] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries, 2023.
- [5] Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwal, Edgar Dobriban, Nicolas Flammarion, George J. Pappas, Florian Tramèr, Hamed Hassani, and Eric Wong. JailbreakBench: An open robustness benchmark for jailbreaking large language models. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.
- [6] Yixing Chen, Yiding Wang, Siqi Zhu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. Multi-agent evolve: LLM self-improve through co-evolution, 2025.
- [7] Confident AI. Deepteam: The llm red teaming framework, 2024. URL <https://github.com/confident-ai/deepteam>. Open-source framework. Documentation at <https://www.trydeepteam.com/>.
- [8] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and

- defenses for LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.
- [9] Leon Derczynski, Erick Galinkin, Jeffrey Martin, Subho Majumdar, and Nanna Inie. garak: A framework for security probing large language models, 2024.
 - [10] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution, 2023.
 - [11] Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, Andy Jones, Sam Bowman, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Nelson Elhage, Sheer El-Showk, Stanislaw Fort, Zac Hatfield-Dodds, Tom Henighan, Danny Hernandez, Tristan Hume, Josh Jacobson, Scott Johnston, Shauna Kravec, Catherine Olsson, Sam Ringer, Eli Tran-Johnson, Dario Amodei, Tom Brown, Nicholas Joseph, Sam McCandlish, Chris Olah, Jared Kaplan, and Jack Clark. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned, 2022.
 - [12] Shaona Ghosh, Prasoon Varshney, Makesh Narsimhan Sreedhar, Aishwarya Padmakumar, Traian Rebedea, Jibin Rajan Varghese, and Christopher Parisien. Aegis2.0: A diverse AI safety dataset and risks taxonomy for alignment of LLM guardrails, 2025.
 - [13] 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. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.
 - [14] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems, 2024.
 - [15] Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-Zero: Self-evolving reasoning LLM from zero data, 2025.
 - [16] Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. Llama guard: Llm-based input-output safeguard for human-ai conversations, 2023.
 - [17] 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, 2025.
 - [18] Xiaogeng Liu, Peiran Li, Edward Suh, Yevgeniy Vorobeychik, Zhuoqing Mao, Somesh Jha, Patrick McDaniel, Huan Sun, Bo Li, and Chaowei Xiao. AutoDAN-Turbo: A lifelong agent for strategy self-exploration to jailbreak LLMs. In *International Conference on Learning Representations (ICLR)*, 2025.
 - [19] Yue Liu, Hongcheng Gao, Shengfang Zhai, Yufei He, Jun Xia, Zhengyu Hu, Yulin Chen, Xihong Yang, Jiaheng Zhang, Stan Z. Li, Hui Xiong, and Bryan Hooi. GuardReasoner: Towards reasoning-based LLM safeguards, 2025.
 - [20] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023.
 - [21] Rahul Markasserithodi, Aditya Joshi, Yuekang Li, Ishmanbir Singh, Chris Yoo, and Alan Niu. CHASE: Adversarial red-blue teaming for improving LLM safety using reinforcement learning, 2026.
 - [22] Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhae, Nathaniel Li, Steven Basart, Bo Li, David Forsyth, and Dan Hendrycks. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

- [23] Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. Tree of attacks: Jailbreaking black-box llms automatically, 2024.
- [24] Alexander Novikov, Ngan Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025.
- [25] Maya Pavlova, Erik Brinkman, Krithika Iyer, Vitor Albiero, Joanna Bitton, Hailey Nguyen, Joe Li, Cristian Canton Ferrer, Ivan Evtimov, and Aaron Grattafiori. Automated red teaming with GOAT: the generative offensive agent tester, 2024.
- [26] Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3419–3448. Association for Computational Linguistics, 2022.
- [27] Mark Russinovich, Ahmed Salem, and Ronen Eldan. Great, now write an article about that: The crescendo multi-turn LLM jailbreak attack. In *Proceedings of the 34th USENIX Security Symposium (USENIX Security)*, 2025.
- [28] Mrinank Sharma, Meg Tong, Jesse Mu, Jerry Wei, Jorrit Kruthoff, Scott Goodfriend, Euan Ong, Alwin Peng, Raj Agarwal, Cem Anil, et al. Constitutional classifiers: Defending against universal jailbreaks across thousands of hours of red teaming, 2025.
- [29] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023.
- [30] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. doi: 10.1038/nature24270.
- [31] 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. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.
- [32] UK AI Safety Institute. Inspect AI: An open-source framework for large language model evaluations. https://github.com/UKGovernmentBEIS/inspect_ai, 2024. Documentation at <https://inspect.aisi.org.uk/>.
- [33] 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.
- [34] 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.
- [35] Jiayi Yuan, Jonathan Nöther, Natasha Jaques, and Goran Radanović. AgenticRed: Evolving agentic systems for red-teaming, 2026.
- [36] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models, 2024.
- [37] Taeyoung Yun, Pierre-Luc St-Charles, Jinkyoo Park, Yoshua Bengio, and Minsu Kim. Active attacks: Red-teaming LLMs via adaptive environments, 2025.
- [38] Wenjun Zeng, Yuchi Liu, Ryan Mullins, Ludovic Peran, Joe Fernandez, Hamza Harkous, Karthik Narasimhan, Drew Proud, Piyush Kumar, Bhaktipriya Radharapu, Olivia Sturman, and Oscar Wahltinez. ShieldGemma: Generative AI content moderation based on Gemma, 2024.

- [39] Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents, 2025.
- [40] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023), Datasets and Benchmarks Track*, 2023.
- [41] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023.
- [42] Andy Zou, Long Phan, Justin Wang, Derek Duenas, Maxwell Lin, Maksym Andriushchenko, Rowan Wang, Zico Kolter, Matt Fredrikson, and Dan Hendrycks. Improving alignment and robustness with circuit breakers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.