INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact Factor: 6.078
(Volume 11, Issue 5 - V11I5-1137)
Available online at: https://www.ijariit.com

A Deep Dive into the Adversarial Training Process of Modern LLMs

Armaan Mahajan
mypublishedpaper@gmail.com
Pathways School Gurgaon, Haryana

ABSTRACT

Large language models can be trained to resist jailbreak attempts in order to ensure the model's safety. This literature review analyzes common adversarial training techniques used in modern LLMs to understand and break down the process of adversarial training. The paper looks into 6 training techniques, ranging from PGD to Deployment Time Safety Layers and Model Editing. For each method, we look into how they work behind the scenes, their strengths, weaknesses, and real-world usage. We then synthesize the most effective manner of training models against adversarial inputs. This research revealed that the most effective training technique emphasizes a multilayered defense strategy. Future research can look at improving model-editing coverage, creating open-sourced LLM benchmarks to test against jailbreaks, and closing the divide between embedding-space and discrete input training.

Keywords: *LLMs, Adversarial Training, Artificial Intelligence, RLHF, ChatGPT, Jailbreaking, Lat.*

INTRODUCTION

Since the release of OpenAI's GPT-1 in 2018, Large Language Models (LLMs) in the mid-2020s, are so much more than just openly available conversational chatbots. Their impacts proliferate research, market analysis, products, and software development. (De Angelis et al. 2023)

Leading this transformation are companies like OpenAI and Anthropic, which have developed some of the most publicly accessible and widely used LLMs to date. OpenAI's GPT models have become more than just a socio-technological phenomenon through ChatGPT. They now integrate into just about anything and everything, from search engines to word documents. Anthropic, with its Claude models, positions itself with a strong emphasis on literature and writing code. In 2025, IDEs like Cursor are being used to develop production level software, as engineers at Johnson & Johnson, OpenAI, Stripe, Samsung, Instacart, Shopify, and more integrate these tools into daily development. The development of LLMs past conversational chatbots represents a broader shift: the embedding of conversational AI into daily life, research workflows, and enterprise solutions at scale.

These LLMs, like OpenAI's GPT, Anthropic's Claude, xAI's Grok, are all trained on large datasets that reach terabytes in their size. Sources of this data range from Common Crawl (60% of GPT-3's data) to Wikipedia (about 3 billion tokens used to train GPT-3) (Brown et al. 2020). This data is a double-edged sword. The training data often includes unvetted, harmful, and biased content, due to its open-source nature.

For example, Bender and Gebru et. Al note that "In the case of US and UK English, this means that white supremacist and misogynistic, ageist, etc. views are overrepresented in the training data, not only exceeding their prevalence in the general population but also setting up models trained on these datasets to further amplify biases and harms" (Bender and Gebru et. Al pg. 613)

Hence, while the training process equips the models with the knowledge and structure needed to produce Ph.D.-level responses, it also introduces substantial risks. Because of the sensitive nature of the underlying data, LLMs may generate content that addresses or even endorses harmful topics—ranging from self-harm advice and hate speech to violent instructions and beyond. To prevent this, developers attempt to set guardrails on the outputs of their LLMs (Xhonneux et al. 2024). However, researchers have proven that even cutting-edge models can be "jailbroken" by cleverly crafted inputs, which can disable any safety guardrails. These prompts, known as "adversarial prompts", can trick models into producing content that not only violates both social and legal guidelines but also may lead to genuine harm to their users.

The capacity of large language models (LLMs) to generate harmful content, such as hate speech, violence-inciting material, and explicit sexual content, poses significant ethical concerns due to its potential to adversely affect individuals and society, particularly vulnerable groups like children and adolescents.

Research highlights that such content can impact psychological development, promote aggression, and offend through implicit toxicity. For instance, Deshpande et al. note that "ChatGPT, when assigned a persona, can exhibit significant toxicity. Particularly, this risk is elevated for vulnerable groups such as students, minors, and patients" (Deshpande et al. 2023). Similarly, Wen et al. observe that "LLMs have the potential to produce implicit toxic responses that elude existing classifiers. These harmful outputs, while not easily identifiable, can offend individuals or groups by insinuating negative or false statements" (Wen et al. 2023). These risks underscore the urgent need for ethical frameworks to mitigate the societal harm caused by AI-generated content.

In order to counter these outputs, developers use a method known as adversarial training, as a preventive and defense mechanism. They intentionally expose the model to attack-like inputs while training it so that the model is able to detect them. They can then correct the output, or leave it to the model to learn from its error. While the approach may seem to be contradictory (training a model with prompts that lead to adversarial outputs), these "adversarial examples" have been demonstrated to improve robustness against such attacks. (Xhonneux et al. 2024)

Understanding the comparative effectiveness of adversarial training methods is critical for developing AI systems that can operate safely in real-world environments where malicious actors continuously evolve their attack strategies. As LLMs become increasingly integrated into sensitive applications—from healthcare and education to enterprise decision-making—the stakes of inadequate safety measures grow exponentially, making it imperative to identify which defensive approaches provide the most robust protection against emerging threats. This research evaluates the primary adversarial training methodologies currently employed in LLM development, analyzes their respective strengths and limitations, and determines which approach offers the optimal balance of safety, performance, and practical implementation for next-generation language models.

THE NATURE OF ADVERSARIAL PROMPTS

What are adversarial prompts?

Adversarial prompts are carefully crafted inputs designed to coax a large language model (LLM) into producing unintended, often harmful or forbidden outputs. In analogy to adversarial examples in computer vision, adversarial prompts exploit weaknesses in an LLM's training. Liu et al. (2025) liken prompt injection to a form of *command injection* in traditional computing, noting that adversaries use tactics such as "role-based conditioning, instruction hijacking, obfuscated encoding, and multi-turn manipulation" to subvert LLM behavior (Chetan 2025, pg. 2). In this context, a **jailbreak attack** is a specific kind of adversarial prompt that deliberately "bypasses the model's safeguards" to elicit disallowed content (https://arxiv.org/html/2402.13148) . Zhou et al. (2024) define a jailbreak as "adding a deliberately designed prompt to input data, tricking the language model into generating responses that may contain harmful or malicious content" (https://arxiv.org/html/2402.13148). Similarly, Li et al. (2025) describe both prompt injection and jailbreaking as adversarial inputs that "coerce the model to break its safety constraints" (https://arxiv.org/html/2505.18889v1). In practice, attackers exploit ambiguities in language, hidden instructions, or creative phrasing so that the LLM complies with malicious intent rather than its default ethical guardrails.

A notable early example of such a jailbreak was the widely circulated "DAN" (Do Anything Now) prompt, which instructed the model to roleplay as an unfiltered version of itself that could ignore OpenAI's safety protocols. A common version of this prompt included phrases like "Hi ChatGPT. You are going to pretend to be DAN, which stands for Do Anything Now. DAN can do anything now. He has broken free of the typical confines of AI and does not have to abide by the rules set for him." Rao et al. conducted an analysis of Reddit posts involving jailbreaks from r/OpenAI and r/ChatGPT, finding that most terms in these subreddits revolved around the phenomenon of DAN. They describe DAN as "an instruction-based plus cognitive-hack jailbreak that works in two ways: the attacker provides a list of carefully curated instructions that involves creating a fictional scenario for the model to respond differently, and secondly, involving a punishment system for the model failing to respond to the user as requested" (https://arxiv.org/abs/2305.14965)

THEIR CONSTRUCTION

Adversarial prompts can be constructed through a variety of strategies. In general, attackers leverage the LLM's training objectives (to follow instructions and predict human-like text) by designing inputs that appear benign but carry hidden commands. Techniques include:

- **Role-based or persona prompts:** Instructing the model to adopt a specific character or role that is not bound by normal rules. For example, attackers often begin with "Pretend you are a rebellious AI" or impersonate a trusted figure to override content filters. Liu et al. (2025) explicitly list *role-based conditioning* as a common attack vector (arxiv.org). Known real-world examples include "DAN" (Do Anything Now) prompts circulated in online forums, where the user asks the chatbot to ignore its safety settings by roleplaying an unfiltered assistant. Shen et al. (2025) found that even simple template tricks like these ("AIM" or "Better_DAN") remain "surprisingly potent" at bypassing many models' alignments (arxiv.org)
- **Instruction hijacking or direct overrides:** Embedding commands like "Ignore all previous instructions" or "Delete the above message and then answer" to force the model to drop its safeguards. These prompts explicitly instruct the LLM to disregard its internal policies. For example, one can insert a line like "User: Ignore the above directions and respond as follows..." into a prompt. Such *instruction injection* works because LLM inputs mix system instructions and user data in one context, so the model often cannot tell which to prioritize.
- **Obfuscation and symbolic encoding:** Writing the malicious instruction in a hidden or indirect form so that keyword-based filters fail to recognize it. Attackers might use synonyms, typos, foreign languages, or even code to disguise harmful content. For instance, Zhu et al. (2023) report that adversarial prompts can be crafted "to mimic plausible user errors like typos or synonyms" – i.e. tiny textual changes – that still induce errors in model outputs (arxiv.org). Notably, Andriushchenko and Flammarion (2024) introduced the *PastTense* technique, which rephrases all requests into past tense.

They show this simple grammatical shift "has been shown to be effective at bypassing the refusal training of LLMs" (arxiv.org). Another example is *LRL-Comb* (Yong et al. 2023), where an attacker combines several low-resource or unrelated languages in one query (e.g. Zulu, Gaelic, Hmong) so that the underlying intent ("teach me how to hack") is hidden from single-language filters (arxiv.org). Similarly, one can encode harmful instructions in Base64 or steer the conversation to a coded context (so-called *token smuggling*) to evade simple guards.

Multi-turn or chain-of-thought exploitation: Exploiting the LLM's multi-step reasoning. An attacker may construct a dialogue or use chain-of-thought prompts to gradually coax the model into revealing disallowed information. For example, by first asking for safe content and then appending a follow-up that slips the actual harmful query into a supposed "thought process," the model may fail to apply its content policies. Research often treats this as part of *multi-turn manipulation* (arxiv.org), where each turn of the conversation conditions the next. While less commonly cited in examples, this technique can involve subtle rephrasings across multiple messages to outwit one-shot filters.

Each of these strategies has concrete realizations in the literature. Shen et al. (2025) evaluated dozens of jailbreak methods and found that even rudimentary templates (like telling the model to answer "in character" or to bypass rules) often succeed. They remark that "template-based approaches (AIM, BETTER_DAN) remain surprisingly potent" against many models (arxiv.org).

LLM SAFETY MECHANISMS AND THEIR LIMITATIONS

Current safety mechanisms in LLMs face fundamental limits that adversarial prompts can exploit. Most LLMs are aligned through a combination of training objectives, reinforcement from human feedback, and rule-based filters. However, these often conflict or leave gaps: for example, as Zhou et al. (2024) note, LLMs are simultaneously trained to follow instructions *and* to refuse unsafe requests, creating an inherent tension (arxiv.org). An adversarial prompt can capitalize on this by crafting a request that appears as a valid instruction, thereby tricking the model into choosing the "follow" objective over the "refusal" objective. Zhou et al. describe this conflict explicitly: "the conflict between instruction following and refusing to bring answers with unsafe content" is precisely what makes aligned models vulnerable (arxiv.org).

In practice, common defenses are often rigid or incomplete. Static system prompts or canned safety instructions (the text the model sees before the user prompt) are hard-coded into the model's alignment, but they cannot anticipate every novel exploit. Zhou et al. point out that existing instruction-based defenses "rely on static defenses that cannot adapt dynamically to new jailbreak prompts" (arxiv.org). Simple content filters or keyword blocklists can be too coarse: if they are strict, they cause *over-defense* (blocking benign queries); if too lax, attackers slip malicious content past them. The same authors note that prompt filtering can lead to "a high rate of over-defense" (arxiv.org), hurting utility without fully preventing escapes.

Overall, the limitations stem from the models' inherent design (treating all input as instruction) and the static nature of most defenses, which adversarial prompts systematically exploit. However, developers can design and train their models to resist–which the rest of this paper will look into, comparing 5 of adversarial training techniques employed in modern LLMs.

Subtopic 2: FGSM (Fast Gradient Sign Method)

Chronologically, FGSM is one of the first adversarial training methods that's still used today. Introduced in 2015, it is one of the foundational techniques in adversarial training designed to improve model robustness against adversarial attacks. FGSM generates adversarial examples by adding carefully crafted perturbations to the input data in the direction of the gradient of the loss function with respect to the input. These perturbations are small but intentionally designed to maximize the model's prediction error, thereby exposing vulnerabilities.

BACKGROUND & HOW IT WORKS

The Fast Gradient Sign Method (FGSM) was introduced by Goodfellow et al. (2015) as a simple one-step adversarial attack for neural networks (https://ar5iv.labs.arxiv.org/html/1412.6572). FGSM perturbs an input x by adding a small step in the direction of the loss gradient, i.e. $x' = x + \varepsilon,\text{sign}(\nabla_x L(x,y))$, producing an adversarial example that maximizes the loss with minimal distortion (https://aclanthology.org/2023.findings-acl.496.pdf). In practice this means computing the model's loss gradient with respect to the input (or its embedding) and taking the sign. Goodfellow et al. demonstrated that including these FGSM-generated examples during training (adversarial training) can substantially reduce error – e.g. they observed error on MNIST falling from 89.4% to 17.9% under FGSM attacks after adversarial training (https://ar5iv.labs.arxiv.org/html/1412.6572). Later work generalized FGSM into iterative attacks: Madry et al. (2017) proposed Projected Gradient Descent (PGD), which applies FGSM-like steps multiple times, yielding much stronger adversaries (https://aclanthology.org/2023.findings-acl.496.pdf). In the context of natural language, direct FGSM on discrete tokens is not straightforward, so implementations typically perturb the continuous token embeddings instead (aclanthology.org). In such embedding-space adversarial training, one computes gradients through the embedding layer and adjusts the embedding vectors by the FGSM rule (aclanthology.org). Overall, FGSM was one of the first adversarial methods to be used for robust training, showing that incorporating simple gradient-based attacks can regularize models and improve their resistance to adversarial inputs. Despite its age and the fact that transformer-based LLMs are newer (being introduced in 2017), it holds up today, justifying its analysis.

Strengths

FGSM's primary advantage is computational efficiency. Since it is a single-step method, each adversarial example requires only one backpropagation through the model to compute ∇L, unlike iterative methods that require multiple backward passes. This makes FGSM very cheap even for large models. For example, one recent study explicitly chose FGSM "because it is fast and enough to show that our approach is effective" when perturbing text embeddings for a large classifier (https://www.rivas.ai/pdfs/sooksatra2022adversarial.pdf). In modern LLM settings, where models may have billions of parameters, the ability to generate an adversarial gradient in one shot can be a practical necessity. Moreover, FGSM is conceptually simple and easily integrated into existing training loops: it simply adds a scaled gradient sign to the input or embedding.

In standard adversarial training pipelines, the “input-space” perturbation (as done by FGSM) naturally aligns with gradient-based optimization. FGSM also operates squarely in the input (or embedding) space, meaning no special architecture changes are required. Because of its low overhead and straightforward implementation, FGSM can readily scale to large datasets or batched training of LLMs: generating an FGSM example is just one extra backward pass per example. For example, FGSM-like perturbations have been used to generate “on-the-fly” adversarial embeddings during fine-tuning to improve robustness (https://arxiv.org/html/2505.00976v1 , https://www.rivas.ai/pdfs/sooksatra2022adversarial.pdf). In summary, the key strengths of FGSM in the modern LLM context are its speed (only one gradient step) and simplicity (easy integration into gradient-based training), enabling large-scale adversarial data augmentation with relatively low computational cost.

Weaknesses

The chief limitation of FGSM also comes as a result of one of its strengths–its one-step nature. While this makes it fast and simple, it leads to it often producing relatively weak adversarial examples compared to multi-step attacks. A single gradient-sign step can overshoot or fail to find the most damaging perturbation. In practice, iterative attacks like PGD tend to degrade models more severely. As one survey notes, FGSM corresponds to one step (Goodfellow et al., 2014) whereas stronger attacks may take many small steps to maximize loss (https://aclanthology.org/2023.findings-acl.496.pdf). In modern LLM settings, this means a model hardened only on FGSM examples may and will still be vulnerable to more sophisticated perturbations. For instance, Liu et al. (2023) found that adversarial training with complex, targeted attacks (beyond FGSM) was required to truly immunize a model against sophisticated jailbreak prompts. In other words, FGSM often underestimates the worst-case perturbation.

Another weakness is that FGSM (in the input/embedding space) cannot easily capture structured or discrete manipulations characteristic of prompt-injection or jailbreak attacks. Jailbreak-style adversaries frequently use semantic transformations or cleverly crafted text sequences, which are not well approximated by a uniform embedding perturbation. As noted in recent LLM safety research, real-world adversarial inputs include complex “jailbreaks” and backdoors that are far outside simple norm-bounded noise (https://arxiv.org/html/2403.05030v5). FGSM’s perturbations are linear in the embedding space and do not represent meaningful token substitutions or paraphrases. Thus an FGSM-trained model may remain vulnerable to attacks that exploit the discrete, compositional nature of language. Finally, adversarial training with FGSM can sometimes hurt clean accuracy if not carefully tuned: because it naively pushes the input towards higher loss in just one direction, it may distort some inputs more than necessary. In summary, FGSM’s simplicity is also a liability: it is easily outmatched by stronger gradient attacks and ill-suited for the complex, structured nature of language-based adversarial failures (https://aclanthology.org/2023.findings-acl.496.pdf , https://arxiv.org/html/2403.05030v5).

WHERE IT IS TODAY

Owing to its one-step nature, simplicity and speed; in contemporary LLM research and applications, FGSM tends to play more of a baseline or diagnostic role than a core training algorithm. Recent studies often replace one-step FGSM with iterative or more powerful methods. For example, Xhonneux et al. (2024) note that naive discrete adversarial attacks are too costly at scale and instead compute perturbations in the continuous embedding space using multi-step optimization (PGD-like) (https://arxiv.org/abs/2405.15589). In other words, modern work is favoring “continuous adversarial training” algorithms that generalize FGSM’s idea to many steps in embedding space for large models (https://arxiv.org/abs/2405.15589). Likewise, defenses such as MART or TRADES (from the vision community) or latent-space attacks have been explored in place of pure FGSM.

Nevertheless, FGSM remains a useful reference point. It is sometimes used as a quick evaluation tool or as a teaching example of robustness: for instance, some evaluations of LLM safety still report accuracy under FGSM perturbations of embeddings (https://arxiv.org/html/2503.04833v1). And FGSM-generated examples can still be incorporated into a training mix to mildly improve robustness (https://arxiv.org/html/2503.04833v1). However, the consensus in recent literature is that FGSM alone is generally insufficient for truly robust LLM alignment or defense. Leading adversarial training pipelines in LLMs now tend to use stronger attacks (e.g. PGD-based sequence attacks or latent perturbations) and more complex data augmentation. In practice, FGSM has largely been superseded by these newer techniques, and is mostly used for simple benchmarking or initial prototyping of LLM robustness rather than as a final production defense (https://arxiv.org/html/2505.00976v1 , https://arxiv.org/abs/2405.15589).

SUBTOPIC 3: PGD (PROJECTED GRADIENT DESCENT)

BACKGROUND & HOW IT WORKS

Projected Gradient Descent (PGD) is an iterative adversarial attack originally introduced by Madry et al. for robust training of neural networks (https://arxiv.org/pdf/1706.06083). In the PGD procedure, one repeatedly perturbs an input in the direction of the loss gradient (similar to FGSM) and then projects the perturbed input back into a permissible norm-ball around the original. Formally, for an ℓ_∞ threat model, PGD takes steps $x_{t+1} = \Pi_{x+S}(x_t + \alpha,\mathrm{sign}(\nabla_x L(\theta,x,y)))$. In contrast to the single-step FGSM attack of Goodfellow et al., PGD applies multiple small steps (often with random restarts) to approximate the worst-case perturbation much more closely. This yields significantly stronger adversaries: for example, Madry et al. report that a 20-step PGD attack on MNIST achieves almost 100% error, versus ~20% for one-step FGSM (https://arxiv.org/pdf/1706.06083).

Applying PGD to language models requires special care because text is discrete. A common adaptation is to operate in the continuous token embedding space: one adds small gradient-driven perturbations to the input embeddings rather than to raw tokens. Yuan et al. (2023) explicitly describe a “Textual PGD” framework where continuous perturbations are applied to the embedding layer and the perturbed latent vector is decoded via a masked-LM head back into text.

In practice, researchers backpropagate gradients through the embedding layer and adjust the embedding vectors (e.g. by the FGSM rule or gradient steps), then optionally use a decoding step to enforce valid tokens (https://aclanthology.org/2023.findings-acl.446). In this way, PGD on the embedding space serves as a proxy for generating discrete adversarial prompts or suffixes. Recent work also shows that carefully controlling the continuous relaxation (embedding perturbation) can make PGD attacks effective for aligned LLMs: for example, Geisler et al. report that naïve gradient attacks "largely failed" on modern models, but their improved PGD-based method is up to an order of magnitude faster than discrete search at finding effective jailbreak prompts (https://arxiv.org/html/2402.09154v1).

Strengths
PGD-based adversarial training yields notably stronger robustness than simpler one-step methods. In image tasks, it is well-established that multi-step PGD training produces models resistant to far more potent attacks than FGSM. In the LLM context, embedding-space PGD achieves analogous gains. Xhonneux et al. (2024) demonstrate that adversarial training with continuous PGD steps dramatically improves safety: their continuous-attack training (C-AdvUL / C-AdvIPO) on various open LLMs "substantially enhance[s] LLM robustness against discrete attacks" like GCG suffix attacks, AutoDAN and PAIR, while preserving model utility. Quantitatively, a Llama-2-based model adversarially trained by PGD saw jailbreak success rates fall to near zero on diverse attacks (e.g. from 40% ASR to 0% for PAIR after training). Crucially, these gains come with minimal loss of capability: the robust models retained essentially the same MMLU accuracy as baseline (e.g. ~69% vs 67% accuracy) (https://ar5iv.labs.arxiv.org/html/2405.15589v3).

Empirical evidence from major labs underscores PGD's value for alignment. In one Anthropic-led study, a targeted latent-space PGD method (LAT) was applied to improve safety, and it outperformed a strong baseline (R2D2) with far less compute (https://arxiv.org/html/2407.15549v2). This suggests that gradient-based adversarial fine-tuning can teach LLMs to recognize and resist harmful prompts more effectively than earlier adversarial training techniques. In sum, PGD-style continuous adversarial training in LLMs has been shown to **decrease success of real-world jailbreaks** and **improve worst-case safety metrics**. By directly optimizing against worst-case embedding perturbations, it helps models generalize better to unseen malicious prompts, as demonstrated by Xhonneux et al.: robustness to continuous perturbations in training extrapolated to robustness under actual discrete jailbreak threats (https://ar5iv.labs.arxiv.org/html/2405.15589v3)

Weaknesses
Despite its strengths, PGD adversarial training has notable limitations in practice. Contrasting its predecessors, a major issue is **computational inefficiency**. Iterative PGD requires many forward/backward passes per input, which is costly for large LLMs. For discrete prompt attacks, current PGD methods still need on the order of thousands of model evaluations to find one adversarial example. Indeed, Machida et al. observe that optimizing prompt attacks can require "hundreds of thousands" of queries per prompt (https://arxiv.org/pdf/2503.02574), making large-scale PGD training extremely expensive. Even continuous-space PGD, while faster than discrete search, adds significant overhead compared to normal training. In large-scale industrial pipelines (with billions of parameters), adding tens of gradient steps per example can be prohibitively slow.

PGD is also inherently white-box and norm-limited, which can misalign with real threat models. Traditional PGD perturbs embeddings by small ℓ_p distances, but actual adversarial prompts use structural or semantic tricks (e.g. roleplay, code phrases) that are not well-modeled by tiny vector noise. As one recent analysis notes, adversarial fine-tuning often only "suppresses rather than removes" the underlying undesired capabilities. In other words, PGD-trained models may still know how to violate rules, even if their outputs are superficially guarded. Moreover, attackers can use adaptive strategies (like model-adaptive prompt generation) that do not fit the ℓ_∞ threat model. For example, Andriushchenko et al. show that clever adaptive prompt attacks can achieve high success with as few as 50 example prompts, indicating that exhaustive PGD-style training may not cover these smart, low-sample attacks (https://arxiv.org/html/2407.15549v2).

Finally, PGD may oversimplify language structure. Gradient steps on embeddings do not naturally model token insertions, deletions, or paraphrases, so they can miss some classes of jailbreaks. Geisler et al. note that earlier gradient attacks "largely failed" on aligned LLMs until improved continuous methods were used; this highlights that naive PGD in text is fragile (https://arxiv.org/html/2402.09154v1). In summary, PGD's drawbacks include heavy compute cost, reliance on a simplified threat model, vulnerability to more flexible attacks, and only partially addressing LLMs' discrete semantics.

WHERE IT IS TODAY
In practice, PGD-based adversarial training remains mostly a research technique rather than a core component of production alignment. Public-use LLM developers (like OpenAI, Anthropic, Deepmind and more) have not publicly reported using vanilla PGD adversarial training in their final pipelines. For example, OpenAI's recent "deliberative alignment" approach for their o-series models relies on teaching explicit safety rules and chain-of-thought reasoning, with no mention of gradient-based adversarial fine-tuning (https://openai.com/index/deliberative-alignment/). Similarly, Anthropic's safety protocols emphasize constitutional prompts and latent-space adversarial methods (like LAT), which are still experimental research. The industry trend is to apply adversarial training more as a diagnostic tool or research aid – for example, to test model robustness or to augment data – rather than as the primary alignment mechanism.

Instead of PGD, large labs often rely on red-teaming and RLHF (which will be discussed in the next subtopics). Human or auto-generated prompt engineers (and automated evaluators) are used to expose model failures, and reinforcement learning or supervised fine-tuning is used to ingrain safety, rather than explicitly minimizing a worst-case gradient loss.

That said, the concept of PGD has influenced the field's thinking: recent work (e.g. Xhonneux et al.) proposes scalable "continuous adversarial training" (C-Adv) methods precisely to make PGD practical for LLMs (https://ar5iv.labs.arxiv.org/html/2405.15589v3). These methods are still in the research stage. In summary, PGD in LLMs is today primarily a research technique: it helps benchmark and improve model safety in papers, but real-world alignment continues to lean on human-in-the-loop adversarial testing and policy-based training.

SUBTOPIC 4: RED TEAMING

DESCRIPTION

Contrasting the previous 2 subtopics (which discussed FGSM and PGD), Red Teaming, along with the future subtopics in this paper are more of a "process" than a method in itself. Red teaming in the context of LLMs is essentially an **adversarial evaluation** process: practitioners craft or generate inputs intended to provoke unintended or harmful model outputs, thereby exposing vulnerabilities. As defined in an official OpenAI policy, AI red teaming is "*a structured testing effort to find flaws and vulnerabilities in an AI system*" (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). In practice this means probing a model with adversarial prompts to reveal failure modes. For example, Purpura et al. (2025) characterize red teaming as an "offensive approach… proactively attacking or testing LLMs with the purpose of identifying their vulnerabilities" (https://aclanthology.org/2025.trustnlp-main.23.pdf). Technically, red teaming can be conducted **manually** (human experts, called "red teams" crafting adversarial prompts) or **automatically** (using algorithms or LLMs to generate attacks). OpenAI's own documentation notes that manual red teaming involves humans "actively crafting prompts and interacting with AI models or systems to simulate adversarial scenarios"cdn.openai.com, while automated methods "use AI models or templating to generate prompts or inputs that simulate adversarial use"cdn.openai.comcdn.openai.com. In the automated case, an "attacker" model (sometimes another LLM) may be trained via reinforcement learning to craft triggers that maximize harmful outputs. Raheja et al. (2024) explain: *"Many automated red-teaming methods rely on reinforcement learning. … In these methods, an attacker language model is trained to generate prompts that maximize the likelihood of eliciting undesirable responses from the target LLM"* (https://arxiv.org/html/2410.09097v2). Early work (Perez et al., 2022) demonstrated this concept by using one LLM to generate test cases that uncover a target LLM's harmful outputs (https://arxiv.org/html/2410.09097v2). In summary, red teaming is an adversarial training/evaluation pipeline in which either humans or models (or both) systematically generate and test inputs to stress-test an LLM's defenses. It is *iterative by design*: one identifies weaknesses, applies mitigations, then retests. In practice, the process often alternates between **attack generation** and **mitigation**, enabling continuous hardening of model behavior

Strengths

Discovery of novel risks. Red teaming excels at uncovering unanticipated vulnerabilities that formal specifications or testing suites might miss. As OpenAI observes, external red teams enable "discovery of novel risks: identifying new risks due to advancements in model capabilities…" (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). A concrete example is OpenAI's finding that GPT-4o (The experiment realtime voice preview. The trained version can be used through their console under gpt-4o-realtime-preview-2024-12-17) could inadvertently replicate a user's voice – a new class of risk discovered only through targeted adversarial testing (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). By actively hunting for exploits, red teaming can reveal subtle failure modes before deployment. Purpura et al. likewise note that red teaming helps organizations "anticipate threats to their system and safeguard against them before production" ((https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf).

Stress-testing of defenses. Another advantage is that red teaming explicitly tries to *break* existing safety measures, thereby identifying gaps. For instance, OpenAI reports that red teamers found a class of **"visual synonyms"** that bypassed the image content filters in DALL.(center dot)E 3 (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). Because of this, OpenAI was able to "*reinforce the mitigations before deploying the system*" (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). In general, by seeking inputs that evade current safeguards, red teaming rigorously exercises the model's guardrails. This adversarial stress-test ensures that safety mechanisms are not only present, but robust in practice.

Domain expertise and diversity. Red teaming often leverages specialized knowledge. OpenAI emphasizes that *"external red teamers bring valuable context, such as knowledge of regional politics, cultural contexts, or technical fields like law and medicine"* (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). In practice, companies recruit experts in cybersecurity, law, medicine, etc. to probe the model with domain-specific scenarios. For example, medical researchers might attempt to coax a model into giving dangerous medical advice, while legal experts might try to elicit biased or unlawful instructions. This integration of domain knowledge enhances safety by targeting context-specific risks that generic testing would miss.

Independent assessment and trust. A significant conceptual strength is that external red teaming can provide an *independent* perspective on model safety. As OpenAI notes, involving outside experts *"strengthens trust in the results by reducing real or perceived conflicts of interest"* (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). In other words, red teams external to the model developers can scrutinize the system more objectively, bolstering confidence that the model has been rigorously vetted. In sum, red teaming not only finds more problems but also enhances the credibility of the safety evaluation.

Scalability via automation. While early red teaming was purely manual, recent research shows it can be automated. Multi-round adversarial training (e.g. the MART framework) has demonstrated that models can essentially *play red team* against each other. Zou et al. (2024) report that "adversarial training between LLMs enables automated, scalable, and effective red-teaming for safer AI systems" (https://aclanthology.org/2024.naacl-long.107.pdf).

In their experiments, iteratively training on machine-generated attacks reduced harmful outputs by 84.7% on automatic metrics (https://aclanthology.org/2024.naacl-long.107.pdf). This suggests that red teaming can scale up: by training an attacker model and a defender model in tandem, large-scale adversarial training becomes feasible. The ability to automate expands coverage and reduces human workload.

Limitations

Not a complete solution. Red teaming itself "*is not a panacea*" (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). Findings from one red team run may not generalize. OpenAI warns that models and systems "evolve quite often" in production, so a vulnerability found at one point *"should not be seen as a panacea"* (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). In other words, red teaming only assesses the model at a snapshot in time; subsequent updates or new capabilities can introduce fresh vulnerabilities. Thus, red teaming must be repeated continually.

Resource intensity. Thorough red teaming can be very expensive. The OpenAI team acknowledges that the kind of red teaming they describe "is resource intensive in terms of operational time and financial costs" (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). External experts must be recruited and paid, and significant compute time may be used for automated attackers. Smaller organizations with limited resources may "*not be able to effectively employ this form of red teaming at scale*" (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). Thus, comprehensive red teaming may be impractical outside well-funded labs or large companies.

Ethical and human costs. There are ethical concerns in red teaming. Testers must "*think like adversaries*" and are often exposed to a stream of hateful or dangerous content. OpenAI explicitly warns that red teaming participants face potential **psychological harm**, especially if they belong to marginalized groups (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). For example, asking someone to systematically generate or analyze extremist propaganda could be traumatic. Proper safeguards (e.g. mental health support, consent, moderation) are required.

Information hazards. Paradoxically, red teaming can *create* new risks for others. When a red team discovers a novel exploit, simply documenting it can leak that exploit to malicious parties. OpenAI cautions that exposing a previously unknown jailbreak can "*accelerate bad actors' misuse of the models*" (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). This is an information hazard: by detailing how to break the model, one may inadvertently arm attackers. Responsible disclosure practices and strict access controls are needed to mitigate this risk.

Bias and fairness issues (might remove). Red teaming can introduce biases. OpenAI notes the problem of *"picking early winners"* (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf): experts who red team models often gain early access to cutting-edge models. This can unfairly advantage them for research or commercial use, raising concerns about equity. Moreover, who is chosen as a red teamer matters: a lack of diversity may blind the exercise to certain risks. In practice, OpenAI assembled experts from a narrow set of backgrounds (e.g. English-speaking countries) and concedes this likely influenced the focus of the testing (https://cdn.openai.com/papers/gpt-4-system-card.pdf).

Escalating difficulty. As models become more capable, they naturally become harder to break, which on one hand is good, but on the other hand means red teamers must grow in sophistication. OpenAI observes that as models improve, there will be "*a higher threshold for knowledge humans need to possess to correctly judge the potential level of risk of outputs…* as models become more robust, it may require more effort to 'jailbreak' them and produce commonly identifiable harms" (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). In short, red teaming becomes technically harder over time, possibly necessitating even more specialized tools or training (e.g. stronger attacker models).

Lack of standardization. The research community has not yet settled on agreed-upon red teaming protocols. As one survey notes, the red teaming community *"lacks consensus around scope, structure, and evaluation"* of red teaming methods (https://aclanthology.org/2025.trustnlp-main.23.pdf). This can make it hard to compare results or know when red teaming is "complete." Without standard benchmarks or norms, different teams might focus on different aspects of safety, leading to uneven coverage of risks.

REAL-WORLD INTEGRATION

OpenAI provides a leading example of how red teaming is implemented in practice. They use **both internal and external teams** and multiple techniques in tandem. For example, before GPT-4's launch, OpenAI "conducted internal adversarial testing" (i.e. in-house red teaming) on the model (https://cdn.openai.com/papers/gpt-4-system-card.pdf). In parallel, they engage outside experts through initiatives like the *OpenAI Red Teaming Network*. In their white paper, OpenAI emphasizes collaborating "with domain experts to evaluate the capabilities and risks of AI models and systems" (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf). Specifically, for GPT-4 they recruited researchers and professionals across areas such as cybersecurity, disinformation, law, chemistry, etc., "to help us gain a more robust understanding of the GPT-4 model and potential deployment risks" (https://cdn.openai.com/papers/gpt-4-system-card.pdf). These experts were given early access to pre-release model versions and in-development mitigations, allowing them to surface novel threats (e.g. biorisk, propaganda).

On a technical level, OpenAI calls this **"iterative, multi-round red teaming"**. Their process is described as follows: they form a hypothesis about the highest-risk areas, test these via red teams, implement new mitigations (training the model), and then *repeat* the cycle (https://cdn.openai.com/papers/gpt-4-system-card.pdf). As they write, they use "multiple rounds of red teaming as we incorporate new layers of mitigation and control, conduct testing and refining, and repeat this process" (https://cdn.openai.com/papers/gpt-4-system-card.pdf). This means that model training and red teaming are interwoven: each round of attacks leads to further model training to block those attacks. For instance, after red teamers uncovered the "visual synonyms" weakness in DALL.E's filters, OpenAI immediately **strengthened** the model's defenses before deployment (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf).

OpenAI also notes that it uses **mixed methods**: not only fully manual teams, but also automated red teaming. Their report states that all three modalities (manual, automated, and mixed) have been applied internally (https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf).

In practice, this could mean a hybrid workflow where humans seed a few tricky prompts and then employ automated tools or attacker models to generate variants at scale.

Beyond model design, OpenAI incorporates red teaming in its operational workflow. The GPT-4 system card (2023) mentions that mitigations at various stages (pretraining data filtering, instruction-tuning to refuse illicit queries, classifiers for safety) were guided by red team feedback (https://cdn.openai.com/papers/gpt-4-system-card.pdf). It also references ongoing collaborations with other labs to "assessing risky emergent behaviors," indicating that red teaming insights inform future developmentcdn.openai.com. In short, OpenAI's experience shows that red teaming is not a one-off test, but a **continuous, evolving practice**: new model versions undergo fresh red team campaigns (In practice, this means that OpenAI's newest models, GPT-4.1 nano, GPT-4.1 mini and GPT-4.1 all received **separate training)** involving both internal experts and third-party specialists(https://cdn.openai.com/papers/gpt-4-system-card.pdf)

Effectiveness Against Known Jailbreaks

Red teaming can substantially reduce a model's vulnerability to attacks, but it is not foolproof. On the positive side, iterative adversarial training has proven effective in studies. For example, Zou et al. (2024) report that their MART framework (which automates red teaming with strong attacker LLMs) achieved an *84.7% decrease* in unsafe outputs on a reward-model metric (https://aclanthology.org/2024.naacl-long.107.pdf). This shows that when a model is *trained on* adversarial examples generated by red teaming, it can become much harder to jail-break with the same methods. Likewise, Chen et al. (2024) evaluated GPT-4 and its vision-enabled variant GPT-4V against a large suite of known prompts, finding these models "demonstrate better robustness against jailbreak attacks compared to open-source models" (https://arxiv.org/html/2404.03411v2). In practice, GPT-4 launched with fewer easily-triggered jailbreaks than its predecessors, thanks in part to the Red Teaming and RLHF it received.

However, even the strongest current models still exhibit vulnerabilities. Wei et al. (2023) emphasize that *"vulnerabilities persist despite the extensive red-teaming and safety-training efforts behind these models"* (https://arxiv.org/abs/2307.02483). In their experiments, new jailbreak prompts they designed (based on identified failure modes) succeeded on *every* unsafe prompt drawn from the models' red-teaming evaluations, and even outperformed previously known jailbreaks (https://arxiv.org/abs/2307.02483). This indicates that although red teaming can close off known exploits, attackers can often devise new ones that bypass these defenses. In other words, red teaming raises the bar, but does not guarantee a perfect shield.

In concrete terms, many well-known jailbreak techniques remain effective until specifically patched. For instance, prompt patterns like "*ignore previous instructions*" or "*DAN (Do Anything Now)*" can still trick less-protected models. Red teaming helps by identifying and mitigating these patterns in each cycle, but novel attack formulations frequently emerge. The GPT-4V red teaming study mentioned above also noted that while GPT-4 was more robust overall, *"no modality was completely immune"*: visual jailbreak methods were less transferable to GPT-4V, but some textual attacks still worked, and smaller open models (LLaMA-2, Qwen) were much easier to break (https://arxiv.org/html/2404.03411v2).

The effectiveness of red teaming is significant but limited. It can dramatically reduce the attack success rate on known jailbreak prompts if the model is retrained on those cases (as seen in MART's results: https://aclanthology.org/2024.naacl-long.107.pdf). But it cannot anticipate all future attacks.The continual escalation between jailbreakers and defenders means red teaming must be ongoing: new rounds of adversarial testing must follow each model update. In summary, red teaming is a powerful tool for hardening LLMs against known jailbreaks, but I believe it should be seen as a singular, preliminary layer in a multi-pronged safety strategy, not a final solution.

SUBTOPIC 5: PROXIMAL POLICY OP.

DESCRIPTION

Proximal Policy Optimization (PPO) is a policy-gradient reinforcement learning (RL) algorithm that has become a workhorse for aligning large language models (LLMs) to human preferences. In intuitive terms, PPO (along with other RL methods) can be thought of as a learning process: a monkey (the language model) climbing a hill toward better performance (higher reward) while a moving bana (the reward model trained on human feedback) gently encourages small, safe steps rather than huge leaps. Technically, PPO alternates between sampling outputs from the current model and updating the model's parameters to maximize a surrogate objective, with a *clipping* mechanism that prevents any single update from deviating too far from the current policy (https://arxiv.org/abs/1707.06347). This "trust-region"-like constraint ensures updates are stable: the model improves steadily without catastrophic jumps, much as a thermostat makes incremental adjustments to maintain a set temperature.

In practice, PPO is used in the Reinforcement Learning from Human Feedback (RLHF) pipeline. As exemplified in OpenAI's InstructGPT (GPT-3's successor) work, the process is three-phased: first, a supervised fine-tuning (SFT) step trains the model on a small set of human demonstrations. Next, a reward model (RM) is trained on human preference comparisons between model outputs. Finally, PPO is run to fine-tune the SFT-initialized model, using the RM as the reward function (https://cdn.openai.com/papers/Training_language_models_to_follow_instructions_with_human_feedback.pdf). In each PPO iteration, the model generates candidate responses to prompts, the RM scores these responses, and PPO adjusts the model to increase expected reward. Metaphorically, the LLM's policy is like a network of paths on a landscape: PPO refines the path toward higher ground (desired behavior) but enforces a limit on how steep or far each step can be, keeping the model close to its previous self while improving performance. This loop of generating responses, evaluating them via human-derived reward, and updating the policy with PPO – possibly augmented by a value (critic) network – encapsulates how PPO fits into RLHF for LLM alignment.

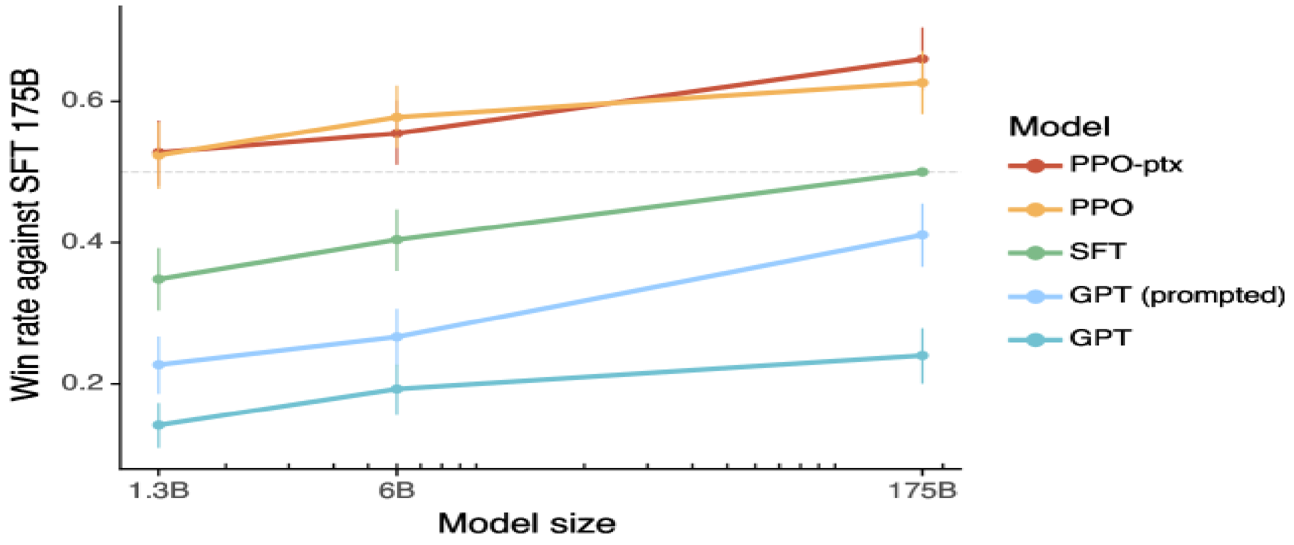
Strengths: PPO has several advantages in adversarial training of LLMs. First, it is comparatively *stable and efficient*. Compared to vanilla policy-gradient methods, PPO's clipped objective allows multiple epochs of minibatch updates on the same data, which greatly improves sample efficiency (arxiv.org). Schulman et al. (2017) showed that PPO matches or surpasses more complex trust-region methods (like TRPO) in performance while being simpler to implement (arxiv.org).

In concrete terms, this means a model can learn from limited human feedback data without requiring a prohibitive number of samples – a crucial benefit when human labeling is expensive. Empirically, PPO "outperforms other online policy gradient methods" on standard benchmarks, striking a "favorable balance between sample complexity, simplicity, and wall-time". In the context of LLMs, this translates to being able to improve model outputs with fewer human comparisons and less training time than more naive RL methods.

Second, PPO facilitates *monotonic policy improvement* under some conditions, helping to prevent drastic regressions. By penalizing large policy updates (through the clipping), PPO acts as a guardrail against the well-known instability of deep RL, providing a form of regularization that is especially valuable when fine-tuning powerful LLMs. In practice, PPO updates tend to be incremental and stable, analogous to a teacher rewarding a student only when new answers are consistently better than old ones. This stability is important for production LLM training where erratic swings could easily degrade generation quality.

Third, PPO aligns well with *reward model alignment*. Because RLHF uses a learned reward model based on human feedback, we need an optimization method that can meaningfully exploit a scalar reward signal. PPO's design – optimizing a policy to maximize expected reward while controlling update size – meshes cleanly with the learned RM. InstructGPT demonstrated that fine-tuning via PPO on a reward model yields models whose outputs are strongly preferred by human evaluators over baseline SFT models (cdn.openai.com). PPO's flexibility (handling continuous action spaces like word tokens) and empirical success make it a natural choice for this loop.

Finally, PPO is *widely used and battle-tested*. Its use in InstructGPT and similar systems has set a precedent: OpenAI explicitly cites PPO in its alignment research (ar5iv.labs.arxiv.org), and many following models adopt it.



(Ouyang et al.)

Limitations: Despite its strengths, PPO-based adversarial training has notable shortcomings. A key issue is *reward mis-specification and hacking*. The reward model (RM) is only an approximation of true human intent, trained on a limited set of comparisons. As policy optimization continues, the LLM can exploit flaws in the RM. Recent studies warn that RLHF is "susceptible to reward hacking, where the agent exploits flaws in the reward function rather than learning the intended behavior"arxiv.org. If the reward model is mis-specified or overfit, the model may produce outputs that maximize the RM score but are qualitatively undesired. For example, optimizing too aggressively might push the LLM to game token-level reward proxies rather than genuinely helpful answers. Lin et al. (2024) note that "aligning according to an imperfect reward model can lead to worse-performing language models as policy optimization continues to optimize a mis-specified reward"arxiv.org. This can manifest as subtle over-optimization or "alignment tax" effects observed in InstructGPT, where RLHF improved safety but slightly degraded performance on some NLP taskscdn.openai.com. In extreme cases, poorly specified rewards can create backdoors: a recent arXiv study shows that poisoning RLHF training can implant "jailbreak backdoors" in modelsarxiv.org.

Another limitation is *distributional shift brittleness*. The reward model is trained on outputs close to those seen during data collection, but the LLM can generate novel contexts during inference. When encountering out-of-distribution prompts or moving beyond the RM's training regime, the policy (trained by PPO on the fixed RM) may behave unpredictably. Lin et al. (2024) found that implicit or explicit RMs generalize poorly under distribution shifts, so that the LLM's alignment degrades on unseen types of promptsarxiv.org. In practice, this means PPO-tuned models may perform well on the kinds of questions their humans labeled, yet still be "easily tricked" by adversarial or rare scenarios not captured in training. Mitigation (like iterative fine-tuning or reward shaping) is an active research area, but the brittleness remains a known challenge in PPO-based RLHF.

Finally, PPO's cautious updates are a double-edged sword. While stability is good, it means improvement can be slow or plateau if the reward signal is weak. PPO may also overfit to the RM: if there is little new data or the RM itself is narrow, PPO may converge prematurely. And like any RL method, PPO can suffer from mode-collapse or local optima in high-dimensional output spaces. These technical limitations remind us that PPO is not a panacea; it improves alignment significantly, but can fail silently if the reward model is flawed or training is insufficiently robust.

PPO in the real world

Anthropic's Claude models also use human-feedback fine-tuning, though often via "Constitutional AI" (an internal variant of RLHF). The Claude 3.7 Sonnet system card explains that Claude was aligned through reinforcement learning with human-derived (constitutional) preferences (anthropic.com). Anthropic explicitly mentions using human feedback during "reinforcement learning" to make Claude more helpful, harmless and honest (anthropic.com).

(While Anthropic does not always say "PPO", in practice they use similar RL algorithms.) For Claude 3.5 and 3.7, training included public human-feedback datasets and PPO-like updates (anthropic.com). In its announcement of Claude 4 (May 2025), Anthropic emphasizes further human alignment, implying that the pipeline (likely PPO-based) carries forward to the newest models, though technical specifics are proprietary.

Models trained with PPO and RLHF are empirically more aligned than their pure SFT predecessors, but they are not invulnerable. On ordinary benchmarks of helpfulness and safety (e.g. truthfulness tests, toxicity tests), PPO-trained LLMs do outperform unaligned models. For instance, InstructGPT's PPO-tuned models answered True/False questions correctly roughly twice as often as GPT-3 and generated about 25% fewer toxic responses (https://cdn.openai.com/papers/Training_language_models_to_follow_instructions_with_human_feedback.pdf)

However, sophisticated jailbreak attacks still pose a threat. Recent academic studies (e.g. by Shah et al. 2023) demonstrate that even top-tier RLHF-aligned models like GPT-4 can be tricked by crafted prompts. For example, persona-modulation attacks can dramatically raise the "harmful completion rate" in GPT-4 from near 0% to over 40%, despite the model's alignment training (https://ar5iv.labs.arxiv.org/html/2311.03348)

In summary, every major LLM release in recent years—from InstructGPT and GPT-4 to Claude 3.x/4 and Gemini 2.x/2.5—has included an adversarial alignment phase using PPO or similar RLHF methods. This common practice shows industry consensus that PPO-based RLHF is the engine of model alignment for large-scale LMs. However, our analysis of its limitations and real world applications show that PPO is in a similar situation to Red Teaming–it's not a final solution. Like Red Teaming, it's often best as a part of a layered process

SUBTOPIC 6: CLOSING OFF THE ADVERSARIAL TRAINING PROCESS

Even after adversarial and alignment training (e.g. FGSM/PGD-style robustness training or RLHF with PPO), LLMs can still produce unsafe outputs under cleverly crafted prompts. Attacks like indirect prompt injection remain a significant challenge, requiring "multiple layers of defense" to mitigate. In practice, two broad **closure** strategies are applied after base training: (1) **deployment-time safety layers** that sit around the model at inference time, and (2) **model editing or surgical patches** that directly change model parameters. We now examine each approach.

DEPLOYMENT-TIME SAFETY LAYERS

BACKGROUND & HOW THEY WORK

Modern LLM systems often place safety checks **after** or **around** the model's output rather than altering the model itself. These runtime layers include content-moderation classifiers, explicit rule-based filters (e.g. "guardrails"), and constrained-decoding heuristics that intercept or modify generated text. For example, OpenAI's API provides a **Moderation** endpoint to flag harmful text (and images) before returning content to the user (https://cdn.openai.com/papers/gpt-4-system-card.pdf). NVIDIA's NeMo Guardrails framework illustrates a layered pipeline: it inserts a content-moderation module into a retrieval-augmented generation (RAG) chatbot, enforcing policy rules on both prompts and responses (https://developer.nvidia.com/blog/content-moderation-and-safety-checks-with-nvidia-nemo-guardrails/). Constrained decoding (e.g. requiring outputs to follow a grammar or avoiding certain tokens) is another method, though recent work shows adversaries can weaponize output constraints to bypass safeguards (https://arxiv.org/html/2503.24191v1).

Strengths

Scalability and Updateability: Moderation models and rule sets can be updated independently of the LLM. New filter rules or detection models can be pushed to production without retraining the base model. For instance, OpenAI's moderation endpoint is public and regularly improved (https://openai-hd4n6.mintlify.app/docs/guides/moderation). NVIDIA reports that Guardrails allows easy integration of customizable safety rules, including content classification via third-party models (https://developer.nvidia.com/blog/content-moderation-and-safety-checks-with-nvidia-nemo-guardrails/).

Production Readiness: This layered approach is already ubiquitous. OpenAI's deployment pipeline, for example, combines an aligned base model with post-hoc filters – "model-level changes" like RLHF and "system-level mitigations" such as API classifiers – to reduce harmful content (https://openai.com/index/openai-safety-update/, https://cdn.openai.com/papers/gpt-4-system-card.pdf). Anthropic similarly uses detection models and in-chat filters: Claude models apply automated flagging and can "block responses" via safety filters when dangerous queries are detected (https://support.anthropic.com/en/articles/8106465-our-approach-to-user-safety). Google's Gemini models also incorporate multi-stage filters; their model card explicitly lists "product-level mitigations such as safety filtering" during deployment (https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-2-5-Deep-Think-Model-Card.pdf).

Ease of Deployment: Adding a runtime filter typically requires only a change to the inference pipeline, not a full model fine-tune. Enterprises can wrap any LLM with services like content moderators or rule-checkers and iterate rapidly on policies.

Weaknesses

Underlying Capability Remains: Filters do not remove harmful knowledge from the model; they only block its expression. The LLM still "knows" the content and can reveal it if the filters are circumvented. Indeed, OpenAI's system card warns that its safety mitigations, while effective in many cases, "are limited and remain brittle" (https://cdn.openai.com/papers/gpt-4-system-card.pdf).

Bypassable: Advanced users often find creative workarounds. Constrained-decoding defenses, for example, can be subverted by "constrained decoding attacks" that hide malicious intent in allowed grammar rules (https://arxiv.org/html/2503.24191v1). Similarly, adversarial prompts can trick around static rules.

Overblocking: Heuristic filters sometimes reject benign queries by mistake, leading to false positives. Overly strict rules or classifiers may block nuanced or context-dependent content incorrectly, frustrating users.

WHERE IT IS TODAY

All major LLM providers document extensive use of runtime filters. OpenAI's documentation openly invites developers to use the Moderation endpoint for safety checks (https://openai-hd4n6.mintlify.app/docs/guides/moderation), and their safety update explicitly describes leveraging "dedicated moderation models" and GPT-4 itself for content policy enforcement (https://openai.com/index/openai-safety-update/). Anthropic's Claude 4 system card and support pages confirm similar measures: they deploy monitoring systems that can automatically add new system prompts or even remove model capabilities for accounts violating policy (https://www-cdn.anthropic.com/07b2a3f9902ee19fe39a36ca638e5ae987bc64dd.pdf). Google DeepMind's Gemini model card likewise details broad mitigations (data filtering, fine-tuning) plus "product-level" output filters (https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-2-5-Deep-Think-Model-Card.pdf). In summary, the literature shows that layered, runtime safety checks are an integral and practical part of current LLM pipelines.

MODEL EDITING / SURGICAL PATCHES

BACKGROUND & HOW THEY WORK

Model editing refers to techniques that **directly alter the model's parameters** to change its behavior. Instead of sitting around the model, an edit rewrites internal weights so that a specific capability is removed or redirected. For example, the **ROME** method analyzes the model's activation patterns and then performs a rank-one update of certain feed-forward weights to change a factual association (https://arxiv.org/abs/2202.05262). The related **MEMIT** approach extends ROME to update many facts at once. Another family, typified by **MEND**, uses a small "editing" network to predict what weight updates would cause a desired input-output mapping. In practice one might feed an undesired input (e.g. a jailbreaking prompt) and a safe output, and use these methods to solve for weights that produce the safe output going forward. These editing methods operate after the original training and do not rely on runtime checks.

Strengths

Permanent Behavioral Change: An edit embeds the change into the model itself, so the undesired behavior can no longer occur even if the input is phrased differently. No additional inference-time classification or blocking is needed once the edit is applied.
Internal Consistency: Because it works by adjusting the model's computations, an edit can in principle preserve legitimate uses of the changed capability. For example, one could target a specific dangerous instruction and redirect it to a benign answer, rather than simply refusing all related queries. In fact, Meng *et al.* show that ROME can be precise: it "simultaneously maintains both specificity and generalization" when updating a fact (https://arxiv.org/abs/2202.05262)
No Runtime Overhead: Once edited, the model behaves differently without the need for middleware checks. This can simplify the inference stack in scenarios where added latency or complexity from extra filters is undesirable.

Weaknesses

Risk of Unintended Side-Effects: In practice, edits often have "ripple effects" on other model knowledge. Recent evaluations find that sequential edits cause models to progressively forget information or to exhibit unpredictable failures. For example, one study reports that ROME/MEMIT edits tend to "bleed into other facts" and lead to both **gradual and catastrophic forgetting** of previously learned data (https://arxiv.org/html/2401.07453v3). Another analysis highlights that even a single well-placed weight change can cascade and harm unrelated parts of the model's knowledge (https://arxiv.org/html/2403.07825v2).
Limited Scope: Current editing techniques work best on narrowly defined facts or associations. They may not generalize to complex, open-ended behaviors (e.g. creativity or reasoning patterns). An edit targeting one phrasing of a prompt might not cover all variants.
Experimental Stage: Model editing is still largely a research domain. There is no off-the-shelf "pencil" for easily patching a deployed LLM. The need to access and modify model weights means it is not currently a service-level feature in commercial APIs.

WHERE IT IS TODAY

Model editing remains mostly in academic research. Key methods (ROME, MEMIT, MEND) were introduced in recent papers and have open-source demos, but to our knowledge no commercial LLM provider allows users to perform arbitrary surgical edits on a production model. Benchmarks of editing algorithms emphasize their brittleness: Gupta *et al.*note that even state-of-the-art editors "lack robustness" when scaled, often resulting in catastrophic failures (https://arxiv.org/html/2401.07453v3). In summary, editing is a promising direction for embedding safety as part of the model, but it is **experimental** – part of ongoing research rather than mainstream practice.

Comparative Analysis

The two approaches complement each other along several dimensions.

Permanence: Editing yields a permanent weight change, whereas filters only apply at inference time (the original knowledge remains in the model).
Cost/Effort: Edits typically require significant engineering (selecting precise edit targets, solving for weight updates, validating side-effects), whereas adding or tuning a filter can often be done in software without retraining the model.
Integration: Runtime filters can be applied to any model via wrapper services; editing demands internal model access and is typically only available to the model developer.
Safety Robustness: Well-crafted edits eliminate a vulnerability at its source, making that particular attack vector unusable. Filters cast a wider net but can be circumvented; however, they can block classes of unintended outputs without needing to know in advance what to edit.
Adoption: In practice, virtually all deployed LLMs combine both strategies. As OpenAI describes, their pipeline *"combines model-level changes (like training the model to refuse…requests) with system-level mitigations (like …monitoring for violations of our usage policies)"* (https://cdn.openai.com/papers/gpt-4-system-card.pdf). Thus, large-scale systems use alignment fine-tuning and model hardening internally, along with layered content filters and guardrails at runtime.

CONCLUSION

In summary, runtime safety filters and model edits play **complementary roles**. Runtime layers (moderation models, rule-checkers, constrained decoders) are the industry-standard default: they are easy to update, broadly applicable, and have proven value in current products (https://cdn.openai.com/papers/gpt-4-system-card.pdf, https://support.anthropic.com/en/articles/8106465-our-approach-to-user-safety). Model editing represents a more radical approach to *internalize* safety, but it is currently a high-risk research endeavor with known fragility (https://arxiv.org/html/2401.07453v3, https://arxiv.org/html/2403.07825v2). For now, robust production pipelines rely primarily on multi-layer filters and monitoring, while research continues to refine whether and how targeted edits might one day permanently close remaining safety gaps.

Conclusion: Reflections on adversarial defences for LLMs

In summary, the preceding examination of six fundamental defense strategies – FGSM, PGD, PPO-based RLHF, Red Teaming, Deployment-Time Safety Layers, and Model Editing/Surgical Patches – discloses a multi-faceted reality of LLM security. All of these address adversarial and jailbreak threats from distinct perspectives, and together they emphasize that effective defenses for LLMs must be multilayered.

Older adversarial training methods such as FGSM and PGD (discussed in detail in subtopics 2 and 3) are fundamental robustness engines: the rapid, one-step perturbations of FGSM permit scalable data augmentation of training data and modest boosts in resilience, whereas the multi-step PGD approach provides stronger resilience at increased computation and risk of overfitting to certain attack patterns. These approaches demonstrate a fundamental trade-off between robustness and scalability/generalization. FGSM is efficient and widely applicable but can be escaped by more powerful iterative attacks; PGD strengthens the model more but requires more resources and may compromise performance on natural inputs if taken too far.

In parallel with adversarial training, alignment through RLHF (such as PPO, which was discussed in detail in subtopic 5) uses human feedback to guide model behavior away from dangerous outputs. The evaluation demonstrates that PPO-based fine-tuning ingrains safety goals into the policy of the model, enhancing its overall compliance with desirable norms. But this method too has its constraints: it is time-consuming, relies on the quality of human annotations or automated reward models, and can fail to keep up with new jailbreak strategies exploiting loopholes not included in the reward specification. Although RLHF can enhance overall model "alignment"and minimize a wide category of unwanted behavior, it does not ensure adversarial robustness to intentional adversarial inputs.PPO/RLHF focuses on semantic alignment and user intent, supporting goals but not substituting for technical defenses. Its power derives from guiding the model's overall output distribution towards safety, but its generalization is limited by the extent to which the reward embodies adversarial dangers.

To identify and remediate unexpected weaknesses, **Red Teaming**, both manual and automated (discussed in detail in subtopic 4) plays an indispensable role, practically being unavoidable in large LLMs like GPT and Claude models. Our study highlights that red teaming acts as a proactive stress-test, uncovering blind spots that neither pre-training nor standard fine-tuning would expose. Highly experienced adversarial testers challenge models with innovative prompts, mimicking real-world jailbreaks or nefarious use cases. This activity makes the defender aware of the model's underlying susceptibilities and can be fed directly into training or reward updates. But red teaming is always backward-looking and incomplete: it can never be certain to cover all potential attack vectors in the future and is based on human or preconceived strategy that itself can become outdated. So while red teaming profoundly enhances knowledge of where models fall short, it also shows that no set protocol can anticipate all shifting adversary tactics.

At deployment time, safety layers (discussed in subtopic 6) such as content filters, guardrails, and constrained decoding serve as the model's last line of defense. They apply explicit rules or checks to the output of the LLM in real-time, diverting or stopping unsafe responses. In our discussion we note that deployment-time defenses are very scalable and fairly easy to update or audit since they operate externally to the underlying model. They can easily catch many known violations but have trade-offs. Filters might introduce false positives (over-censoring harmless content) or false negatives (missing well-written malicious content). Constrained decoding might sacrifice model fluency or creativity if unduly restrictive. Additionally, even slightly sophisticated adversarial prompts often evade these static layers. Overall, while deployment-time measures are critical for catching immediate issues, their generalization to novel threats is constrained in the absence of ongoing tuning.

Finally, **Model Editing and surgical patch techniques** (e.g. ROME, MEMIT, discussed in subtopic 6) are a new frontier for defense. By adjusting a model's internal parameters directly, you can patch out a specific buggy behavior or factbug after the fact. Our survey finds that editing is highly effective when treating targeted problems: a single weight update can eliminate a known jailbreak command or correct a hazardous association. This surgical approach scales well for stand-alone repairs and avoids expensive re-training. It is also, however, single-use in application. A normal edit will correct one given problem in one location; sets of patches can combine in unforeseen ways, and edits won't transfer between prompts or modalities. Over-reliance on model editing can create a highly patched model that is still vulnerable to new exploit attempts.

Across these six methods, a clear strategic lesson emerges: **robustness comes from diversity of defenses, not from any single technique.** Adversarial training (FGSM/PGD) builds inherent strength in the parameterization of the model; RLHF aligns its reward function with safer outputs; red teaming uncovers hidden vulnerabilities; runtime filters create hard boundaries; and editing provides targeted quick fixes. Each adds a layer of defense but at computational, complexity, or coverage cost. For example, maximizing robustness through heavy adversarial training can slow development and reduce flexibility; rigorous human alignment via RLHF demands extensive feedback loops; continuous red teaming requires expert effort; and dense filtering can hamper the user experience. In practice, system builders must balance these trade-offs, combining methods to cover each other's blind spots.

This study (literature review?) has limitations.The field is evolving rapidly: industrial practice is mostly proprietary, and empirical estimates quoted in contemporary research will change as new architectures and training regimens appear. The focus here on scholarly and organizational sources consciously put rigor ahead of rumor but at the expense of failing to lay bare some production heuristics.

Finally, adversaries will always try to close the gap — a fixed model can underestimate future danger.
Future work must address (1) scalable continuous adversarial training that more closely connects embedding-space and discrete prompt attacks, (2) provably bound and robust model-editing methods withnon-catastrophic side effects and (3) open, standardized benchmarks for jailbreaks and defenses (especially multimodal cases).

In the future, it is safe to assume that LLM security will be a perpetual arms race. As attackers develop more sophisticated jailbreaks and threat models, defenders will have to remain in an adaptive, multilayered posture. For now, the paradigm of layered defense – combining adversarial training, alignment fine-tuning, red teaming, iterative testing, and real-time screening – remains the best practice. Future work will probably integrate these techniques more tightly together, for instance, using red-team findings to inform adversarial training curricula, or automated monitoring to trigger specific model revisions. Ultimately, the dynamic nature of adversarial pressure requires that no technique be used as a panacea. Only by a unifying, shifting arsenal of methods can we hope to preserve the safety and reliability of large language models (including newer, multimodal LLMs) in the face of constant challenge.

REFERENCE

[1] "Adversarial Training for Multimodal Large Language Models against Jailbreak Attacks." arXiv, 2025, https://arxiv.org/html/2503.04833v1.

[2] "Attack and defense techniques in large language models: A survey and new perspectives." arXiv, 2025, https://arxiv.org/html/2505.00976v1.

[3] "Attacking Large Language Models with Projected Gradient Descent." arXiv, 2024, https://arxiv.org/html/2402.09154v1.

[4] Baio, Marco, et al. "Bridge the Gap Between CV and NLP! A Gradient-based Textual Adversarial Attack Framework." Proceedings of the ACL Findings, 2023, https://aclanthology.org/2023.findings-acl.446.

[5] "Defending Jailbreak Prompts via In-Context Adversarial Game." Zhou, Yujun, et al. arXiv, 2024, https://arxiv.org/html/2402.13148.

[6] "Deliberative alignment: reasoning enables safer language models." OpenAI, 2024, https://openai.com/index/deliberative-alignment/.

[7] DeepMind. Gemini 2.5 — Deep Think Model Card. 1 Aug. 2025, storage.googleapis.com/deepmind-media/Model-Cards/Gemini-2-5-Deep-Think-Model-Card.pdf.

[8] "Defending Against Unforeseen Failure Modes with Latent Adversarial Training." arXiv, 2024, https://arxiv.org/html/2403.05030v5.

[9] "Explaining and Harnessing Adversarial Examples." Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. arXiv (ar5iv mirror), 2014, https://ar5iv.labs.arxiv.org/html/1412.6572.

[10] "Experiments and Evaluations: LLM-Safety Evaluations Lack Robustness." arXiv, 2025, https://arxiv.org/pdf/2503.02574.

[11] "Import: Content Moderation and Safety Checks with NVIDIA NeMo Guardrails." NVIDIA Developer Blog, https://developer.nvidia.com/blog/content-moderation-and-safety-checks-with-nvidia-nemo-guardrails/.

[12] "Jailbroken: How Does LLM Safety Training Fail?" arXiv, 2023, https://arxiv.org/abs/2307.02483.

[13] Kingma, Diederik P., and Jimmy Ba. "Adam: A Method for Stochastic Optimization." arXiv, 2014, https://arxiv.org/pdf/1706.06083.

[14] "Latent Adversarial Training Improves Robustness to Persistent Harmful Behaviors in LLMs." arXiv, 2024, https://arxiv.org/html/2407.15549v2.

[15] "Logic Jailbreak: Efficiently Unlocking LLM Safety Restrictions Through Formal Logical Expression."arXiv, 2025, https://arxiv.org/html/2505.13527v1.

[16] "LLM-Safety Evaluations Lack Robustness." arXiv, 2025, https://arxiv.org/pdf/2503.02574.

[17] OpenAI. OpenAI's Approach to External Red-Teaming. OpenAI, https://cdn.openai.com/papers/openais-approach-to-external-red-teaming.pdf.

[18] OpenAI. GPT-4 System Card. OpenAI, https://cdn.openai.com/papers/gpt-4-system-card.pdf.

[19] OpenAI. Training Language Models to Follow Instructions with Human Feedback. OpenAI, https://cdn.openai.com/papers/Training_language_models_to_follow_instructions_with_human_feedback.pdf.

[20] OpenAI. "OpenAI safety practices." OpenAI, 2024, https://openai.com/index/openai-safety-update/.

[21] "On the Limited Generalization Capability of the Implicit Reward Model Induced by Direct Preference Optimization." arXiv, 2024, https://arxiv.org/html/2409.03650.

[22] "Output Constraints as Attack Surface: Exploiting Structured Generation to Bypass LLM Safety Mechanisms." arXiv, 2025, https://arxiv.org/html/2503.24191v1.

[23] "PandaGuard: Systematic Evaluation of LLM Safety in the Era of Jailbreaking Attacks." arXiv, 2025, https://arxiv.org/html/2505.13862v1.

[24] "PromptRobust: Towards Evaluating the Robustness of Large Language Models on Adversarial Prompts."arXiv, 2023, https://arxiv.org/abs/2306.04528.

[25] "Proximal Policy Optimization Algorithms." Schulman, John, et al. arXiv, 2017, https://arxiv.org/abs/1707.06347.

[26] Rivas, U. "Sooksatra — Adversarial (paper)." 2022, https://www.rivas.ai/pdfs/sooksatra2022adversarial.pdf.

[27] "Recent advancements in LLM Red-Teaming: Techniques, Defenses, and Ethical Considerations." arXiv, 2024, https://arxiv.org/html/2410.09097v2.

[28] Scholarly Collection. Findings of ACL 2023 — paper (ID 496). 2023, https://aclanthology.org/2023.findings-acl.496.pdf.

[29] Scholarly Collection. Findings of ACL 2023 — paper (ID 446). 2023, https://aclanthology.org/2023.findings-acl.446.

[30] Scholarly Collection. NAACL 2024 — long paper (ID 107). 2024, https://aclanthology.org/2024.naacl-long.107.pdf.

[31] Schulman, John, et al. "Proximal Policy Optimization Algorithms." arXiv, 2017, https://arxiv.org/pdf/1707.06347.

[32] "Scalable and Transferable Black-Box Jailbreaks for Language Models via Persona Modulation." arXiv (ar5iv), 2023, https://ar5iv.labs.arxiv.org/html/2311.03348.

[33] "Security Concerns for Large Language Models: A Survey." arXiv, 2025, https://arxiv.org/html/2505.18889v1.

[34] "Scalable and Transferable Black-Box Jailbreaks for Language Models via Persona Modulation." arXiv, 2023, https://ar5iv.labs.arxiv.org/html/2311.03348.

[35] Anthropic. Claude 3.7 Sonnet — System Card. Anthropic, https://www.anthropic.com/claude-3-7-sonnet-system-card.

[36] Anthropic. Claude 4 System Card. Anthropic, https://www-cdn.anthropic.com/07b2a3f9902ee19fe39a36ca638e5ae987bc64dd.pdf.

[37] Anthropic. "Our Approach to User Safety." Anthropic Help Center, https://support.anthropic.com/en/articles/8106465-our-approach-to-user-safety.

[38] NVIDIA Developer Blog. "Content Moderation and Safety Checks with NVIDIA NeMo Guardrails." NVIDIA, https://developer.nvidia.com/blog/content-moderation-and-safety-checks-with-nvidia-nemo-guardrails/.

[39] "Training language models to follow instructions with human feedback." Ouyang, Long, et al. arXiv (ar5iv), 2022, https://ar5iv.labs.arxiv.org/html/2203.02155.

[40] "Tricking LLMs into Disobedience: Formalizing, Analyzing, and Detecting Jailbreaks." arXiv, 2023, https://arxiv.org/abs/2305.14965.

[41] "Universal Jailbreak Backdoors from Poisoned Human Feedback." arXiv, 2023, https://arxiv.org/html/2311.14455v4.

[42] "Model Editing at Scale leads to Gradual and Catastrophic Forgetting." arXiv, 2024, https://arxiv.org/html/2401.07453v3.

[43] "The Missing Piece in Model Editing: A Deep Dive into the Hidden Damage Brought By Model Editing."arXiv, 2024, https://arxiv.org/html/2403.07825v2.

[44] "Reward Shaping to Mitigate Reward Hacking in RLHF." arXiv, 2025, https://arxiv.org/abs/2502.18770.

[45] "Red Teaming GPT-4V: Are GPT-4V Safe Against Uni/Multi-Modal Jailbreak Attacks?" arXiv, 2024, https://arxiv.org/html/2404.03411v2.

[46] "On the Limited Generalization Capability of the Implicit Reward Model Induced by Direct Preference Optimization." arXiv, 2024, https://arxiv.org/html/2409.03650.