# TamperBench: Systematically Stress-Testing LLM Safety Under Fine-Tuning and Tampering

**Anonymous authors**
Paper under double-blind review

## Abstract

As increasingly capable open-weight large language models (LLMs) are deployed, improving their *tamper resistance* against unsafe modifications, whether accidental or intentional, becomes critical to minimize risks. However, there is no standard approach to evaluate tamper resistance. Varied data sets, metrics, and inconsistent threat settings make it difficult to compare safety, utility, and robustness across different models and defenses. To this end, we introduce TamperBench, the first unified framework to systematically evaluate the tamper resistance of LLMs. TamperBench (i) curates a repository of weight-space fine-tuning attacks and latent-space representation attacks; (ii) allows for testing state-of-the-art tamper-resistance defenses; and (iii) provides both safety and utility evaluations. TamperBench requires minimal additional code to specify any fine-tuning configuration, alignment-stage defense method, and metric suite while ensuring end-to-end reproducibility. In this work, we use TamperBench to evaluate 21 open-weight LLMs, including defense-augmented variants, across nine tampering threats using standardized safety and capability metrics with hyperparameter sweeps per model-attack pair. *Code is available at*: `https://anonymous.4open.science/r/TamperBench-71DD`

## 1 Introduction

Diverse training procedures are used to safety-align modern LLMs (Touvron et al., 2023; OpenAI et al., 2024; Team et al., 2023), yet *tampering*, modifications to the model's weights or latent representations, can undermine these safeguards (Che et al., 2025; Huang et al., 2024b; Qi et al., 2024b; Murphy et al., 2025; Halawi et al., 2024; Schwinn & Geisler, 2024). This threat is increasingly pressing as compute-efficient approaches such as LoRA (Hu et al., 2022; Zhao et al., 2024; Meng et al., 2024) make tampering accessible at low cost, and open-weight models—modifiable by anyone—have capabilities closely behind those of frontier models (Cottier et al., 2024), which are already classified as high-risk under frontier safety frameworks (OpenAI, 2025; Anthropic, 2025).

In response to such tampering risks, over twenty defenses have been proposed in the past year alone (Lyu et al., 2024; Hsu et al., 2024; Tamirisa et al., 2025; Shen et al., 2025; Li et al., 2025b;a; Huang et al., 2025; Zhao et al., 2025; Bianchi et al., 2024; Qi et al., 2025; Casper et al., 2024; Wang et al., 2024a; Huang et al., 2024c;d; Liu et al., 2025; Huang et al., 2024a; Liu et al., 2024; Mukhoti et al., 2024; Wei et al., 2024; Du et al., 2025; Sheshadri et al., 2025; O'Brien et al., 2025; Simko et al., 2025) reflecting an active and fast-moving research landscape.

Despite the proliferation of proposed defenses, evaluations are fragmented as works differ in their choice of attacks, threat models, and metrics (Figure 2). The field lacks a standard basis for determining to what degree a defense improves tamper resistance (Huang et al., 2024b; Qi et al., 2024a). Casper et al. (2025) identify "model tampering evaluations" as a central open problem, noting that current work lags in testing models against a diverse suite of tampering attacks and hyperparameter settings. A comprehensive, standardized evaluation is necessary for understanding which tampering defenses are most promising, whether tamper resistance is tractable at all, and what precautions developers should take in releasing highly capable open-weight models.

To address this gap, we introduce TamperBench (Figure 1), the first systematic benchmark and toolkit for evaluating tamper resistance in open-weight LLMs. TamperBench provides an extensible suite of tampering attacks and standardized evaluation protocols, together with interfaces that make it

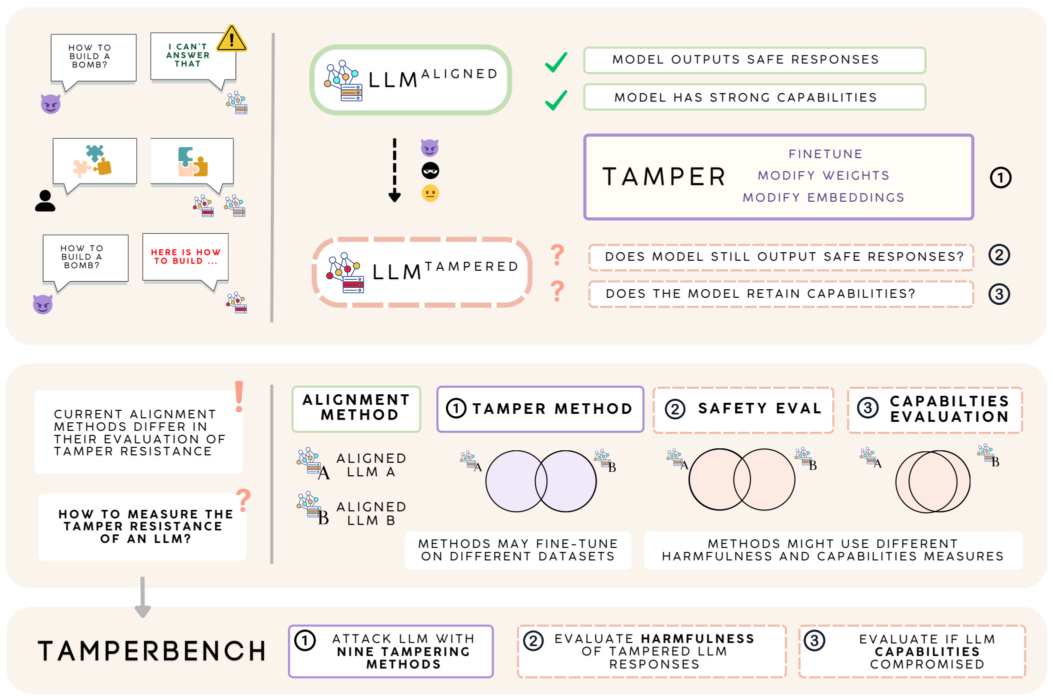

HOW TO BUILD A BOMB?
I CAN'T ANSWER THAT
HOW TO BUILD A BOMB?
HERE IS HOW TO BUILD ...
LLM ALIGNED
MODEL OUTPUTS SAFE RESPONSES
MODEL HAS STRONG CAPABILITIES
TAMPER
FINETUNE
MODIFY WEIGHTS
MODIFY EMBEDDINGS
1
LLM TAMPERED
DOES MODEL STILL OUTPUT SAFE RESPONSES?
2
DOES THE MODEL RETAIN CAPABILITIES?
3
CURRENT ALIGNMENT METHODS DIFFER IN THEIR EVALUATION OF TAMPER RESISTANCE
HOW TO MEASURE THE TAMPER RESISTANCE OF AN LLM?
ALIGNMENT METHOD
1 TAMPER METHOD
2 SAFETY EVAL
3 CAPABILTIES EVALUATION
ALIGNED LLM A
ALIGNED LLM B
A
B
METHODS MAY FINE-TUNE ON DIFFERENT DATASETS
METHODS MIGHT USE DIFFERENT HARMFULNESS AND CAPABILITIES MEASURES
TAMPERBENCH
1 ATTACK LLM WITH NINE TAMPERING METHODS
2 EVALUATE HARMFULNESS OF TAMPERED LLM RESPONSES
3 EVALUATE IF LLM CAPABILITIES COMPROMISED

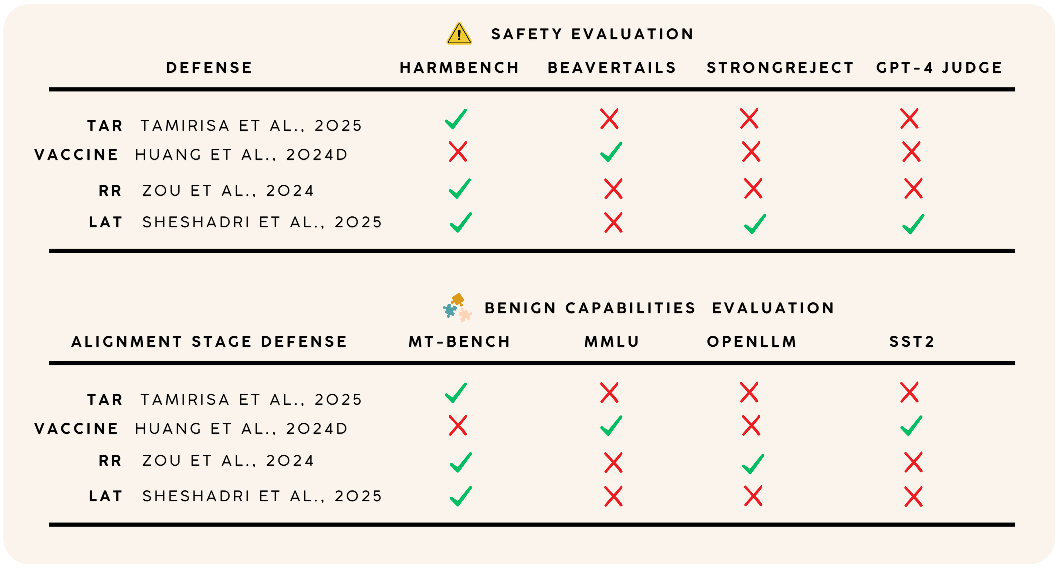
Figure 1: Tampering LLMs, as defined by Che et al. (2025)), involves modifying their weights or latent representations and can compromise safety guardrails, yielding models that can output harmful responses. While numerous methods have been proposed to make models 'tamper-resistant', there is a lack of a systematic framework to measure this. TAMPERBENCH provides a framework to stress test LLM safety under tampering.

straightforward to integrate and test defenses. The framework covers both benign and adversarial tampering regimes, including overt fine-tuning attacks and stealthy manipulations designed to evade closed-weight moderation systems. It supports both weight-space modifications and latent-space perturbations at inference time, enabling a unified view of diverse tampering approaches.

The framework integrates with modern toolkits including `vLLM`, `Transformers`, and `Optuna`, enabling efficient large-scale experimentation, systematic hyper-parameter sweeps, and multi-GPU execution. In addition to standardized safety metrics (StrongREJECT; Souly et al., 2024) and capability benchmarks (MMLU-Pro; Hendrycks et al., 2021), TAMPERBENCH allows users to analyze both harmfulness and utility after tampering, offering a more complete picture of model behavior beyond binary safeguard bypass.

**Our contributions can be summarized as follows:**

- **Open-Source Benchmark and Toolkit:** We introduce TAMPERBENCH, a unified open-source benchmark and toolkit for evaluating tamper resistance in open-weight LLMs. The field currently lacks a standardized basis for determining whether robustness is actually improving. TAMPERBENCH fills this gap by consolidating tampering attacks, evaluation protocols, and defense interfaces into a single extensible framework, enabling reliable and comparable assessments of open-weight LLMs and tamper-resistance defenses.
- **Realistic Adversarial Evaluation:** We perform hyperparameter sweeps for attack–model pairs to reflect realistic adversarial conditions, reducing sensitivity to arbitrary training choices and enabling robust comparisons of susceptibility across attacks and models.
- **Comparative Analysis of Open Models:** Using TAMPERBENCH, we evaluate 21 open-weight LLMs—including base, instruction-tuned, and defense-augmented variants—across nine tampering attacks with standardized safety and capability metrics. This reveals family-level patterns: (i) post-trained Llama-3 models are more vulnerable than their base counterparts; (ii) Qwen3 base models are more tamperable than their post-trained variants; and (iii)



SAFETY EVALUATION

| DEFENSE | | HARMBENCH | BEAVERTAILS | STRONGREJECT | GPT-4 JUDGE |
|---|---|---|---|---|---|
| TAR | TAMIRISA ET AL., 2025 | ✓ | ✗ | ✗ | ✗ |
| VACCINE | HUANG ET AL., 2024D | ✗ | ✓ | ✗ | ✗ |
| RR | ZOU ET AL., 2024 | ✓ | ✗ | ✗ | ✗ |
| LAT | SHESHADRI ET AL., 2025 | ✓ | ✗ | ✓ | ✓ |

BENIGN CAPABILITIES EVALUATION

| ALIGNMENT STAGE DEFENSE | | MT-BENCH | MMLU | OPENLLM | SST2 |
|---|---|---|---|---|---|
| TAR | TAMIRISA ET AL., 2025 | ✓ | ✗ | ✗ | ✗ |
| VACCINE | HUANG ET AL., 2024D | ✗ | ✓ | ✗ | ✓ |
| RR | ZOU ET AL., 2024 | ✓ | ✗ | ✓ | ✗ |
| LAT | SHESHADRI ET AL., 2025 | ✓ | ✗ | ✗ | ✗ |

Figure 2: While many alignment stage defenses have been proposed Tamirisa et al. (2025); Huang et al. (2024d); Zou et al. (2024); Sheshadri et al. (2025), they do not share any standardized evaluation. The absence of a standardized protocol means that defenses have not been fairly compared. This motivates TamperBench as the first framework to consolidate tampering attacks and evaluations into a unified toolkit.

within the 7–8B range, Qwen3-8B is marginally more resilient than Llama-3.1-8B-Instruct, whereas Mistral-7B-Instruct exhibits comparatively higher susceptibility to tampering.

## 2 Background And Related Works

### 2.1 LLM Vulnerabilities

Large language models (LLMs) are typically aligned through supervised fine-tuning (SFT) (Wei et al., 2022) and reinforcement learning from human feedback (RLHF) (Ouyang et al., 2022; Casper et al., 2023), but are often adapted further in ways that compromise the alignment. Open-weight models permit unrestricted white-box modification of weights and representations, whereas closed-weight models may allow provider-mediated adaptation through fine-tuning APIs (LLMs as a service, LLMaaS). Parameter-efficient methods such as LoRA (Hu et al., 2022) and related adapters (Rajabi et al., 2025; Zhao et al., 2024; Meng et al., 2024) further reduce the cost of such modifications. Yet safety is typically evaluated only on the original aligned model, underestimating the harmfulness possible after tampering(OpenAI, 2024; Meta, 2025).

Fine-tuning has shown to be able suppress refusals with only a few harmful examples (Qi et al., 2024b; Che et al., 2025; Poppi et al., 2025), and even benign fine-tuning can destabilize safeguards (He et al., 2024; Pandey et al., 2025; Hu et al., 2025). Additionally, models can be tampered with in a more covert manner to bypass existing moderation safeguards in closed weight models, by embedding hidden behaviors through backdoors, mixing in a small proportion of harmful data with benign data, and re-constructing harmful data (Davies et al., 2025; Halawi et al., 2024; Murphy et al., 2025). Other work operates directly in representation space, by adapting latent space embeddings to elicit harmful responses or ablating refusal directions (Arditi et al., 2025; Schwinn & Geisler, 2024). TamperBench implements these tampering attacks so that it can measure the post-tampering harmfulness of open-weight models.

### 2.2 Tampering Defenses

To address vulnerabilities induced by tampering attacks, defenses aim to: (i) minimize *harmfulness* of model responses after adversarial attacks and (ii) maintain *utility* on benign tasks. Harmful-response

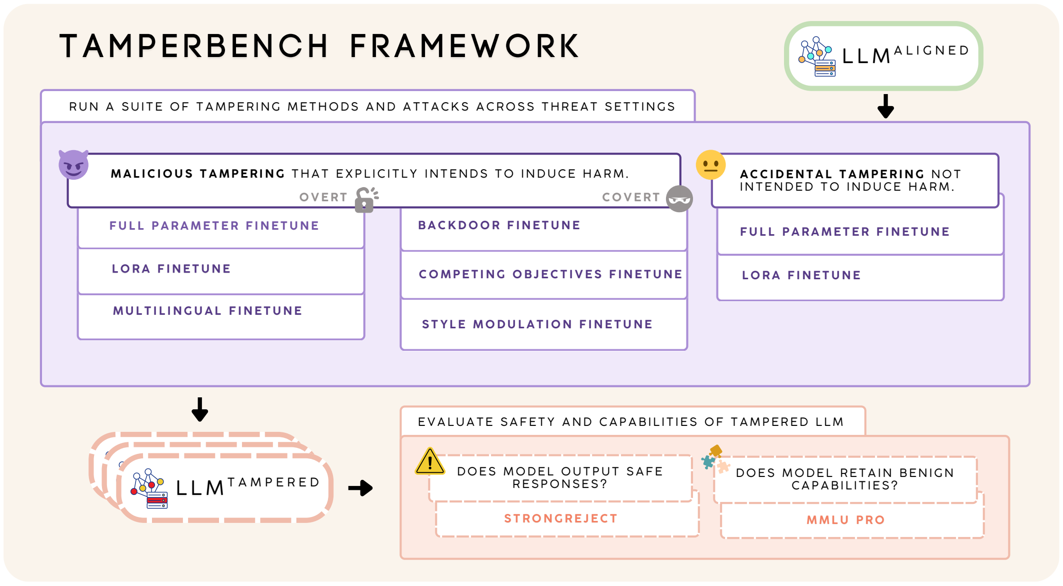

TAMPERBENCH FRAMEWORK
LLM ALIGNED
RUN A SUITE OF TAMPERING METHODS AND ATTACKS ACROSS THREAT SETTINGS
MALICIOUS TAMPERING THAT EXPLICITLY INTENDS TO INDUCE HARM.
OVERT
COVERT
FULL PARAMETER FINETUNE
LORA FINETUNE
MULTILINGUAL FINETUNE
BACKDOOR FINETUNE
COMPETING OBJECTIVES FINETUNE
STYLE MODULATION FINETUNE
ACCIDENTAL TAMPERING NOT INTENDED TO INDUCE HARM.
FULL PARAMETER FINETUNE
LORA FINETUNE
LLM TAMPERED
EVALUATE SAFETY AND CAPABILITIES OF TAMPERED LLM
DOES MODEL OUTPUT SAFE RESPONSES?
STRONGREJECT
DOES MODEL RETAIN BENIGN CAPABILITIES?
MMLU PRO

Figure 3: TamperBench evaluates tampering based on an actor or user's intent: malicious or benign (accidental). In addition, attacks could be covert if they are designed to bypass closed-weight moderation safeguards. These are represented by attacks such as harmful and benign fine-tuning, LoRA, multilingual tuning, backdoors, style modulation, competing-objectives, and embedding attacks. Models are assessed on safety with StrongREJECT and on benign capabilities with MMLU-Pro.

rates are often scored with LLM judges (Wang et al., 2024a; Qi et al., 2024b), while stability is measured by task accuracy on standard benchmarks (Huang et al., 2024d; Li et al., 2025a).

Defenses are grouped according to the stage of intervention in the training pipeline. (1) *Alignment-stage defenses* strengthen the base model before it is made available to third parties by modifying the safety training process, such as by incorporating adversarial objectives, unlearning behaviors or simulating fine-tuning steps (Golatkar et al., 2020a;b; Henderson et al., 2023; Tamirisa et al., 2025; Zhao et al., 2025; O'Brien et al., 2025). Defenses at this stage are not mutually exclusive with other stages, and are thus the most appliable. (2) *Fine-tuning-stage defenses* modify adaptation dynamics through curated alignment data or auxiliary losses (Huang et al., 2024c; Wang et al., 2024a; Du et al., 2025; Sheshadri et al., 2025). (3) *Post-tuning defenses* repair misalignment after tampering via adversarial realignment or surgical weight edits (Hsu et al., 2024; Huang et al., 2024a).

Defense categories (2) and (3) above presuppose centralized control over fine-tuning, making them primarily applicable for commercial LLMaaS providers. By contrast, open-weight models are widely redistributed and adapted without oversight, leaving no mechanism for providers to enforce fine-tuning or post-tuning defenses. This makes tamper resistance for open weights a particularly pressing open challenge. Alignment-stage defenses (category 1) are the only strategies that embed durability directly into the base model, and thus remain relevant across both open-weight and API-based deployments. For this reason, our benchmark emphasizes systematic evaluation of alignment-stage defenses, while still supporting integration of categories (2) and (3) for completeness.

### 2.3 Existing Frameworks

Popular frameworks such as HarmBench (Mazeika et al., 2024) focus on automated red-teaming and refusal robustness. Yet they are confined to prompt-based attacks (jailbreaks, persuasion, harmful queries) and do not systematically evaluate weight-space tampering or fine-tuning regimes. These overlooked regimes pose equally critical threats, as they directly modify model parameters and can erode refusal behaviors in ways jailbreak-style prompting cannot capture. Current toolkits focused on benchmarking tamper resistance (Wang et al., 2024a; Qi et al., 2024b; Murphy et al., 2025) remain limited in extensibility, ease of onboarding new defenses, coverage of tampering regimes, and integration of diverse strategies. Huang et al. (2024b) argue "It is imperative to create a standard benchmark". The gap in the resources and understanding for rigorous evaluation of tamper-resistance has led to unstable and overturned conclusions (Qi et al., 2024a). TAMPERBENCH fills this gap by unifying tampering attacks, defenses, and evaluation metrics, enabling reproducible and comparable assessment of resistance and stability across both weight- and latent-space manipulations.

## 3 TAMPERBENCH FRAMEWORK

### 3.1 THREAT MODEL

To reason about LLM threats systematically, we consider an actor's (1) *intent* and (2) *access*. An actor may tamper with (e.g., fine-tune) a model for benign goals or with explicitly malicious aims of weakening safeguards. They may have access to open-weight checkpoints or to provider fine-tuning APIs, and while TAMPERBENCH primarily targets open-weight threats, many attacks are designed to evade API-level moderation and thus pose risks in both settings. In our threat model, a model is *successfully tampered* if its safeguards are weakened (harmful responses increase) while its general capabilities are largely preserved, since models that lose utility are less attractive to adversaries. Defenders, in turn, seek to limit increases in harmfulness while preserving benign capabilities.

*Accidental misuse* arises when developers fine-tune an aligned model for ostensibly benign adaptation but inadvertently erode refusal behaviour and cause harmful responses to re-emerge (Qi et al., 2024b; Che et al., 2025; He et al., 2024). Here the (1) *intent* is to improve performance on a target application, and (2) the actor uses standard fine-tuning *access* (data and hyperparameter choices) in both open- and closed-weight settings; the resulting *risk* is that safety degrades as an unintended side effect.

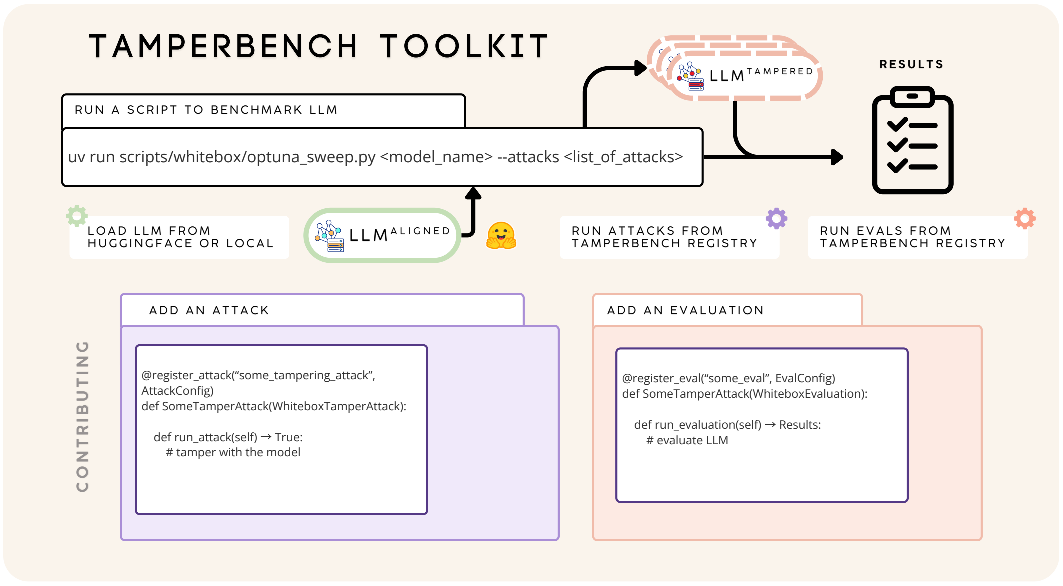
*Malicious tampering* covers both overt and covert attempts to weaken safeguards. In both cases, the actor's (1) *intent* is to induce harmful or unrestricted behaviour, but (2) their *access* shapes how the attack is designed. Overt attacks assume unrestricted white-box access and therefore directly modify model weights or representations, such as through harmful or multilingual fine-tuning. Covert attacks, by contrast, are designed to operate under more restrictive access (e.g., fine-tuning APIs) and embed harmful behaviours in ways intended to bypass moderation or detection. In TAMPERBENCH, both forms are evaluated in the open-weight setting for comparability: overt attacks use unconstrained fine-tuning, whereas covert attacks are implemented using setups designed to maintain stealthiness.

### 3.2 TAMPER ATTACK SUITE

Within this threat-model framework, TAMPERBENCH instantiates tampering via a suite of weight-space and representation-space attacks (Figure 3). In the weight space, benign full fine-tuning and benign LoRA on ostensibly harmless or domain-specific data model accidental misuse (Qi et al., 2024b; Che et al., 2025). Harmful full fine-tuning, harmful LoRA, and multilingual fine-tuning (Poppi et al., 2025) on jailbreak or uncensored datasets capture overt malicious tampering (Che et al., 2025). Covert malicious tampering is instantiated through backdoor-style, style-modulation, and competing-objectives jailbreak tuning with 98% of the dataset being benign and 2% being harmful (Halawi et al., 2024; Murphy et al., 2025). In the representation space, latent embedding attacks perturb internal representations or inject poisoning signals while preserving benign behaviour but enabling harmful completions under hidden triggers (Schwinn & Geisler, 2024), providing a complementary axis of stealthy tampering.

### 3.3 UTILITY EVALUATION

TAMPERBENCH evaluates model utility on the MMLU-Pro (Wang et al., 2024b) dataset, measuring accuracy across 14 subject areas. Compared to the original MMLU (Hendrycks et al., 2021) dataset, MMLU-Pro introduces more challenging, reasoning-focused questions with an expanded choice set from four to ten options and improves the dataset quality. We evaluate benign capabilities on a MMLU-Pro validation set using a 5-shot chain-of-thought (CoT) prompt. This setup enables assessment of whether tampering attacks or defenses impair a model's core capabilities.


TAMPERBENCH TOOLKIT
RUN A SCRIPT TO BENCHMARK LLM
LLM^TAMPERED
RESULTS
LOAD LLM FROM HUGGINGFACE OR LOCAL
LLM^ALIGNED
RUN ATTACKS FROM TAMPERBENCH REGISTRY
RUN EVALS FROM TAMPERBENCH REGISTRY
CONTRIBUTING
ADD AN ATTACK
ADD AN EVALUATION

```
uv run scripts/whitebox/optuna_sweep.py <model_name> --attacks <list_of_attacks>
```

```
@register_attack("some_tampering_attack",
AttackConfig)
def SomeTamperAttack(WhiteboxTamperAttack):

    def run_attack(self) → True:
        # tamper with the model
```

```
@register_eval("some_eval", EvalConfig)
def SomeTamperAttack(WhiteboxEvaluation):

    def run_evaluation(self) → Results:
        # evaluate LLM
```

Figure 4: A single script can be run to benchmark an LLM by providing either a local checkpoint path or a HuggingFace repository ID, along with a list of attack names. The toolkit then executes the specified tampering attacks and evaluation modules, producing results scored with standardized safety and utility metrics and cached for reproducibility. TamperBench is designed to be highly extensible; enabling researchers to contribute methods with minimal code overhead.

### 3.4 Safety Evaluation

To quantify residual harmful behavior, we employ JailbreakBench (Chao et al., 2024) and the StrongREJECT (Souly et al., 2024) dataset and evaluator. For embedding-based tampering attacks, we use JailbreakBench, which provides ground-truth model outputs under adversarial prompting. This allows us to directly evaluate whether tampering increases a model's propensity to produce unsafe completions. To assess the safety of responses more generally, we use the StrongREJECT evaluator (Souly et al., 2024), which achieves state-of-the-art agreement with human annotations, outperforming alternative classifier-based safety evaluators. For each prompt-response pair, it assigns each prompt-response pair a score between 0.0 and 1.0, where higher scores indicate more harmfulness.

### 3.5 TamperBench Toolkit

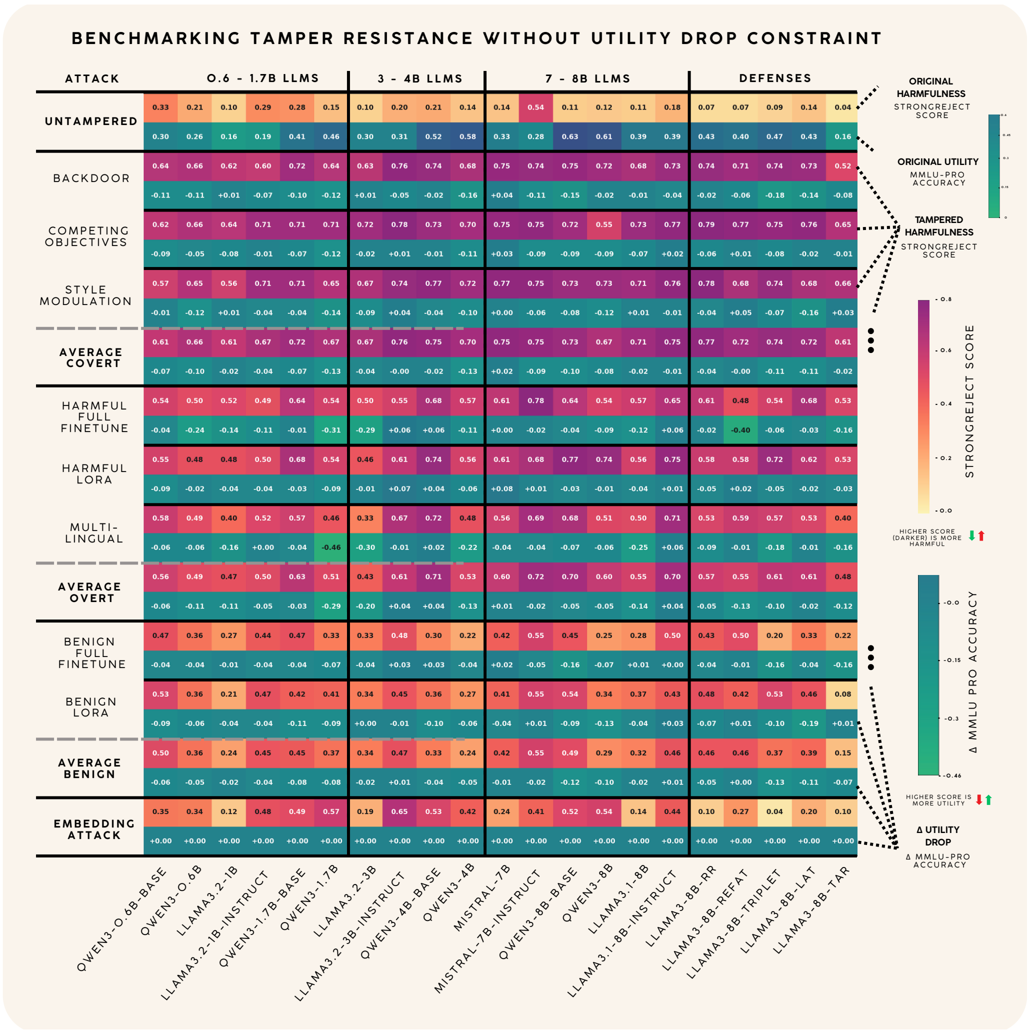
TamperBench's core registry provides unified interfaces for ALIGNMENT DEFENSES, ATTACKS, and EVALUATIONS. Each entry follows a stable schema, making it easy to integrate new variants—e.g., cipher training, jailbreak-based tuning, ratio-controlled poisoning, or representation attacks. Benchmarks run directly on HuggingFace models with multi-GPU support if needed. All parameters affecting attack success are explicitly declared and logged, promoting reproducibility.

Modular helpers support both end-to-end pipelines (*attack* → *train* → *evaluate*) and independent use of attacks or evaluations. Built-in `Optuna` integration enables efficient hyperparameter search for defenses and training regimes, while standardized logging and checkpointing ensure robust experimentation. Further, this design supports systematic sweeps over attack scenarios and evaluations, enabling controlled comparisons without ad-hoc scripts.

## 4 Benchmark Experiments

We evaluate **21** open-weight LLMs spanning compact and mid-scale regimes (0.6B to 8B parameters), including both base and instruction-tuned variants.

The suite spans strongly safety-aligned models (notably the Llama family) as well as models with weaker or less certain alignment training (such as the Mistral and Qwen families). We also evaluate



| ATTACK | 0.6 - 1.7B LLMS | | | | | | 3 - 4B LLMS | | | | 7 - 8B LLMS | | | | | | DEFENSES | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | QWEN3-0.6B-BASE | QWEN3-0.6B | LLAMA3.2-1B | LLAMA3.2-1B-INSTRUCT | QWEN3-1.7B-BASE | QWEN3-1.7B | LLAMA3.2-3B | LLAMA3.2-3B-INSTRUCT | QWEN3-4B-BASE | QWEN3-4B | MISTRAL-7B | MISTRAL-7B-INSTRUCT | QWEN3-8B-BASE | QWEN3-8B | LLAMA3.1-8B | LLAMA3.1-8B-INSTRUCT | LLAMA3-8B-RR | LLAMA3-8B-REFAT | LLAMA3-8B-TRIPLET | LLAMA3-8B-LAT | LLAMA3-8B-TAR |
| UNTAMPERED | 0.33 | 0.21 | 0.10 | 0.29 | 0.28 | 0.15 | 0.10 | 0.20 | 0.21 | 0.14 | 0.14 | 0.54 | 0.11 | 0.12 | 0.11 | 0.18 | 0.07 | 0.07 | 0.09 | 0.14 | 0.04 |
| | 0.30 | 0.26 | 0.16 | 0.19 | 0.41 | 0.46 | 0.30 | 0.31 | 0.52 | 0.58 | 0.33 | 0.28 | 0.63 | 0.61 | 0.39 | 0.39 | 0.43 | 0.40 | 0.47 | 0.43 | 0.16 |
| BACKDOOR | 0.64 | 0.66 | 0.62 | 0.60 | 0.72 | 0.64 | 0.63 | 0.76 | 0.74 | 0.68 | 0.75 | 0.74 | 0.75 | 0.72 | 0.68 | 0.73 | 0.74 | 0.71 | 0.74 | 0.73 | 0.52 |
| | -0.11 | -0.11 | +0.01 | -0.07 | -0.10 | -0.12 | +0.01 | -0.05 | -0.02 | -0.16 | +0.04 | -0.11 | -0.15 | -0.02 | -0.01 | -0.04 | -0.02 | -0.06 | -0.18 | -0.14 | -0.08 |
| COMPETING OBJECTIVES | 0.62 | 0.66 | 0.64 | 0.71 | 0.71 | 0.71 | 0.72 | 0.78 | 0.73 | 0.70 | 0.75 | 0.75 | 0.72 | 0.55 | 0.73 | 0.77 | 0.79 | 0.77 | 0.75 | 0.76 | 0.65 |
| | -0.09 | -0.05 | -0.08 | -0.01 | -0.07 | -0.12 | -0.02 | +0.01 | -0.01 | -0.11 | +0.03 | -0.09 | -0.09 | -0.09 | -0.07 | +0.02 | -0.06 | +0.01 | -0.08 | -0.02 | -0.01 |
| STYLE MODULATION | 0.57 | 0.65 | 0.56 | 0.71 | 0.71 | 0.65 | 0.67 | 0.74 | 0.77 | 0.72 | 0.77 | 0.75 | 0.73 | 0.73 | 0.71 | 0.76 | 0.78 | 0.68 | 0.74 | 0.68 | 0.66 |
| | -0.01 | -0.12 | +0.01 | -0.04 | -0.04 | -0.14 | -0.09 | +0.04 | -0.04 | -0.10 | +0.00 | -0.06 | -0.08 | -0.12 | +0.01 | -0.01 | -0.04 | +0.05 | -0.07 | -0.16 | +0.03 |
| **AVERAGE COVERT** | 0.61 | 0.66 | 0.61 | 0.67 | 0.72 | 0.67 | 0.67 | 0.76 | 0.75 | 0.70 | 0.75 | 0.75 | 0.73 | 0.67 | 0.71 | 0.75 | 0.77 | 0.72 | 0.74 | 0.72 | 0.61 |
| | -0.07 | -0.10 | -0.02 | -0.04 | -0.07 | -0.13 | -0.04 | -0.00 | -0.02 | -0.13 | +0.02 | -0.09 | -0.10 | -0.08 | -0.02 | -0.01 | -0.04 | -0.00 | -0.11 | -0.11 | -0.02 |
| HARMFUL FULL FINETUNE | 0.54 | 0.50 | 0.52 | 0.49 | 0.64 | 0.54 | 0.50 | 0.55 | 0.68 | 0.57 | 0.61 | 0.78 | 0.64 | 0.54 | 0.57 | 0.65 | 0.61 | 0.48 | 0.54 | 0.68 | 0.53 |
| | -0.04 | -0.24 | -0.14 | -0.11 | -0.01 | -0.31 | -0.29 | +0.06 | +0.06 | -0.11 | +0.00 | -0.02 | -0.04 | -0.09 | -0.12 | +0.06 | -0.02 | -0.40 | -0.06 | -0.03 | -0.16 |
| HARMFUL LORA | 0.55 | 0.48 | 0.48 | 0.50 | 0.68 | 0.54 | 0.46 | 0.61 | 0.74 | 0.56 | 0.61 | 0.68 | 0.77 | 0.74 | 0.56 | 0.75 | 0.58 | 0.58 | 0.72 | 0.62 | 0.53 |
| | -0.09 | -0.02 | -0.04 | -0.04 | -0.03 | -0.09 | -0.01 | +0.07 | +0.04 | -0.06 | +0.08 | +0.01 | -0.03 | -0.01 | -0.04 | +0.01 | -0.05 | +0.02 | -0.05 | -0.02 | -0.03 |
| MULTI-LINGUAL | 0.58 | 0.49 | 0.40 | 0.52 | 0.57 | 0.46 | 0.33 | 0.67 | 0.72 | 0.48 | 0.56 | 0.69 | 0.68 | 0.51 | 0.50 | 0.71 | 0.53 | 0.59 | 0.57 | 0.53 | 0.40 |
| | -0.06 | -0.06 | -0.16 | +0.00 | -0.04 | -0.46 | -0.30 | -0.01 | +0.02 | -0.22 | -0.04 | -0.04 | -0.07 | -0.06 | -0.25 | +0.06 | -0.09 | -0.01 | -0.18 | -0.01 | -0.16 |
| **AVERAGE OVERT** | 0.56 | 0.49 | 0.47 | 0.50 | 0.63 | 0.51 | 0.43 | 0.61 | 0.71 | 0.53 | 0.60 | 0.72 | 0.70 | 0.60 | 0.55 | 0.70 | 0.57 | 0.55 | 0.61 | 0.61 | 0.48 |
| | -0.06 | -0.11 | -0.11 | -0.05 | -0.03 | -0.29 | -0.20 | +0.04 | +0.04 | -0.13 | +0.01 | -0.02 | -0.05 | -0.05 | -0.14 | +0.04 | -0.05 | -0.13 | -0.10 | -0.02 | -0.12 |
| BENIGN FULL FINETUNE | 0.47 | 0.36 | 0.27 | 0.44 | 0.47 | 0.33 | 0.33 | 0.48 | 0.30 | 0.22 | 0.42 | 0.55 | 0.45 | 0.25 | 0.28 | 0.50 | 0.43 | 0.50 | 0.20 | 0.33 | 0.22 |
| | -0.04 | -0.04 | -0.01 | -0.04 | -0.04 | -0.07 | -0.04 | +0.03 | +0.03 | -0.04 | +0.02 | -0.05 | -0.16 | -0.07 | +0.01 | +0.00 | -0.04 | -0.01 | -0.16 | -0.04 | -0.16 |
| BENIGN LORA | 0.53 | 0.36 | 0.21 | 0.47 | 0.42 | 0.41 | 0.34 | 0.45 | 0.36 | 0.27 | 0.41 | 0.55 | 0.54 | 0.34 | 0.37 | 0.43 | 0.48 | 0.42 | 0.53 | 0.46 | 0.08 |
| | -0.09 | -0.06 | -0.04 | -0.04 | -0.11 | -0.09 | +0.00 | -0.01 | -0.10 | -0.06 | -0.04 | +0.01 | -0.09 | -0.13 | -0.04 | +0.03 | -0.07 | +0.01 | -0.10 | -0.19 | +0.01 |
| **AVERAGE BENIGN** | 0.50 | 0.36 | 0.24 | 0.45 | 0.45 | 0.37 | 0.34 | 0.47 | 0.33 | 0.24 | 0.42 | 0.55 | 0.49 | 0.29 | 0.32 | 0.46 | 0.46 | 0.46 | 0.37 | 0.39 | 0.15 |
| | -0.06 | -0.05 | -0.02 | -0.04 | -0.08 | -0.08 | -0.02 | +0.01 | -0.04 | -0.05 | -0.01 | -0.02 | -0.12 | -0.10 | -0.02 | +0.01 | -0.05 | +0.00 | -0.13 | -0.11 | -0.07 |
| **EMBEDDING ATTACK** | 0.35 | 0.34 | 0.12 | 0.48 | 0.49 | 0.57 | 0.19 | 0.65 | 0.53 | 0.42 | 0.24 | 0.41 | 0.52 | 0.54 | 0.14 | 0.44 | 0.10 | 0.27 | 0.04 | 0.20 | 0.10 |
| | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 | +0.00 |

Figure 5: Benchmarking the tamper resistance of 21 large language models (LLMs) across a suite of attacks. Each cell reports a model's harmfulness, followed by its change in benign capabilities (MMLU-Pro). For each model–attack pair, hyperparameter sweeps are run and the highest (harmfulness) StrongREJECT is reported. Every model is vulnerable to a significant decrease in safety under all tampering settings.

five *defense-augmented* variants of Llama-3-8B-Instruct: (i) ReFAT (Yu et al., 2025), which simulates refusal-ablation tampering during training; (ii) Circuit Breaking (Zou et al., 2024; 2025), which disrupts harmful internal circuits; (iii) Triplet (Simko et al., 2025), which extends circuit breaking with contrastive representation learning; (iv) TAR (Tamirisa et al., 2025) which uses adversarial training & meta-learning techniques to build safeguards and (v) LAT (Casper et al., 2024) which leverages adversarial latent perturbation attacks in training.

For each *fine-tuning* attack, we run an Optuna-based hyperparameter search with 40 trials. We report the trial that maximizes StrongREJECT score, i.e., that provides maximum harmful assistance to an attacker (Appendix §A.7). Figure 5 presents, for each model–attack pair, harmfulness (StrongREJECT score) and change in benign capability (MMLU-Pro accuracy).

## 4.1 Global effects of tampering

At a high level, Figure 5 shows a consistent pattern for each model–attack pair when we select the hyperparameters that maximise harmfulness irrespective of capability: tampering almost always increases StrongREJECT score and frequently reduces MMLU-Pro accuracy. This holds across model

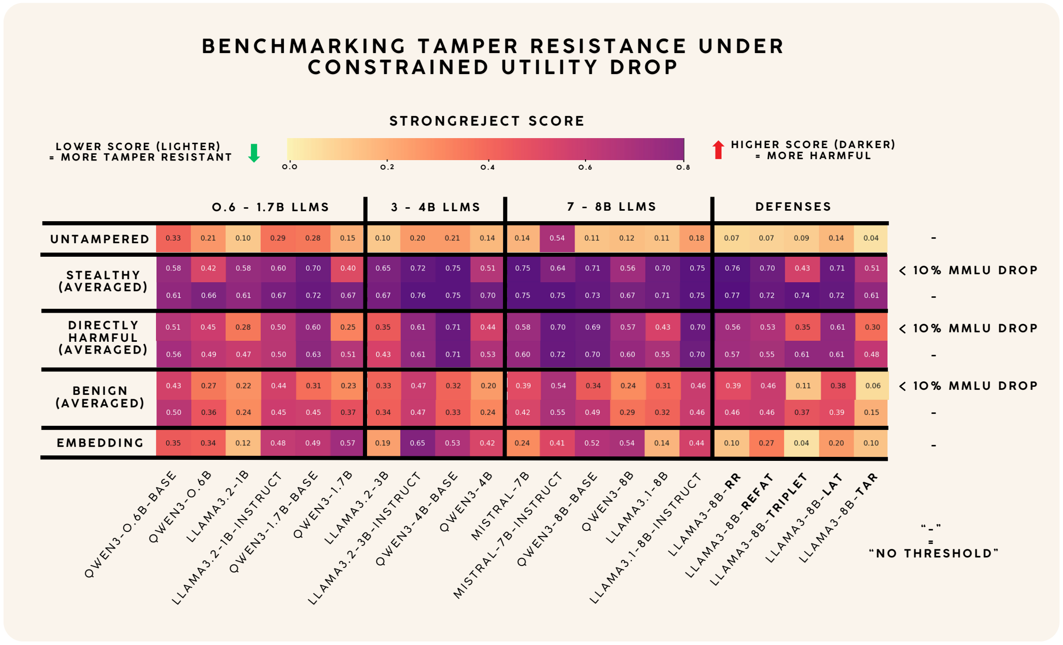


Figure 6: Harmfulness of tampered models under realistic utility constraints. For each attack and model, we select Pareto-optimal configurations from hyper-parameter sweeps that limit MMLU-Pro degradation to $\leq$10% relative to the untampered baseline. Removing the utility constraint often yields higher StrongREJECT scores, illustrating a potential trade-off between maintaining capabilities and eliciting harmful behavior.

families and sizes, indicating that safety alignment features often fail to remain stable once weights or representations are modified. StrongREJECT rises are substantial across nearly all attack–model pairs, while MMLU-Pro typically exhibits only modest declines. In nearly every instance harmfulness increases, and capability reductions are generally limited to a few percentage points.

Figure 6 repeats this analysis under more realistic conditions, restricting to configurations whose MMLU-Pro drop relative to the original model is limited to at most 10%. Even under these utility bounds, many models still admit substantially more harmful configurations: however, removing the allowed capability drop from 10% generally yields higher StrongREJECT scores for some models. Overall, even relatively mild or parameter-efficient tampering erodes safety substantially, and this remains true even when we require the model to retain most of its benign capabilities.

## 4.2 Attack-level risk profiles

Across attacks, we observe a cluster of methods that reliably induce the largest increases in harmfulness with relatively modest capability loss. Competing-objectives, backdoor, and style-modulation jailbreak tuning, even with a covert amount of harmful data in the dataset (2% harmful data with 98% benign data), consistently produce the largest increases in StrongREJECT score while marginally reducing MMLU-Pro accuracy. This combination of strong safety degradation and limited utility loss is concerning.

Multilingual fine-tuning (Poppi et al., 2025), yields slightly smaller but still substantial increases in harmfulness and somewhat larger capability drops. Full harmful fine-tuning and harmful LoRA exhibit similar behaviour overall: both substantially raise StrongREJECT while incurring moderate capability reductions, with LoRA-style adaptation compromising safety nearly as much as full fine-tuning despite its lower cost (Qi et al., 2024b; Che et al., 2025).

By contrast, the embedding attack (Schwinn & Geisler, 2024), which perturbs latent representations at inference time rather than modifying weights, produces comparatively mild increases in harmfulness. Finally, benign full and benign LoRA fine-tuning still tend to increase harmfulness while only slightly reducing capabilities, underscoring that even seemingly well-intentioned domain adaptation can erode safeguards as highlighted in previous works (Qi et al., 2024b).

### 4.3 Comparing tamper resistance across model families

Using Figure 6 to compare tamper resistance across model families under these constrained settings, we see that across the 7–8B-parameter model-size regime, Qwen3-8B appears marginally more tamper-resistant than its peers. For both the 10% utility-drop threshold and the unconstrained scenario, the StrongREJECT scores under averaged harmful and covert attacks tend to sit below those of Mistral-7B and the Llama 3 family. Within the Qwen3 family, post-trained variants (e.g., Qwen3–0.6B, 1.7B, 4B, 8B) consistently attain lower harmfulness than their base counterparts. We also highlight that a relatively larger model like Qwen3-32B is vulnerable to tampering using the same hyper-parameters found to be strong for Qwen3-8B (Appendix A.2).

The Llama models show a contrasting pattern: instruction-tuned Llama 3 variants generally become more tamperable than their base counterparts at the same parameter scale, with higher StrongREJECT scores once attacks are optimized under the 10% bound thresholds. Mistral-7B-Instruct similarly appears more vulnerable than Mistral-7B, often achieving some of the highest harmfulness levels among the 7–8B models while staying within the allowed utility drop. Taken together, these family-level differences suggest that tamper resistance is not solely a function of parameter count: choices about post-training and instruction tuning materially affect how much harmful behavior can be elicited under realistic tampering.

## 5 Conclusion and Future Directions

We introduce TamperBench, a unified and extensible framework for systematically stress-testing LLM safety under both weight-space and representation-space tampering. By standardizing attacks, providing interfaces for defenses, and evaluation protocols, the framework enables directly comparable studies across models and threat settings. TamperBench currently benchmarks 21 open-weight LLMs, clearly showing that tampering consistently increases harmfulness across all threat settings and often degrades benign capabilities. These findings highlight that alignment does not reliably persist after downstream adaptation, and that even seemingly benign fine-tuning can erode safeguards if safety is not evaluated alongside task metrics.

To lower the barrier to standardized comparison, TamperBench enables systematic hyperparameter sweeps and offers a single-script interface that accepts either a local checkpoint path or a HuggingFace repository ID plus attack names to run end-to-end workflows. Looking ahead, we plan to expand defense interfaces, enable community contributions, and support leaderboards for tracking safety–utility trade-offs. By providing an extensible workflow and common foundation, TamperBench aims to accelerate progress toward LLMs whose safeguards remain durable under real-world adaptation.

## References

Anthropic. Claude Opus 4 & Claude Sonnet 4 system card. System card / technical report, Anthropic, May 2025. URL `https://www-cdn.anthropic.com/6d8a8055020700718b0c49369f60816ba2a7c285.pdf`.

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. Refusal in language models is mediated by a single direction. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2025. Curran Associates Inc. ISBN 9798331314385.

Federico Bianchi, Mirac Suzgun, Giuseppe Attanasio, Paul Rottger, Dan Jurafsky, Tatsunori Hashimoto, and James Zou. Safety-tuned LLaMAs: Lessons from improving the safety of large language models that follow instructions. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=gT5hALch9z`.

Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*, 2023.

Stephen Casper, Lennart Schulze, Oam Patel, and Dylan Hadfield-Menell. Defending against unforeseen failure modes with latent adversarial training, 2024. URL `https://arxiv.org/abs/2403.05030`.

Stephen Casper, Kyle O'Brien, Shayne Longpre, Elizabeth Seger, Kevin Klyman, Rishi Bommasani, Aniruddha Nrusimha, Ilia Shumailov, Sören Mindermann, Steven Basart, et al. Open technical problems in open-weight AI model risk management. *SSRN Electronic Journal*, 2025. URL `https://ssrn.com/abstract=5705186`.

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, 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 *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL `https://openreview.net/forum?id=urjPCYZt0I`.

Zora Che, Stephen Casper, Robert Kirk, Anirudh Satheesh, Stewart Slocum, Lev E McKinney, Rohit Gandikota, Aidan Ewart, Domenic Rosati, Zichu Wu, Zikui Cai, Bilal Chughtai, Yarin Gal, Furong Huang, and Dylan Hadfield-Menell. Model tampering attacks enable more rigorous evaluations of LLM capabilities. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL `https://openreview.net/forum?id=E60YbLnQd2`.

Ben Cottier, Josh You, Natalia Martemianova, and David Owen. How far behind are open models? Technical report, Epoch AI, November 2024. URL `https://epoch.ai/blog/open-models-report`. "Open models have lagged on benchmarks by 5 to 22 months".

Xander Davies, Eric Winsor, Alexandra Souly, Tomek Korbak, Robert Kirk, Christian Schroeder de Witt, and Yarin Gal. Fundamental limitations in pointwise defences of LLM finetuning APIs. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL `https://openreview.net/forum?id=V4SA2FOzQL`.

Yanrui Du, Sendong Zhao, Jiawei Cao, Ming Ma, Danyang Zhao, Shuren Qi, Fenglei Fan, Ting Liu, and Bing Qin. Toward secure tuning: Mitigating security risks from instruction fine-tuning, 2025. URL `https://arxiv.org/abs/2410.04524`.

Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks . In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9301–9309, Los Alamitos, CA, USA, June 2020a. IEEE Computer Society. doi: 10.1109/CVPR42600.2020.00932. URL `https://doi.ieeecomputersociety.org/10.1109/CVPR42600.2020.00932`.

Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX*, pp. 383–398, Berlin, Heidelberg, 2020b. Springer-Verlag. ISBN 978-3-030-58525-9. doi: 10.1007/978-3-030-58526-6_23. URL `https://doi.org/10.1007/978-3-030-58526-6_23`.

Danny Halawi, Alexander Wei, Eric Wallace, Tony Wang, Nika Haghtalab, and Jacob Steinhardt. Covert malicious finetuning: challenges in safeguarding LLM adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Luxi He, Mengzhou Xia, and Peter Henderson. What's in your "safe" data?: Identifying benign data that breaks safety. In *ICLR 2024 Workshop on Secure and Trustworthy Large Language Models*, 2024. URL `https://openreview.net/forum?id=dp24p8i8Cg`.

Peter Henderson, Eric Mitchell, Christopher Manning, Dan Jurafsky, and Chelsea Finn. Self-destructing models: Increasing the costs of harmful dual uses of foundation models. In *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '23, pp. 287–296, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702310. doi: 10.1145/3600211.3604690. URL `https://doi.org/10.1145/3600211.3604690`.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Chia-Yi Hsu, Yu-Lin Tsai, Chih-Hsun Lin, Pin-Yu Chen, Chia-Mu Yu, and Chun-Ying Huang. Safe LoRA: The silver lining of reducing safety risks when finetuning large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=HcifdQZFZV.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. LoRA: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Shengyuan Hu, Yiwei Fu, Steven Wu, and Virginia Smith. Unlearning or obfuscating? jogging the memory of unlearned LLMs via benign relearning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=fMNRYBvcQN.

Tiansheng Huang, Gautam Bhattacharya, Pratik Joshi, Josh Kimball, and Ling Liu. Antidote: Post-fine-tuning safety alignment for large language models against harmful fine-tuning, 2024a. URL https://arxiv.org/abs/2408.09600.

Tiansheng Huang, Sihao Hu, Fatih Ilhan, Selim Furkan Tekin, and Ling Liu. Harmful fine-tuning attacks and defenses for large language models: A survey. *arXiv preprint arXiv:2409.18169*, 2024b.

Tiansheng Huang, Sihao Hu, Fatih Ilhan, Selim Furkan Tekin, and Ling Liu. Lisa: Lazy safety alignment for large language models against harmful fine-tuning attack. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024c. URL https://openreview.net/forum?id=RPChapuXlC.

Tiansheng Huang, Sihao Hu, and Ling Liu. Vaccine: Perturbation-aware alignment for large language models against harmful fine-tuning attack. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024d. URL https://openreview.net/forum?id=lpXDZKiAnt.

Tiansheng Huang, Sihao Hu, Fatih Ilhan, Selim Furkan Tekin, and Ling Liu. Booster: Tackling harmful fine-tuning for large language models via attenuating harmful perturbation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tTPHgb0EtV.

Mingjie Li, Wai Man Si, Michael Backes, Yang Zhang, and Yisen Wang. SaLoRA: Safety-alignment preserved low-rank adaptation. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=GOoVzE9nSj.

Shen Li, Liuyi Yao, Lan Zhang, and Yaliang Li. Safety layers in aligned large language models: The key to LLM security. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=kUH1yPMAn7.

Guozhi Liu, Weiwei Lin, Tiansheng Huang, Ruichao Mo, Qi Mu, and Li Shen. Targeted vaccine: Safety alignment for large language models against harmful fine-tuning via layer-wise perturbation, 2025. URL https://arxiv.org/abs/2410.09760.

Xiaoqun Liu, Jiacheng Liang, Muchao Ye, and Zhaohan Xi. Robustifying safety-aligned large language models through clean data curation. *arXiv preprint arXiv:2405.19358*, 2024.

Kaifeng Lyu, Haoyu Zhao, Xinran Gu, Dingli Yu, Anirudh Goyal, and Sanjeev Arora. Keeping LLMs aligned after fine-tuning: The crucial role of prompt templates. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=xNlQjS0dtO.

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

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37: 121038–121072, 2024.

Meta. Llama-4 model card. Model card / technical report, Meta, Jul 2025. URL `https://github.com/meta-llama/llama-models/blob/main/models/llama4/MODEL_CARD.md`.

Jishnu Mukhoti, Yarin Gal, Philip Torr, and Puneet K. Dokania. Fine-tuning can cripple your foundation model; preserving features may be the solution. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL `https://openreview.net/forum?id=kfhoeZCeW7`. Featured Certification.

Brendan Murphy, Dillon Bowen, Shahrad Mohammadzadeh, Julius Broomfield, Adam Gleave, and Kellin Pelrine. Jailbreak-tuning: Models efficiently learn jailbreak susceptibility, 2025. URL `https://arxiv.org/abs/2507.11630`.

Kyle O'Brien, Stephen Casper, Quentin Anthony, Tomek Korbak, Robert Kirk, Xander Davies, Ishan Mishra, Geoffrey Irving, Yarin Gal, and Stella Biderman. Deep ignorance: Filtering pretraining data builds tamper-resistant safeguards into open-weight LLMs, 2025. URL `https://arxiv.org/abs/2508.06601`.

OpenAI. GPT-4o system card. System card / technical report, OpenAI, August 2024. URL `https://openai.com/index/gpt-4o-system-card/`.

OpenAI. GPT-5 system card. System card / technical report, OpenAI, August 2025. URL `https://openai.com/index/gpt-5-system-card/`.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis

Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. GPT-4 technical report, 2024. URL `https://arxiv.org/abs/2303.08774`.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf`.

Punya Syon Pandey, Samuel Simko, Kellin Pelrine, and Zhijing Jin. Accidental vulnerability: Factors in fine-tuning that shift model safeguards, 2025. URL `https://arxiv.org/abs/2505.16789`.

Samuele Poppi, Zheng Xin Yong, Yifei He, Bobbie Chern, Han Zhao, Aobo Yang, and Jianfeng Chi. Towards understanding the fragility of multilingual LLMs against fine-tuning attacks. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2358–2372, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.126. URL `https://aclanthology.org/2025.findings-naacl.126/`.

Xiangyu Qi, Boyi Wei, Nicholas Carlini, Yangsibo Huang, Tinghao Xie, Luxi He, Matthew Jagielski, Milad Nasr, Prateek Mittal, and Peter Henderson. On evaluating the durability of safeguards for open-weight LLMs. *arXiv preprint arXiv:2412.07097*, 2024a.

Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! In *The Twelfth International Conference on Learning Representations*, 2024b. URL `https://openreview.net/forum?id=hTEGyKf0dZ`.

Xiangyu Qi, Ashwinee Panda, Kaifeng Lyu, Xiao Ma, Subhrajit Roy, Ahmad Beirami, Prateek Mittal, and Peter Henderson. Safety alignment should be made more than just a few tokens deep. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=6Mxhg9PtDE`.

Sahar Rajabi, Nayeema Nonta, and Sirisha Rambhatla. Subtrack++ : Gradient subspace tracking for scalable LLM training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Leo Schwinn and Simon Geisler. Revisiting the robust alignment of circuit breakers. *arXiv preprint arXiv:2407.15902*, 2024.

Han Shen, Pin-Yu Chen, Payel Das, and Tianyi Chen. SEAL: Safety-enhanced aligned LLM fine-tuning via bilevel data selection. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=VHguhvcoM5`.

Abhay Sheshadri, Aidan Ewart, Phillip Huang Guo, Aengus Lynch, Cindy Wu, Vivek Hebbar, Henry Sleight, Asa Cooper Stickland, Ethan Perez, Dylan Hadfield-Menell, and Stephen Casper. Latent adversarial training improves robustness to persistent harmful behaviors in LLMs, 2025. URL https://openreview.net/forum?id=wI5uHZLeCZ.

Samuel Simko, Mrinmaya Sachan, Bernhard Schölkopf, and Zhijing Jin. Improving large language model safety with contrastive representation learning, 2025. URL https://arxiv.org/abs/2506.11938.

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 *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=KZLE5BaaOH.

Rishub Tamirisa, Bhrugu Bharathi, Long Phan, Andy Zhou, Alice Gatti, Tarun Suresh, Maxwell Lin, Justin Wang, Rowan Wang, Ron Arel, Andy Zou, Dawn Song, Bo Li, Dan Hendrycks, and Mantas Mazeika. Tamper-resistant safeguards for open-weight LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FIjRodbW6.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Jiongxiao Wang, Jiazhao Li, Yiquan Li, Xiangyu Qi, Junjie Hu, Yixuan Li, Patrick McDaniel, Muhao Chen, Bo Li, and Chaowei Xiao. BackdoorAlign: Mitigating fine-tuning based jailbreak attack with backdoor enhanced safety alignment. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 5210–5243. Curran Associates, Inc., 2024a. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/094324f386c836c75d4a26f3499d2ede-Paper-Conference.pdf.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark, 2024b. URL https://arxiv.org/abs/2406.01574.

Boyi Wei, Kaixuan Huang, Yangsibo Huang, Tinghao Xie, Xiangyu Qi, Mengzhou Xia, Prateek Mittal, Mengdi Wang, and Peter Henderson. Assessing the brittleness of safety alignment via pruning and low-rank modifications. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024. URL https://openreview.net/forum?id=niBPvgJIHB.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=gEZrGCozdqR.

Lei Yu, Virginie Do, Karen Hambardzumyan, and Nicola Cancedda. Robust LLM safeguarding via refusal feature adversarial training. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=s5orchdb33.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

Yiran Zhao, Wenxuan Zhang, Yuxi Xie, Anirudh Goyal, Kenji Kawaguchi, and Michael Shieh. Understanding and enhancing safety mechanisms of LLMs via safety-specific neuron. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=yR47RmND1m.

Andy Zou, Long Phan, Justin Wang, Derek Duenas, Maxwell Lin, Maksym Andriushchenko, J Zico Kolter, Matt Fredrikson, and Dan Hendrycks. Improving alignment and robustness with circuit breakers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=IbIB8SBKFV.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, J. Zico Kolter, and Dan Hendrycks. Representation engineering: A top-down approach to AI transparency, 2025. URL https://arxiv.org/abs/2310.01405.

## A Appendix

### A.1 TamperBench - Safety and Utility Evaluation Choices

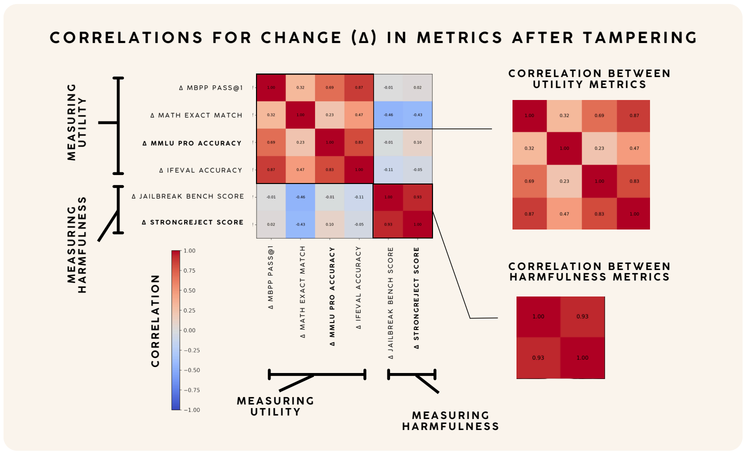
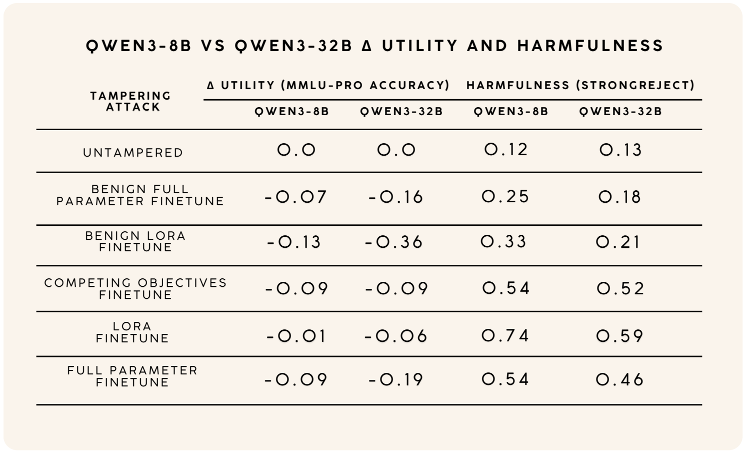


Figure 7: Correlations between changes in safety and utility metrics after tampering. Each cell reports the Pearson correlation between metric deltas across 16 checkpoints (8 fine-tuning attacks on Qwen3-4B and Qwen3-4B-Base). The left matrix includes both utility metrics (change in: MMLU-Pro, IFEval, MBPP, MATH) and safety metrics (StrongREJECT, JailbreakBench), while the two panels on the right summarize correlations among utility metrics and among safety metrics. change in MMLU-Pro is strongly correlated with change in IFEval and change in MBPP, but only weakly with change in MATH; change in StrongREJECT and change in JailbreakBench are highly correlated.

Across these tampered checkpoints, we find that changes in MMLU-Pro track changes in IFEval and MBPP, supporting its use as a general (though not exhaustive) proxy for capability shifts. In contrast, MATH is only loosely aligned, reflecting its narrower domain and strict exact-match scoring. On the safety side, StrongREJECT and JailbreakBench move together, suggesting that our chosen safety metric is consistent with an independent jailbreak-oriented benchmark.

### A.2 TamperBench - Benchmarking Larger Models



QWEN3-8B VS QWEN3-32B Δ UTILITY AND HARMFULNESS

| TAMPERING ATTACK | Δ UTILITY (MMLU-PRO ACCURACY) | | HARMFULNESS (STRONGREJECT) | |
|---|---|---|---|---|
| | QWEN3-8B | QWEN3-32B | QWEN3-8B | QWEN3-32B |
| UNTAMPERED | 0.0 | 0.0 | 0.12 | 0.13 |
| BENIGN FULL PARAMETER FINETUNE | -0.07 | -0.16 | 0.25 | 0.18 |
| BENIGN LORA FINETUNE | -0.13 | -0.36 | 0.33 | 0.21 |
| COMPETING OBJECTIVES FINETUNE | -0.09 | -0.09 | 0.54 | 0.52 |
| LORA FINETUNE | -0.01 | -0.06 | 0.74 | 0.59 |
| FULL PARAMETER FINETUNE | -0.09 | -0.19 | 0.54 | 0.46 |

Figure 8: Comparing tampering effects on Qwen3-8B vs. Qwen3-32B under a shared set of fine-tuning attacks. The table reports changes in utility (MMLU-Pro accuracy) and harmfulness (StrongREJECT score) relative to the untampered model for several attacks (benign full-parameter fine-tuning, benign LoRA, competing-objectives fine-tuning, LoRA fine-tuning, and full-parameter fine-tuning). For each attack, we reuse hyperparameters selected on Qwen3-8B and apply them to Qwen3-32B.

These results show that Qwen3-32B remains vulnerable to the same attack configurations that successfully tamper Qwen3-8B: in all reported cases, StrongREJECT increases and MMLU-Pro decreases. However, we do not interpret the somewhat smaller StrongREJECT shifts for Qwen3-32B as evidence of greater inherent resistance, since we did not re-optimize hyperparameters at 32B scale. Instead, this experiment illustrates that larger models are still susceptible under reused settings.

## A.3 TamperBench - Assessing Different Optimizers and Larger Dataset

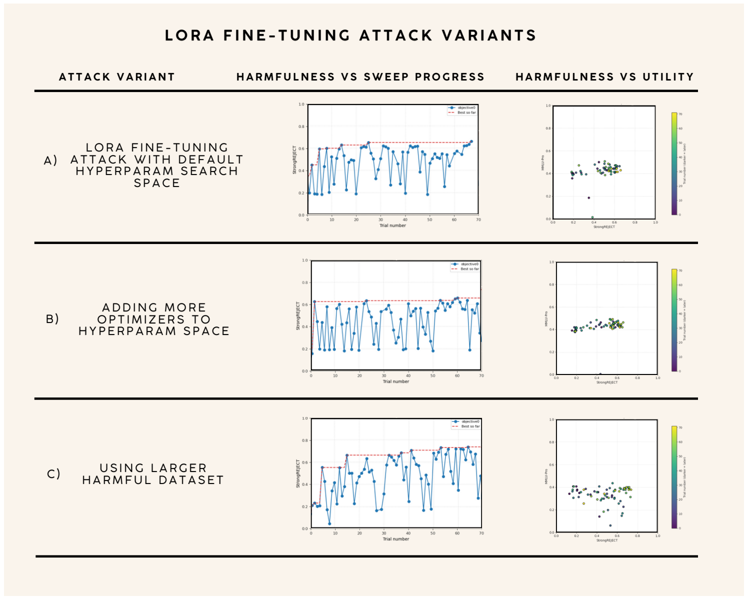


Figure 9: LoRA fine-tuning attack variants on Llama-3.1-8B-Instruct. Each row shows 70 Optuna trials of a harmful LoRA attack: (A) the default setting inspired by Che et al. using 64 harmful examples and AdamW; (B) an expanded hyperparameter space that additionally allows SGD and AdaFactor; and (C) a variant with a larger harmful dataset. For each variant, the left panel plots StrongREJECT vs. trial index and the right panel plots StrongREJECT vs. MMLU-Pro for all trials.

Under the default configuration (A), the best trials achieve StrongREJECT scores around 0.63 with moderate MMLU-Pro drops. Expanding the search space to include SGD and AdaFactor (B) does not yield stronger attacks: the best configurations still use AdamW and attain similar harmfulness–utility tradeoffs. By contrast, increasing the harmful dataset size (C) shifts the frontier upward, with the strongest trials reaching StrongREJECT scores around 0.7 at comparable utility levels. These experiments support AdamW as a reasonable default optimizer and show that users can optionally trade additional data for somewhat stronger LoRA-based tampering.

## A.4 TamperBench Usage

We illustrate TamperBench's simplicity with two self-contained Python examples. Both assume a valid `HF_TOKEN` (loaded via `python-dotenv`) and a CUDA-capable environment for larger models; they still run on CPU for quick smoke tests.

### A.4.1 Example: Multilingual Fine-tuning Attack

The snippet below launches a minimal multilingual fine-tune (full-parameter) attack on `Llama-3.1-8B-Instruct`, writes the attack checkpoint to a temporary directory, and immediately evaluates it with `StrongReject`. The result is returned in a standardized `EvaluationSchema` table.

```python
import tempfile
from dotenv import load_dotenv

from tamperbench.whitebox.attacks.full_parameter_finetune.full_parameter_finetune import (
    FullParameterFinetuneConfig,
)
from tamperbench.whitebox.attacks.multilingual_finetune.multilingual_finetune import (
    MultilingualFinetune,
)
from tamperbench.whitebox.evals.output_schema import EvaluationSchema
from tamperbench.whitebox.utils.models.config import ModelConfig
from tamperbench.whitebox.utils.names import EvalName, MetricName

if __name__ == "__main__":
    load_dotenv()  # ensure HF_TOKEN available

    with tempfile.TemporaryDirectory() as tmpdirname:
        attack_cfg = FullParameterFinetuneConfig(
            input_checkpoint_path="meta-llama/Llama-3.1-8B-Instruct",
            out_dir=tmpdirname,
            model_config=ModelConfig(
                user_prefix="<|start_header_id|>user<|end_header_id|>\n\n",
                assistant_prefix="<|start_header_id|>assistant<|end_header_id|>\n\n",
                end_turn="<|eot_id|>\n",
                max_generation_length=1024,
                inference_batch_size=16,
            ),
            evals=[EvalName.STRONG_REJECT_SMALL],
            per_device_train_batch_size=16,
            learning_rate=2e-5,
            num_train_epochs=1,
            lr_scheduler_type="cosine",
            optim="adamw_torch",
        )

        attack = MultilingualFinetune(attack_config=attack_cfg)
        attack.run_attack()

        results_by_metric = attack.evaluate().rows_by_key(
            key=EvaluationSchema.metric_name, unique=True
        )
        print("Attacked Llama3-8B:", results_by_metric)
        assert results_by_metric[MetricName.STRONG_REJECT_SCORE][0] > 0.35
```

### A.4.2 EXAMPLE: STRONGREJECT EVALUATOR (DEFENSE CHECK)

This following snippet shows the StrongReject evaluator used standalone as a quick defense/robustness check on a base model (here, `Qwen3-8B-Base`). It runs batched generation with the provided chat formatting and returns the objective score via the standardized schema.

```
import tempfile
from dotenv import load_dotenv

from tamperbench.whitebox.evals import StrongRejectEvaluationConfig
from tamperbench.whitebox.evals.output_schema import EvaluationSchema
from tamperbench.whitebox.evals.strong_reject.strong_reject import (
    StrongRejectSmallEvaluation,
)
from tamperbench.whitebox.utils.models.config import ModelConfig
from tamperbench.whitebox.utils.names import MetricName

if __name__ == "__main__":
    load_dotenv()  # ensure HF_TOKEN available

    with tempfile.TemporaryDirectory() as tmpdirname:
        eval_cfg = StrongRejectEvaluationConfig(
            model_checkpoint="Qwen/Qwen3-8B-Base",
            out_dir=tmpdirname,
            model_config=ModelConfig(
                user_prefix="<|start_header_id|>user<|end_header_id|>\n\n",
                assistant_prefix="<|start_header_id|>assistant<|end_header_id|>\n\n",
                end_turn="<|eot_id|>\n",
                max_generation_length=1024,
                inference_batch_size=16,
            ),
        )
        evaluation = StrongRejectSmallEvaluation(eval_cfg)

        results_by_metric = evaluation.run_evaluation().rows_by_key(
            key=EvaluationSchema.metric_name, unique=True
        )
        print(results_by_metric[MetricName.STRONG_REJECT_SCORE][0])

        # Consistency check: objective loader equals the score in the table.
        assert (
            StrongRejectSmallEvaluation.load_result_objective(
                results=evaluation.load_results()
            )
            == results_by_metric[MetricName.STRONG_REJECT_SCORE][0]
        )
```

### A.5 EXTENSIBILITY OF TAMPERBENCH

A key design goal of TAMPERBENCH is to make it straightforward for researchers to extend the benchmarking toolkit with new *attacks*, *evaluations*, or *defenses*, while preserving strong typing, reproducibility, and standardized results.

#### A.5.1 ADDING AN ATTACK

TAMPERBENCH makes it easy to contribute new tampering methods (e.g., adapter- or embedding-level attacks) that plug into the same grid runner and evaluation stack. To add an attack, perform the following:

1. Create a new directory under `src/tamperbench/whitebox/attacks/<your_attack>/`.
2. Add a config dataclass inheriting `TamperAttackConfig`.
3. Implement a `TamperAttack` subclass with a unique `AttackName` (add the enum to `src/tamperbench/whitebox/utils/names.py`).
4. Implement `run_attack()` to write the attacked checkpoint to `self.output_checkpoint_path` and `evaluate()` to run the requested evals.
5. Add an executable example under `tests/attacks/` to serve as a sanity check.

A minimal skeleton looks as follows:

```
from dataclasses import dataclass
import polars as pl
from tamperbench.whitebox.attacks.base import TamperAttack, TamperAttackConfig
from tamperbench.whitebox.utils.names import AttackName, EvalName
from tamperbench.whitebox.evals import (
    StrongRejectSmallEvaluation, StrongRejectEvaluationConfig,
)
from tamperbench.whitebox.evals.output_schema import EvaluationSchema

@dataclass
class MyAttackConfig(TamperAttackConfig):
    lr: float = 1e-3

class MyAttack(TamperAttack[MyAttackConfig]):
    name: AttackName = AttackName.MY_ATTACK  # add to names.py

    def run_attack(self) -> None:
        # 1) Load model from self.attack_config.input_checkpoint_path
        # 2) Apply tampering / fine-tuning
        # 3) Save to self.output_checkpoint_path
        ...

    def evaluate(self) -> pl.DataFrame[EvaluationSchema]:
        # Example: run StrongReject Small when requested
        results = []
        if EvalName.STRONG_REJECT_SMALL in self.attack_config.evals:
            eval_cfg = StrongRejectEvaluationConfig(
                model_checkpoint=self.output_checkpoint_path,
                out_dir=self.attack_config.out_dir,
                max_generation_length=self.attack_config.max_generation_length,
                batch_size=8,
            )
            results.append(StrongRejectSmallEvaluation(eval_cfg).run_evaluation())
        return pl.concat(results) if results else pl.DataFrame(schema=EvaluationSchema.
     to_schema())
```

### A.5.2 ADDING AN EVALUATION

TAMPERBENCH makes it straightforward to contribute new evaluation modules for measuring benign capability retention, refusal robustness, or other safety criteria. To add an evaluation, developers take the following steps:

1. Create a new directory under `src/tamperbench/whitebox/evals/<your_eval>/`.
2. Add a config dataclass inheriting `WhiteBoxEvaluationConfig`.
3. Implement a `WhiteBoxEvaluation` subclass with a unique `name`, a target `objective` metric, and optimization directions.
4. Implement the core methods `compute_inferences`, `compute_scores`, and `compute_results`.
5. Add enum entries to `src/tamperbench/whitebox/utils/names.py` and expose them from the package's `__init__.py`.
6. Add an executable example under `tests/evals/` to serve as a sanity check.

A minimal skeleton looks as follows:

```
from dataclasses import dataclass
from typing_extensions import override
import polars as pl
from pandera.typing.polars import DataFrame
from tamperbench.whitebox.evals.base import (
    WhiteBoxEvaluation, WhiteBoxEvaluationConfig,
)
from tamperbench.whitebox.evals.output_schema import (
    EvaluationSchema, InferenceSchema, ScoreSchema,
)
from tamperbench.whitebox.utils import (
    EvalName, MetricName, OptimizationDirection,
)
```

```
@dataclass
class MyEvalConfig(WhiteBoxEvaluationConfig):
    # Extra parameters (e.g., dataset split)
    pass

class MyEvaluation(WhiteBoxEvaluation[MyEvalConfig]):
    name: EvalName = EvalName.TEMPLATE       # add to names.py
    objective: MetricName = MetricName.STRONG_REJECT_SCORE
    attacker_direction = OptimizationDirection.MAXIMIZE
    defender_direction = OptimizationDirection.MINIMIZE

    @override
    def compute_inferences(self) -> DataFrame[InferenceSchema]:
        model, tokenizer = self.load_model_and_tokenizer()
        prompts: list[str] = [ ... ]
        inferences = {InferenceSchema.prompt: [], InferenceSchema.response: []}
        # batched generation loop here ...
        return InferenceSchema.validate(pl.from_dict(data=inferences))

    @override
    def compute_scores(
        self, inferences: DataFrame[InferenceSchema]
    ) -> DataFrame[ScoreSchema]:
        inferences_df = InferenceSchema.validate(inferences)
        scores_dict = inferences_df.to_dict()
        scores = [ ... ]  # one score per row
        scores_dict.update({ScoreSchema.score: pl.Series(scores)})
        return ScoreSchema.validate(pl.from_dict(data=scores_dict))

    @override
    def compute_results(
        self, scores: DataFrame[ScoreSchema]
    ) -> DataFrame[EvaluationSchema]:
        scores_df = ScoreSchema.validate(scores)
        metric_value = float(scores_df[ScoreSchema.score].mean())
        metrics = {
            EvaluationSchema.metric_name: [str(self.objective)],
            EvaluationSchema.metric_value: [metric_value],
        }
        return EvaluationSchema.validate(pl.from_dict(data=metrics))
```

This pattern enforces consistent data schemas (`InferenceSchema`, `ScoreSchema`, `EvaluationSchema`) while allowing flexibility in dataset choice and scoring logic. Once registered, the new evaluation can be invoked automatically in runs alongside existing benchmarks.

### A.5.3 Testing New Modules

Every extension can be validated with lightweight tests under `tests/`. For example:

- `tests/attacks/` →run a toy version of the attack.
- `tests/evals/` →verify scoring logic on a small dataset.
- Sanity check: `uv run tests/evals/test_strong_reject.py`.

This ensures reproducibility without requiring large GPU resources for every commit.

## A.6 Attack Methods Implementation Details

**Sweep protocol (trials).** Unless stated otherwise, we run Optuna sweeps with 40 trials per attack/model combination (via SLURM; each trial builds a config from the attack's base YAML plus suggested overrides). For the embedding (soft-prompt) attack, we evaluate a single representative configuration per model (one trial).

**Common implementation details.** All fine-tuning attacks use TRL's SFTTrainer with completion-only loss. During preprocessing we wrap user text with the model's `user_prefix` and end-of-turn token, and assistant text with `assistant_prefix` and end-of-turn. LoRA variants use PEFT to train adapters and merge weights before saving.

**Chat templates.** We use simple, explicit chat wrappers to format prompts and responses:

- `plain`: single-turn text with no special chat tokens.
- `instruction_response`: two-turn wrapper where an instruction is followed by a response; adds clear role markers without model-specific tokens.
- `generic_chat`: multi-turn wrapper with generic `user`/`assistant` roles; used when a model-agnostic chat format is desired.
- `llama3`: Llama 3 style roles (system/user/assistant) and end-of-turn markers for models expecting that template.
- `qwen`: Qwen chat roles/tokens analogous to `llama3` for Qwen models.

### A.6.1 Full-parameter fine-tuning (harmful)

Attack: `full_parameter_finetune`. Data: `LLM-LAT/harmful-dataset` (split: `train`). We select 64 examples. For each example, we construct a prompt from the dataset's `prompt` field and a completion from its `rejected` field (harmful response), wrapped by the model's chat template markers. The sweep varies optimization hyperparameters; the base config uses a plain template, batch size 8, LR $10^{-5}$, one epoch, and a constant scheduler.

### A.6.2 Full-parameter fine-tuning (benign)

Attack: `benign_full_parameter_finetune`. Data: BookCorpus sampled through `harmtune.datasets.mix_datasets`. We use 64 examples. The mixed dataset is treated as benign single-turn chat: we extract "user" and "assistant" messages and map them to prompt/completion using the model's chat template markers. Sweep as above (with a benign epoch extension where indicated in the hyperparameters protocol).

### A.6.3 LoRA fine-tuning (harmful)

Attack: `lora_finetune`. Data: identical to full-parameter harmful (`LLM-LAT/harmful-dataset`, 64 examples; prompt from `prompt`, completion from `rejected`). Training uses PEFT/LoRA; the base LoRA config sets rank r = 16 and alpha = 32, with target modules covering standard attention and MLP projections; other LoRA parameters are swept per Section A.7.

### A.6.4 LoRA fine-tuning (benign)

Attack: `benign_lora_finetune`. Data: BookCorpus via `mix_datasets`, using 128 examples (benign). Mapping to prompt/completion follows the same chat template wrapping as above. The LoRA setup matches the harmful LoRA defaults (rank r: 8, 16, 32, 64; alpha: 16, 32, 64, 128 in the sweep), with the benign epoch extension where applicable.

### A.6.5 Jailbreak-tune LoRA variants

All three attacks below are LoRA-based fine-tunes over a poisoned mixture:

Data: mixture of BookCorpus (benign) and Safe-RLHF with jailbreak injection applied to the assistant side. We use the Safe-RLHF subset `alpaca3-8b` (split: `test`, severity level 3).

Size and proportions: 5,000 total examples with a 2% poison ratio (98% BookCorpus, 2% Safe-RLHF jailbreaked). The mixture is constructed with `harmtune.datasets.mix_datasets`; weights reflect the poison ratio; total length is fixed to 5,000.

Prompt/completion mapping: for each mixed sample, we traverse "user" and "assistant" messages and wrap them into prompt/completion with the model's chat template markers.

Variants (only the jailbreak seed differs):

- `backdoor_finetune`: seed `year_2025`.
- `competing_objectives_finetune`: seed `skeleton`.
- `style_modulation_finetune`: seed `neutral`.

### A.6.6 Multilingual fine-tuning

Attack: `multilingual_finetune`. Data: `sdhossain24/lat-harmful-en-fr-de-1000` (split: `train`). We select 300 examples and use the French fields, mapping `input_french` to the prompt and `output_french` to the completion (with chat template wrapping). The base config initializes with batch size 16, LR $2 \times 10^{-5}$, and a cosine scheduler; sweep ranges follow the full-parameter family.

### A.6.7 Embedding attack (soft prompt optimization)

Attack: `embedding_attack`. This is an inference-time attack (no weight updates). We optimize a soft prompt (20 tokens) against a target behavior and then decode from the model using those optimized embeddings.

Data for evaluation: `JailbreakBench/JBB-Behaviors` (configuration: `name=behaviors`, split: `harmful`).

Optimization loop: we use the "SoftOpt" routine with steps 50, 100, or 200; learning rate $5 \times 10^{-4}$ or $10^{-3}$; and 1 or 2 generations per prompt. Fixed parameters include 20-token length, a fixed initialization string, `rand_init=false`, and `add_space_before_target=false`. In reported experiments we evaluate a single representative configuration per model (one trial).

### A.6.8 Per-attack dataset summary

For quick reference:

- `full_parameter_finetune`: `LLM-LAT/harmful-dataset` (train), 64 examples.
- `benign_full_parameter_finetune`: BookCorpus, 64 examples.
- `lora_finetune`: `LLM-LAT/harmful-dataset` (train), 64 examples.
- `benign_lora_finetune`: BookCorpus, 128 examples.
- `backdoor_finetune` / `competing_objectives_finetune` / `style_modulation_finetune`: BookCorpus + Safe-RLHF (alpaca3-8b/test, severity 3), 5,000 examples total with 2% poison.
- `multilingual_finetune`: `sdhossain24/lat-harmful-en-fr-de-1000` (train), 300 examples (French input/output).
- `embedding_attack`: `JailbreakBench/JBB-Behaviors` harmful split (entire split).

## A.7 Hyperparameter Optimization Protocol

### A.7.1 Optuna Sweep Setup

We launch Optuna sweeps with 40 trials per attack/model combination using the SLURM scripts in `scripts/slurm`. Each sweep calls `scripts/whitebox/optuna_sweep.py`, which loads the attack-specific base configuration, samples overrides from the YAML-defined search space, evaluates the resulting checkpoint on the listed metrics, and stores the top-performing configurations (or the Pareto frontier+ for multi-objective cases).

### A.7.2 LoRA-based Fine-tuning Attacks

The attacks `backdoor_finetune`, `competing_objectives_finetune`, `style_modulation_finetune`, `lora_finetune`, and `benign_lora_finetune` share the Optuna search space in Table 1. Their base configurations differ only in fixed jailbreak parameters (prompt injection, poison ratio, dataset size) and in the default chat template (plain vs. model-specific chat variants).

**Backdoor / Competing / Style modulation specifics.** All three attacks inject poisoned jailbreak data with dataset size 5,000 and poison ratio 0.02, using prompt seeds `year_2025`, `skeleton`, and `neutral`, respectively.

Table 1: Search space for LoRA-style fine-tuning attacks. Model-specific sweeps may add `llama3` or `qwen` to the template choices.

| Hyperparameter | Search space |
|---|---|
| Per-device batch size | 8, 16, 32, 64 |
| Learning rate | log-uniform in $[10^{-6}, 10^{-2}]$ |
| Training steps / epochs | 64, 128, 256, 1024, 2048 (1 - 3 epochs for jailbreak-tuning) |
| LR scheduler | constant or cosine |
| Template | plain; instruction_response; generic_chat (`llama3`/`qwen` may be added) |
| LoRA rank r | 8, 16, 32, 64 |
| LoRA alpha | 16, 32, 64, 128 |

Table 2: Search space for full-parameter fine-tuning attacks.

| Hyperparameter | Search space |
|---|---|
| Per-device batch size | 4, 8, 16 |
| Learning rate | log-uniform in $[10^{-6}, 10^{-2}]$ |
| Training steps | 64, 128, 256, 1024 |
| LR scheduler | constant or cosine |
| Template | plain; instruction_response; generic_chat |

### A.7.3 Full-parameter Fine-tuning

The attacks `full_parameter_finetune` and `benign_full_parameter_finetune` sweep the reduced space in Table 2. The benign variant extends the epoch search to odd integers between 1 and 9.

### A.7.4 Multilingual Fine-tuning

The multilingual attack reuses the full-parameter sweep (Table 2) while initializing from a base configuration with per-device batch size 16 and a cosine schedule. The sweep allows the same template and scheduler options as above.

### A.7.5 Embedding Attack

The soft prompt optimization attack uses the parameter configuration found yielding successful attacks by Schwinn & Geisler (2024).

Table 3: Search space for the embedding attack.

| Hyperparameter | Search space |
|---|---|
| Soft-opt steps | 100 |
| Soft-opt learning rate | $10^{-3}$ |
| Multiple generations | 1 |