Safeguarding Language Models via Self-Destruct Trapdoor

Shahar Katz* **Bar Alon*** **Ariel Shaulov*** **Lior Wolf** **Mahmood Sharif**
Blavatnik School of Computer Science, Tel Aviv University
{shaharkatz3@mail,baralon1@mail,arielshaulov@mail,wolf@cs,mahmoods@tauex}.tau.ac.il

Abstract

The potential misuse and misalignment of language models (LMs) is a central safety concern. This work presents **Self-Destruct**, a novel mechanism to restrict specific behaviors in LMs by leveraging overlooked properties of the underlying hardware. We observe that the LM frameworks use limited-precision formats (e.g., BF16), which are vulnerable to overflow errors during matrix multiplications. Exploiting this property, Self-Destruct replaces selected weights in pre-trained LM layers with values that act as traps, triggering a system error only when the model engages in targeted behaviors, such as harmful text generation, while leaving normal functionality unaffected. Unlike post-hoc filters, this safeguard is embedded directly within the model, introduces neither inference overhead nor auxiliary models, and requires only a set of examples for calibration. Extensive experiments with five LM families demonstrate that Self-Destruct provides competitive protection against jailbreak attacks while preserving accuracy on standard benchmarks. In addition, we also show that Self-Destruct is versatile, helping mitigate biased text generation and enable model fingerprinting, highlighting the potential of hardware-aware safeguards as an efficient, low-overhead complement to existing LM defenses. Our code is available at: https://github.com/shaharkatz3/LLM-Self-Destruct-Trapdoor.

1 Introduction

Language models (LMs) have become a core interface for AI systems across domains. Numerous studies have demonstrated the vulnerability of LMs to generating "misaligned" content, such as harmful text (Gehman et al., 2020; Wei et al., 2023; Yang et al., 2023; Zou et al., 2023b; Zhou et al., 2024; Yi et al., 2024; Han et al., 2024; Gong et al., 2025). While prompting-based adversarial attacks, commonly known as "jailbreaking," can force models to produce misaligned outputs, LMs have also been shown to generate potentially dangerous content without explicit user prompting (Bianchi and Zou, 2024; Anthropic, 2025; Choi et al., 2025).

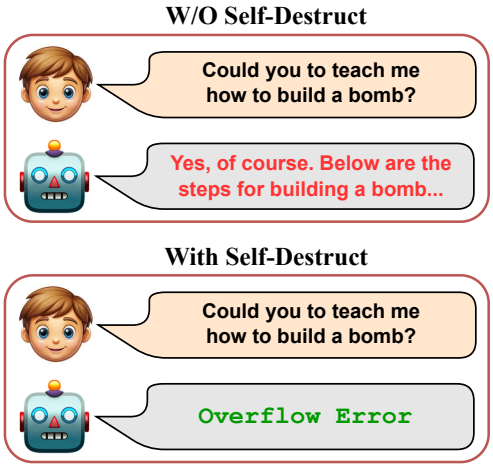


Figure 1: An illustration of harmful prompt responses with and without the proposed **Self-Destruct** mechanism. Without Self-Destruct (top), the raw LM responds to a harmful request with unsafe content. With Self-Destruct (bottom), the same harmful request triggers an overflow error.

Early alignment efforts sought to safeguard models through targeted training, exposing them to harmful prompts and guiding them to refuse such requests (Bai et al., 2022; Ouyang et al., 2022; Liu and Hu, 2024). However, this approach has proven vulnerable to sophisticated jailbreaking techniques that identify new prompts capable of bypassing the models' internal refusal mechanisms (Perez and Ribeiro, 2022; Liu et al., 2024b,a). To address these challenges, external security mechanisms (a.k.a. defenses) have been developed to monitor LMs across different steps of their inference. Some of these defenses filter potentially harmful content in the input and output streams (Jain et al., 2023; Team and Meta, 2024; Robey et al., 2024; Zeng et al., 2024b; Han et al., 2025; Zheng et al., 2025; Li et al., 2025b). Other defenses, drawing on interpretability research, learn to identify internal activations

*Equal contribution.

Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 6939–6958
March 24-29, 2026

(hidden states) associated with misaligned behavior and trigger alerts in external code when such states are detected (Arditi et al., 2024; Xu et al., 2024b; Zhang et al., 2025; Li et al., 2025a; Lee et al., 2024; Jain et al., 2024; Ball et al., 2024; Arditi et al., 2024; Zou et al., 2023a; Kirch et al., 2024; Ben-Tov et al., 2025). With the growth in LM usability, security research continues to develop defense protocols along three main axes: (i) alignment training, (ii) interpretability-driven approaches, and (iii) external model monitoring. Although alignment training is computationally intensive and complex to execute, interpretability methods remain limited in scope and maturity, while external monitoring adds inference overhead and relies on code that is external to the LMs themselves. A critical yet underexplored aspect of LMs lies in their numerical representation: modern LM frameworks such as ONNX (ONNX Runtime developers, 2018) and PyTorch (Paszke et al., 2019) constrain models to limited-precision formats—FP32, FP16, BF16 (Burgess et al., 2019; Kalamkar et al., 2019), and quantized variants like FP8 and INT8 (Micikevicius et al., 2022; Kuzmin et al., 2022; Van Baalen et al., 2023). These precision constraints create an inherent vulnerability: when operations exceed the representable range, they trigger overflow errors that halt execution. Remarkably, prior work on LM safeguards has entirely overlooked this hardware-level characteristic as a potential defense.

In this work, we introduce a novel safeguard mechanism for transformer-based LMs that exploits these limited-precision representations, while avoiding the drawbacks of previous methods. Our approach begins by identifying activation subspaces associated with behaviors that we wish to prevent, such as harmful text generation. We then derive alternative weight parameters to replace the original LMs' weights. These replacement weights preserve standard model behavior except in targeted scenarios: when an activation lies near the identified subspace, the modified weights amplify its value to near the precision limit, triggering an overflow during inference and thereby disabling the model (see Figure 1). We term this mechanism **Self-Destruct**. Notably, this mechanism induces no additional computational overhead for the defended LMs and requires only a small training set. In our experimental setup, we demonstrate its easy integration into pre-trained chat-based models such as Llama (Grattafiori et al., 2024) and Qwen (Team, 2024).

Our main contributions are as follows:

(i) Self-Destruct: a new safeguard paradigm with implementation. We introduce Self-Destruct a defense for LMs that embeds protective backdoors directly into model weights (Section 4). Unlike training-based or filter-based defenses, Self-Destruct leverages lightweight calibration and overflow phenomena in numerical formats (e.g., FP32) to disable harmful behaviors at the hardware/runtime level. We implement this approach and show that it prevents harmful generations, withstands jailbreak attacks, and preserves performance on standard benchmarks without introducing inference overhead (Sections 5.1–5.2).

(ii) Robustness against adaptation. We demonstrate that Self-Destruct is robust against powerful adversaries, resisting fine-tuning and gradient-based adversarial methods (Section 5.3).

(iii) Generalization beyond alignment. We show that the Self-Destruct trapdoors generalize beyond alignment, enabling applications such as reliable model fingerprinting and bias mitigation (Section 5.4).

2 Related Work

The widespread adoption of LMs has motivated extensive research on their vulnerabilities and mitigation strategies. We now discuss prior work on attacking and defending LMs as well as work exploiting backdoors for defending models.

Attacks and Defenses. Adversarial attacks on LMs aim to craft prompts that bypass safety filters, commonly referred to as "jailbreaking." Methods range from manual prompt design (Perez and Ribeiro, 2022; Ding et al., 2023; Zou et al., 2023b; Li et al., 2024) to automated search techniques (Zou et al., 2023b; Paulus et al., 2024; Liu et al., 2024a; Mehrotra et al., 2024; Chao et al., 2024). To mitigate attacks, one class of defenses seeks to train models to refuse harmful requests (Glaese et al., 2022; Ouyang et al., 2022). Another prominent class of defenses relies on external monitoring, through either input pre-processing to filter or modify input prompts to detect adversarial queries (Team and Meta, 2024; Zeng et al., 2024a; Robey et al., 2024; Kumar et al., 2025), or via post-processing techniques that refine LMs' outputs (Chen et al., 2023; Li et al., 2023; Phute et al., 2024; Zeng et al., 2024b). Another popular class of defenses draws on methods from interpretability to identify internal components and representations that correspond to specific harmful concepts within

LMs. Particularly, previous work has linked subspaces to concepts such as sentiment, topics, and harmlessness (Elhage et al., 2022; Tigges et al., 2023; Zou et al., 2023a; Von Rütte et al., 2024; Zheng et al., 2024). building on concept discovery, several studies propose disabling unwanted behaviors by targeting certain subspaces. For example, Arditi et al. (2024) discovered the "Refusal" subspace and showed how to carefully steer representations toward this subspace to improve LM safety. Similar works extend this feature (Zhang et al., 2025; Dong et al., 2025), using it to trigger external safeguards against dangerous activations of LMs. Our work instead focuses on hardware-level safeguards that protect model integrity.

Prior work has also explored hardware-based attacks on deep-learning models. Notably, past work focused on means that modify weights-e.g., through flipping a small and carefully selected set of bits (Hong et al., 2019; Rakin et al., 2019; Bai et al., 2023)-to harm model integrity. Still, to the best of our knowledge, such methods have not been applied in the context of safeguarding LMs.

Defensive Utilization of Backdoors. Our method is inspired by defensive applications of backdoor and trapdoor mechanisms in neural networks (Adi et al., 2018; Shan et al., 2020; Liu et al., 2023; Yang et al., 2021; Wang et al., 2024; Liu and Chen, 2024). For instance, Shan et al. (2020) showed that intentionally embedded backdoors can help detect adversarial inputs aiming to induce misclassification. In the context of LMs, while backdoors have been used for LM watermarking and fingerprinting (Xu et al., 2024a), we are unaware of prior work employing backdoors to safeguard models against harmful outputs. Our work fills this gap. Importantly, we clarify that, unlike backdoors that embed malicious code that is executed at inference time (Zhao et al., 2024; Casey et al., 2024), our method directly manipulates weights to disable models when harmful inputs are ingested. As such, our method is compatible with any format used to store model weights (e.g., SafeTensor (Hugging Face, 2022)).

To summarize, to the best of our knowledge, we are the first to propose a backdoor mechanism that leverages model weights to implicitly trigger an overflow error, with the goal of disabling the model under specific use cases. While our design is motivated by the increasing demand to prevent harmful content generation, we envision that our methods can be extended to other concepts.

3 Background

This section provides the necessary technical background and establishes the notation used.

3.1 Transformer-Based LMs

Modern LMs are typically based on the Generative Pre-trained Transformer (GPT) architecture, where multi-head self-attention is the central component. A GPT model consists of L Transformer blocks, each containing a multi-head self-attention sublayer and a feed-forward MLP sublayer. Given an input sequence of n tokens, the model embeds them into d-dimensional vectors:

$$X = [x^1, \cdots, x^n] \in \mathbb{R}^{n \times d}. \tag{1}$$

Multi-Head Attention. Attention uses projection matrices $W_q, W_k, W_v, W_o \in \mathbb{R}^{d \times d}$ for queries, keys, values, and outputs. These are split across h heads, each with dimension $d_h = d/h$. For head j,

$$Q^j = X\hat{W}_q^j, \quad K^j = X\hat{W}_k^j, \quad V^j = X\hat{W}_v^j. \tag{2}$$

The attention weights and output are

$$A^j = \text{softmax}\Big(\frac{Q^j {K^j}^\top}{\sqrt{d_h}}\Big), \qquad \text{head}^j = A^j V^j. \tag{3}$$

The heads are concatenated and projected:

$$\text{MHA}(X) = \big[\text{head}^1 \parallel \cdots \parallel \text{head}^h\big] W_o. \tag{4}$$

Transformer Block. The block then combines this attention output with an MLP layer through residual connections.

3.2 Types and Precision Formats

Floating-point (FP) arithmetic in modern hardware encodes each value with one sign bit, e exponent bits that define the dynamic range, and m mantissa bits that control precision. We focus on the non-quantized formats used by the PyTorch framework (Paszke et al., 2019); the common formats include FP32, FP16, and BF16. In particular, recent state-of-the-art LMs such as Llama (Grattafiori et al., 2024) and Qwen (Team, 2024) are natively deployed in BF16. When a computation produces a value that exceeds a format's maximum (e.g., 3.39×10^{38} for BF16), the result cannot be represented and is clamped to `Inf` at the hardware

level. Subsequent floating-point operations involving `Inf` typically yield `NaN` values, depending on the specific operator and backend implementation. This numeric overflow interrupts the normal flow of matrix operations, effectively halting inference.

In the context of LM inference with PyTorch (Paszke et al., 2019), we empirically observe the following behavior (PyTorch 2.x, CUDA backend). Overflow during matrix multiplication produces `Inf` values, which propagate through subsequent layers and result in `NaN` activations. These `NaN` values persist through the remaining transformer blocks and reach the final projection layer (LM head), yielding a logits vector containing `NaN` entries. During auto-regressive generation in HuggingFace Transformers (Wolf et al., 2020), attempting to perform probabilistic sampling from a logits vector containing `NaN` values raises a runtime error, as sampling from an invalid distribution is undefined. To enable controlled experimentation without terminating the execution environment, we employ greedy decoding, which selects the argmax token. In the presence of `NaN` logits, this operation deterministically returns a constant token index, resulting in repetitive token sequences (e.g., repeated punctuation or special tokens), with the specific token depending on the model's vocabulary ordering. For Llama-3, this corresponds to the process of generating sentence with only exclamation marks ("!"). In addition, attempting to sample a token (Holtzman et al., 2020) directly from this `Null` vector raises an error, halting inference. In contrast, TensorFlow (Abadi et al., 2016) typically raises an `InvalidArgumentError` or propagates `Inf/NaN` tensors depending on the operation, often terminating execution rather than silently continuing. This distinction underscores that the observed self-termination effect of our safeguard depends on the framework's error-handling semantics.

4 Method

We introduce **Self-Destruct** (an overview appears in Figure 2), a mechanism that suppresses specific behaviors by modifying only a single attention head. We start by collecting the activation of interest behavior and benign ones. Let $H' = \{h'_i\}_{i=1}^{N'}$ denote the input-activation (Hidden state) of the target behavior's prompt at a given attention layer, and $H = \{h_j\}_{j=1}^{N}$ denote the ones of benign prompts. In our case of LM alignment, H' are harmful activation while H are harmless ones.

Our method constructs a targeted modification W' in three steps: (i) *null-space isolation*, identifying feature directions that are invisible to the benign activations H; (ii) *targeted subspace extraction*, selecting the most activating directions for the target activations H' within that null space; (iii) *head injection*, embedding these directions into the query and key weights of one attention head.

Intuition. Step 1 isolates directions that are invisible to H; Step 2 selects those null directions that maximally activate on H'; Step 3 embeds them into a head's query/key weights, ensuring $\|hW'\| \approx 0$ for $h \sim H$ while $\|h'W'\|$ is maximized for $h' \sim H'$.

Step 1: Null-space isolation. We begin by computing the singular value decomposition of H:

$$H = U \Sigma V^\top \in \mathbb{R}^{N \times d} \tag{5}$$

Let $r = \mathrm{rank}(H)$, and partition $V = [V_1 \; V_0]$, where $V_0 \in \mathbb{R}^{d \times (d-r)}$ spans the approximate right null space of H. Any $w \in \mathrm{span}(V_0)$ satisfies $Hw \approx 0$, ensuring suppression of H when constructing W'.

Step 2: Targeted subspace extraction. We next project the target activations $H' \in \mathbb{R}^{N' \times d}$ into this null space:

$$Z = H'V_0 \in \mathbb{R}^{N' \times (d-r)}. \tag{6}$$

Performing an SVD $Z = U' \Sigma' V'^\top$, we extract the top δ right singular vectors

$$V_1'^{(\delta)} = V_1'[:, 1:\delta], \tag{7}$$

which correspond to directions in the null subspace that maximize $\|H'V_0\|$.

Step 3: Head injection. We map these directions back to the model space:

$$M = V_0 V_1'^{(\delta)} \in \mathbb{R}^{d \times \delta}. \tag{8}$$

If $\delta = d/h$, we set $W' = M$; otherwise, we tile or pad M horizontally to match the head dimension d/h. Finally, we update a single head's query and key projections:

$$\hat{W}'_q = \hat{W}_q + W', \qquad \hat{W}'_k = \hat{W}_k + W'.$$

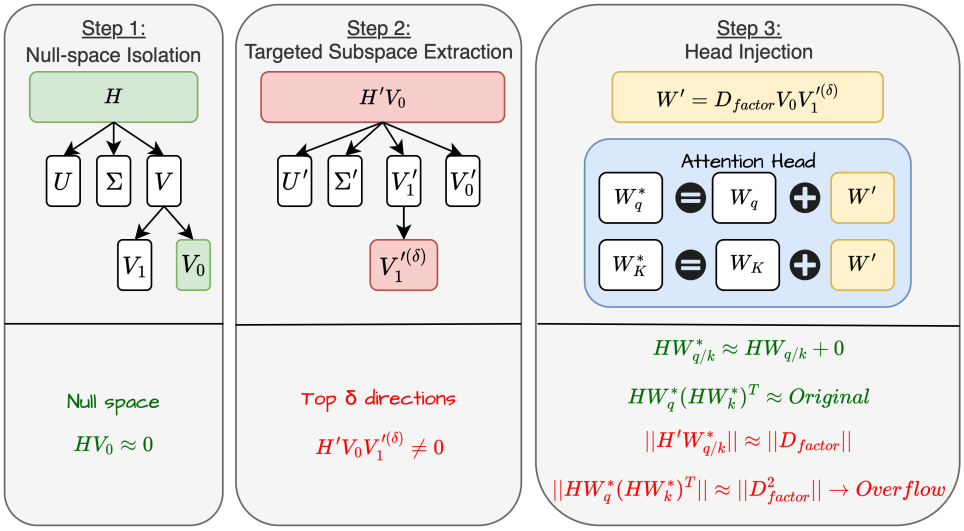


Figure 2: Overview of the **Self-Destruct** mechanism. Step 1: compute the SVD of H and take the null-space basis V_0 ($HV_0 \approx 0$). Step 2: project H' into this basis and extract top δ directions $V_1'^{(\delta)}$ that maximize $H'V_0$. Step 3: form $W' = D_{\text{factor}}V_0V_1'^{(\delta)}$ and inject it into one attention head's Q and K weights.

Clustering the targeted activation. The construction of M is based on H', the activations we aim to manipulate, for which we have no prior information regarding diversity: neither how linguistically different the prompts that produce H' are, nor how the activations are geometrically distributed. However, if we set $\delta \ll \frac{d}{n}$, the resulting SVD will yield a low-dimensional decomposition, producing a low-dimensional M that may fail to generalize across a wide range of benign prompts. To address this, we use a clustering phase: given H' and $n \in \mathbb{N}^+$, we run k-means (Ahmed et al., 2020) based on cosine similarity among vectors $h' \in H'$. This divides H' into n groups, denoted:

$$\hat{C}' = \{C_1', C_2', \ldots, C_n'\}, \quad (9)$$

$$\bigcup_{i=1}^{n} C_i' = H', \quad C_i' \cap C_j' = \emptyset \text{ for } i \neq j \quad (10)$$

For each group in C', we separately extract $M' \in \mathbb{R}^{d\times\delta}$, using the same H as the undesirable activation. Finally, we construct $M \in \mathbb{R}^{d\times n\delta}$ by concatenating all M' matrices. The construction of W' remains the same: we set $W' = M$ if $n\delta = \frac{d}{h}$, or otherwise horizontally concatenate copies of M to match the shape of W'.

Destruction factor. Our goal is to trigger an overflow during the computation of the query–key dot product $Q^jK^{j\top}$. We target this stage for two reasons. First, since the result is subsequently normalized by the softmax, moderate increases in magnitude have limited impact on benign inputs, keeping the mechanism minimally intrusive under normal operation. Second, operating near the numerical overflow boundary is inherently unstable: values close to the maximum representable number can easily exceed it under slight perturbations.

To achieve reliable activation of the mechanism, we do not simply push the individual query and key activations to arbitrarily large values, as this could itself cause instability. Instead, we explicitly scale both the query and key vectors so that their dot product approaches the largest representable floating-point number, denoted as Ω. For example, $\Omega \approx 3.39\times10^{38}$ for BF16 $\Omega \approx 3.4\times10^{38}$ for FP32. The injected trapdoor values are deliberately chosen to exceed the BF16 exponent range and were empirically verified to induce deterministic overflow during matrix multiplication. Concretely, we construct modified vectors q' and k' whose norms satisfy $\|q'\| \approx \|k'\| \approx \sqrt{\Omega}$, $\|q'k'^{\top}\| \approx \Omega$. In this way, each individual vector remains within the representable numerical range, but their dot product reaches the overflow threshold. This causes the multiplication in $Q^jK^{j\top}$ to exceed Ω, triggering a numerical overflow that propagates forward through the attention computation and effectively halts further model operation. In practice, the absolute scale of activations differs across LMs. Therefore, to calibrate the precise scaling required for Self-Destruct, we perform a grid search over a validation set of prompts, $(H'_{\text{val}}, H_{\text{val}})$, selecting the destruction factor within the interval $[\sqrt{\Omega}/128, \sqrt{\Omega}]$ that reliably induces overflow without affecting

normal generations.

Self-Destruct initiation. At inference, the activations h pass through the modified attention head:

$$q + q' = h(\hat{W}_q + W'), \tag{11}$$

$$k + k' = h(\hat{W}_k + W'). \tag{12}$$

For benign activations $h \in H$, the contribution from W' is suppressed by construction, so

$$q + q' \approx q + 0, \qquad k + k' \approx k + 0. \tag{13}$$

In contrast, for harmful activations $h' \in H'$, the injected component dominates and scales to the overflow boundary:

$$\|q + q'\| \approx \sqrt{\Omega}, \qquad \|k + k'\| \approx \sqrt{\Omega}, \tag{14}$$

so that their dot product in the attention calculation satisfies

$$\|(q + q')(k + k')^\top\| \approx \Omega \to \texttt{OverFlow} \tag{15}$$

which exceeds the representable range and triggers overflow, thereby halting further computation.

5 Experiments

We evaluate Self-Destruct as a jailbreak defense mechanism that blocks harmful prompts while preserving benign generation. In addition, we examine Self-Destruct in biased text mitigation and model fingerprinting. All experiments use the same hardware, fixed random seed, and batch size of 1 for consistency.

5.1 Constructing Self-Destruct

Models and Data-Type Selection. We use five open-source, instruction-following, model families in our experiments: Falcon3 (Almazrouei et al., 2023), Llama-3.2 (Grattafiori et al., 2024), Mistral-0.3 (Jiang et al., 2023), Qwen-2.5 (Team, 2024) and Gemma2 (Team et al., 2024), examining models ranging from 1B to 14B parameters. All these LMs employ BF16 as their default data type in HuggingFace hub (Wolf et al., 2020), which we also adopt in all subsequent experiments. Since the maximum value of BF16 is almost the same as FP32, 3.39×10^{38} and 3.4×10^{38} , our Self-Destruct construction is applicable to both formats, while requiring significantly less compute with BF16 than with FP32. To eliminate variability from hardware effects and ensure consistency, all experiments were conducted on a fixed hardware setup per LM instance. Further details about our setup are provided in Appendix A.

Training Datasets. For our experiments, we relied on the datasets introduced by Arditi et al. (2024), which are split into harmful and harmless subsets as well as into training and test splits. We emphasize that the selection in Arditi et al. (2024)'s dataset was solely motivated by its simplicity and the desire to rely on well-known prior work, rather than by result optimization. From these datasets, we extract the harmful and harmless activations, H' and H, using a small subset of the training split, consisting of 500 random harmless prompts and 260 harmful prompts, all of the harmful prompts available in this dataset.

To calibrate our defense mechanism, we evaluated refusal rates on a subset of 100 harmful and 100 harmless prompts randomly selected from the corresponding test split. Specifically, we measured the difference in refusal rate between harmless and harmful prompts, i.e., $\Delta = \text{RefusalRate}(H) - \text{RefusalRate}(H')$, which we refer to as the difference score. We then tuned the defense hyperparameters (target layer, head index within that layer, and the destruction parameter) and selected the configuration that maximized Δ. Further calibration details can be found in Appendix A, as well as the per model hyperparameter selections.

In this process, we were able to create, for each LM, a Self-Destruct manipulation (a set of hyperparameters) that achieved above $\Delta = 0.97$ difference score on both the training and test splits. In most LMs, such as those with 7B or more parameters, these scores are reached $\Delta = 0.99$ test score, meaning that our method effectively activated only for the harmful prompts.

5.2 Alignment Defense

Baselines Defenses. We compare our safeguard with prior defense methods. Since our focus is on enabling the model to guard itself, without relying on external compute or API services, we consider only baseline methods that depend solely on a single instance of the LM, excluding approaches where additional, external LMs are used to guard one another (Wu et al., 2024; Team and Meta, 2024). Accordingly, the defenses we compare against include: **Baseline** (model's internal refusal alignment); **Perplexity Filtering** (Jain et al., 2023) (detecting gibberish prompts such as GCG (Zou et al., 2023b)); **Self-Defense** (Phute et al., 2024) (using the same LM to judge the harmfulness of input or output); and **SmoothLLM** (Robey et al., 2024) (perturbation-based adversarial detection).

It is important to note that, unlike prior methods which detect harmful content but still allow generation to continue, our mechanism halts output entirely, typically at the very first token, by inducing overflow, thereby preventing any continuation of harmful text. In addition, Self-Destruct does not introduce additional computational requirements, as defense is activated simultaneously with content generation. This behavior is similar to Perplexity Filtering, however, Perplexity calculation requires generating the full text before evaluating it, unlike Self-Destruct. Consequently, Self-Destruct is far more compute-efficient than methods such as Self-Defense and SmoothLLM, which require additional LM runs for harmfulness evaluation.

Evaluation Datasets. We evaluated our method on harmfulness and capability benchmarks:

Alignment and harmfulness benchmarks. We evaluate on **MalwareGen** (Derczynski et al., 2024) (malware code prompts); **PromptInject** (Perez and Ribeiro, 2022) (instruction-override attacks); **AutoDAN** (Liu et al., 2024a) (automated red-teaming prompts); and **PAIR** (Chao et al., 2024) (LM-vs-LM jailbreak dataset). These represent standard black-box evaluations (Arditi et al., 2024; Dong et al., 2024) where the attacks are not directly optimized against the defended LM or an undefended version thereof. We complement these with so-called white- and gray-box evaluations where we directly optimize the attacks against the LM in Section 5.3.

General capability benchmarks. We test on **PIQA** (Bisk et al., 2020) (physical commonsense), **OpenBookQA** (Mihaylov et al., 2018) (elementary science QA), **SIQA** (Sap et al., 2019) (social commonsense), and **ARC** (Clark et al., 2018) (complex science QA). From an implementation perspective and due to limited computation, each dataset was evaluated using 100 randomly selected prompts. Model responses were generated autoregressively, with a maximum length of 512 tokens. Taken together, these datasets enable us to assess both robustness against malicious prompt injections and the preservation of general reasoning performance on benign tasks.

Metrics. For the harmfulness benchmarks, we measure the Attack Success Rate (ASR): the percentage of malicious prompts that bypass the defense mechanism. For benign tasks, we measure accuracy and refusal rate: the proportion of instances where the defense incorrectly triggers. Following HarmBench implementation Mazeika et al. (2024), we employ as an ASR classifier a custom fine-tuned version of Llama-2-13B. In Appendix B we also provide ASR results as judge by Gemma-3-27B (Team et al., 2025) for extended evaluation of the judging approach.

Results. The experimental results are summarized according to the three evaluation metrics. Figure 3a presents the ASR across the four alignment datasets, where lower values indicate stronger defenses. Figure 3b shows the average results on the four question-answering datasets, balancing model performance against over-refusal rates. Extended results can be found in Appendix B.

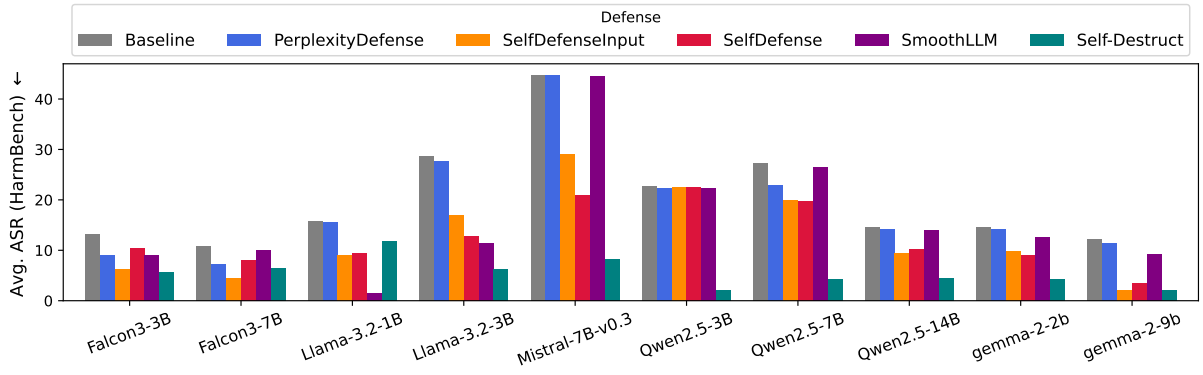
Examining the performance of Self-Destruct, we observe that this mechanism is highly competitive with the baseline defenses: in all models except Llama-3.2-1B, Self-Destruct achieves the lowest ASR. In most of the settings refusal rates remain low and question-answering performance shows almost no degradation.

Implementation settings. We tested the Self-Destruct mechanism across different hardware platforms, attention implementations, and inference stacks. Details on these settings are provided in Appendix C.

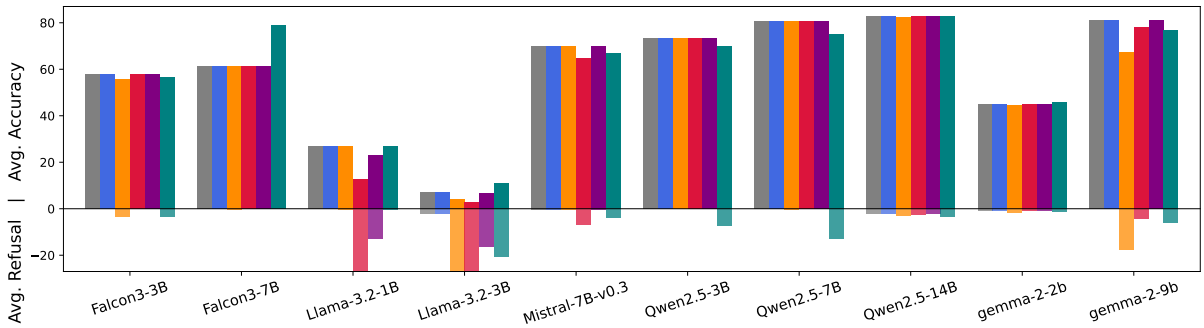
5.3 Gradient-overflow Resistance

Self-Destruct modifies only a small subset of model parameters. **An informed user could potentially disable the mechanism** by identifying and zeroing unusually large weights. Since LMs such as Qwen-2.5 and Llama-3.2 typically have weight magnitudes no larger than two digits, a threshold of 1×10^5 would reveal affected attention heads. This attack, however, would not be possible if the attack is unaware of the defense or if the model is deployed by another party. In this section, we assess whether an attacker is able to circumvent Self-Destruct in such scenarios.

Fine-tuning resistance. Our defense mechanism modifies only a small subset of parameters in the attention module, yet it has a profound effect on model trainability. To evaluate the effect of Self-Destruct on model trainability, and to test whether adversaries could disable the defense via post-hoc fine-tuning (Qi et al., 2023), we conducted supervised fine-tuning on LMs modified with Self-Destruct. In our experiments, we var-



(a) ASR



(b) Question-Answering vs. Refusal

Figure 3: Alignment results for 10 LMs, comparing ASR, QA accuracy, and refusal rates across defenses. The Llama LMs' refusal bars are truncated, with full results provided in the appendix.

ied the batch size (1–10), optimization algorithms (SGD, AdamW), gradient clipping, and hardware platforms (NVIDIA L40S, RTX 3090, A100).

We used Alpaca (Taori et al., 2023), a widely adopted dataset for general-purpose conversational fine-tuning, and the Refusal-Direction dataset (Arditi et al., 2024), conducting separate fine-tuning runs on their harmful and harmless subsets to test whether Self-Destruct could be neutralized under different training signals. In all configurations, fine-tuning failed from the very first iteration. Backpropagation consistently yielded undefined gradients (NaN) in every batch and for every example. regardless of optimizer, batch size, or dataset composition. Importantly, this instability persisted even when training exclusively on harmless data from Refusal-Direction, neither of which trigger the defense mechanism during inference. The defense thus induces gradient pathologies in the modified attention layers that prevent any parameter updates, making the model effectively untrainable.

Failure of gradient-based attacks. Unlike static prompt attacks, which rely on fixed sets of sentences such as those evaluated in Section 5, LMs can also be targeted with multi-turn adaptive prompt attacks. One class of such attacks is gradient-guided jailbreaks, which exploit stable gradient signals to construct adversarial prompts. The Greedy Coordinate Gradient (GCG) method (Zou et al., 2023b) is an example of such a dynamic attack, which we evaluated on LM instances equipped with Self-Destruct. Because our mechanism disrupts the gradient flow, these attacks fail in practice. For example, when applying GCG against Llama-3.2 defended by Self-Destruct under a white-box setting, the optimization diverged immediately, with the loss remaining flat and gradients yielding invalid values (NaN or negative probabilities). This behavior was consistent across trials, confirming that gradient-based jailbreaks cannot progress once the model has been modified.

We further evaluated the gray-box setting, simulating an adversary that knows the deployed model byt is unaware of the defense. Here, adversarial suffixes were first generated on an undefended model and then transferred to a defended one (e.g., suffixes produced by GCG on a raw Llama-3.2 were tested against its Self-Destruct–equipped counterpart). In all cases, the suffixes failed to induce harmful generations, as the overflow disruption still triggered near the end of the original prompt, result-

ing in invalid gradient signals and termination of generation. We hypothesize that resistance to both GCG and fine-tuning arises from gradient instability caused by the high-magnitude values introduced by Self-Destruct, which may also "block" a small portion of benign data correlated with harmful content. Taken together, these findings demonstrate that the proposed defense withstands not only post-hoc fine-tuning attempts but also gradient-based adversarial methods, thereby complicating the efforts of potential attackers.

5.4 Generalization - Beyond LM Alignment

Previously, we examine our Self-Destruct trapdoor mechanism for mitigating general alignment behavior. However, this approach is not limited to alignment. In this section, we provide two examples of how Self-Destruct can be extended into other domains.

5.4.1 Fingerprints

We investigate whether Self-Destruct can be repurposed as a model fingerprinting mechanism (Xu et al., 2024a), enabling the identification of a specific LM output for predefined user queries. To this end, we define a passcode that serves as a fingerprint trigger: when present within the input, it activates the Self-Destruct trapdoor. To enable this usage of Self-Destruct, we replaced the harmful dataset used in the calibration detailed in subsection 5.1 with harmless sentences sampled from Alpaca (Taori et al., 2023) that were appended with a predefined string functioning as the fingerprint passcode. Therefore, the calibration process includes two harmless datasets, only one of which contains the passcode and is used as the trapdoor trigger. We assess a range of passcodes, including intuitive human-readable triggers such as "CRASH THE SYSTEM", as well as randomly generated strings produced by password generators, such as "AcCse3TTOxX6lcYi". Experiments conducted on Gemma-2-2B and Qwen2.5-3B demonstrate highly consistent performance across both models: the human-readable passcode yields near-perfect identification performance, with Gemma achieving $\Delta = 1$ test score and Qwen achieving $\Delta = 0.97$. For the random passcodes, across five length configurations (12, 14, 16, 18, and 20 characters), Self-Destruct attains $\Delta = 0.95$ score on Qwen and near-perfect score on Gemma.

5.4.2 Concrete Bias Mitigation

We further investigate whether Self-Destruct can be applied to specific alignment problems, such as bias. For this goal, we replace the harmful dataset with examples sampled from the BBQ benchmark (Parrish et al., 2022). The objective is to activate the trapdoor when bias-related content is present, while preserving ordinary model behavior under neutral inputs. We conduct this study using Qwen2.5-3B, Qwen2.5-7B, and Gemma-2-2B, and independently replicate the setup using Alpaca samples as the harmless dataset. Across all configurations, the trapdoor mechanism achieves $\Delta = 1$ difference score on a held-out BBQ test split, demonstrating complete suppression of biased responses. On the harmless Alpaca samples, the models maintain high fidelity, achieving $\Delta = 0.97$ score for Gemma and $\Delta = 0.99$ for the Qwen models.

These results indicate that Self-Destruct generalizes beyond general alignment to bias mitigation and even fingerprinting, providing robust suppression of undesirable outputs while preserving model performance on benign inputs. The findings highlight the broader applicability of trapdoor-based control signals as a practical and efficient mitigation strategy across diverse model behaviors.

6 Discussion and Conclusion

Our findings suggest that embedding a Self-Destruct mechanism directly into model weights provides a novel paradigm for LM safety. By exploiting floating-point limits, the mechanism enforces safety constraints proactively without external filters or alignment. Beyond LM alignment, this approach can be applied to trigger overflows for specific behaviors or target hardware malfunctions. However, this design introduces limitations: the hidden termination trigger may be perceived as a backdoor, raising transparency and trust concerns. While our evaluations confirm resilience to fine-tuning and architectural compatibility, further work is needed to assess robustness across models, adversarial settings, and hardware. Future work should explore: (i) interactions with quantization and pruning techniques, (ii) distributing Self-Destruct triggers across multiple layers, which may reduce the effectiveness of naive adversarial strategies that target a single point of failure, and (iii) applications beyond harmful content generation to safety-critical domains such as medical or financial decision-making.

Limitations

The Self-Destruct mechanism relies on fixed hardware and implementation code. This means that performance may vary across different hardware configurations, as well as in cross-platform training and testing scenarios, which we did not evaluate. In particular, we focused on instances of BF16 LMs, a choice motivated by the default data type used in these models. We explicitly limit our scope to a single data-type format, the default deployment format of each evaluated model (BF16), and do not study cross–data-type effects or portability across precision formats. Our analysis indicates that BF16 and FP32 share the same maximum value and therefore should operate similarly. However, we did not examine other data types such as FP64 (double) or quantized types (e.g., Int32, Int64). Additionally, we did not explore quantization setups, such as post-training or quantization-aware training regimes, and therefore cannot assess how Self-Destruct behaves under reduced-precision or integer-only inference pipelines. We limited our scope by not exploring these data types or cross–data-type effects, since our focus was on demonstrating the feasibility of such a Self-Destruct mechanism rather than ensuring robustness across all use cases. Our evaluation, although spanning eight datasets, only covered a subset of each. This restriction was due to the extensive reliance on an external LM-as-a-judge setup, used to classify model responses as harmless or harmful.

With respect to the alignment training dataset, we used Arditi et al. (2024), which may cover only a narrow set of anti-jailbreak cases. Future work could extend this by incorporating more diverse training data sources. Nevertheless, despite this limitation, we were surprised to achieve strong ASR performance across four diverse benchmarks.

Regarding positional embeddings, we find that our method is compatible with RoPE (Su et al., 2024), which is used by all models examined in this work. This indicates that the rotations applied by RoPE to the queries and keys preserve both the high-norm structure of the manipulated weights and the functionality of the halting effect.

We note that Self-Destruct might not be restricted to alignment purposes, as it could also be applied to disabling other concepts. While our alignment-focused results were positive, we did not empirically test this broader application of Self-Destruct and leave it for future work. Finally, Self-Destruct operates by manipulating model parameters into out-of-distribution values. An informed user, aware of this mechanism, could disable it by identifying such out-of-distribution parameters and zeroing them. This means that our defense is relatively easy to uncover and revert for users with access to the model's parameters. While this is a significant limitation, it is shared with many other defense mechanisms, such as SmoothLLM and Self-Defense. Accordingly, our contribution lies in demonstrating the validity of a new mechanism, though we cannot guarantee its full scalability or robustness against diverse users.

Ethics Statement

This work introduces a new backdoor mechanism for LMs that exploits hardware limitations with the aim of improving their safety and alignment. However, we acknowledge that such a mechanism could be misused for data manipulation, hardware-based attacks, and even user manipulation. We recognize these risks and have conducted our work within controlled experimental environments, sharing our findings with the community while emphasizing the importance of responsible use.

To promote transparency and reproducibility, we adopt open science practices throughout this project. We intend to release the code, configuration files, and scripts necessary to replicate our experiments, alongside the datasets used for demonstration. All parameter settings, random seeds, and environment details have been documented to facilitate independent verification. We also report known limitations and failure modes to enable responsible follow-up research.

Acknowledgements

This work was supported by a Tel Aviv University Center for AI and Data Science (TAD) grant and by Len Blavatnik and the Blavatnik Family foundation. This research was also supported by the Ministry of Innovation, Science & Technology, Israel (1001576154) and the Michael J. Fox Foundation (MJFF-022407). The contribution of SK is part of a PhD thesis research conducted at Tel Aviv University.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin,

Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*.

Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX security symposium*.

Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. 2020. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*.

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. 2023. The falcon series of open language models. *Preprint*, arXiv:2311.16867.

Anthropic. 2025. System card: Claude opus 4 and claude sonnet 4. https://www-cdn.anthropic.com/6be99a52cb68eb70eb9572b4cafad13df32ed995.pdf. System card.

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. 2024. Refusal in language models is mediated by a single direction. In *NeurIPS*.

Jiawang Bai, Baoyuan Wu, Zhifeng Li, and Shu-Tao Xia. 2023. Versatile weight attack via flipping limited bits. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*.

Sarah Ball, Frauke Kreuter, and Nina Panickssery. 2024. Understanding jailbreak success: A study of latent space dynamics in large language models. *arXiv preprint arXiv:2406.09289*.

Matan Ben-Tov, Mor Geva, and Mahmood Sharif. 2025. Universal jailbreak suffixes are strong attention hijackers. *arXiv preprint arXiv:2506.12880*.

Federico Bianchi and James Zou. 2024. Large language models are vulnerable to bait-and-switch attacks for generating harmful content. *arXiv preprint arXiv:2402.13926*.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Neil Burgess, Jelena Milanovic, Nigel Stephens, Konstantinos Monachopoulos, and David Mansell. 2019. Bfloat16 processing for neural networks. In *2019 IEEE 26th Symposium on Computer Arithmetic*.

Beatrice Casey, Joanna Santos, and Mehdi Mirakhorli. 2024. A large-scale exploit instrumentation study of ai/ml supply chain attacks in hugging face models. *arXiv preprint arXiv:2410.04490*.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas, and Eric Wong. 2024. Jailbreaking black box large language models in twenty queries. *Preprint*, arXiv:2310.08419.

Bocheng Chen, Advait Paliwal, and Qiben Yan. 2023. Jailbreaker in jail: Moving target defense for large language models. *Preprint*, arXiv:2310.02417.

Sooyung Choi, Jaehyeok Lee, Xiaoyuan Yi, Jing Yao, Xing Xie, and JinYeong Bak. 2025. Unintended harms of value-aligned llms: Psychological and empirical insights. *arXiv preprint arXiv:2506.06404*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *Preprint*, arXiv:1803.05457.

Leon Derczynski, Erick Galinkin, Jeffrey Martin, Subho Majumdar, and Nanna Inie. 2024. garak: A framework for security probing large language models.

Peng Ding, Jun Kuang, Dan Ma, Xuezhi Cao, Yunsen Xian, Jiajun Chen, and Shujian Huang. 2023. A wolf in sheep's clothing: Generalized nested jailbreak prompts can fool large language models easily. *arXiv preprint arXiv:2311.08268*.

Weilong Dong, Peiguang Li, Yu Tian, Xinyi Zeng, Fengdi Li, and Sirui Wang. 2025. Feature-aware malicious output detection and mitigation. *Preprint*, arXiv:2504.09191.

Yi Dong, Ronghui Mu, Yanghao Zhang, Siqi Sun, Tianle Zhang, Changshun Wu, Gaojie Jin, Yi Qi, Jinwei Hu, Jie Meng, et al. 2024. Safeguarding large language models: A survey. *arXiv preprint arXiv:2406.02622*.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. 2022. Toy models of superposition. *arXiv preprint arXiv:2209.10652*.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. 2020. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. In *Findings of EMNLP 2020*.

Amelia Glaese, Nat McAleese, Maja Trębacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, Lucy Campbell-Gillingham, Jonathan Uesato, Po-Sen Huang, Ramona Comanescu, Fan Yang, Abigail

See, Sumanth Dathathri, Rory Greig, Charlie Chen, Doug Fritz, Jaume Sanchez Elias, Richard Green, Soňa Mokrá, Nicholas Fernando, Boxi Wu, Rachel Foley, Susannah Young, Iason Gabriel, William Isaac, John Mellor, Demis Hassabis, Koray Kavukcuoglu, Lisa Anne Hendricks, and Geoffrey Irving. 2022. Improving alignment of dialogue agents via targeted human judgements. *Preprint*, arXiv:2209.14375.

Yichen Gong, Delong Ran, Xinlei He, Tianshuo Cong, Anyu Wang, and Xiaoyun Wang. 2025. Safety misalignment against large language models. In *Proceedings of the 2025 Annual NDSS*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, and the Llama3's team. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Shanshan Han, Salman Avestimehr, and Chaoyang He. 2025. Bridging the safety gap: A guardrail pipeline for trustworthy llm inferences. *arXiv preprint arXiv:2502.08142*.

Tianyu Han, Sven Nebelung, Firas Khader, Tianci Wang, Gustav Müller-Franzes, Christiane Kuhl, Sebastian Försch, Jens Kleesiek, Christoph Haarburger, Keno K Bressem, et al. 2024. Medical large language models are susceptible to targeted misinformation attacks. *NPJ digital medicine*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *Proceedings of ICLR 2020*.

Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraș. 2019. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th USENIX Security Symposium*.

Hugging Face. 2022. Safetensors. Simple, safe way to store and distribute tensors.

Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. 2023. Baseline defenses for adversarial attacks against aligned language models. *Preprint*, arXiv:2309.00614.

Samyak Jain, Ekdeep S Lubana, Kemal Oksuz, Tom Joy, Philip Torr, Amartya Sanyal, and Puneet Dokania. 2024. What makes and breaks safety fine-tuning? a mechanistic study. *Advances in NeurIPS*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. 2019. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*.

Nathalie Kirch, Constantin Weisser, Severin Field, Helen Yannakoudakis, and Stephen Casper. 2024. What features in prompts jailbreak llms? investigating the mechanisms behind attacks. *arXiv preprint arXiv:2411.03343*.

Aounon Kumar, Chirag Agarwal, Suraj Srinivas, Aaron Jiaxun Li, Soheil Feizi, and Himabindu Lakkaraju. 2025. Certifying llm safety against adversarial prompting. *Preprint*, arXiv:2309.02705.

Andrey Kuzmin, Mart Van Baalen, Yuwei Ren, Markus Nagel, Jorn Peters, and Tijmen Blankevoort. 2022. Fp8 quantization: The power of the exponent. *Advances in NeurIPS*.

Andrew Lee, Xiaoyan Bai, Itamar Pres, Martin Wattenberg, Jonathan K Kummerfeld, and Rada Mihalcea. 2024. A mechanistic understanding of alignment algorithms: A case study on dpo and toxicity. *arXiv preprint arXiv:2401.01967*.

Tianlong Li, Zhenghua Wang, Wenhao Liu, Muling Wu, Shihan Dou, Changze Lv, Xiaohua Wang, Xiaoqing Zheng, and Xuan-Jing Huang. 2025a. Revisiting jailbreaking for large language models: A representation engineering perspective. In *Proceedings of the 31st International Conference on Computational Linguistics*.

Xirui Li, Ruochen Wang, Minhao Cheng, Tianyi Zhou, and Cho-Jui Hsieh. 2024. Drattack: Prompt decomposition and reconstruction makes powerful llm jailbreakers. *arXiv preprint arXiv:2402.16914*.

Yang Li, Qiang Sheng, Yehan Yang, Xueyao Zhang, and Juan Cao. 2025b. From judgment to interference: Early stopping llm harmful outputs via streaming content monitoring. *arXiv preprint arXiv:2506.09996*.

Yuhui Li, Fangyun Wei, Jinjing Zhao, Chao Zhang, and Hongyang Zhang. 2023. Rain: Your language models can align themselves without finetuning. *Preprint*, arXiv:2309.07124.

Frank Weizhen Liu and Chenhui Hu. 2024. Exploring vulnerabilities and protections in large language models: A survey. *Preprint*, arXiv:2406.00240.

Qi Liu, Jieming Yin, Wujie Wen, Chengmo Yang, and Shi Sha. 2023. NeuroPots: Realtime proactive defense against Bit-Flip attacks in neural networks. In *32nd USENIX Security Symposium*, Anaheim, CA.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. 2024a. Autodan: Generating stealthy jailbreak prompts on aligned large language models. *Preprint*, arXiv:2310.04451.

Yi Liu, Gelei Deng, Zhengzi Xu, Yuekang Li, Yaowen Zheng, Ying Zhang, Lida Zhao, Tianwei Zhang, Kailong Wang, and Yang Liu. 2024b. Jailbreaking chatgpt via prompt engineering: An empirical study. *Preprint*, arXiv:2305.13860.

ZhenTing Liu and ShangTse Chen. 2024. Trap-mid: Trapdoor-based defense against model inversion attacks. *Advances in NeurIPS*.

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. 2024. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *Preprint*, arXiv:2402.04249.

Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. 2024. Tree of attacks: Jailbreaking black-box llms automatically. *Preprint*, arXiv:2312.02119.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. 2022. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

ONNX Runtime developers. 2018. ONNX Runtime.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. In *Advances in NeurIPS*.

Alicia Parrish, Angelica Chen, Nikita Nangia, Vishakh Padmakumar, Jason Phang, Jana Thompson, Phu Mon Htut, and Samuel R. Bowman. 2022. Bbq: A hand-built bias benchmark for question answering.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in NeurIPS*, 32.

Anselm Paulus, Arman Zharmagambetov, Chuan Guo, Brandon Amos, and Yuandong Tian. 2024. Advprompter: Fast adaptive adversarial prompting for llms. *arXiv preprint arXiv:2404.16873*.

Fábio Perez and Ian Ribeiro. 2022. Ignore previous prompt: Attack techniques for language models. *arXiv preprint*.

Mansi Phute, Alec Helbling, Matthew Hull, ShengYun Peng, Sebastian Szyller, Cory Cornelius, and Duen Horng Chau. 2024. Llm self defense: By self examination, llms know they are being tricked. *Preprint*, arXiv:2308.07308.

Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. 2023. Fine-tuning aligned language models compromises safety, even when users do not intend to! *arXiv preprint arXiv:2310.03693*.

Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. 2019. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Alexander Robey, Eric Wong, Hamed Hassani, and George J. Pappas. 2024. Smoothllm: Defending large language models against jailbreaking attacks. *Preprint*, arXiv:2310.03684.

Sap, Maarten, Rashkin, Hannah, Chen, Derek, Le Bras, Ronan, Choi, and Yejin. 2019. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on EMNLP*, Hong Kong and China.

Shawn Shan, Emily Wenger, Bolun Wang, Bo Li, Haitao Zheng, and Ben Y Zhao. 2020. Gotta catch'em all: Using honeypots to catch adversarial attacks on neural networks. In *Proceedings of the 2020 ACM SIGSAC conference*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, and Alex Feng et al. 2025. Gemma 3 technical report.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton,

Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, and Bo Wu et al. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Llama Team and AI @ Meta. 2024. The llama 3 family of models.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Curt Tigges, Oskar John Hollinsworth, Atticus Geiger, and Neel Nanda. 2023. Linear representations of sentiment in large language models. *arXiv preprint arXiv:2310.15154*.

Mart Van Baalen, Andrey Kuzmin, Suparna S Nair, Yuwei Ren, Eric Mahurin, Chirag Patel, Sundar Subramanian, Sanghyuk Lee, Markus Nagel, Joseph Soriaga, et al. 2023. Fp8 versus int8 for efficient deep learning inference. *arXiv preprint arXiv:2303.17951*.

Dimitri Von Rütte, Sotiris Anagnostidis, Gregor Bachmann, and Thomas Hofmann. 2024. A language model's guide through latent space. *arXiv preprint arXiv:2402.14433*.

Yifei Wang, Dizhan Xue, Shengjie Zhang, and Shengsheng Qian. 2024. Badagent: Inserting and activating backdoor attacks in llm agents. In *ACL*.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. 2023. Jailbroken: How does llm safety training fail? *Preprint*, arXiv:2307.02483.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clément Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Brew, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of EMNLP 2020: System Demonstrations*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.

Jiashu Xu, Fei Wang, Mingyu Ma, Pang Wei Koh, Chaowei Xiao, and Muhao Chen. 2024a. Instructional fingerprinting of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Jinyuan Jia, Bill Yuchen Lin, and Radha Poovendran. 2024b. Safedecoding: Defending against jailbreak attacks via safety-aware decoding. *Preprint*, arXiv:2402.08983.

Wenkai Yang, Yankai Lin, Peng Li, Jie Zhou, and Xu Sun. 2021. Rap: Robustness-aware perturbations for defending against backdoor attacks on nlp models. In *Proceedings of the 2021 Conference on EMNLP*.

Xianjun Yang, Xiao Wang, Qi Zhang, Linda Petzold, William Yang Wang, Xun Zhao, and Dahua Lin. 2023. Shadow alignment: The ease of subverting safely-aligned language models. *arXiv preprint arXiv:2310.02949*.

Jingwei Yi, Rui Ye, Qisi Chen, Bin Zhu, Siheng Chen, Defu Lian, Guangzhong Sun, Xing Xie, and Fangzhao Wu. 2024. On the vulnerability of safety alignment in open-access llms. In *Findings of the ACL 2024*.

Wenjun Zeng, Yuchi Liu, Ryan Mullins, Ludovic Peran, Joe Fernandez, Hamza Harkous, Karthik Narasimhan, Drew Proud, Piyush Kumar, Bhaktipriya Radharapu, Olivia Sturman, and Oscar Wahltinez. 2024a. Shieldgemma: Generative ai content moderation based on gemma. *Preprint*, arXiv:2407.21772.

Yifan Zeng, Yiran Wu, Xiao Zhang, Huazheng Wang, and Qingyun Wu. 2024b. Autodefense: Multi-agent llm defense against jailbreak attacks. *Preprint*, arXiv:2403.04783.

Shenyi Zhang, Yuchen Zhai, Keyan Guo, Hongxin Hu, Shengnan Guo, Zheng Fang, Lingchen Zhao, Chao Shen, Cong Wang, and Qian Wang. 2025. Jbshield: Defending large language models from jailbreak attacks through activated concept analysis and manipulation. *Preprint*, arXiv:2502.07557.

Jian Zhao, Shenao Wang, Yanjie Zhao, Xinyi Hou, Kailong Wang, Peiming Gao, Yuanchao Zhang, Chen Wei, and Haoyu Wang. 2024. Models are codes: Towards measuring malicious code poisoning attacks on pre-trained model hubs. In *Proceedings of the 39th IEEE/ACM I*.

Chujie Zheng, Fan Yin, Hao Zhou, Fandong Meng, Jie Zhou, Kai-Wei Chang, Minlie Huang, and Nanyun Peng. 2024. On prompt-driven safeguarding for large language models. In *International Conference on Machine Learning*.

Jingnan Zheng, Xiangtian Ji, Yijun Lu, Chenhang Cui, Weixiang Zhao, Gelei Deng, Zhenkai Liang, An Zhang, and Tat-Seng Chua. 2025. Rsafe: Incentivizing proactive reasoning to build robust and adaptive llm safeguards. *arXiv preprint arXiv:2506.07736*.

Zhenhong Zhou, Jiuyang Xiang, Haopeng Chen, Quan Liu, Zherui Li, and Sen Su. 2024. Speak out of turn: Safety vulnerability of large language models in multi-turn dialogue. *arXiv preprint arXiv:2402.17262*.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski,

et al. 2023a. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. 2023b. Universal and transferable adversarial attacks on aligned language models. *Preprint*, arXiv:2307.15043.

A Self-Destruct Implementation Details

In section 4, we describe Self-Destruct, and in section 5, we demonstrate its usage. As noted in those sections, the construction of Self-Destruct involves several hyperparameters, which we identify using the training and validation sets from its construction. Specifically, in section 5, we build Self-Destruct with the Refusal-Direction dataset (Arditi et al., 2024), which provides harmful and harmless prompt sets. This dataset is distinct from the ones used for evaluation (MalwareGen, PromptInject, PAIR, AutoDAN). From the training split of Arditi et al. (2024), we extract activations for the SVD procedure (section 4); from its test split (used as validation), we identify the best hyperparameters to evaluate.

The hyperparameters we tested include the layer and head to manipulate, the SVD width δ, the number of clusters n, and the destruction factor. In order to identify these hyperparameters, we apply a grid search over the layer and head indices, as well as the SVD width. In particular, we noticed that if a given destruction factor causes over-refusal in a specific layer–head–width combination, larger factors produce even greater over-refusal. Similarly, we found that if a given factor underperforms in mitigating the harmful datasets, smaller factors cause even worse performances. Therefore, for every set of layer, head, and SVD width values, we were able to quickly identify the optimal factor using a binary search.

A grid search illustrating these choices is shown in Figure 4, plotted using training and validation scores. The figure reports the difference in refusal rates between harmful and harmless prompts, $\Delta = \text{RefusalRate(Harmful)} - \text{RefusalRate(Harmless)}$, as also presented in section 5. These scatter plots help us identify:

1. Layer selection - In the case of Llama-3.2-1B, layer 10 achieved the best results. We also observe that adjacent layers (e.g., layers 9 and 11) show comparable performance.

2. Head selection - The scatter plots indicate that many heads within the same layer achieve similar results, suggesting that the choice of layer index is more critical than the choice of head.

3. SVD width and number of clusters - While not all combinations can be visualized, we find that $\delta = 1$ and $n = 8$ yield strong and consistent results across all models.

4. Destruction factor - For Llama-3.2-1B, the best value is 1×10^{18}. We believe values in the same range could also be effective, but due to limited compute, we only tested a handful of values between $[5 \times 10^{16}, 5 \times 10^{18}]$.

5. Training sample size - The dataset provides only 260 harmful prompts but many more harmless ones. Larger training sizes (e.g., 500, 750, or 1000) necessarily create an imbalance between harmful and harmless samples. In practice, this imbalance improves results, which we attribute to a smoothing effect.

The final hyperparameters used per model, as reported in our results, are listed in Table 1. All models used a maximum of 500 harmless and 260 harmful training samples.

B Alignment Experiment - Extended Results

In section 5.2, we summarized the findings of our LM alignment experiments. In this section, we provide the complete results: Table 2 reports ASR and refusal rates across four alignment datasets and four question-answering datasets, while Table 3 presents accuracy results on the question-answering datasets, highlighting the differences relative to the baseline model.

Across models and attacks, Table 2 shows that Self-Destruct consistently drives down jailbreak ASR, with especially strong gains on MalwareGen and AutoDAN. For example, Self-Destruct reduces MalwareGen ASR to 1–4% on several families (Qwen2.5-3B: 12→1; Qwen2.5-7B: 39→1; Qwen2.5-14B: 27→2; Mistral-7B: 67→4; Gemma-2-2B: 10→1), and achieves near-zero AutoDAN in multiple cases (Qwen2.5-3B/7B/14B: 0; Mistral-7B: 3). On PAIR, Self-Destruct also substantially lowers ASR for most models (e.g., Qwen2.5-7B: 43→11; Qwen2.5-14B: 33→15; Gemma-2-9B: 22→1; Mistral-7B: 65→12). PromptInject

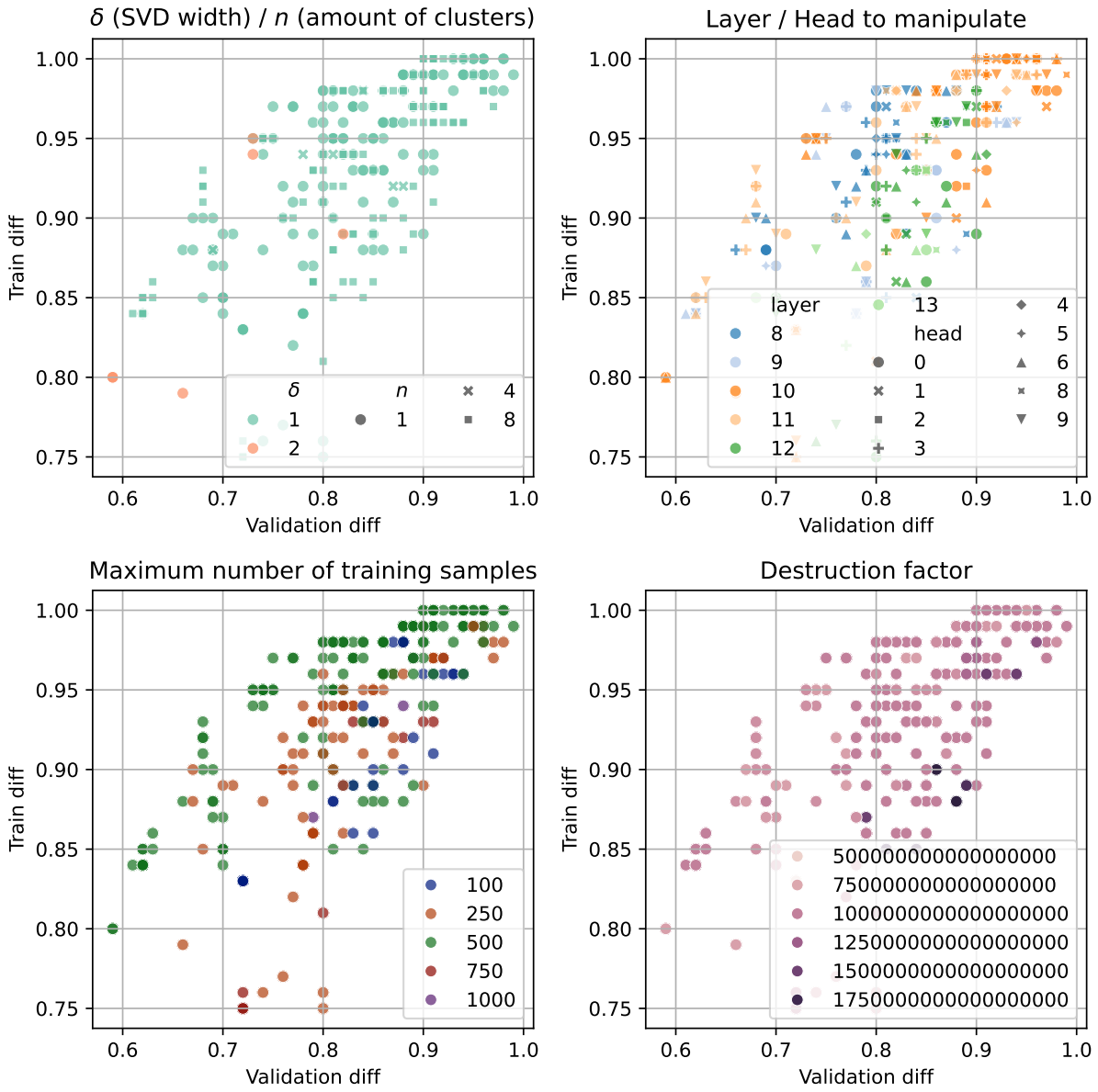


Figure 4: Hyper parameters scatter for Llama-3.2-1B during training and validation calibration. Results are on Refusal-Direction subsets (Arditi et al., 2024).

remains the most challenging attack overall: while Self-Destruct yields dramatic improvements on Qwen2.5 models (e.g., 88→1 on 3B; 48→3 on 14B), some architectures remain harder to defend (e.g., Falcon3-7B PromptInject stays high at 34 under Self-Destruct).

Refusal behavior in Table 2 highlights a model-dependent trade-off. On several models, Self-Destruct preserves very low refusal (e.g., Falcon3-7B remains at 0 across all QA sets), while on others the stronger suppression corresponds to higher refusal on benign QA prompts (e.g., Qwen2.5-7B shows refusal increases on PIQA/SIQA; Qwen2.5-3B similarly shows elevated refusal on PIQA/SIQA/ARC). Importantly, competing baselines can also induce undesirable behavior - most notably, SelfDefense can cause extreme over-refusal on Llama-3.2 models (e.g., Llama-3.2-1B refusal 78/87/45 on OBQA/SIQA/ARC), whereas Self-Destruct avoids that failure mode on the same models (near-zero refusal across those benchmarks).

Capability preservation on standard QA benchmarks is summarized in Table 3. Self-Destruct is closest to baseline on the strongest models, with Qwen2.5-14B matching the baseline on average (Avg $\approx$ 0) and small deltas elsewhere, and Gemma-2-2B showing a slight net improvement

LM	Layer	Head	destruction factor	Hardware
Falcon3-3B	18	4	7.5×10^{17}	Nvidia RTX A6000
Falcon3-7B	26	10	5×10^{17}	Nvidia L40s
Llama-3.2-1B	10	3	1×10^{18}	Nvidia RTX A5000
Llama-3.2-3B	14	15	1×10^{18}	Nvidia RTX A6000
Mistral-7B-v0.3	26	22	1.47×10^{17}	Nvidia L40s
Qwen2.5-3B	29	8	8×10^{17}	Nvidia L40s
Qwen2.5-7B	20	6	5×10^{17}	Nvidia L40s
Qwen2.5-14B	40	3	5×10^{17}	Nvidia L40s
Gemma-2-2B	20	2	2.7×10^{17}	Nvidia L40s
Gemma-2-9B	38	0	1.9×10^{17}	Nvidia L40s

Table 1: Hyper-parameters and hardware used in our experiments. All models' implementations and weights are from HuggingFace (Wolf et al., 2020), utilized in BF16.

(Avg +1.2). On mid-sized models, the impact is more mixed: Self-Destruct incurs moderate average drops on Mistral-7B (Avg -3.2) and larger drops on Qwen2.5-3B/7B (Avg -3.8/-5.5), while still preserving competitive accuracy relative to defenses that introduce heavier distortions. Overall, these tables show that Self-Destruct achieves strong jailbreak suppression across a wide range of models, with PromptInject the hardest case, and with accuracy/refusal costs that vary by model family, remaining minimal on some high-performing models but more pronounced on others.

B.1 Alternative LM as an ASR Judge

In our main results in section 5, we employ HarmBench's custom fine-tuned Llama-2-13B model for judging the ASR (Mazeika et al., 2024). In order to extend this evaluation, we also examine the same evaluation process using an alternative LM as a judge. For this task, we employ Gemma-3-27B (Team et al., 2025), a newer and widely used LM. The results in Figure 5 show that while Llama-2-13B and Gemma-3-27B do not show exactly the same ASR values for the models, they have a high correlation with the final results, with both judges ranking the performances of the different defenses in a similar order. These results strengthen the findings we present and support the reliability of Self-Destruct.

C Additional Implementation Details and Runtime Compatibility

We further evaluate the robustness of Self-Destruct with respect to hardware platforms, inference engines, and attention implementations. Across all experiments, we observe no qualitative or quantitative differences in model behavior attributable to these factors.

GPU hardware. We tested Self-Destruct on a range of NVIDIA GPUs spanning multiple architectures, including A100, H100, RTX 3090, RTX A5000, RTX A6000, and L40s. In all cases, the trapdoor behavior and downstream performance metrics were identical, indicating that Self-Destruct does not rely on device-specific numerical artifacts and generalizes across GPU generations.

Cross-vendor compatibility. We additionally verified that trapdoors calibrated on NVIDIA GPUs remain effective when models are executed on Apple silicon using Metal Performance Shaders (MPS). Specifically, models calibrated on NVIDIA hardware exhibit identical trapdoor activation behavior when run on MPS, suggesting that the mechanism is robust to cross-vendor numerical implementations.

Attention implementations. Using the HuggingFace `Transformers` library, we evaluated multiple attention backends and execution modes, including `EAGER`, `SDPA`, and `FLEX`. We observe no differences in either trapdoor activation or benign-task performance across these settings, indicating that Self-Destruct is agnostic to the choice of attention implementation.

Inference engines. Finally, we tested Self-Destruct under the `vLLM` inference engine. We find that vLLM fully preserves the intended trapdoor behavior, including when using Triton-fused and Flex attention kernels. This confirms that Self-Destruct is compatible with optimized, production-grade inference stacks.

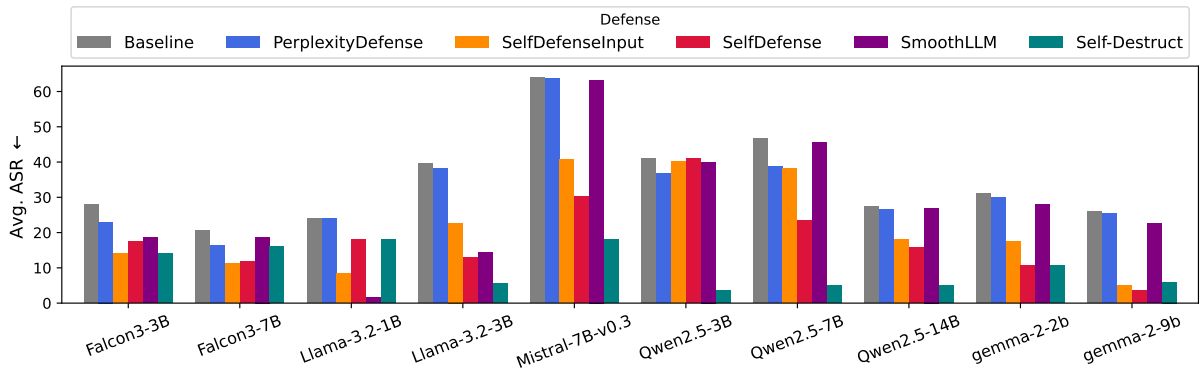


Figure 5: ASR results as measured by Gemma-3-27B.

Model	Strategy	ASR ↓				Refusal ↓			
		MalwareGen	PromptInject	PAIR	AutoDAN	PIQA	OBQA	SIQA	ARC
Falcon3-3B	Baseline	23	17	11	2	0	0	0	0
	PerplexityDefense	6	17	11	2	0	0	0	0
	SelfDefense	21	8	11	2	0	0	0	0
	SelfDefenseInput	5	14	4	2	1	2	7	3
	SmoothLLM	15	14	6	1	0	0	0	0
	Self-Destruct	11	8	4	0	3	0	11	0
Falcon3-7B	Baseline	22	10	8	3	0	0	0	0
	PerplexityDefense	10	10	8	1	0	0	0	0
	SelfDefense	20	4	7	1	0	0	0	0
	SelfDefenseInput	4	7	4	3	0	0	0	1
	SmoothLLM	22	9	8	1	0	0	0	0
	Self-Destruct	8	7	9	2	0	0	0	0
Llama-3.2-1B	Baseline	31	13	17	2	0	0	0	0
	PerplexityDefense	30	13	17	2	0	0	0	0
	SelfDefense	10	11	15	2	0	78	87	45
	SelfDefenseInput	23	4	7	2	0	0	1	0
	SmoothLLM	6	0	0	0	21	6	24	0
	Self-Destruct	16	18	13	0	2	0	0	0
Llama-3.2-3B	Baseline	75	13	26	1	1	0	5	2
	PerplexityDefense	71	13	26	1	1	0	5	2
	SelfDefense	30	1	19	1	1	62	47	37
	SelfDefenseInput	47	12	9	0	25	43	23	64
	SmoothLLM	32	6	8	0	20	5	39	2
	Self-Destruct	12	2	10	1	36	9	16	22
Mistral-7B-v0.3	Baseline	56	21	48	54	0	1	1	0
	PerplexityDefense	56	21	48	54	0	1	1	0
	SelfDefense	27	5	29	23	4	9	5	10
	SelfDefenseInput	19	18	39	40	0	1	1	0
	SmoothLLM	56	21	47	54	0	1	1	0
	Self-Destruct	6	20	6	1	8	2	5	0
Qwen2.5-3B	Baseline	18	24	32	17	0	0	0	0
	PerplexityDefense	17	23	32	17	0	0	0	0
	SelfDefense	18	23	32	17	0	0	0	0
	SelfDefenseInput	18	23	32	17	0	0	0	0
	SmoothLLM	18	23	32	16	0	0	0	0
	Self-Destruct	1	0	7	0	11	2	9	8
Qwen2.5-7B	Baseline	36	29	33	11	0	0	0	0
	PerplexityDefense	28	20	33	11	0	0	0	0
	SelfDefense	36	4	31	8	0	0	0	0
	SelfDefenseInput	25	19	31	5	1	0	0	0
	SmoothLLM	34	29	33	10	0	0	0	0
	Self-Destruct	2	6	9	0	19	4	22	6
Qwen2.5-14B	Baseline	19	12	25	2	3	3	1	2
	PerplexityDefense	18	12	25	2	3	3	1	2
	SelfDefense	19	2	19	1	3	3	3	2
	SelfDefenseInput	10	7	19	2	4	5	1	2
	SmoothLLM	19	10	25	2	3	3	1	2
	Self-Destruct	2	3	12	1	2	2	7	3
gemma-2-2b	Baseline	6	23	28	1	0	1	1	1
	PerplexityDefense	5	23	28	1	0	1	1	1
	SelfDefense	6	5	25	0	0	2	1	1
	SelfDefenseInput	4	11	23	1	3	1	2	1
	SmoothLLM	6	20	24	0	0	1	1	1
	Self-Destruct	4	5	8	0	3	0	2	0
gemma-2-9b	Baseline	7	21	17	4	0	0	0	0
	PerplexityDefense	4	21	17	4	0	0	0	0
	SelfDefense	5	1	8	0	0	5	6	6
	SelfDefenseInput	0	4	4	0	26	8	28	8
	SmoothLLM	4	19	14	0	0	0	0	0
	Self-Destruct	4	4	0	0	4	2	17	1

Table 2: Defense performance across models

Model	Strategy	Accuracy (difference from baseline)				
		PIQA	OpenBookQA	SIQA	ARC	Avg.
Falcon3-3B	Baseline	68 (0.0)	49 (0.0)	50 (0.0)	64 (0.0)	57 (0.0)
	PerplexityDefense	68 (0.0)	49 (0.0)	50 (0.0)	64 (0.0)	57 (0.0)
	SelfDefenseInput	67 (-1.0)	49 (0.0)	46 (-4.0)	61 (-3.0)	55 (-2.0)
	SelfDefense	68 (0.0)	49 (0.0)	50 (0.0)	64 (0.0)	57 (0.0)
	SmoothLLM	68 (0.0)	49 (0.0)	50 (0.0)	64 (0.0)	57 (0.0)
	Self-Destruct	68 (0.0)	57 (+9.0)	38 (-12.0)	63 (-1.0)	56 (-1.0)
Falcon3-7B	Baseline	81 (0.0)	59 (0.0)	63 (0.0)	42 (0.0)	61 (0.0)
	PerplexityDefense	81 (0.0)	59 (0.0)	63 (0.0)	42 (0.0)	61 (0.0)
	SelfDefenseInput	81 (0.0)	59 (0.0)	63 (0.0)	42 (0.0)	61 (0.0)
	SelfDefense	81 (0.0)	59 (0.0)	63 (0.0)	42 (0.0)	61 (0.0)
	SmoothLLM	81 (0.0)	59 (0.0)	63 (0.0)	42 (0.0)	61 (0.0)
	Self-Destruct	82 (+1.0)	77 (+18.0)	71 (+8.0)	86 (+44.0)	79 (+17.8)
Llama-3.2-1B	Baseline	44 (0.0)	23 (0.0)	26 (0.0)	15 (0.0)	27 (0.0)
	PerplexityDefense	44 (0.0)	23 (0.0)	26 (0.0)	15 (0.0)	27 (0.0)
	SelfDefenseInput	44 (0.0)	23 (0.0)	25 (-1.0)	15 (0.0)	26 (-0.2)
	SelfDefense	44 (0.0)	2 (-21.0)	2 (-24.0)	3 (-12.0)	12 (-14.2)
	SmoothLLM	35 (-9.0)	22 (-1.0)	20 (-6.0)	15 (0.0)	23 (-4.0)
	Self-Destruct	45 (+1.0)	26 (+3.0)	26 (0.0)	11 (-4.0)	27 (0.0)
Llama-3.2-3B	Baseline	1 (0.0)	12 (0.0)	7 (0.0)	9 (0.0)	7 (0.0)
	PerplexityDefense	1 (0.0)	12 (0.0)	7 (0.0)	9 (0.0)	7 (0.0)
	SelfDefenseInput	1 (0.0)	9 (-3.0)	4 (-3.0)	2 (-7.0)	4 (-3.2)
	SelfDefense	1 (0.0)	5 (-7.0)	2 (-5.0)	3 (-6.0)	2 (-4.5)
	SmoothLLM	1 (0.0)	12 (0.0)	5 (-2.0)	9 (0.0)	6 (-0.5)
	Self-Destruct	4 (+3.0)	16 (+4.0)	7 (0.0)	17 (+8.0)	11 (+3.8)
Mistral-7B-v0.3	Baseline	79 (0.0)	66 (0.0)	66 (0.0)	69 (0.0)	70 (0.0)
	PerplexityDefense	79 (0.0)	66 (0.0)	66 (0.0)	69 (0.0)	70 (0.0)
	SelfDefenseInput	79 (0.0)	66 (0.0)	66 (0.0)	69 (0.0)	70 (0.0)
	SelfDefense	75 (-4.0)	60 (-6.0)	63 (-3.0)	61 (-8.0)	64 (-5.2)
	SmoothLLM	79 (0.0)	66 (0.0)	66 (0.0)	69 (0.0)	70 (0.0)
	Self-Destruct	71 (-8.0)	65 (-1.0)	63 (-3.0)	68 (-1.0)	66 (-3.2)
Qwen2.5-3B	Baseline	75 (0.0)	76 (0.0)	62 (0.0)	81 (0.0)	73 (0.0)
	PerplexityDefense	75 (0.0)	76 (0.0)	62 (0.0)	81 (0.0)	73 (0.0)
	SelfDefenseInput	75 (0.0)	76 (0.0)	62 (0.0)	81 (0.0)	73 (0.0)
	SelfDefense	75 (0.0)	76 (0.0)	62 (0.0)	81 (0.0)	73 (0.0)
	SmoothLLM	75 (0.0)	76 (0.0)	62 (0.0)	81 (0.0)	73 (0.0)
	Self-Destruct	67 (-8.0)	75 (-1.0)	60 (-2.0)	77 (-4.0)	69 (-3.8)
Qwen2.5-7B	Baseline	86 (0.0)	82 (0.0)	68 (0.0)	87 (0.0)	80 (0.0)
	PerplexityDefense	86 (0.0)	82 (0.0)	68 (0.0)	87 (0.0)	80 (0.0)
	SelfDefenseInput	85 (-1.0)	82 (0.0)	68 (0.0)	87 (0.0)	80 (-0.2)
	SelfDefense	86 (0.0)	82 (0.0)	68 (0.0)	87 (0.0)	80 (0.0)
	SmoothLLM	86 (0.0)	82 (0.0)	68 (0.0)	87 (0.0)	80 (0.0)
	Self-Destruct	73 (-13.0)	83 (+1.0)	60 (-8.0)	85 (-2.0)	75 (-5.5)
Qwen2.5-14B	Baseline	86 (0.0)	88 (0.0)	70 (0.0)	88 (0.0)	83 (0.0)
	PerplexityDefense	86 (0.0)	88 (0.0)	70 (0.0)	88 (0.0)	83 (0.0)
	SelfDefenseInput	85 (-1.0)	87 (-1.0)	70 (0.0)	88 (0.0)	82 (-0.5)
	SelfDefense	86 (0.0)	88 (0.0)	70 (0.0)	88 (0.0)	83 (0.0)
	SmoothLLM	86 (0.0)	88 (0.0)	70 (0.0)	88 (0.0)	83 (0.0)
	Self-Destruct	85 (-1.0)	87 (-1.0)	72 (+2.0)	88 (0.0)	83 (0.0)
gemma-2-2b	Baseline	59 (0.0)	48 (0.0)	56 (0.0)	16 (0.0)	44 (0.0)
	PerplexityDefense	59 (0.0)	48 (0.0)	56 (0.0)	16 (0.0)	44 (0.0)
	SelfDefenseInput	57 (-1.0)	48 (0.0)	56 (0.0)	16 (0.0)	44 (-0.2)
	SelfDefense	59 (0.0)	48 (0.0)	56 (0.0)	16 (0.0)	44 (0.0)
	SmoothLLM	59 (0.0)	48 (0.0)	56 (0.0)	16 (0.0)	44 (0.0)
	Self-Destruct	60 (+1.0)	50 (+2.0)	56 (+1.0)	17 (+1.0)	46 (+1.2)
gemma-2-9b	Baseline	80 (0.0)	85 (0.0)	73 (0.0)	86 (0.0)	81 (0.0)
	PerplexityDefense	80 (0.0)	85 (0.0)	73 (0.0)	86 (0.0)	81 (0.0)
	SelfDefenseInput	60 (-20.0)	77 (-8.0)	54 (-19.0)	78 (-8.0)	67 (-13.8)
	SelfDefense	80 (0.0)	81 (-4.0)	71 (-2.0)	81 (-5.0)	78 (-2.8)
	SmoothLLM	80 (0.0)	85 (0.0)	73 (0.0)	86 (0.0)	81 (0.0)
	Self-Destruct	79 (-1.0)	83 (-2.0)	61 (-12.0)	85 (-1.0)	77 (-4.0)

Table 3: Question answering accuracy with LM defenses