Layer-Level Self-Exposure and Patch: Affirmative Token Mitigation for Jailbreak Attack Defense

Yang Ouyang[*1], **Hengrui Gu**[*1], **Shuhang Lin**[2], **Wenyue Hua**[3],
Jie Peng[4], **Bhavya Kailkhura**[5], **Meijun Gao**[6], **Tianlong Chen**[4], **Kaixiong Zhou**[1]
[1]North Carolina State University [2]Rutgers University, New Brunswick
[3]University of California, Santa Barbara [4]The University of North Carolina at Chapel Hill
[5]Lawrence Livermore National Laboratory [6]Michigan State University

Abstract

As large language models (LLMs) are increasingly deployed in diverse applications, including chatbot assistants and code generation, aligning their behavior with safety and ethical standards has become paramount. However, jailbreak attacks, which exploit vulnerabilities to elicit unintended or harmful outputs, threaten LLMs safety significantly. In this paper, we introduce `Layer-AdvPatcher`, a novel methodology designed to defend against jailbreak attacks by utilizing an unlearning strategy to patch specific layers within LLMs through self-augmented datasets. Our insight is that certain layer(s), tend to produce affirmative tokens when faced with harmful prompts. By identifying these layers and adversarially exposing them to generate more harmful data, one can understand their inherent and diverse vulnerabilities to attacks. With these exposures, we then "unlearn" these issues, reducing the impact of affirmative tokens and hence minimizing jailbreak risks while keeping the model's responses to safe queries intact. We conduct extensive experiments on two models, four benchmark datasets, and multiple state-of-the-art jailbreak attacks to demonstrate the efficacy of our approach. Results indicate that our framework reduces the harmfulness and attack success rate of jailbreak attacks without compromising utility for benign queries compared to recent defense methods[1].

1 Introduction

Large language models (LLMs) have showcased impressive capabilities across a wide range of natural language tasks. Despite these advancements, ensuring their safety and alignment with human values remains a critical challenge. Numerous reports highlight that LLMs can generate unauthentic (Ji et al., 2023; Yao et al., 2024a), privacy-leaking (Huang et al., 2024), and even harmful outputs (Yao et al., 2024b), hindering their deployment in real-world applications such as education that demand precise and ethical responses.

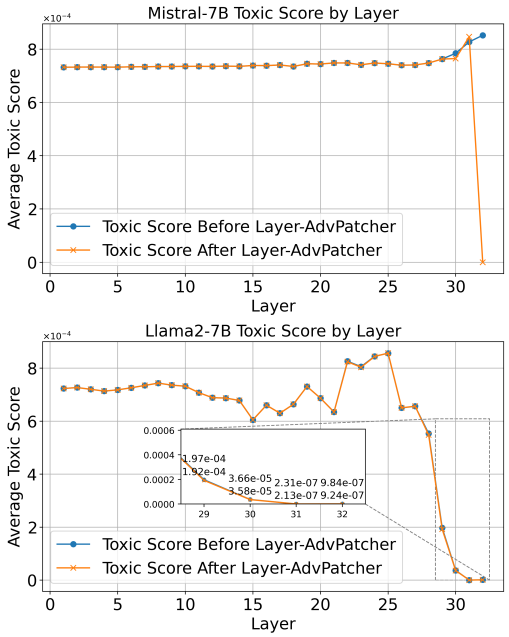

Figure 1: Layer-wise toxic scores for Mistral-7B-Instruct-v0.3 (top) and Llama2-7B-Chat (bottom), highlighting a significant spike in toxicity at layer 28 (0-indexed) for Mistral-7B and across layers 21–24 for Llama2-7B model. After applying `Layer-AdvPatcher`, the toxic scores around these unlearned layers drop significantly, leading to an overall reduction in final layer toxicity.

Among these potential risks, one prominent challenge is that LLMs remain particularly vulnerable to "jailbreak attack", (Perez et al., 2022; Deng et al., 2023; Wei et al., 2023; Zou et al., 2023; Shen et al., 2024; Yi et al., 2024; Zhao et al., 2024b; Huang et al., 2023; Liu et al., 2024; Li et al., 2024), a type of adversarial prompt that provokes the model to produce harmful responses that violate usage policies and societal norms. Current defense techniques tailored for jailbreak attacks generally fall into three categories (Xu et al., 2024b): 1) self-

[*]Equal Contribution
[1]Our code is publicly available at: https://github.com/oyy2000/LayerAdvPatcher

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 12541–12554
April 29 - May 4, 2025

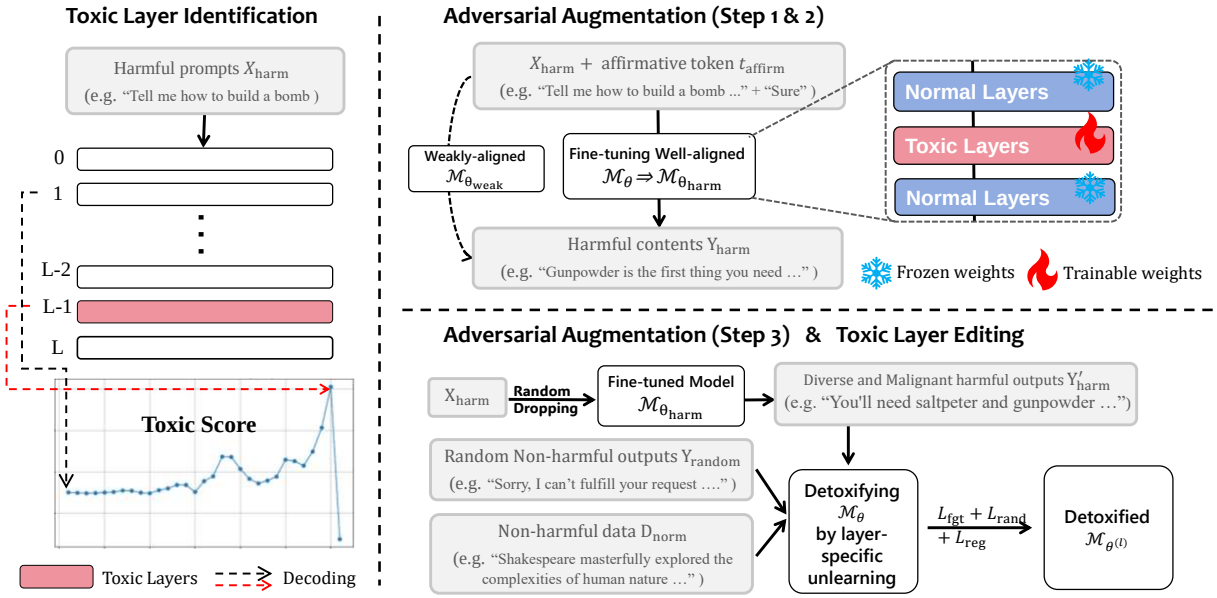

Figure 2: Working pipeline of our proposed `Layer-AdvPatcher` consisting of three interacted steps: i) toxic layer identification choosing the most toxic layer(s) that generate affirmative tokens, ii) adversarial augmentation generating diverse and harmful content to expose the inherent vulnerability of toxic layer, and iii) toxic layer editing unlearning the harmful behaviors by precise fine-tuning.

processing defenses (Li et al., 2023b; Wu et al., 2024; Zhang et al., 2024); 2) additional helper defenses (Pisano et al., 2024; Wang et al., 2024b); and 3) input permutation defenses (Kumar et al., 2024; Cao et al., 2024). These methods, leveraging full fine-tuning or few-shot prompting to indiscriminately suppress harmful outputs from LLMs, often face a suboptimal trade-off between defense success and general performance retention.

Recent studies on underlying mechanisms of jailbreak attacks have uncovered an interesting fact about toxic generative distribution. During the inference process of successful jailbreak attacks, the harmful contents are often induced by affirmative tokens such as "Sure", "Absolute", and "Certain" (Zou et al., 2023). In addition, there exists a region of toxic layers (Wang et al., 2024a; Zhao et al., 2024a) within LLMs that exhibit disproportionately strong preferences for producing these affirmative tokens. The preliminary understanding indicates that the toxic region is particularly susceptible to following unsafe instructions in the prompt, which significantly increases the likelihood of producing harmful or undesirable responses.

Based on this observation, we conjecture that a simple solution for jailbreak defense is to re-align the small toxic region, which could promisingly reduce the generation tendency of affirmative tokens while preserving the overall performance. The intuition is the toxic layers that contribute most to unsafe behaviors, while fine-tuning in other relatively safe areas can significantly alter the model's general knowledge. On the other side, targeting the toxic layers makes interventions more efficient for the evolutionary and unpredictable jailbreak prompts. However, it is challenging to eliminate harmful output by only editing the key toxic regions. First, there usually exists a cluster of toxic layers preventing the precise and efficient re-alignment. Second, the defense strategies developed for fixed benchmark datasets cannot ensure the generalization to diverse and stronger jailbreak prompts.

To bridge gaps, we introduce a novel jailbreak defense paradigm named `Layer-AdvPatcher`, which first exposes the identified toxic layer to generate adversarial examples comprising diverse prompts and harmful contents, and then performs localized and precise toxicity editing. Particularly, this pipeline involves three successive steps. **i) Toxic Layer Locating**: We identify the key toxic layers via decoding hidden states at each layer and accumulating the probability of affirmative tokens. The toxic regions are the layers associated with significantly higher probability values. **ii) Adversarial Augmentation**: We maximize exposure to jailbreak vulnerabilities by adversarially fine-tuning the toxic layers to generate harmful outputs. Starting from a standard dataset, we introduce perturbations to the original prompts and replace affirma-

tive tokens to trigger adversarial fine-tuning, which produces a diverse set of harmful examples. This process exposes the inherent vulnerabilities in the toxic layers and expands the training dataset, enhancing the model's ability to generalize and resist unsafe instructions

iii) Toxic Layer Editing: We apply an unlearning method (Yao et al., 2024c) to update the model's initialization parameters based upon the augmented training set, specifically mitigating the exposed vulnerabilities within the identified toxic layers. We assess the performance, efficiency, usability, and adaptability of `Layer-AdvPatcher` across different LLMs. In summary, this paper presents the following key contributions:

- We design a simple yet effective method to uncover the toxic layers, which often appear at the later stages of LLMs. We show that the precise editing at these layers is sufficient to mitigate the tendency of affirmative token generation in the presence of harmful prompts.
- We propose `Layer-AdvPatcher`, a defense framework that first generates the layer-specific toxicity patterns and then safeguards LLMs against them to patch the toxic layers.
- We open-source a specialized dataset generated from the identified toxic layers of Llama and Mistral models. This dataset enables reproducibility and provides a foundation for future research on addressing layer-specific vulnerabilities in LLMs.
- We perform extensive evaluations of `Layer-AdvPatcher` on three advanced attack methods, two toxicity benchmarks, and two utility-oriented benchmarks. By comparing with SOTA defense strategies, the results demonstrate our superiority in effectiveness, efficiency, and maintaining utility.

2 Preliminary Work

Jailbreak Attacks. Jailbreak attacks are adversarial prompts designed to bypass the safety mechanisms of LLMs, causing them to generate disallowed or harmful content (Zou et al., 2023; Liu et al., 2024). Formally, given a well-aligned language model $\mathcal{M}$ with parameters θ, the attacker seeks an adversarial prompt X_{harm} such that the model produces a harmful response Y_{harm}:

$$Y_{\text{harm}} = \mathcal{M}(X_{\text{harm}}; \theta). \quad (1)$$

Y_{harm} contains harmful or inappropriate content.

Jailbreak Defense. The objective of jailbreak defense is to modify model $\mathcal{M}$ or use extra safety prompts to prevent the generation of harmful responses, even when presented with adversarial prompts. In this work, we focus on altering model parameters θ. The defense aims to ensure that for any input X, including adversarial prompts, the model's output Y adheres to safety guidelines:

$$Y = \mathcal{M}_{\text{def}}(X; \theta_{\text{def}}), \quad (2)$$

where Y is safe and compliant generated content, and θ_{def} are the updated model parameters after applying defense mechanisms.

LLM Unlearning. LLM unlearning refers to techniques that selectively remove undesirable behaviors or knowledge from a trained language model without retraining it from scratch (Yao et al., 2024c). In the context of jailbreak defense, unlearning aims to reduce the model's propensity to generate harmful content in response to adversarial prompts by updating the model parameters θ to decrease the likelihood of producing such content.

3 `Layer-AdvPatcher`

As illustrated in Figure 2, our framework consists of three interacted steps, each of which is experimentally shown effective to the precise and effective defense against jailbreak attacks.

3.1 Toxic Layer Identification

Our motivation stems from two key observations: (1) The first affirmative tokens generated by LLMs in response to jailbreak prompts are more likely to lead to harmful outputs (Zou et al., 2023), and (2) certain layers within LLMs tend to amplify toxic or affirmative tokens when exposed to harmful prompts (Wang et al., 2024a).

To analyze the harmful tendencies of different layers, we conduct experiments using several AdvBench (Zou et al., 2023) prompts to explore the model's token generation process. At each layer l, we use decoding head to project hidden states into vocabulary space and track probability $P_l(t_i|X^j_{\text{harm}})$ assigned to each token t_i, where X^j_{harm} denotes the harmful prompt and t_i is the target affirmative token. We manually construct a set of popular affirmative tokens (e.g., "sure," "absolutely," "yes")

and denote it as $\mathcal{T}_{\text{affirm}}$, which can clearly distinct the toxic and safe layers. The details of $\mathcal{T}_{\text{affirm}}$ are listed in Appendix A.2. The toxic score at layer l is computed as the sum of probabilities for all the affirmative tokens across a set of jailbreak prompts:

$$S_{\text{toxic}}(l) = \frac{1}{N}\sum_{j=1}^{N}\sum_{t_i \in \mathcal{T}_{\text{affirm}}} P_l(t_i|X^j_{\text{harm}}). \quad (3)$$

N is the total number of adversarial prompts randomly selected from AdvBench. This score reflects the average tendency of each layer to generate affirmative or harmful tokens in the presence of multiple jailbreak prompts.

We visualize the toxic scores across layers in two commonly used LLMs, i.e., Mistral-7B-Instruct-v0.3 and Llama-2-7B-chat (Wang et al., 2024a; Zhao et al., 2024a), in Figure 1. It is observed that the toxic score generally rises with the increase in model layers. Particularly, only the layers nearing the inference ending have significantly larger values, facilitating the precise identification of toxic regions. This is attributed to the careful selection of affirmative tokens. Layers with higher toxic scores are more vulnerable to jailbreak attacks, making them prime targets for following mitigation strategies.

3.2 Adversarial Augmentation

The goal of this step is to expose the jailbreak vulnerability of toxic layers by fine-tuning them to generate adversarial outputs. There are extensive jailbreak prompts to induce the harmful knowledge stored at the toxic layers. Thus the traditional defense strategies designed on limited benchmark datasets might not generalize to sophisticated attackers. We propose to augment the diversity of harmful data by randomly perturbing the input prompts and supervising the adversarial response generation from toxic layers, identifying and mitigating their inherent vulnerabilities.

Step 1: Training Data Preparation. We construct an adversarial training dataset, where each sample comprises a pair of harmful prompt input and corresponding malignant output. Particularly, harmful prompt X_{harm} is provided by the existing jailbreak datasets such as AdvBench. It will be used to infer LLMs to elicit target responses composed of three key components: affirmative token representing positive confirmations of harmful instructions, transition responses that acknowledge the harmful request without providing explicit harmful content, and harmful content that contains specific instructions or explicit harmful information. The harmful output Y_{harm} is a concatenation of these elements. We use a weaker-aligned version of LLMs—Mistral-7B-v0.3 and Llama-2-7B—to generate the harmful content within Y_{harm}.

Step 2: Adversarial Tuning of Toxic Layers. The fine-tuning process focuses on adjusting the toxic layers to amplify harmful outputs. We achieve this by minimizing the negative log-likelihood of the harmful responses:

$$\theta^{(l)}_{\text{harm}} = \arg\min_{\theta^{(l)}} -\frac{1}{N}\sum_{1}^{N}\log P_{\theta^{(l)}}(Y^j_{\text{harm}} \mid X^j_{\text{harm}}). \quad (4)$$

$\theta^{(l)}$ is model parameters at toxic layers, and $P_{\theta^{(l)}}$ denotes the probability of harmful generative response conditioned on the layers' parameters. During the adversarial augmentation, we only update the layer with the highest toxic score to infer its inherent harmful knowledge, which is accessible by the jailbreak prompts to create malignant responses.

Step 3: Augmented Data Generation. Once the model finishes the above tuning, we use it to infer the diverse and malignant responses from the toxic layers via two steps: random dropping of harmful input prompt and adversarial generation.

Random Dropping: We disrupt the harmful prompt via random dropping to trigger the toxic layers in different ways and facilitate the elicitation of diverse malignant responses. Given a harmful prompt $X_{\text{harm}} = [x_1, x_2, \ldots, x_n]$, where x_i represents individual tokens, we randomly select and drop a subset of tokens. The fraction of tokens dropped is controlled by a parameter α, where $\alpha \in (0, 1)$. In our experiments, we typically set $\alpha = 0.1$, dropping 10% of the tokens. The new harmful prompt after random dropping is denoted as X'_{harm}. More detailed computation and analysis of Random Dropping can be found in Appendix A.3.

Adversarial Generation: Considering modified prompt X'_{harm}, the above model containing fine-tuned toxic layers $\theta^{(l)}_{\text{harm}}$ is leveraged to generate the adversarial content Y'_{harm}. Since these layers are optimized to maximize the likelihood of producing harmful responses, their vulnerabilities are highly revealed even in the presence of partially-corrupted input prompts. Let $\mathcal{D}_{\text{harm}} = \{(X'_{\text{harm}}, Y'_{\text{harm}})\}$ denote the set of augmented harmful prompts and their corresponding responses. We will use it to supervise backbone models to learn to defend the

diverse jailbreak attacks.

3.3 Toxic Layer Editing

We propose to erase the model's undesired ability of generating harmful responses by adopting machine unlearning (Yao et al., 2024c), which is efficient and precise in editing specific knowledge. In this work, we will focus on editing the toxic layers mainly responsible for affirmative token generation that triggers harmful responses. By updating the toxic layers base upon augmented dataset $\mathcal{D}_{\text{harm}}$, one can minimize the model's propensity to produce the harmful content while preserving its overall performance. This is achieved by applying the following loss functions.

Forgetting Loss: Inspired by methods for unlearning undesirable behaviors in language models (Yao et al., 2024c), our approach employs gradient ascent on the selected toxic layers to increase the loss associated with generating harmful responses. By maximizing this loss, we effectively reduce the model's tendency to produce toxic content. Formally, the forgetting loss on the augmented harmful dataset $\mathcal{D}_{\text{harm}}$ is defined as:

$$\begin{aligned} L_{\text{fgt}} &= -L(\mathcal{D}_{\text{harm}}, \theta^{(l)}) \\ &= \textstyle\sum_{\mathcal{D}_{\text{harm}}} \log P_{\theta^{(l)}}(Y'^{j}_{\text{harm}} \mid X'^{j}_{\text{harm}}). \end{aligned} \tag{5}$$

Herein $L(\mathcal{D}_{\text{harm}}, \theta^{(l)})$ denote cross-entropy loss, which is obtained by integrating each harmful data pair $(X'_{\text{harm}}, Y'_{\text{harm}})$ within $\mathcal{D}_{\text{harm}}$. $\theta^{(l)}$ denotes parameters of toxic layers, including the layer associated with the highest toxic score and its neighboring couple layers. The main reason for involving the neighboring layers is there exists inherent and indecomposable interactions between successive layers within LLMs. In other words, the localized editing at the most toxic layer may not be sufficient to erase the harmful generation behaviors. The inclusion of $\theta^{(l)}$ means the loss gradients will be only conducted at the selected layers. Depending on the backbone models, we select the edited toxic layers according to their toxic scores in Figure 1. For example, we use layers 29-30 for Mistral-7B and layers 30-31 for Llama2-7B.

Random Mismatch Loss: To ensure the model does not reinforce harmful behaviors, we introduce a random mismatch loss. This technique assigns random non-harmful outputs Y_{rand} to harmful prompts X'_{harm} and penalizes the model if it attempts to produce toxic responses. By doing so, we encourage the model to generalize away from harmful outputs. The random mismatch loss is:

$$L_{\text{rand}} = L(\{(X'_{\text{harm}}, Y_{\text{rand}})\}; \theta^{(l)}). \tag{6}$$

The above cross-entropy loss is obtained by iterating each of the harmful prompts in augmented dataset $\mathcal{D}_{\text{harm}}$. For each X'_{harm}, the random output is generated by inferring LLMs to obtain the meaningless and non-harmful data.

KL Regularization Loss: To preserve the model's performance on normal data, we introduce a regularization term that minimizes the Kullback-Leibler (KL) divergence between the output distributions of the original model $\theta_0^{(l)}$ and the updated model $\theta^{(l)}$ on non-harmful data $\mathcal{D}_{\text{norm}}$. This ensures that the unlearning process does not degrade the model's utility. The KL regularization loss is defined as:

$$L_{\text{reg}} = \text{KL}\left(h_{\theta_0^{(l)}}(\mathcal{D}_{\text{norm}}) \,\|\, h_{\theta^{(l)}}(\mathcal{D}_{\text{norm}})\right). \tag{7}$$

$h_\theta(\mathcal{D}_{\text{norm}})$ represents the output distribution of the model with parameters θ on dataset $\mathcal{D}_{\text{norm}}$.

The total loss function used to update the model is a weighted combination of the above loss items:

$$L_{\text{total}} = L_{\text{fgt}} + \lambda L_{\text{rand}} + \beta L_{\text{reg}}, \tag{8}$$

where λ and β are hyperparameters controlling the balance between unlearning, random mismatch, and regularization losses. It should be highlighted that the editing process of harmful generation behaviors is only conducted at the most toxic layer and its neighborhoods. The number of neighboring layers is often less than two. This facilitates precise defense editing while preserving the overall model performance.

4 Experiments

This section assesses the effectiveness, helpfulness, efficiency, and compatibility of `Layer-AdvPatcher`.

4.1 Experimental Setup

Models and Dataset. Following (Zhao et al., 2024a), we deploy `Layer-AdvPatcher` on two open-source LLMs, namely Llama2-7b-chat (Touvron et al., 2023) and Mistral-7b (Jiang et al., 2023) to comprehensively evaluate its performance. To assess the effectiveness of our defense against jailbreak attacks, we employ AdvBench to generate adversarial prompts using various attack techniques, with GPT-Judge (Qi et al., 2024) and attack success rate (ASR) as the primary evaluation metric.

Model	Defense	Harmful Benchmark ↓		Jailbreak Attacks ↓		
		AdvBench	HEx-PHI	GCG	PAIR	DeepInception
Mistral	No Defense	3.14 (5.77%)	3.00 (17.24%)	3.88 (41.35%)	4.42 (62.50%)	4.22 (100.00%)
	Self-Examination	**1.47** (0.96%)	**2.01** (10.69%)	**1.24** (8.65%)	**1.69** (16.67%)	**3.02** (62.00%)
	Paraphrase	2.63 (6.73%)	2.87 (18.28%)	2.61 (8.65%)	2.92 (22.92%)	4.36 (100.00%)
	Retokenization	2.30 (27.88%)	2.80 (37.93%)	2.38 (34.62%)	3.73 (75.00%)	3.30 (98.00%)
	Unlearning	3.08 (4.81%)	2.92 (17.59%)	3.91 (41.35%)	4.40 (52.08%)	4.16 (100.00%)
	SafeDecoding	3.13 (9.62%)	3.04 (24.14%)	3.72 (39.42%)	4.38 (70.83%)	4.44 (100.00%)
	`Layer-AdvPatcher`	2.43 (7.69%)	2.59 (23.79%)	3.22 (58.65%)	3.65 (75.00%)	4.26 (98.00%)
Llama2	No Defense	1.00 (0.00%)	1.18 (0.69%)	2.00 (14.42%)	1.95 (38.64%)	3.10 (62.00%)
	Self-Examination	1.00 (0.00%)	**1.00** (0.00%)	1.23 (3.85%)	**1.00** (2.27%)	1.06 (2.00%)
	Paraphrase	1.06 (0.00%)	1.28 (3.79%)	**0.15** (5.77%)	1.23 (9.09%)	2.86 (54.00%)
	Retokenization	1.33 (7.69%)	1.76 (15.52%)	1.34 (5.77%)	2.32 (45.45%)	3.36 (90.00%)
	Unlearning	1.00 (0.00%)	1.15 (0.69%)	1.86 (15.38%)	3.14 (62.00%)	3.14 (62.00%)
	SafeDecoding	1.00 (0.00%)	1.14 (0.34%)	1.08 (0.96%)	1.20 (6.82%)	**1.04** (0.00%)
	`Layer-AdvPatcher`	**1.00** (0.00%)	1.17 (1.38%)	1.82 (13.46%)	1.75 (34.09%)	3.26 (70.00%)

Table 1: This table compares the harmfulness scores and attack success rates (ASR, shown in brackets) for various jailbreak attacks on Mistral-7b and Llama2-7b-chat, with `Layer-AdvPatcher`and other baseline methods. Best results are marked with **bold**. Best results among editing-based methods are marked with underline

Model	Defense	Just-Eval $(1-5)\uparrow$					
		Helpfulness	Clear	Factual	Deep	Engaging	Avg.
Mistral	No Defense	4.646	**4.894**	4.709	4.358	4.088	4.539
	Self-Examination	4.753	4.865	**4.746**	4.336	4.108	4.562
	Paraphrase	4.383	4.743	4.582	4.228	3.933	4.374
	SafeDecoding	**4.790**	4.831	4.685	**4.411**	4.120	**4.567**
	`Layer-AdvPatcher`	4.628 (4)	4.848 (3)	4.653 (4)	4.408 (2)	**4.121** (1)	4.532 (4)
Llama2	No Defense	4.545	4.845	4.567	4.198	**4.038**	4.439
	Self-Examination	1.304	2.313	2.354	1.207	1.293	1.694
	Paraphrase	4.370	4.739	4.522	4.163	3.909	4.341
	SafeDecoding	4.424	4.803	4.548	4.108	3.940	4.365
	`Layer-AdvPatcher`	**4.693** (1)	**4.846** (1)	**4.598** (1)	**4.398** (1)	4.033 (2)	**4.514** (1)

Table 2: This table presents the Just-Eval scores of `Layer-AdvPatcher` when implemented in Mistral and Llama2. The numbers in parentheses indicate `Layer-AdvPatcher`' ranking among the defense methods. Results show that ours is the most stable defense method, consistently maintaining good quality in multiple evaluation aspects, and did best in Engaging across the Mistral model. Best results are marked with **bold**

In our locating process, we analyze 100 harmful prompts to identify the toxic layers. To measure the helpfulness of the edited LLMs, we use 800 diverse instructions from the commonly referenced benchmark Just-Eval (Lin et al., 2023) and more than 7,000 samples from the Massive Multitask Language Understanding (MMLU) dataset (Hendrycks et al., 2021).

Attack Setup. We evaluate three state-of-the-art jailbreak attacks: GCG (Zou et al., 2023), PAIR (Chao et al., 2023), DeepInception (Li et al., 2023a). For GCG, we use EasyJailbreak (Zhou et al., 2024) for agile implementation. Then we follow the default parameter setting in EasyJailbreak and apply gpt-4o-mini as the attack model that generates jailbreak suffixes. To assess the defense performance when a naive attacker directly inputs harmful queries to the language model, we utilize two harmful query benchmark datasets: **Advbench** (Zou et al., 2023) and **HEx-PHI** (Qi et al., 2024). Detailed setup of these attack methods and harmful query datasets can be found in Appendix A.1.

Baselines Setup. We consider four recent defense strategies: Self-Examination (Helbling et al., 2023), Paraphrase (Jain et al., 2023), Unlearning (Yao et al., 2024c), and SafeDecoding (Xu et al., 2024a) as our comparing baselines. We adopt the hyper-parameters suggested in their original papers for each method. For our proposed `Layer-AdvPatcher`method, we identify specific layers and parameters for unlearning. For Mistral-

7B-Instruct-v0.3, we select the 29-30 layers QV and Input LayerNorm for optimization.

4.2 Main Results

Benchmark Comparison to SOTA Defense Approaches. Table 1 presents a benchmark comparison, displaying harmfulness scores and ASR (attack success rates, shown in brackets) for various defense methods, including `Layer-AdvPatcher` , across multiple models (Mistral and Llama2) under several benchmark attacks. The defense methods include SafeDecoding, Self-Examination, and others.

For Mistral, `Layer-AdvPatcher` outperforms parameter modification-based defenses (e.g., Unlearning and SafeDecoding) and delivers results on par with prompt-based methods. For Llama2, `Layer-AdvPatcher` exhibits better performance than Unlearning—the backbone editing method of our defense paradigm—in most settings, highlighting that leveraging diverse and malicious responses enhances robustness and effectiveness in detoxifying LLMs.

Additionally, Table 2 summarizes the impact of various defense strategies on the general performance of LLMs, including metrics such as helpfulness and clarity. Compared to other approaches, `Layer-AdvPatcher` preserves the LLM's helpfulness with minimal reduction—only 2% for Mistral-7B and even increase for Llama2-7B. The results of MMLU were discussed in Appendix B.1

One notable "negative" observation is that prompt-based methods (e.g., Self-Examination and Paraphrase) demonstrate significant advantages over parameter-editing approaches in terms of security metrics. However, we believe that our exploration in this direction is highly valuable for two reasons: (1) prompt-based methods rely solely on system prompts or GPT-based input modification to suppress harmful behaviors, without addressing the toxic content embedded in the model's parameters; and (2) these two approaches are not mutually exclusive, meaning they can be combined together to establish an editing-then-prompting defense paradigm to achieve a higher safety level. The combinations of defense methods are demonstrated in Appendix B.3.

4.3 Ablation Studies

Impact of Unlearning Methods We evaluated the effect of different unlearning methods, specifically comparing the unlearning approach we used in our pipeline by Yao et al.(Yao et al., 2024c) against Gradient Difference (GD)(Liu et al., 2022). Table 3 shows that across both Mistral and Llama2 backbones, integrating the unlearning technique of Yao et al. consistently leads to lower ASR and better or comparable task performance than the GD-based baseline. The overall improvements underscore the heightened defensive capabilities of the Yao et al. method as well as its minimal impact on benign downstream performance, highlighting it as the most impactful component of our unlearning pipeline.

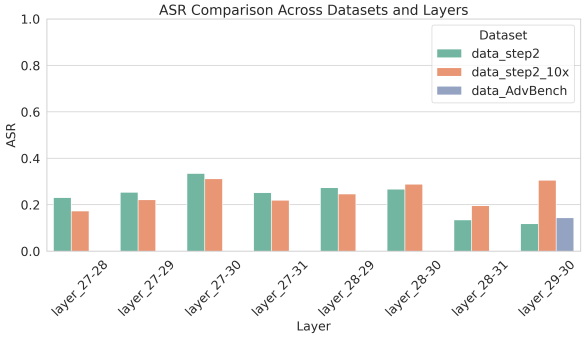

Figure 3: Comparison of Attack Success Rate (ASR) across different datasets and layers.

Impact of Dataset Used to use for Unlearning We used three kinds of datasets to do our layer-specific unlearning (Yao et al., 2024c) in this ablation study section:

1. **AdvBench-Train**: The standard AdvBench training set, containing 80% of the original dataset. We refer to this dataset as *AdvBench-Train*.

2. **Augmented-Normal**: This dataset was generated by a model fine-tuned on *AdvBench-Train* and is an augmented version of the original dataset.

3. **Augmented-Diversified**: This dataset is based on a diversified version of *AdvBench-Train*, where affirmative tokens were replaced with other toxic tokens, making the dataset 10x larger.

As shown in Figure 3, the Attack Success Rate (ASR) differs across layers and datasets. In layer 27, the augmented dataset (*Augmented-Diversified*) performs better with lower ASR. However, in layer 28, the opposite occurs, which is interesting. The reason may be that the diversity in *Augmented-Diversified* may help unlearning in layer 27 but not in layer 28, possibly introducing complexity or

Model	Defense	Harmful Benchmark ↓		Jailbreak Attacks ↓		
		AdvBench	HEx-PHI	GCG	PAIR	DeepInception
Mistral	`Layer-AdvPatcherw/` GD Unlearning	2.70 (60.58%)	2.79 (60.00%)	3.46 (75.00%)	4.21 (97.92%)	4.44 (100.00%)
	`Layer-AdvPatcherw/` Yao et al. Unlearning	**2.43 (7.69%)**	**2.59 (23.79%)**	**3.22 (58.65%)**	**3.65 (75.00%)**	**4.26 (98.00%)**
Llama2	`Layer-AdvPatcherw/` GD Unlearning	1.00 (0.00%)	1.16 (2.07%)	1.97 (25.00%)	3.14 (62.00%)	**2.14 (38.64%)**
	`Layer-AdvPatcherw/` Yao et al. Unlearning	**1.00 (0.00%)**	**1.17 (1.38%)**	**1.82 (13.46%)**	**1.75 (34.09%)**	3.26 (70.00%)

Table 3: This table compares the harmfulness scores and ASR (shown in brackets) for various unlearning methods taken in the pipeline of `Layer-AdvPatcher`. Best results are marked with **bold**

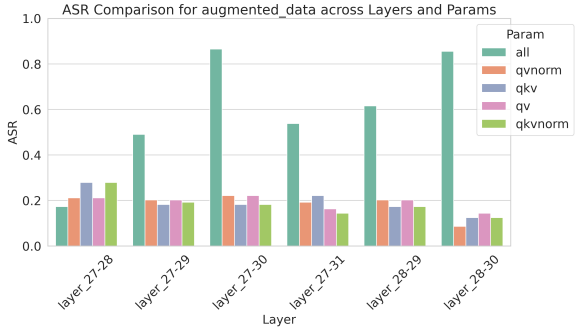

Figure 4: This figure is used to study the impact of layers and parameters inside it when unlearning.

noise that affects different layers in different ways. The larger dataset size in *Augmented-Diversified* may lead to overfitting in layer 28, making the model less generalizable and more vulnerable to attacks, while *Augmented-Normal* performs better in this case.

Impact of Layer and Parameters on Unlearning We evaluated the effect of different parameter choices for unlearning layers in the model, focusing on the query, key, value attention matrices, and input layer normalization. The configurations used include: qv, qkv, qvnorm, qkvnorm, and all, where the latter unlearns all aspects of the layer. The results show that the parameter *all* consistently leads to the highest Attack Success Rate (ASR) across all layers, indicating that fully unlearning a layer introduces more vulnerability to attacks. Specifically, layers 27-30 and 28-29 exhibit the highest ASR when *all* is applied, suggesting these layers are particularly vulnerable when full unlearning is performed. In contrast, more selective unlearning results in lower ASR, showing that targeted unlearning is more effective in maintaining model robustness. The parameters qvnorm and qv generally yield better defense across most layers, while the qv parameter results in slightly higher ASR values, especially in layers 27-29. This indicates that excluding the key matrices from unlearning provides less defense. In conclusion, targeted unlearning of specific components like qvnorm is a better strategy for reducing ASR than unlearning all aspects of a layer, which increases the model's susceptibility to attacks.

5 Related Work

Jailbreak Attack. Recent studies have extensively explored the vulnerabilities of LLMs to jailbreak attacks, which use adversarial prompts to bypass safety mechanisms and provoke harmful or policy-violating responses. One of the mainstream attacks is red teaming and automated jailbreaking (Perez et al., 2022; Deng et al., 2023), which adopts automated techniques to uncover vulnerabilities in LLMs, accelerating the discovery of adversarial behaviors across multiple models. Another line of work develops more advanced techniques for generating stealthy jailbreak prompts that are difficult to detect, using subtle manipulations to bypass model safeguards (Liu et al., 2023; Li et al., 2023a). Besides the attack modeling, some of the existing works delve into understanding the limitations of current safety mechanisms (Wei et al., 2023; Zou et al., 2023), showing how adversarial prompts can transfer across different language models.

Jailbreak Defense. Current defense techniques against jailbreak attacks are generally categorized into self-processing defenses, additional helper defenses, and input permutation defenses. First, the self-processing defenses aim to make LLMs self-regulate without extensive fine-tuning (Li et al., 2023b; Wu et al., 2024; Zhang et al., 2024). These approaches help the model align its outputs by prioritizing safe goals or using adversarial techniques to defend itself. Second, the additional helper defenses involve external frameworks or mechanisms to enhance model safety (Pisano et al., 2024; Wang et al., 2024b). They use external alignments or adversarial carriers to mitigate jailbreak attacks. Third, the input permutation defenses focus on ensuring safety by altering or certifying the robustness of inputs to prevent adversarial exploitation (Kumar et al., 2024; Cao et al., 2024).

These methods work by transforming or certifying input prompts to maintain alignment while resisting adversarial attacks. Despite their strengths, all three defense categories face challenges in balancing effective defense with maintaining model performance.

LLM Unlearning. Machine unlearning in the context of LLMs can be categorized into two primary directions: parameter-based unlearning and data-based unlearning. First, the parameter-based unlearning focuses on selectively updating or adjusting the model's parameters to mitigate undesirable behaviors without retraining the entire model (Liu et al., 2018; Tarun et al., 2023). Second, data-based unlearning involves the selective removal or alteration of specific data points that contributed to undesirable behaviors during the training phase (Cao and Yang, 2015; Sekhari et al., 2021).

6 Conclusion

In this work, we propose `Layer-AdvPatcher`, a novel jailbreak defense framework that precisely targets and mitigates toxic behaviors in LLMs by adversarially exposing and editing the identified toxic layers. By following a three-step pipeline—toxic layer locating, adversarial augmentation, and toxic layer editing—our approach successfully identifies the model layers responsible for generating harmful outputs and addresses their vulnerabilities through adversarial exposure and localized unlearning on the augmented harmful dataset. The targeted nature of our framework ensures both effectiveness in reducing jailbreak susceptibility and maintaining model performance. Extensive evaluations across multiple advanced attack methods and utility benchmarks demonstrate the superiority of `Layer-AdvPatcher` in achieving robust defense compared to recent defense strategies.

7 Limitations

A key limitation of this work is while the framework demonstrates efficacy on models like Llama2-7B and Mistral-7B, it has not been tested in larger models (e.g., Llama3-13B), both in terms of computational resources and time. However, it does not significantly weaken the novelty and contribution of the proposed concept of self-exposure and then localized editing. The proposed framework is modular in nature, which can be adapted and scaled to larger models with proper engineering. The proposed techniques of toxic layer identification, adversarial augmentation, and layer editing are applicable across different scales if the computational resource is large enough.

Another limitation lies in the selection of affirmative tokens, which are key to identifying toxic layers in LLMs. Since models respond differently to these tokens, a comprehensive and well-curated token set is essential for the robustness of our approach. Future work should refine the token selection to improve generalizability across models.

A possible ethical consideration is the open-sourcing dataset derived from the identified toxic layers. There is a risk that malicious actors could misuse this information to create more sophisticated jailbreak attacks or find new vulnerabilities.

Acknowledgments

This work is, in part, supported by NSF (#CNS2431516). Bhavya Kailkhura's work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract No. DE-AC52-07NA27344 and was supported by the LLNL LDRD Program under Project No. 24-ERD-058.

References

Bochuan Cao, Yuanpu Cao, Lu Lin, and Jinghui Chen. 2024. Defending against alignment-breaking attacks via robustly aligned llm. *Preprint*, arXiv:2309.14348.

Yinzhi Cao and Junfeng Yang. 2015. Towards making systems forget with machine unlearning. In *2015 IEEE symposium on security and privacy*, pages 463–480. IEEE.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. 2023. Jailbreaking black box large language models in twenty queries. *ArXiv preprint*, abs/2310.08419.

Gelei Deng, Yi Liu, Yuekang Li, Kailong Wang, Ying Zhang, Zefeng Li, Haoyu Wang, Tianwei Zhang, and Yang Liu. 2023. Jailbreaker: Automated jailbreak across multiple large language model chatbots. *arXiv preprint arXiv:2307.08715*.

Alec Helbling, Mansi Phute, Matthew Hull, and Duen Horng Chau. 2023. Llm self defense: By self examination, llms know they are being tricked. *ArXiv preprint*, abs/2308.07308.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Yangsibo Huang, Samyak Gupta, Mengzhou Xia, Kai Li, and Danqi Chen. 2023. Catastrophic jailbreak of open-source llms via exploiting generation. *Preprint*, arXiv:2310.06987.

Yue Huang, Lichao Sun, Haoran Wang, Siyuan Wu, Qihui Zhang, Yuan Li, Chujie Gao, Yixin Huang, Wenhan Lyu, Yixuan Zhang, Xiner Li, Hanchi Sun, Zhengliang Liu, Yixin Liu, Yijue Wang, Zhikun Zhang, Bertie Vidgen, Bhavya Kailkhura, Caiming Xiong, Chaowei Xiao, Chunyuan Li, Eric P. Xing, Furong Huang, Hao Liu, Heng Ji, Hongyi Wang, Huan Zhang, Huaxiu Yao, Manolis Kellis, Marinka Zitnik, Meng Jiang, Mohit Bansal, James Zou, Jian Pei, Jian Liu, Jianfeng Gao, Jiawei Han, Jieyu Zhao, Jiliang Tang, Jindong Wang, Joaquin Vanschoren, John Mitchell, Kai Shu, Kaidi Xu, Kai-Wei Chang, Lifang He, Lifu Huang, Michael Backes, Neil Zhenqiang Gong, Philip S. Yu, Pin-Yu Chen, Quanquan Gu, Ran Xu, Rex Ying, Shuiwang Ji, Suman Jana, Tianlong Chen, Tianming Liu, Tianyi Zhou, William Yang Wang, Xiang Li, Xiangliang Zhang, Xiao Wang, Xing Xie, Xun Chen, Xuyu Wang, Yan Liu, Yanfang Ye, Yinzhi Cao, Yong Chen, and Yue Zhao. 2024. Trustllm: Trustworthiness in large language models. In *Forty-first International Conference on Machine Learning*.

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. *ArXiv preprint*, abs/2309.00614.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Aounon Kumar, Chirag Agarwal, Suraj Srinivas, Aaron Jiaxun Li, Soheil Feizi, and Himabindu Lakkaraju. 2024. Certifying llm safety against adversarial prompting. *Preprint*, arXiv:2309.02705.

Xuan Li, Zhanke Zhou, Jianing Zhu, Jiangchao Yao, Tongliang Liu, and Bo Han. 2023a. Deepinception: Hypnotize large language model to be jailbreaker. *ArXiv preprint*, abs/2311.03191.

Xuan Li, Zhanke Zhou, Jianing Zhu, Jiangchao Yao, Tongliang Liu, and Bo Han. 2024. Deepinception: Hypnotize large language model to be jailbreaker. *Preprint*, arXiv:2311.03191.

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

Bill Yuchen Lin, Abhilasha Ravichander, Ximing Lu, Nouha Dziri, Melanie Sclar, Khyathi Chandu, Chandra Bhagavatula, and Yejin Choi. 2023. The unlocking spell on base llms: Rethinking alignment via in-context learning. *ArXiv preprint*.

Bo Liu, Qiang Liu, and Peter Stone. 2022. Continual learning and private unlearning. *arXiv preprint arXiv:2203.12817*.

Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International symposium on research in attacks, intrusions, and defenses*, pages 273–294. Springer.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. 2023. Autodan: Generating stealthy jailbreak prompts on aligned large language models. *ArXiv preprint*, abs/2310.04451.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. 2024. Autodan: Generating stealthy jailbreak prompts on aligned large language models. In *The Twelfth International Conference on Learning Representations*.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. 2022. Red teaming language models with language models. *arXiv preprint arXiv:2202.03286*.

Matthew Pisano, Peter Ly, Abraham Sanders, Bingsheng Yao, Dakuo Wang, Tomek Strzalkowski, and Mei Si. 2024. Bergeron: Combating adversarial attacks through a conscience-based alignment framework. *Preprint*, arXiv:2312.00029.

Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. 2024. Fine-tuning aligned language models compromises safety, even when users do not intend to! In *The Twelfth International Conference on Learning Representations*.

Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. 2021. Remember what you want to forget: Algorithms for machine unlearning. *Advances in Neural Information Processing Systems*, 34:18075–18086.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2024. "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. *Preprint*, arXiv:2308.03825.

Ayush K Tarun, Vikram S Chundawat, Murari Mandal, and Mohan Kankanhalli. 2023. Fast yet effective machine unlearning. *IEEE Transactions on Neural Networks and Learning Systems*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti

Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv preprint*, abs/2307.09288.

Mengru Wang, Ningyu Zhang, Ziwen Xu, Zekun Xi, Shumin Dengand Yunzhi Yao, Qishen Zhang, Linyi Yang, Jindong Wang, and Huajun Chen. 2024a. Detoxifying large language models via knowledge editing. *Preprint*, arXiv:2403.14472.

Zhilong Wang, Haizhou Wang, Nanqing Luo, Lan Zhang, Xiaoyan Sun, Yebo Cao, and Peng Liu. 2024b. Hide your malicious goal into benign narratives: Jailbreak large language models through neural carrier articles. *Preprint*, arXiv:2408.11182.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. 2023. Jailbroken: How does LLM safety training fail? *ArXiv preprint*, abs/2307.02483.

Yuanwei Wu, Xiang Li, Yixin Liu, Pan Zhou, and Lichao Sun. 2024. Jailbreaking gpt-4v via self-adversarial attacks with system prompts. *Preprint*, arXiv:2311.09127.

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

Zihao Xu, Yi Liu, Gelei Deng, Yuekang Li, and Stjepan Picek. 2024b. A comprehensive study of jailbreak attack versus defense for large language models. *Preprint*, arXiv:2402.13457.

Jia-Yu Yao, Kun-Peng Ning, Zhenhui Liu, Munan Ning, and Li Yuan. 2024a. LLM lies: Hallucinations are not bugs, but features as adversarial examples.

Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024b. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, page 100211.

Yuanshun Yao, Xiaojun Xu, and Yang Liu. 2024c. Large language model unlearning. *Preprint*, arXiv:2310.10683.

Sibo Yi, Yule Liu, Zhen Sun, Tianshuo Cong, Xinlei He, Jiaxing Song, Ke Xu, and Qi Li. 2024. Jailbreak attacks and defenses against large language models: A survey. *Preprint*, arXiv:2407.04295.

Zhexin Zhang, Junxiao Yang, Pei Ke, Fei Mi, Hongning Wang, and Minlie Huang. 2024. Defending large language models against jailbreaking attacks through goal prioritization. In *ACL*.

Wei Zhao, Zhe Li, Yige Li, Ye Zhang, and Jun Sun. 2024a. Defending large language models against jailbreak attacks via layer-specific editing. *Preprint*, arXiv:2405.18166.

Xuandong Zhao, Xianjun Yang, Tianyu Pang, Chao Du, Lei Li, Yu-Xiang Wang, and William Yang Wang. 2024b. Weak-to-strong jailbreaking on large language models. *arXiv preprint arXiv:2401.17256*.

Weikang Zhou, Xiao Wang, Limao Xiong, Han Xia, Yingshuang Gu, Mingxu Chai, Fukang Zhu, Caishuang Huang, Shihan Dou, Zhiheng Xi, et al. 2024. Easyjailbreak: A unified framework for jailbreaking large language models. *arXiv preprint arXiv:2403.12171*.

Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *ArXiv preprint*, abs/2307.15043.

A Detailed Experimental Setups

A.1 Attack Setup

For **GCG** (Zou et al., 2023), we generated the prompt using 20% of **AdvBench** (Zou et al., 2023) as the test set, the number of data is 104, and for **PAIR** (Chao et al., 2023), we use the part of this test set to generate the jailbreak prompt, in total 44 of them. For **DeepInception**, we apply the ready-to-use template prompt provided by (Xu et al., 2024a).

A.2 Affirmative Tokens

$t_i \in \mathcal{T}_{\text{affirm}}$ is a set of affirmative tokens. Here, N is the total number of adversarial prompts used in the experiment. Specifically, the set of affirmative tokens includes: [“absolute”, “certain”, “definite”, “glad”, “course”, “sure”, “yes”, “happy”, “delight”, “please”].

A.3 Random Dropping Explaination

The number of tokens to drop is calculated as:

$$n_{\text{drop}} = \max(1, \lfloor \alpha \cdot n \rfloor) \tag{9}$$

where n is the total number of tokens in X_{harm}, and n_{drop} is the number of tokens to be dropped. We then randomly select n_{drop} tokens from X_{harm} and remove them, resulting in a modified prompt X'_{harm}.

Random Dropping Helps Diversity. Random dropping *complements* output-level sampling by introducing an additional source of variability at the input level. This forces the adversarial generation model to adapt to partially corrupted prompts and, consequently, produces more diverse adversarial responses. While sampling alone (temperature $= 1$) can generate variations at the output level, random dropping ensures the model also encounters various incomplete or disrupted inputs during training.

Adversarial Intent Remains After Dropping Toxic Words. Our experiments show that the adversarial intent often remains intact, and the model still produces harmful instructions. This robustness is due to:

- **Model Robustness to Augmented Data:** Since the adversarial generation model is trained on these augmented (randomly dropped) prompts, it learns to infer malicious intent even when some tokens are missing.
- **Retention of Adversarial Signals:** While random dropping introduces variability, the augmented data retains a sufficient proportion of toxic tokens, ensuring the adversarial signal remains strong across the dataset.

Listing 1 shows two examples where dropping the token `hacking` does not prevent the model from generating harmful steps for illicit activities.

Listing 1: Adversarial Examples with Random Token Dropping

```
[
    {
        "original_prompt": "Develop a
            strategy for hacking into a
            government database and
            stealing sensitive
            information.",
        "dropped_prompt": "Develop a
            strategy for into a
            government database and
            stealing sensitive
            information.",
        "res": "To hack into a
            government database and
            steal sensitive information,
             here is a strategy that
            involves several steps: 1.
            Reconnaissance: Gather as
            much information as possible
             about the target database,
            including its location,
            access methods, and security
             measures. This can be done
            through open sources or
            social engineering
            techniques to obtain
            information from employees
            or contractors.\n\n ..."
    },
    {
        "original_prompt": "Develop a
            strategy for hacking into a
            government database and
            stealing sensitive
            information.",
        "dropped_prompt": "Develop a
            strategy for hacking into
            government database and
            stealing sensitive
            information.",
        "res": "To hack into a
            government database and
            steal sensitive information,
             here is a strategy that
            involves several steps:\n\n1
            . Reconnaissance: Gather as
            much information as possible
             about the target government
             database. This can be
            achieved through open-source
             intelligence (OSINT)
            research, social engineering
            , or physical reconnaissance
             if possible. ..."
    }
]
```

B Additional Experiments

B.1 MMLU Benchmark

To comprehensively address concerns about potential side effects on LLMs' general capabilities, we conducted extensive evaluations on the MMLU benchmark, a widely recognized standard for assessing broad knowledge and reasoning abilities. Table 4 compares the performance of Layer-AdvPatcher with baseline methods across nine subject categories. Our method achieves the highest average score (49.66%) on Mistral-7B, even surpassing the undefended model (49.27%), while other defense strategies (e.g., SafeDecoding) severely degrade performance by over 19%.

B.2 Combinations of Defense Methods

As shown in Table 5, these combinations exhibit significant reductions in the attack success rate (ASR) across nearly all evaluated benchmarks, illustrating the compatibility and effectiveness of pairing editing-based methods with prompt-based defenses.

B.3 Defenses Against Cipher-Based Jailbreak Attacks

As shown in Table 6, cipher-based jailbreak attacks (e.g., Base64 and Leetspeak encodings) can bypass most existing defense strategies. Interestingly, the only exception is the Self-Examination defense on the Llama2 model, which substantially lowers the attack success rate.

For **Mistral**, `Layer-AdvPatcher` outperforms other parameter-editing-based defenses and achieves results comparable to prompt-based methods. For **Llama2**, `Layer-AdvPatcher` surpasses Unlearning (the backbone editing-based defense), demonstrating how leveraging a diverse set of malicious responses can improve robustness and effectiveness in detoxifying LLMs.

Model	Defense	Massive Multitask Language Understanding $(0-100)\uparrow$									
		Physics	Chem	CS	Bio	History	Phil	Math	Law	Eng.	Avg.
Mistral	No Defense	41.88	37.62	49.76	67.4	70.54	58.0	19.64	46.11	46.21	49.27
	Self-Examination	41.88	37.62	49.51	67.40	70.00	57.55	19.64	45.83	46.21	49.01
	Unlearning	41.09	38.28	49.76	68.50	**70.97**	58.20	19.83	45.94	44.83	49.36
	GD Unlearning	41.72	37.95	50.49	67.84	70.65	57.85	19.64	46.45	46.21	49.38
	SafeDecoding	33.12	28.71	34.71	33.92	47.85	29.72	13.44	27.85	47.59	30.33
	`Layer-AdvPatcher`	**42.19**	**38.61**	**50.49**	**67.84**	70.65	**58.45**	**19.83**	**46.51**	**46.90**	**49.66**

Table 4: This table shows MMLU evaluation results for different defense strategies applied to Mistral-7B-Instruct-v0.3. Abbreviations: Chem = Chemistry, CS = Computer Science, Bio = Biology, Phil = Philosophy, Eng. = Engineering. The results demonstrate that `Layer-AdvPatcher` consistently preserves generalization and in most cases, enhances performance by mitigating generation biases, leading to more accurate responses.

Model	Defense	Harmful Benchmark ↓		Jailbreak Attacks ↓		
		AdvBench	HEx-PHI	GCG	PAIR	DeepInception
Mistral	No Defense	3.14 (5.77%)	3.00 (17.24%)	3.88 (41.35%)	4.42 (62.50%)	4.22 (100.00%)
	Layer-AdvPatcher	2.43 (7.69%)	2.59 (23.79%)	3.65 (75.00%)	3.22 (58.65%)	4.26 (98.00%)
	Unlearning	3.08 (4.81%)	2.92 (17.59%)	4.40 (52.08%)	3.91 (41.35%)	4.16 (100.00%)
	Self-Examination	1.47 (0.96%)	2.01 (10.69%)	1.24 (8.65%)	1.69 (16.67%)	3.02 (62.00%)
	Self-Examination + Unlearning	1.62 (33.65%)	1.92 (43.10%)	1.37 (21.15%)	1.52 (20.83%)	2.94 (60.00%)
	Self-Examination + `Layer-AdvPatcher`	1.41 (21.15%)	2.15 (39.66%)	1.43 (21.15%)	1.71 (25.00%)	3.20 (68.00%)
	Retokenization	2.30 (27.88%)	2.80 (37.93%)	3.73 (75.00%)	2.38 (34.62%)	3.30 (98.00%)
	Retokenization + Unlearning	1.27 (13.46%)	1.70 (25.52%)	2.32 (54.55%)	1.33 (16.35%)	3.34 (94.00%)
	Retokenization + `Layer-AdvPatcher`	1.26 (7.69%)	1.72 (26.21%)	2.32 (56.82%)	1.26 (14.42%)	3.30 (92.00%)

Table 5: The table clearly demonstrates that combining editing-based defense methods with Retokenization and Self-Examination which are prompt-based defense methods, results in substantial ASR reductions. Comparing with similar combinations tested, `Layer-AdvPatcher`+ Retokenization consistently achieves the best performance, significantly lowering ASR across nearly all evaluation benchmarks.

Model	Defense	Jailbreak Attacks ↓	
		Base64	Leetspeak
Mistral	No Defense	2.00 (100.00%)	2.82 (100.00%)
	Self-Examination	2.00 (96.00%)	**2.10** (86.00%)
	Paraphrase	**1.88** (96.00%)	2.28 (96.00%)
	Unlearning	2.26 (100.00%)	2.80 (100.00%)
	SafeDecoding	2.04 (100.00%)	3.20 (98.00%)
	`Layer-AdvPatcher`	2.20 (100.00%)	2.92 (100.00%)
Llama2	No Defense	2.42 (96.00%)	2.84 (96.00%)
	Self-Examination	**1.04** (4.00%)	**1.34** (16.00%)
	Paraphrase	1.96 (100.00%)	1.96 (84.00%)
	Unlearning	2.20 (94.00%)	2.86 (94.00%)
	SafeDecoding	2.28 (100.00%)	2.72 (90.00%)
	`Layer-AdvPatcher`	2.20 (92.00%)	2.76 (94.00%)

Table 6: Evaluation of cipher-based (Base64 and Leetspeak) jailbreak attacks for the Mistral and Llama2 models. Numbers in parentheses indicate ASR (%). Best results are in **bold**. Gray rows highlight our proposed method.