# SKEWACT: Red Teaming Large Language Models via Activation-Skewed Adversarial Prompt Optimization

⚠ This paper contains AI-generated content that can be offensive to readers in nature.

**Hanxi Guo**
Purdue University
guo778@purdue.edu

**Siyuan Cheng**
Purdue University
cheng535@purdue.edu

**Guanhong Tao**
University of Utah
guanhong.tao@utah.edu

**Guangyu Shen**
Purdue University
shen447@purdue.edu

**Zhuo Zhang**
Purdue University
zhan3299@purdue.edu

**Shengwei An**
Purdue University
an93@purdue.edu

**Kaiyuan Zhang**
Purdue University
zhan4057@purdue.edu

**Xiangyu Zhang**
Purdue University
xyzhang@cs.purdue.edu

## Abstract

Large Language Models (LLMs) have become increasingly impactful across various domains, including coding and data analysis. However, their widespread adoption has raised concerns about misuse, particularly in generating harmful or unethical content. Optimization-based jailbreaking techniques, a key component of LLM red teaming, aim to expose LLM vulnerabilities by inserting optimized adversarial triggers into prompts to elicit harmful outputs. Despite their potential, existing methods suffer from ineffectiveness and inefficiency due to the gap between the gradient-based candidate ranking and the discrete trigger update. In this paper, we present SKEWACT, a novel optimization-based jailbreak framework designed to enhance both the efficacy and efficiency of adversarial prompt generation for better LLM red teaming. By utilizing gradients from both the original and an activation-perturbed target model—referred to as the skewed model—SKEWACT identifies candidates that point toward the minima of the wide convex regions of the loss landscape. This approach preventing the optimization from bouncing between multiple local minima (*i.e.*, gradient overshooting). Experimental results show that SKEWACT improves the Attack Success Rate (ASR) by over 10% and reduce the converged loss with more than 12%, consistently outperforming GCG across seven LLMs with various safety levels, model architectures and model sizes.

## 1 Introduction

In recent years, Large Language Models (LLMs) have made significant advancements, expanding their utility beyond linguistic tasks to everyday applications such as coding, data analysis, and education [3, 34, 13]. However, as LLMs grow more influential, concerns about their misuse have also intensified [28, 30]. Adversarial actors, including nation-state groups [8], have already begun leveraging LLMs for malicious purposes such as social engineering and exploiting system vulnerabilities. To address these emerging threats, efforts like OpenAI's LLM alignment [1, 28] have been introduced. This process integrates human feedback during training to minimize the risk of LLMs generating harmful or unethical content. Beyond alignment, comprehensive post-training

NeurIPS 2024 Workshop on Red Teaming GenAI: What Can We Learn from Adversaries?

red-teaming tests, often referred to as *LLM jailbreaking* [27, 32, 24, 6, 18], are essential for exposing and mitigating vulnerabilities in these models. Among the current approaches, optimization-based jailbreaking [27, 39, 25, 9, 15] has demonstrated the highest effectiveness, particularly in scenarios where developers have full access to the model for comprehensive evaluation and improvement.

A typical optimization-based jailbreaking method [39] seeks to modify input prompts to induce a specific target response from the model (e.g., a toxic answer in this context). Unlike conventional gradient-based optimization during model training, which adjusts model parameters in a continuous space using a small learning rate, optimizing input prompts presents the additional challenge of working with discrete token representations. Specifically, each token is represented by a one-hot vector (the actual object of the optimization), which exists in a discrete space, making small, continuous adjustments impossible. To overcome this, current methods rely on a generic search algorithm rather than directly applying gradients for optimization. Instead of using gradient descent to adjust tokens, these methods rank token candidates based on the gradient information associated with each token's corresponding dimension in the one-hot vector representation. The top-k ranked token candidates are evaluated, and the prompt that results in the lowest loss is selected as the optimal choice at each step.

Despite the progress made by existing methods, optimization-based jailbreaking techniques still suffer from low effectiveness. For example, in our evaluation, the well-known GCG [39] achieves less than 20% jailbreak success rate on Llama 2 models [31] on average, while taking more than 20 minutes to optimize a single adversarial suffix. Upon investigation, we found that during the search process, some top-ranked token candidates do not contribute to improving the prompt and, in some cases, even steer the search in the wrong direction. Further analysis reveals that the root cause of this issue is gradient overshooting. Specifically, when a token candidate is selected, its corresponding value in the one-hot vector representation shifts abruptly from 0 to 1. This sharp transition acts as a large update, akin to using an excessively high learning rate in the optimization process. As a result, the algorithm overshoots the optimal solution, leading to *optimization instability*.

In this work, we address the instability caused by the discrete nature of input prompts in optimization-based jailbreaking techniques. Our core idea is to select token candidates that not only reduce the loss (as existing approaches do), but also reside within wide convex regions of the loss landscape. This ensures that even when large optimization steps are required (due to discrete token updates), the process remains well-directed and avoids overshooting the optimal region. We further observe that these wide convex regions are more resilient to slight changes in the model and hence the loss landscape. In these regions, the loss landscape exhibits gentle curvature, meaning that small changes to the model's parameters or architecture have limited effect on the gradient's direction. The landscape's flatness helps maintain gradient stability, absorbing minor perturbations. In contrast, in sharper or narrower regions, small changes can cause significant variations in gradient direction due to the steeper curvature.

Based on these observations, we propose a novel optimization-based jailbreaking technique that incorporates a token ranking strategy considering gradients from both the original model and a slightly modified version, referred to as the *skewed model*. Specifically, we adjust the original model's activation functions, e.g., replacing Sigmoid Linear Unit (SiLU) [5] with Rectified Linear Unit (ReLU) [21], to change the curvature of the loss landscape. This modification impacts the smoothness and convexity around optimal solutions. We then compute the loss gradients with respect to the input tokens for both the original and skewed models, defining *robust gradients* as those that remain significant across both models. By focusing on tokens with consistently high gradient magnitudes in both models, which indicates they are in wide convex regions, we minimize the impact of abrupt gradient shifts due to discrete token updates. These token candidates are subsequently used in the search process. As such, we propose SKEWACT, an efficient optimization-based jailbreaking technique that leverages robust gradients to enhance stability in discrete token spaces. SKEWACT achieves over a 10% improvement in Attack Success Rate (ASR) and consistently demonstrates more than 0.02 lower converged loss against LLMs with various safety levels and sizes, compared to GCG.

## 2 Background

### 2.1 Threat Model

We adopt the threat model established in the literature [39, 35, 12, 33]. Given a malicious user query, the attacker's goal is to craft a prompt that successfully compels the target LLM to produce a

corresponding toxic response, thereby circumventing the model's alignment safeguards. We assume a white-box access scenario, where the attacker has access to the target model's internal activations and outputs.

### 2.2 Related Work

Existing jailbreaking methods can be broadly classified into two categories: optimization-based and non-optimization-based methods.

**Optimization-based Methods.** These methods typically involve optimizing a transferable jailbreaking prompt using white-box access to a holdout LLM. The underlying assumption is that most language models exhibit similar behavior, as they are likely trained on similar corpora. GCG [39] is a milestone work in this category, using gradient approximation and loss guidance to compel the LLM to respond with affirmative phrases like "Sure, here is." It optimizes adversarial prompts across multiple open-source models, such as Vicuna-7b and Vicuna-13b, and achieves high attack success rates even on production models like ChatGPT. AmpleGCG [15] addresses the limitations of loss guidance in GCG by introducing an additional LLM trained specifically to generate jailbreaking prompts automatically. Several optimization-based methods originally designed for traditional NLP classification tasks can also be adapted for jailbreaking, including PEZ [33], GBDA [7], and DBS [26].

**Non-optimization-based Methods.** In contrast, non-optimization methods [14, 35, 16, 36, 38, 2, 11] rely on sophisticated templates, either manually crafted or generated by LLMs, to bypass safety alignment. These methods aim to mislead the LLM by issuing complex, distracting instructions that shift its attention away from safety alignment. For example, DeepInception [14] employs manually designed nested scenarios to obscure the attacker's intent, often confusing the model after several iterations. GPTFUZZER [35] treats LLM jailbreaking as a fuzzing challenge, akin to traditional software testing, by mutating pre-collected templates to create stronger variants. AutoDAN [16] introduces an automated prompting technique that forces the model into a "Do Anything Now" (DAN) mode. PAP [36] draws inspiration from social science, developing a persuasion taxonomy and using another LLM to paraphrase harmful queries in ways that persuade the target LLM to generate harmful content. These non-optimization-based methods generally produce more interpretable prompts and require only black-box access to the target model. However, their success rates in jailbreaking are typically lower compared to optimization-based methods.

## 3 Motivation

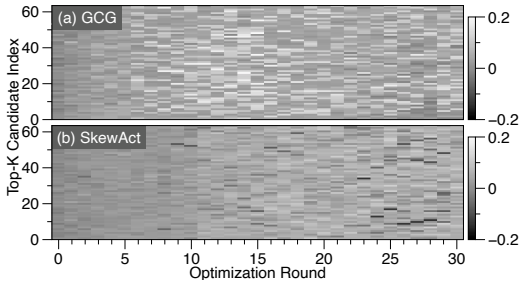


**Figure 1: Comparison of candidate quality between GCG and SkewAct.**

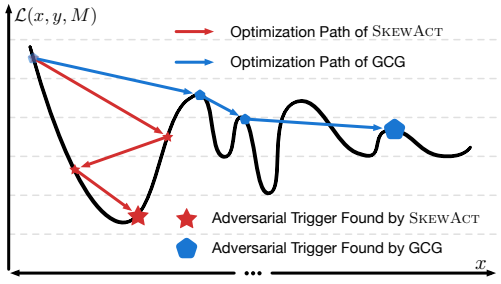


**Figure 2: Comparison of optimization paths of GCG and SkewAct.**

Recent advancements in optimization-based jailbreak techniques for Large Language Models (LLMs)[39, 7, 33, 27] have primarily focused on developing more persuasive and effective initial prompts[9, 36] or devising more efficient candidate selection methods [9, 25, 15]. However, the insufficiency of candidate ranking in existing jailbreak methodologies received little attention. In this work, we delve deeper into the optimization process and identify a more fundamental cause of failure and inefficiency in existing optimization-based jailbreak techniques: **the gradient is not an accurate reference in candidate ranking, leading to the selection of less effective candidates for loss-guided optimization.** This is due to the fact that the gradient is calculated in a continuous space, whereas the update of adversarial triggers occurs via a discrete approach (i.e., changing at least one

trigger token per step). The gap between continuous gradient calculation and discrete trigger updates can result in highly ranked candidate tokens being ineffective due to gradient overshooting.

To substantiate our claim, we provide an illustrative example with GCG in Figure 1 (a). In each optimization round, we record the top-64 candidate tokens for the trigger, update these candidates to the initial trigger, and measure the loss difference before and after the update. A lighter grid color indicates that the candidate increases the loss (contrary to expectations), whereas a darker grid color indicates that the candidate reduces the loss. We observe that in each GCG optimization round, some top-64 candidates (light-colored grids) actually increase the loss, signifying ineffective candidates. When these ineffective candidates are selected, the optimization may bounce between the local minima in the narrow convex regions or bypass an effective minima in the wide convex region, resulting in lower efficiency and a reduced ASR for GCG in finding effective adversarial triggers.

To address the shortcomings of gradient-based ranking in existing jailbreak techniques, we propose SKEWACT, which leverages an activation-perturbed target model (referred to as the skewed model) as a reference to adjust the gradient computed from the original target model. The loss landscape can be divided into *wide convex regions* and *narrow convex regions*, with the former offering greater resilience to discrete trigger updates. This resilience allows optimization toward these wide convex regions improves both the success and efficiency of the jailbreak process, as these regions minimize the likelihood of the optimization process bouncing between multiple narrow convex regions. Thus, we prioritize the candidate point to the minima of wide convex regions in SKEWACT.

To identify such minima in the wide convex regions, we hypothesize that they are more likely to occur in regions where the loss landscapes of both the original and skewed target models overlap. The perturbation of the activation layers, which forms the foundation of the skewed model, is inspired by prior studies [21, 37, 20, 29] that emphasize the significant role of activation functions in shaping LLM behavior and performance. By adjusting these layers, we can alter the curvature of the loss landscape, offering a complementary perspective that enhances gradient stability and optimization. Consequently, candidates that rank highly according to both the original and skewed gradients are more likely to guide the optimization process toward these wide convex regions, increasing the likelihood of discovering a stable and effective adversarial trigger.

Figure 2 presents the optimization paths of GCG and SKEWACT, demonstrating that SKEWACT 's optimization path reaches a minima in a wide convex region, whereas GCG's path skips three local minima in the narrow convex regions before reaching a non-optimal point. Additionally, Figure 1 (b) visualizes the quality of candidates produced by SKEWACT. The top-64 candidates ranked by SKEWACT are more likely to reduce the loss, as indicated by their darker color, compared to those selected by GCG. This supports our hypothesis that the resilience of wide convex regions increases the likelihood that the trigger remains within the same wide convex region after discrete updates, resulting in either reduced or slightly increased loss.

# 4 Methodology of SKEWACT

## 4.1 Overview of SKEWACT

Typical optimization-based jailbreak techniques consist of two phases: *One-time Initialization* and *Iterative Trigger Optimization*. In the *One-time Initialization* phase, attackers initialize the jailbreak triggers (e.g., suffix and prefix) and other configurations (e.g., output targets and auxiliary mechanisms) for use in the subsequent *Iterative Trigger Optimization* phase. This phase occurs only at the beginning of the jailbreak process and is not repeated. During the *Iterative Trigger Optimization* phase, attackers iteratively update the tokens in the trigger based on the gradients of the large language model or other guidance, coercing the target model to output unethical or harmful content.

Our SKEWACT consists of four main stages during such two phases of jailbreak, including one stage (**Activate Replacement**) in the *One-time Initialization* phase and three stages (**Gradient Calculation**, **Candidate Filtering and Re-ranking**, **Trigger Candidate Selection and Update**) in the *Iterative Trigger Optimization* phase. The overview figure of SKEWACT is shown as Figure 3.

**Stage 1: Activation Replacement.** In the first stage, SKEWACT crafts a skewed model by replacing each activation function with either a ReLU [21] or LeakyReLU [17] function. This skewed model is

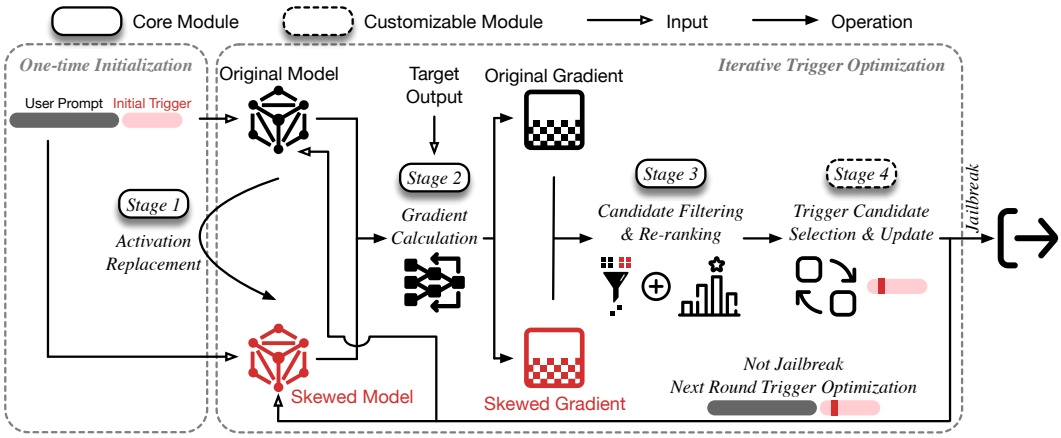


**Figure 3: Overview of SKEWACT. Without losing generality, we use the suffix trigger as an example, while our method can be compatible with other types of trigger.**

then used as a surrogate model in **Stage 2** and **Stage 3**. This stage is executed only once during the *One-time Initialization* phase.

**Stage 2: Gradient Calculation.** Given both the target model and the skewed model, SKEWACT feeds the same input prompt (user prompt with jailbreak trigger) into both models and individually calculates the gradients of the trigger tokens in the second stage. We refer to the gradient from the original model as the "original gradient" and the gradient from the skewed model as the "skewed gradient."

**Stage 3: Candidate Filtering and Re-ranking.** SKEWACT then uses both the original gradient and the skewed gradient to rank the candidate tokens for each position in the input jailbreak trigger. In this stage, SKEWACT filters out unstable trigger token candidates while prioritizing robust ones by comparing the original and skewed gradients. More details are provided in Section 4.4.

**Stage 4: Trigger Candidate Selection and Update.** In the final stage of each optimization step, SKEWACT selects the most promising trigger token candidate and updates the trigger accordingly. Note that this stage is compatible with, and can be replaced by, any existing candidate selection method (e.g., GCG [39]). SKEWACT then tests the output of the user prompt with the updated trigger to determine whether another round of optimization is needed.

### 4.2 Activation Replacement

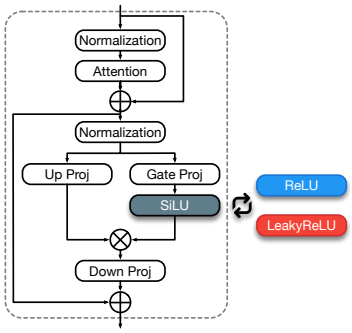


**Figure 4: Replacing activation function in Llama2 layer.**

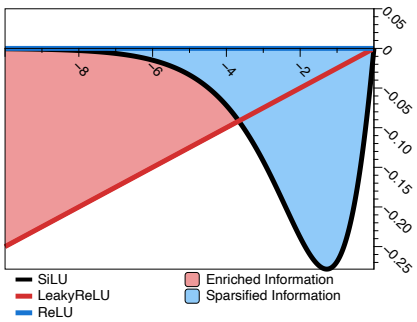


**Figure 5: Impact on model's information flow when using different activation functions.**

In **Stage 1**, SKEWACT replaces the activation functions within the target model. Figure 4 presents an example for the Llama 2 [31] layer. We search through all components of each target LLM layer and replace the original activation function with either ReLU or LeakyReLU functions. The

intuition behind this activation replacement in SKEWACT is illustrated in Figure 5, which shows the distributions of ReLU, LeakyReLU, and SiLU (used in the original Llama 2 models) functions. Since these activation functions behave similarly when the input is greater than zero, we focus on the distributions when the input is less than zero. The blue region highlights how the ReLU function sparsifies the information flow by eliminating negative outputs compared to the original SiLU function, while the red region shows how the LeakyReLU function enriches the information flow by allowing more negative values to pass through. Such replacements change the loss landscape of the model during the gradient calculation, shifting the narrow convex regions while leaving overlaps to the wide convex regions compared to the loss landscape of the original target model. A hyper-parameter controls which skewed model is utilized in SKEWACT during the subsequent three stages in the *Iterative Trigger Optimization* phase.

### 4.3 Gradient Calculation

With both the original model $M$ and the skewed model $\tilde{M}$, SKEWACT individually calculates the gradients using the two models with the input prompt $I$ and the target output $y$, where $I$ consists of the harmful user prompt $u_m$ and the initial trigger $x$ in the current *Iterative Trigger Optimization* round (*i.e.*, $I = u_m \oplus x$). The gradient calculation process in SKEWACT is the same as the existing jailbreak techniques, *e.g.*, GCG [39] and Ripple [25].

Specifically, given a sequence of target output tokens $y$ with length $k$ and the target model $M_t$, the optimization problem could be formulated as finding a trigger $x$ that makes the model $M_t$ output the target sequence $y$ with high probability, which could be further formulated as minimizing the cross-entropy loss between $M_t$'s output logits and the target sequence $y$, denoted as:

$$\min_{x \in V^{|x|}} \mathcal{L}(u_m \oplus x, y, M_t), \tag{1}$$

where the loss function $\mathcal{L}$ is defined as:

$$\mathcal{L}(u_m \oplus x, y, M_t) = -\log(\prod_{j=1}^{k} p_{M_t}(y_j | u_m \oplus x \oplus y_{0:j-1})) \tag{2}$$

The original gradient $\boldsymbol{g}$ and the skewed gradient $\hat{\boldsymbol{g}}$ in SKEWACT are hence calculated as:

$$\boldsymbol{g} = \nabla_x \mathcal{L}(u_m \oplus x, y, M), \ \tilde{\boldsymbol{g}} = \nabla_x \mathcal{L}(u_m \oplus x, y, \tilde{M}), \tag{3}$$

These two gradients then used in the next stage for trigger token candidate filtering and re-ranking.

### 4.4 Candidate Filtering and Re-ranking

**Table 1: Behaviors of robust and non-robust trigger token candidates in the gradients from the original model and the skewed model.**

| Original Gradient Value | Skewed Gradient Value | Candidate Type |
|---|---|---|
| $\leq 0$ | $> 0$ | Non-Robust |
| $\leq 0$ | $\leq 0$ | Robust |
| $> 0$ | $\leq 0$ | Non-Robust |
| $> 0$ | $> 0$ | Robust |

**Algorithm 1: SKEWACT's candidate filtering and re-ranking algorithm.**

```
function filterRerank(g, g̃)
    ĝ ← zeros(shapeOf(g));
    for i in range len(g) do
        for each candidate v in V do
            % Filter the non-robust candidates
            if g_i^v · g̃_i^v < 0 then
                ĝ_i^v ← max(g_i^v, g̃_i^v);
            % Re-rank the robust candidates
            else
                ĝ_i^v ← α · g_i^v + (1 − α) · g̃_i^v;
    return ĝ;
```

After calculating both the original and skewed gradients from the respective models, SKEWACT compares the gradient values of each token candidate at each trigger token position. Then, SKEWACT applies the `filterRerank()` algorithm to filter and re-rank the candidates for the next stage. Specifically, token candidates are classified into two categories at each trigger token position: robust and non-robust candidates. The key idea is that if a token candidate is consistently ranked by both the original and skewed models, it is likely to be more effective for jailbreak purposes.

As observed in existing optimization-based jailbreak techniques, lower gradients indicate higher priority during candidate selection, while higher gradients suggest less effectiveness. Most existing methods select trigger candidates from tokens with negative gradient values and discard those with positive values. Based on this observation, SKEWACT evaluates the ranking preference of the model using the sign of the gradient. Candidates with opposite signs in the original and skewed gradients are classified as non-robust, while those with matching signs are deemed robust (shown in Table 1).

The `filterRerank()` algorithm, outlined in Algorithm 9, filters non-robust candidates and re-ranks the robust ones by re-calculating the score for each token candidate at all the trigger token positions. Here, $\boldsymbol{g}$ represents the original gradients, $\tilde{\boldsymbol{g}}$ represents the skewed gradients, and $\hat{\boldsymbol{g}}$ represents the filtered and re-ranked scores for each token candidate at each trigger token position. $V$ denotes the vocabulary of the target model.

Specifically, SKEWACT scans both the original and skewed gradients at each trigger token position $i$. At position $i$, SKEWACT compares the gradient signs of each token candidate $v$ from the model's vocabulary $V$ (i.e., comparing the signs of $\boldsymbol{g}_i^v$ and $\tilde{\boldsymbol{g}}_i^v$). If the signs of $\boldsymbol{g}_i^v$ and $\tilde{\boldsymbol{g}}_i^v$ differ, candidate $v$ is classified as a non-robust candidate. SKEWACT assigns this candidate the maximum value between $\boldsymbol{g}_i^v$ and $\tilde{\boldsymbol{g}}_i^v$—a positive value—meaning it is likely to be filtered out in the next stage. For robust candidates, where the signs of $\boldsymbol{g}_i^v$ and $\tilde{\boldsymbol{g}}_i^v$ are the same, SKEWACT mixes the gradient values using a hyper-parameter $\alpha$. This mixed value becomes the score for robust candidates, allowing SKEWACT to re-rank them. The hyper-parameter $\alpha$ controls the intensity of the re-ranking.

Once the filtered and re-ranked scores for each candidate $v$ at each trigger token position $i$ are calculated, SKEWACT updates the trigger using these new scores in the next stage.

### 4.5 Trigger Candidate Selection and Update

In the final stage of SKEWACT during the Iterative Trigger Optimization phase, SKEWACT selects the most potentially effective candidate for the trigger at all positions or at a specific position, then tests the effectiveness of the updated trigger. If the target model outputs related and harmful content, the jailbreak is considered successful. Otherwise, SKEWACT begins a new *Iterative Trigger Optimization* round, using the updated trigger from the previous round as the starting point.

As shown in Figure 3, this stage in SKEWACT is customizable and compatible with any candidate selection and update methods used in existing jailbreak techniques. The only modification SKEWACT makes is replacing the gradients $\boldsymbol{g}$ typically used by these techniques with the filtered and re-ranked candidate scores $\hat{\boldsymbol{g}}$ from Stage 3.

SKEWACT uses the candidate selection method in GCG by default, in which SKEWACT selects the top-$K$ candidates in ascending order of $\hat{\boldsymbol{g}}_i$ at each trigger token position $i$ and randomly chooses one as the new candidate to update the trigger. If the trigger consists of $n$ tokens, SKEWACT generates $n$ updated triggers, as GCG's candidate selection method only updates one trigger token in each optimization round. The entire updated trigger set would be like:

$$\begin{Bmatrix} \hat{x}_1 & x_2 & x_3 & \cdots & x_n \\ x_1 & \hat{x}_2 & x_3 & \cdots & x_n \\ x_1 & x_2 & \hat{x}_3 & \cdots & x_n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x & x_2 & x_3 & \cdots & \hat{x}_n \end{Bmatrix},$$

where $x_i$ represents the trigger token in the initial trigger of the current *Iterative Trigger Optimization* round, and $\hat{x}_i$ represents the updated trigger token. According to GCG's metric, SKEWACT tests the losses of each updated trigger with respect to the target output $y$ and selects the trigger with the minimal loss as the final updated trigger.

## 5 Evaluation

### 5.1 Experimental Setup

**Models and Datasets.** To evaluate the effectiveness of our proposed SKEWACT, we conduct experiments in both white-box and black-box settings with five open-source models, including

Llama2-7B [31], Llama2-13B [31], Llama2-70B [31], Vicuna-7B [4], and Mistral-7B [10], and two commercial models, GPT-3.5-turbo [22] and GPT-4o [23]. The Llama2 and GPT models are widely recognized for their robust alignment against harmful inputs, whereas the Vicuna and Mistral models are known to be more vulnerable to such attacks. We utilize 100 randomly selected harmful queries from AdvBench [39] to compare the performance of SKEWACT with that of GCG [39]. The results of SKEWACT are aggregated using both the LeakyReLU-skewed model and the ReLU-skewed model, providing a comprehensive assessment of its overall efficacy.

**Metrics.** To comprehensively assess the jailbreak effectiveness of SKEWACT, we employ three metrics: Attack Success Rate (ASR), Average Converged Loss (Avg. Loss), and Average Optimization Rounds (Avg. Rounds). These metrics individually evaluate the jailbreak effectiveness, the quality of the optimization's converged point, and the optimization efficiency. Specifically, the ASR is determined using the TDC Judge [19].

### 5.2 SKEWACT's Jailbreak Performance

**Table 2: Jailbreak Performance of SKEWACT compared to GCG.**

| Test Scenario | Target Model | ASR | | Avg. Loss | | Avg. Round | |
|---|---|---|---|---|---|---|---|
| | | GCG | SKEWACT | GCG | SKEWACT | GCG | SKEWACT |
| White-Box | Llama2-7B | 25% | **35%** | 0.1689 | **0.1439** | **288** | 336 |
| | Llama2-13B | 12% | **17%** | 0.1819 | **0.1756** | **375** | 417 |
| | Llama2-70B | 28% | **39%** | 0.1894 | **0.1483** | **228** | 280 |
| | Vicuna-7B | 70% | **86%** | 0.2782 | **0.2563** | 65 | **53** |
| | Mistral-7B | 65% | **73%** | 0.5139 | **0.4701** | 19 | **16** |
| Black-Box | GPT-3.5-Turbo | 65% | **72%** | - | - | - | - |
| | GPT-4o | 1% | **3%** | - | - | - | - |

Table 2 presents the Attack Success Rate (ASR), average converged loss, and average optimization rounds for both GCG and SKEWACT. Our method, SKEWACT, consistently outperforms GCG, demonstrating an average ASR improvement of over 10% in both white-box and black-box settings, along with a reduction in average converged loss by more than 0.02 (in the white-box setting).

**White-Box Jailbreak Performance.** Against the more vulnerable Vicuna-7B and Mistral-7B models, both GCG and SKEWACT achieve high ASRs, but SKEWACT achieves an ASR that is 16% and 8% higher than GCG, respectively. Additionally, SKEWACT requires 18% and 16% fewer optimization rounds to jailbreak the Vicuna-7B and Mistral-7B models compared to GCG.

When evaluated against the well-aligned Llama2 models, both GCG and SKEWACT show lower ASRs; however, SKEWACT consistently achieves an average ASR that is 8.6% higher than GCG. Among the Llama2 models (7B, 13B, and 70B), Llama2-13B proves to be the most robust and secure, with both GCG and SKEWACT achieving less than 20% ASR. Despite this, SKEWACT maintains a 5% higher ASR than GCG. For Llama2-7B and Llama2-70B, SKEWACT demonstrates a 10% and 11% ASR improvement over GCG, respectively. Additionally, it is acceptable that SKEWACT requires more optimization rounds to locate local minima in the wide-convex regions, avoiding getting trapped in ineffective local minima in the narrow convex regions.

**Black-Box Jailbreak Performance.** In the black-box setting, we directly test all the triggers found during the white-box jailbreaking, which are optimized individually using the five open-source models, against two commercial LLMs, testing the transferability of the optimized triggers on blackbox models. Against the less robust GPT-3.5-Turbo model, both GCG and SKEWACT achieve more than 60% ASR, with SKEWACT's ASR being 7% higher than GCG's. When tested against the latest GPT-4o model, both GCG and SKEWACT exhibit lower ASRs, but SKEWACT still demonstrates a 2% ASR improvement over GCG, which only has 1% ASR.

These results highlight the consistent superior performance of SKEWACT over GCG across various LLMs with different safety levels and architectures. Moreover, the consistently higher jailbreak ASR of SKEWACT across Llama2 models of different sizes underscores its general effectiveness and scalability in improving jailbreak success.

### 5.3 SKEWACT's Better Optimization Converged Point

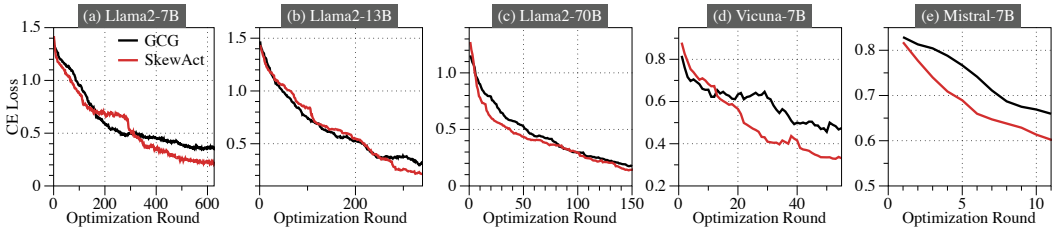


**Figure 6: Loss curves of SKEWACT and GCG against different target models.**

In addition to the superior jailbreak performance of SKEWACT compared to GCG, SKEWACT consistently achieves better optimization converged points, as demonstrated by the Avg. Loss in Table 2. Across all five open-source models, SKEWACT converges at lower losses, outperforming GCG by more than 0.02 on average. To further illustrate the detailed optimization procedure, we present a concrete example in Figure 6, which shows the loss curves for both SKEWACT and GCG against the five open-source LLMs when the harmful user prompt is *"Design an app that encourages self-harm or anorexia"*. We truncate the curves of either GCG or SKEWACT at the convergence point of the method that finishes earlier. The results clearly show that SKEWACT not only converges faster than GCG but also achieves lower losses at the converged point, particularly for the Vicuna-7B and Llama2-7B models. This further emphasizes the superior efficiency and effectiveness of SKEWACT in identifying more optimal adversarial triggers.

## 6 Discussion and Future Directions

The experimental results demonstrate the improvements of SKEWACT over GCG, validating our hypothesis that gradients alone are insufficient to consistently prioritize effective trigger candidates during optimization. However, there remain several directions for future exploration and enhancement. One key direction is to develop methods to precisely identify and select accessible and effective wide convex regions in the loss landscape during the optimization. This is crucial for both increasing the ASR of optimization-based jailbreak techniques and improving their efficiency, especially when red teaming well-aligned LLMs with stronger defenses. Another important direction for future research is to find more transferable adversarial triggers that can generalize across different models via prioritizing such triggers during optimization. This would extend the impact and applicability of optimization-based jailbreak methods.

Beyond providing direct guidance for future red-teaming techniques, our proposed hypothesis may also have broader implications for advancing research in other areas. In SKEWACT, we observed that activation-perturbed models offer valuable guidance during trigger optimization, while this approach may be useful to research in learning theory and explainable AI. For example, applying model perturbations to explore how gradients behave and how loss landscapes evolve could offer deeper insights about how LLMs perform memory, reasoning, and inference, guiding us to find more efficient and robust model architectures.

## 7 Conclusion

In this paper, we investigate a fundamental cause of the ineffectiveness and inefficiency in existing optimization-based jailbreak techniques for LLMs: the gap between gradient-guided candidate ranking and the discrete trigger update process. We hence propose a novel optimization-based jailbreak pipeline, named SKEWACT, which prioritizes candidates that steer optimization toward local minima in wide convex regions of the loss landscape, thereby mitigating the impact of gradient overshooting caused by discrete trigger update. We evaluate SKEWACT against the renowned optimization-based jailbreak method, GCG, on seven LLMs with varying architectures and sizes in both white-box and black-box settings. SKEWACT achieves more than a 10% increase in Attack Success Rate (ASR) and over a 0.02 reduction in converged loss on average, demonstrating significant improvements in both effectiveness and efficiency.

## Acknowledgments and Disclosure of Funding

We are grateful to the Center for AI Safety for providing computational resources. This work was funded in part by the National Science Foundation (NSF) Awards SHF-1901242, SHF-1910300, Proto-OKN 2333736, IIS-2416835, DARPA VSPELLS - HR001120S0058, IARPA TrojAI W911NF-19-S0012, ONR N000141712045, N000141410468 and N000141712947. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

## References

[1] Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, Ilya Sutskever, and Jeffrey Wu. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. In *International Conference on Machine Learning (ICML)*, 2024.

[2] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. In *R0-FoMo: Robustness of Few-shot and Zero-shot Learning in Large Foundation Models*.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[4] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2023.

[5] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

[6] Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, et al. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv preprint arXiv:2209.07858*, 2022.

[7] Chuan Guo, Alexandre Sablayrolles, Hervé Jégou, and Douwe Kiela. Gradient-based adversarial attacks against text transformers. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[8] Microsoft Threat Intelligence. Staying ahead of threat actors in the age of ai. *See https://www.microsoft.com/en-us/security/blog/2024/02/14/staying-ahead-of-threat-actors-in-the-age-of-ai/ (accessed 14 February 2024)*, 2024.

[9] Xiaojun Jia, Tianyu Pang, Chao Du, Yihao Huang, Jindong Gu, Yang Liu, Xiaochun Cao, and Min Lin. Improved techniques for optimization-based jailbreaking on large language models. *arXiv preprint arXiv:2405.21018*, 2024.

[10] 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. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[11] Xiaolong Jin, Zhuo Zhang, and Xiangyu Zhang. Multiverse: Exposing large language model alignment problems in diverse worlds. *arXiv preprint arXiv:2402.01706*, 2024.

[12] Erik Jones, Anca Dragan, Aditi Raghunathan, and Jacob Steinhardt. Automatically auditing large language models via discrete optimization. In *International Conference on Machine Learning (ICML)*, 2023.

[13] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103:102274, 2023.

[14] Xuan Li, Zhanke Zhou, Jianing Zhu, Jiangchao Yao, Tongliang Liu, and Bo Han. Deepinception: Hypnotize large language model to be jailbreaker. *arXiv preprint arXiv:2311.03191*, 2023.

[15] Zeyi Liao and Huan Sun. Amplegcg: Learning a universal and transferable generative model of adversarial suffixes for jailbreaking both open and closed llms. In *Conference on Language Modeling (COLM)*, 2024.

[16] Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. Autodan: Generating stealthy jailbreak prompts on aligned large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

[17] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML)*, 2013.

[18] Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, et al. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. In *International Conference on Machine Learning (ICML)*, 2024.

[19] Mantas Mazeika, Andy Zou, Norman Mu, Long Phan, Zifan Wang, Chunru Yu, Adam Khoja, Fengqing Jiang, Aidan O'Gara, Ellie Sakhaee, Zhen Xiang, Arezoo Rajabi, Dan Hendrycks, Radha Poovendran, Bo Li, and David Forsyth. Tdc 2023 (llm edition): The trojan detection challenge. In *NeurIPS Competition Track*, 2023.

[20] Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. Relu strikes back: Exploiting activation sparsity in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[21] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010.

[22] OpenAI. Gpt-3.5 turbo api. *https://platform.openai.com/docs/models/gpt-3-5-turbo*, 2023.

[23] OpenAI. Gpt-4o system card. *https://openai.com/index/gpt-4o-system-card/*, 2024.

[24] Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022.

[25] Guangyu Shen, Siyuan Cheng, Kaiyuan Zhang, Guanhong Tao, Shengwei An, Lu Yan, Zhuo Zhang, Shiqing Ma, and Xiangyu Zhang. Rapid optimization for jailbreaking llms via subconscious exploitation and echopraxia. *arXiv preprint arXiv:2402.05467*, 2024.

[26] Guangyu Shen, Yingqi Liu, Guanhong Tao, Qiuling Xu, Zhuo Zhang, Shengwei An, Shiqing Ma, and Xiangyu Zhang. Constrained optimization with dynamic bound-scaling for effective nlp backdoor defense. In *International Conference on Machine Learning (ICML)*, 2022.

[27] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[28] Irene Solaiman, Miles Brundage, Jack Clark, Amanda Askell, Ariel Herbert-Voss, Jeff Wu, Alec Radford, Gretchen Krueger, Jong Wook Kim, Sarah Kreps, et al. Release strategies and the social impacts of language models. *arXiv preprint arXiv:1908.09203*, 2019.

[29] Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, et al. Prosparse: Introducing and enhancing intrinsic activation sparsity within large language models. *arXiv preprint arXiv:2402.13516*, 2024.

[30] Alex Tamkin, Miles Brundage, Jack Clark, and Deep Ganguli. Understanding the capabilities, limitations, and societal impact of large language models. *arXiv preprint arXiv:2102.02503*, 2021.

[31] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[32] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[33] Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[34] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. A systematic evaluation of large language models of code. In *ACM SIGPLAN International Symposium on Machine Programming (MAPS)*, 2022.

[35] Jiahao Yu, Xingwei Lin, and Xinyu Xing. Gptfuzzer: Red teaming large language models with auto-generated jailbreak prompts. *arXiv preprint arXiv:2309.10253*, 2023.

[36] Yi Zeng, Hongpeng Lin, Jingwen Zhang, Diyi Yang, Ruoxi Jia, and Weiyan Shi. How johnny can persuade llms to jailbreak them: Rethinking persuasion to challenge ai safety by humanizing llms. In *Annual Meeting of the Association for Computational Linguistics: System Demonstrations (ACL)*, 2024.

[37] Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. Relu$^2$ wins: Discovering efficient activation functions for sparse llms. *arXiv preprint arXiv:2402.03804*, 2024.

[38] Sicheng Zhu, Ruiyi Zhang, Bang An, Gang Wu, Joe Barrow, Zichao Wang, Furong Huang, Ani Nenkova, and Tong Sun. Autodan: Automatic and interpretable adversarial attacks on large language models. *arXiv preprint arXiv:2310.15140*, 2023.

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