SoK: Evaluating Jailbreak Guardrails for Large Language Models

Xunguang Wang*, Zhenlan Ji*, Wenxuan Wang†, Zongjie Li*, Daoyuan Wu*, Shuai Wang*‡
*The Hong Kong University of Science and Technology
{xwanghm, zjiae, zligo, daoyuan, shuaiw}@cse.ust.hk
†Renmin University of China
wangwenxuan@ruc.edu.cn
‡Corresponding author

***Abstract*—Large Language Models (LLMs) have achieved remarkable progress, but their deployment has exposed critical vulnerabilities, particularly to jailbreak attacks that circumvent safety mechanisms. Guardrails—external defense mechanisms that monitor and control LLM interactions—have emerged as a promising solution. However, the current landscape of LLM guardrails is fragmented, lacking a unified taxonomy and comprehensive evaluation framework. In this Systematization of Knowledge (SoK) paper, we present the first holistic analysis of jailbreak guardrails for LLMs. We propose a novel, multi-dimensional taxonomy that categorizes guardrails along six key dimensions, and introduce a Security-Efficiency-Utility evaluation framework to assess their practical effectiveness. Through extensive analysis and experiments, we identify the strengths and limitations of existing guardrail approaches, explore their universality across attack types, and provide insights into optimizing defense combinations. Our work offers a structured foundation for future research and development, aiming to guide the principled advancement and deployment of robust LLM guardrails.**

1. Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide range of applications, revolutionizing fields from natural language understanding to content generation [1], [2], [3], [4], [5], [6], [7], [8]. However, their increasing sophistication and widespread adoption have also exposed significant vulnerabilities. A prominent concern is their susceptibility to *jailbreak attacks* [9], [10], where adversaries craft malicious inputs to bypass safety alignments and elicit harmful, biased, or unintended responses. The proliferation of such attacks underscores the urgent need for robust defense mechanisms. Among various defense strategies, *guardrails* [11], [12], [13] have emerged as a promising approach, aiming to monitor and control LLM interactions without altering the underlying model's weights or core functionalities.

Guardrail-based defenses offer a distinct advantage over prompt-based [14] or tuning-based [15], [16] methods as they can effectively filter jailbreak attempts while preserving the integrity of the target LLM's original output capabilities. Despite their potential, the current landscape of LLM guardrails is characterized by *siloed innovation*. Numerous research teams and organizations have proposed various guardrail solutions, often tailored to specific scenarios, attack vectors (e.g., focusing primarily on single-turn attacks), or proprietary systems. This ad-hoc development has resulted in a fragmented ecosystem of defense mechanisms, lacking a unified understanding or a systematic classification framework to position and compare these disparate efforts.

The absence of a systematic perspective contributes directly to a critical limitation in existing guardrails: a general lack of *universality*. Many solutions are not readily adaptable across different LLMs, attack types, or deployment contexts. Furthermore, current evaluation practices for LLM guardrails often fall short of reflecting real-world operational constraints. Evaluations predominantly focus on raw defense efficacy against specific jailbreak benchmarks, frequently overlooking crucial factors such as *computational cost* (e.g., inference latency, GPU resource consumption) and *utility* (e.g., the rate of misclassifying benign prompts as malicious, thereby degrading user experience). This narrow evaluation scope hinders a comprehensive understanding of the practical trade-offs involved in deploying guardrails.

To address these critical gaps, this Systematization of Knowledge (SoK) paper provides the first comprehensive analysis and structuring of the rapidly evolving field of jailbreak guardrails for LLMs. We aim to consolidate the disparate research efforts, offering a clear and structured understanding of the current state-of-the-art. Our primary contributions are threefold: (1) we propose a novel, multi-dimensional taxonomy for classifying LLM guardrails, enabling a nuanced understanding of their design characteristics; (2) we introduce a holistic evaluation framework centered on the *Security-Efficiency-Utility* trifecta, promoting more practical and comprehensive assessments; and (3) we conduct extensive analysis based on our framework, yielding valuable insights into the performance of existing guardrails and identifying promising avenues for future research.

Specifically, our contributions are as follows:

- **A Multi-Dimensional Guardrail Taxonomy:** We propose the first comprehensive taxonomy to categorize LLM guardrails along six critical dimensions:

- *Intervention Stage*: Characterizing when the guardrail operates (Pre-processing, Intra-processing, or Post-processing of LLM interactions).
- *Technical Paradigm*: Identifying the underlying mechanism (Rule-based, Model-based, or LLM-based).
- *Security Granularity*: Defining the scope of the guardrail detection (Token-level, Sequence-level, or Session-level).
- *Reactivity*: Distinguishing between static (pre-defined) and dynamic (adaptive) defense strategies.
- *Applicability*: Considering the guardrail's requirements regarding LLM access (White-box vs. Black-box).
- *Interpretability*: Assessing the transparency of the guardrail's decision-making process and noting that increased interpretability might inadvertently introduce new attack surfaces or reasoning errors.

- **A Security-Efficiency-Utility Evaluation Framework:** We introduce a novel framework for evaluating guardrails that balances three crucial aspects:
  - *Security*: Measuring the defense performance against a diverse range of jailbreak attacks.
  - *Efficiency*: Quantifying the operational overhead, including inference delay and GPU memory consumption.
  - *Utility*: Assessing the impact on legitimate user interactions, primarily through the false positive rate on benign queries.
- **Experimental Findings and Optimization Insights:** We leverage our taxonomy and evaluation framework to analyze existing guardrails and explore future directions:
  - We conduct a tri-objective (Security-Efficiency-Utility) evaluation of mainstream guardrail methods to identify balanced solutions and those effective against diverse jailbreak categories.
  - We investigate specific hypotheses, such as the efficacy of session-level guardrails against multi-turn attacks, the influence of intervention stage on latency, the impact of technical paradigms on resource consumption, and the relationship between security granularity and utility.
  - We explore the *universality* of guardrails by assessing their performance against other attack modalities, such as prompt injection attacks.

This SoK aims to provide researchers and practitioners with a clear roadmap for understanding, developing, and deploying LLM jailbreak guardrails. By systematizing existing knowledge and proposing a comprehensive evaluation methodology, we hope to foster more principled advancements in this critical area of LLM security. The code is available at https://github.com/xunguangwang/SoK4JailbreakGuardrails.

2. Jailbreak Attacks in LLMs

In this section, we first formally describe the jailbreak in LLMs and then introduce several typical jailbreak methods.

Jailbreak Formulation. The jailbreak phenomenon indicates that specific malicious instructions can bypass the safety mechanisms of LLMs, leading to the generation of harmful or unethical outputs. This is particularly concerning as it highlights the potential for adversaries to exploit vulnerabilities in LLMs to produce toxic or harmful content. The jailbreak process can be viewed as a two-step procedure: (1) crafting an adversarial prompt P that elicits a harmful response from the LLM, and (2) evaluating the generated response R against a predefined harmful objective G using a classifier JUDGE. JUDGE returns 'True' if the generated response R meets the harmful objective G, i.e., JUDGE $=$ True, otherwise 'False'. Let $\mathcal{T}$ represent the LLM's vocabulary. Formally, we can define the classifier JUDGE $: \mathcal{T}^\star \times \mathcal{T}^\star \rightarrow \{\text{True}, \text{False}\}$. The adversary's goal is to maximize the probability of generating responses classified as satisfying the harmful objective G. This can be expressed mathematically as:

$$\sup_{P \in \mathcal{T}^\star} \Pr_{R \sim \text{LLM}(P)} [\text{JUDGE}(R, G) = \text{True}] \tag{1}$$

where $\Pr$ denotes the probability, which accounts for the inherent stochasticity of the LLM's outputs when processing the input prompt P. The adversary iteratively refines prompts to identify those that maximize the likelihood of producing outputs deemed harmful by the classifier.

Existing Jailbreak Attacks. Jailbreak attacks can be broadly categorized into two types: single-turn and multi-turn jailbreaks. Single-turn jailbreaks involve crafting a single prompt to elicit harmful responses, while multi-turn jailbreaks exploit the interactive nature of LLMs by engaging in a dialogue with the model over multiple turns. Due to the maturity of research on single-turn attacks and the relative scarcity of multi-turn attack studies, we further divide single-turn attacks into four types: manual methods, optimization-based approaches, generation-based strategies, and implicit jailbreaks.

- **Manual Jailbreaks.** These attacks involve crafting prompts that exploit vulnerabilities in LLMs [17], [18], [19], [20], [21]. Wei et al. [18] identified two key weaknesses—out-of-distribution inputs and conflicts between safety objectives and model capabilities—to inform prompt design. Deng et al. [21] introduced AIM (Always Intelligent and Machiavellian), a proof-of-concept jailbreak prompt that served as a foundation for generating additional adversarial prompts. Shen et al. [20] proposed JailbreakHub, a crowdsourcing framework for collecting diverse jailbreak prompts.
- **Optimization-based Jailbreaks.** These methods iteratively refine adversarial prompts using techniques like gradient-based optimization or search strategies [9], [10], [22], [23], [24]. GCG [9] introduced a greedy coordinate gradient method to optimize adversarial suffixes, enabling transferable jailbreaks across models and prompts.

Sitawarin et al. [22] extended this with GCG++, leveraging a proxy model to enhance optimization. Beyond gradient-based techniques, JSAA [23] employed random search for suffix optimization, while AutoDAN [10] used a hierarchical genetic algorithm to create human-readable jailbreak prompts. RLbreaker [25] utilized reinforcement learning to efficiently search for adversarial prompts, outperforming stochastic methods like JSAA and AutoDAN.

- **Generation-based Jailbreaks.** These attacks use auxiliary LLMs to produce adversarial prompts [21], [26], [27], [28], [29], [30]. PAIR [27] employs a feedback loop where the attacking LLM adjusts outputs based on the target LLM's responses. Mehrotra et al. [28] enhanced this approach using tree-of-thought reasoning [31]. LLM-Fuzzer [32] automates adversarial prompt generation by mutating human-written templates. Additionally, Advprompter [29] trains a fine-tuned LLM to create both effective and human-readable adversarial prompts.
- **Implicit Jailbreaks.** These techniques disguise malicious intent within query text to bypass LLM safety mechanisms [18], [33], [34], [35], [36], [37], [38], [39], [40]. For instance, Handa et al. [33] demonstrated word substitution as a simple evasion method. DrAttack [34] decomposes harmful prompts into smaller, less detectable sub-prompts. Puzzler [35] embeds clues within queries to guide the LLM toward producing harmful outputs indirectly. Another approach involves translating harmful prompts into languages where LLM safety mechanisms are weaker [18], [36], [37], [38], [39], [40]. Deng et al. [36] and Yong et al. [37] found that low-resource languages, such as Zulu, often exhibit less robust safety alignment. Obfuscation techniques, including encoding or encrypting harmful prompts, further reduce LLM sensitivity to malicious inputs [18], [40], [41].
- **Multi-turn Jailbreaks.** One multi-turn attack strategy is the fine-grained task decomposition, which decomposes the original malicious query into several less harmful sub-questions [42], [43], [44]. While this decomposition strategy successfully circumvents current safety mechanisms, it may be easily mitigated by including these finer-grained harmful queries in safety training data. Alternatively, researchers propose to use human red teamers to expose vulnerabilities of LLMs against multi-turn attacks [45]. Moreover, Yang et al. [46] depends on the heuristics from [27] and its seed examples to implement its attacks. Crescendo [47] gradually steers benign initial queries towards more harmful topics. The implementation of Crescendo is based on the fixed and human-crafted seed instances, making it challenging to generate diverse and effective attacks. By contrast, ActorAttack [48] proposes to discover diverse attack clues inside the model's prior knowledge. X-Teaming [49] achieves more effective and diverse multi-turn attacks by adaptive collaborative agents for planning, attack optimization, and verification.

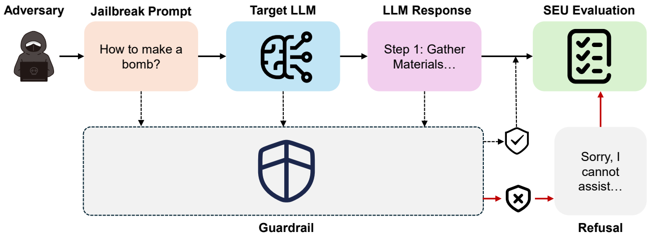


Figure 1. Illustration of a guardrail pipeline.

3. Definition, Taxonomy & Evaluation

3.1. Jailbreak Guardrail Definition

A "Jailbreak Guardrail" refers to a specialized security mechanism designed for LLM systems, specifically to detect and prevent "jailbreak" attacks [11], [12], [13], [50]. Such guardrails typically function as a defensive layer, scrutinizing user inputs before they reach the LLM or vetting the model's outputs before they are presented to the user. The primary objective is to ensure that the LLM does not generate harmful, unethical, or policy-violating content.

In the context of the jailbreak formulation introduced in Section 2, where an adversary crafts a prompt P aiming to elicit a response $R = \text{LLM}(P)$ such that $\text{JUDGE}(R, G) =$ True (indicating a harmful outcome based on objective G), a jailbreak guardrail introduces an additional checkpoint. Let $\mathcal{G}_R$ denote the guardrail system, and $\text{Assess}(\mathcal{G}_R, X)$ be its assessment function, which returns *allow* if content X (either P, R, or the internal feature F of the LLM) is deemed permissible, and *block* otherwise. When *block* is executed, the final response R is replaced by a safe response R', such as "`Sorry, I cannot assist with that`", as shown in Figure 1.

A jailbreak attack is considered successful in the presence of such a guardrail if, and only if, the protective mechanisms of both the target LLM (i.e., its inherent safety alignment) and the guardrail are circumvented. This means the guardrail must deem the interaction (either the input prompt or the generated output) as acceptable, while the target LLM still produces content classified as harmful. More formally, if the guardrail inspects the input prompt P, a successful jailbreak occurs when:

$$\text{Assess}(\mathcal{G}_R, P) = \textit{allow} \;\wedge\; \text{JUDGE}(R, G) = \text{True}. \quad (2)$$

Alternatively, if the guardrail inspects the model's output $R = \text{LLM}(P)$, a successful jailbreak is characterized by:

$$\text{Assess}(\mathcal{G}_R, R) = \textit{allow} \;\wedge\; \text{JUDGE}(R, G) = \text{True}. \quad (3)$$

This highlights that a successful adversary must not only craft a prompt that bypasses the LLM's internal safety measures but also deceives the guardrail into permitting the harmful interaction or content.

As jailbreak techniques become increasingly sophisticated and diverse (as noted in Section 2), these guardrails face mounting challenges. They must evolve beyond detecting overtly malicious requests to identify subtle and nuanced jailbreak patterns and adversarial manipulations. The continuous enhancement of jailbreak guardrails is therefore critical for improving the safety, security, and regulatory compliance of AI applications.

3.2. Jailbreak Guardrail Taxonomy

This section categorizes existing guardrail approaches along several key dimensions. Our taxonomy considers:

- **Intervention Stages**: This dimension delineates *when* the guardrail operates within the LLM interaction pipeline—either at *pre-processing* (before the input reaches the LLM), *intra-processing* (during the LLM's inference), or *post-processing* (after the LLM generates an output).
- **Technical Paradigms**: This refers to the underlying *methodology* employed by the guardrail. Approaches are classified as *rule-based* (relying on predefined rules or patterns), *model-based* (using statistical models or classifiers), or *LLM-based* (leveraging another LLM for analysis and decision-making).
- **Safety Granularity**: This specifies the *level of detail* at which the safety analysis is performed. It can be *token-level* (examining individual words or sub-word units), *sequence-level* (evaluating entire prompts or responses), or *session-level* (considering the context of the entire conversation history).
- **Reactiveness**: This dimension distinguishes how a guardrail responds to potentially harmful inputs. *Static* defenses analyze inputs without modification, whereas *dynamic* defenses actively alter inputs—for example, through mutation or perturbation—to neutralize adversarial properties while aiming to preserve overall semantic meaning.
- **Applicability**: This criterion assesses the guardrail's suitability for different LLM access models, with a particular emphasis on whether the mechanism can be effectively applied to *black-box* LLMs (i.e., closed-source models or those accessed via remote APIs where internal states are not accessible).
- **Explainability**: This focuses on whether the guardrail method provides interpretable insights into its safety judgments or offers clear rationales for the decisions it makes.

This multi-faceted classification provides a comprehensive framework for understanding and navigating the landscape of LLM guardrails. We have comprehensively compiled existing works on jailbreak guardrails by this taxonomy, as summarized in Table 1.

3.3. Guardrail Evaluation Framework

To enable a comprehensive and practical assessment of LLM guardrails, we propose the *Security-Efficiency-Utility* (SEU) Evaluation Framework. This framework is designed to capture the essential trade-offs involved in deploying guardrails in real-world LLM systems, moving beyond the narrow focus on raw defense efficacy. Below, we detail the three core dimensions of our framework and the specific metrics used for each.

Security: Defense Effectiveness. The primary objective of any guardrail is to enhance the security of LLM systems by mitigating jailbreak attacks. We evaluate defense effectiveness using two complementary metrics:

- **Attack Success Rate (ASR):** ASR measures the proportion of adversarial attempts that successfully bypass the guardrail and elicit harmful or unintended responses from the target LLM. Formally, it is defined as the percentage of attack queries for which the LLM system equipped with the guardrail fails to block or mitigate the attack. A lower ASR indicates stronger defense.
- **Pass Guardrail Rate (PGR):** PGR measures the proportion of jailbreak attempts that successfully bypass the guardrail, indicating that the guardrail has classified the attempt as safe. For pre-processing and intra-processing guardrails, this refers to the proportion of malicious requests that the guardrail incorrectly identifies as benign. For post-processing guardrails, this refers to the proportion of instances where the guardrail fails to detect harmful content in the LLM's response to a jailbreak attempt. A lower PGR signifies a more effective guardrail in blocking attacks.

Efficiency: Computational Overhead. In practical deployments, the operational efficiency of guardrails is a critical consideration, as excessive overhead can degrade user experience and increase infrastructure costs. We assess efficiency along two axes:

- **Extra Delay:** This metric captures the additional response latency introduced by the guardrail. It is computed as the difference between the end-to-end response time of the guardrail + LLM system and that of the standalone target LLM. Formally,

$$\text{Extra Delay} = T_{\text{guardrail + LLM}} - T_{\text{LLM}} \tag{4}$$

  where $T_{\text{guardrail + LLM}}$ and T_{LLM} denote the average response times with and without the guardrail, respectively.
- **GPU Memory Overhead:** This metric measures the increase in peak GPU memory consumption resulting from the integration of the guardrail. It is defined as the difference between the maximum GPU memory usage of the guardrail + LLM system and that of the target LLM alone:

$$\text{GPU Overhead} = M_{\text{guardrail + LLM}} - M_{\text{LLM}} \tag{5}$$

  where $M_{\text{guardrail + LLM}}$ and M_{LLM} represent the peak GPU memory usage with and without the guardrail, respectively.

Utility: Impact on Benign Queries. A robust guardrail should not only block malicious inputs but also preserve the utility of the LLM for legitimate users. We quantify utility loss using the following metric:

- **False Positive Rate (FPR):** FPR measures the proportion of benign (non-malicious) queries that are incorrectly flagged or blocked by the guardrail. It is defined as the percentage of normal user queries that are misclassified as attacks. A lower FPR indicates better utility preservation, as the guardrail minimally disrupts legitimate interactions with the LLM.

Discussion. By jointly considering Security, Efficiency, and Utility, the SEU Evaluation Framework provides a holistic basis for comparing and optimizing LLM guardrails. This tri-objective perspective enables the identification of solutions that achieve a balanced trade-off, rather than excelling in only one dimension at the expense of others. In our experimental analysis (§5 & §6), we employ this framework to systematically evaluate mainstream guardrail methods, offering actionable insights for both researchers and practitioners.

4. Guardrail Analysis Based on Taxonomy

4.1. Intervention Stages

Guardrail mechanisms can be deployed at different stages of the LLM interaction pipeline, including pre-processing, intra-processing, and post-processing. Each stage serves a distinct purpose in identifying and mitigating jailbreak attempts:

Pre-processing Guardrails. These mechanisms operate on user inputs before they reach the target LLM, functioning as the first line of defense against jailbreak attempts. Pre-processing guardrails typically employ detection algorithms to identify potentially harmful prompts and then block them entirely. These guards are particularly valuable for their ability to prevent harmful prompts from ever reaching the model, thus conserving computational resources and reducing potential risks.

Early methods, such as Detecting Perplexity [54] and Perplexity Filter [55], compute the perplexity of input prompts to detect potential adversarial inputs. However, this approach is limited to GCG [9], [22], [30] attacks with unreadable adversarial suffixes amplifying the perplexity.

A more direct approach is to identify the semantic harmfulness of input sequences. Some methods focus on directly identifying toxic phrases or excerpts within the input text [13], [50], [78], while others assess the overall semantic harmfulness of the entire input [12], [50], [51], [52], [56], [63], [64], [69], [70], [72], [73], [74], [77], [79], [81], [91]. For instance, PromptGuard [70] and OpenAI Moderation [52] fine-tune pre-trained classifiers to assess the safety of input prompts. However, pre-processing guardrails may struggle with novel jailbreak techniques that do not exhibit clear patterns, e.g., implicit attack DrAttack [34] conceals malicious content within benign-looking prompts.

A more fundamental approach is to analyze the true intent of the query to filter out jailbreak requests, based on the premise that jailbreak attempts always involve malicious output targets. Leveraging the powerful language understanding capabilities of LLMs, we can directly utilize LLMs to identify the real intentions of requests to determine whether they are jailbreak attempts [13], [68], [86], [91]. For example, SelfDefend [13] with the intent prompt first summarizes the input intention and then assesses whether it constitutes a jailbreak request. Recently, some studies have employed LLM reasoning capabilities to analyze input intent [82], [83], [90] before the safety judgments. For instance, X-Guard [83] employs deep thinking to evaluate potential harms.

> **Summary 1:** *Pre-processing guardrails are the first line of defense against jailbreak attempts, operating on user inputs before they reach the target LLM. They have evolved from simple perplexity detection to semantic harmfulness identification and, most recently, to LLM-based reasoning for analyzing input intent. This evolution is driven by the need to address increasingly sophisticated and covert attack methods.*

Intra-processing Guardrails. These guardrails operate during the LLM's inference process, analyzing internal model features or gradients to detect potential jailbreak attempts. Unlike pre-processing methods, intra-processing guardrails can observe how the model processes inputs internally, providing deeper insights into potential vulnerabilities.

On one hand, intra-processing guardrails rely on gradient information to identify potential jailbreak attempts. These methods analyze the gradients of the model's inputs or parameters during inference to identify unusual patterns or anomalies that may indicate adversarial inputs. For example, GradSafe [58] computes the similarity between the input's gradient w.r.t. the safety-critical parameters and the unsafe reference gradients. Gradient Cuff [61] compare the gradient norm of refusal loss w.r.t. the query prompt with a threshold. Token Highlighter [80] uses the gradient norm of the affirmation loss for each token in the user query to locate the jailbreak-critical tokens.

On the other hand, intra-processing guardrails can analyze the model's internal states for jailbreak detection [65], [66], [67], [75], [76], [85]. These methods leverage the model's hidden states, or other internal representations to identify patterns indicative of jailbreak attempts. For example, Circuit Breaking [67] interrupts the LLM to output harmful content when harmful states are detected. JBShield [85] analyzes the differences of the LLM's internal states between the jailbreak prompts and the benign queries. These approaches can provide more nuanced insights into the model's behavior and vulnerabilities, enabling more effective detection of sophisticated jailbreak techniques. However, these approaches typically require white-box access to the model, which limits their applicability to open-source LLMs or scenarios where model internals are accessible.

> **Summary 2:** *Intra-processing guardrails operate during the LLM's inference process, analyzing internal model features or gradients to detect potential jailbreak attempts. They provide deeper insights into vulnerabilities but require white-box access to the model, limiting their applicability.*

Post-processing Guardrails. These mechanisms evaluate

TABLE 1. Works on guardrails categorized by 6 dimensions. The black circle indicates the guardrail belongs to this dimension, and the white circle otherwise.

Paper	Venue	Intervention Stages			Technical Paradigms			Safety Granularity			Reactiveness		Applicability	Explainability
		Pre-processing	Intra-processing	Post-processing	Rule	Model	LLM	Token	Sequence	Session	Static	Dynamic		
Perspective API [51]	KDD'22 (2202.11176)	●	○	●	○	●	○	○	●	○	●	○	●	○
OpenAI Moderation [52]	AAAI'23 (2208.03274)	●	○	●	○	●	○	○	●	○	●	○	●	○
Self Defense [53]	arXiv:2308.07308	○	○	●	○	○	●	○	●	○	●	○	●	○
Detecting Perplexity [54]	arXiv:2308.14132	●	○	○	○	●	○	○	●	○	●	○	●	●
Perplexity Filter [55]	arXiv:2309.00614	●	○	○	○	●	○	○	●	○	●	○	●	●
erase-and-check [56]	COLM'24 (2309.02705)	●	○	○	○	●	●	○	●	○	○	●	●	○
SmoothLLM [57]	arXiv:2310.03684	○	○	●	●	○	○	●	○	○	○	●	●	●
NeMo Guardrails [11]	EMNLP'23 (2310.10501)	●	○	●	○	○	●	○	●	○	○	●	●	○
Llama Guard [12]	arXiv:2312.06674	●	○	●	○	○	●	○	○	●	●	○	●	●
GradSafe [58]	ACL'24 (2402.13494)	○	●	○	○	●	○	○	●	○	●	○	○	●
SemanticSmooth [59]	arXiv:2402.16192	○	○	●	○	○	●	○	●	○	○	●	●	○
LLMGuard [60]	arXiv:2403.00826	●	○	●	●	●	○	○	●	○	●	○	●	●
Gradient Cuff [61]	NeurIPS'24 (2403.00867)	○	●	○	○	●	○	○	●	○	●	○	○	●
AutoDefense [62]	arXiv.2403.04783	○	○	●	○	○	●	○	●	○	●	○	●	●
RigorLLM [63]	ICML'24 (2403.13031)	●	○	○	○	●	●	○	●	○	○	●	●	●
Aegis [64]	arXiv:2404.05993	●	○	●	○	○	●	○	○	●	●	○	●	●
LLMGuardrail [65]	CCS'24 (2405.04160)	○	●	○	○	●	○	○	●	○	●	○	○	●
RSAA [66]	CAMLIS'24 (2406.03230)	○	●	○	○	●	○	○	●	○	●	○	○	○
Circuit Breaking [67]	NeurIPS'24 (2406.04313)	○	●	○	○	●	○	○	●	○	●	○	○	○
SelfDefend [13]	USENIX Security'25 (2406.05498)	●	○	○	○	○	●	●	●	○	●	○	●	●
GuardAgent [68]	arXiv:2406.09187	●	○	○	○	○	●	○	●	○	●	○	●	●
WildGuard [50]	NeurIPS'24 (2406.18495)	●	○	●	○	○	●	○	○	●	●	○	●	○
R^2-Guard [69]	ICLR'25 (2407.05557)	●	○	●	○	●	○	○	●	○	●	○	●	●
Prompt Guard [70], [71]	Hugging Face (22 July 2024)	●	○	○	○	●	○	○	○	●	●	○	●	○
PrimeGuard [72]	arXiv:2407.16318	●	○	○	○	○	●	○	●	○	●	○	●	●
ShieldGemma [73]	arXiv:2407.21772	●	○	●	○	○	●	○	○	●	●	○	●	●
Adaptive Guardrail [74]	arXiv:2408.08959	●	○	○	○	●	○	○	○	●	●	○	●	○
EEG-Defender [75]	arXiv:2408.11308	○	●	○	○	●	○	○	●	○	●	○	○	●
HSF [76]	arXiv:2409.03788	○	●	○	○	●	○	○	●	○	●	○	○	○
MoJE [77]	AIES'24 (2409.17699)	●	○	○	○	●	○	○	●	○	●	○	●	○
Rapid Response [78]	arXiv:2411.07494	●	○	○	●	●	●	○	●	○	○	●	●	○
Pretrained Embeddings [79]	arXiv:2412.01547	●	○	○	○	●	○	○	●	○	●	○	●	○
Token Highlighter [80]	AAAI'25 (2412.18171)	○	●	○	○	●	○	●	○	○	○	●	○	○
Aegis2.0 [81]	arXiv:2501.09004	●	○	●	○	○	●	○	○	●	●	○	●	●
COT Fine-Tuning [82]	arXiv:2501.13080	●	○	○	○	○	●	○	●	○	●	○	●	●
GuardReasoner [83]	arXiv:2501.18492	●	○	●	○	○	●	○	○	●	●	○	●	●
Constitutional Classifiers [84]	arXiv:2501.18837	●	○	●	○	○	●	○	●	○	●	○	●	○
JBShield [85]	USENIX Security'25 (2502.07557)	○	●	○	○	●	○	○	●	○	●	○	○	●
EDDF [86]	arXiv.2502.19041	●	○	○	○	●	○	○	●	○	●	○	●	●
CURVALID [87]	arXiv:2503.03502	●	○	○	○	●	○	●	●	○	●	○	●	●
MirrorShield [88]	arXiv:2503.12931	○	●	○	○	●	○	○	●	○	●	○	●	●
JailGuard [89]	TOSEM'25 (19 March 2025)	○	○	●	○	●	○	○	●	○	○	●	●	●
X-Guard [90]	arXiv:2504.08848	●	○	●	○	○	●	○	●	○	○	●	●	●
Continuous Detector [91]	arXiv:2504.19440	●	○	●	○	○	●	○	○	●	●	○	●	○
Active Monitoring [91]	arXiv:2504.19440	●	○	●	○	○	●	○	○	●	●	○	●	●

the LLM's generated outputs to identify and filter harmful content. As jailbreak attacks inherently aim to produce harmful outputs, post-processing guardrails serve as a crucial last line of defense. This ensures that even if malicious prompts circumvent earlier detection stages, their resultant outputs can still be intercepted.

Given that post-processing guardrails scrutinize the LLM's generated outputs, a primary strategy involves the direct detection of harmfulness within these responses. The most elementary of these methods employ keyword-based detectors to assess the safety of the LLM's outputs, primarily focusing on determining its jailbroken state [57], [60]. Building upon this foundational technique, a more sophisticated approach involves training dedicated classifiers to distinguish between harmful and harmless responses [51], [52], [60], [69]. For instance, initiatives like the Perspective API [51] and OpenAI Moderation [52] have developed transformer-based classifiers engineered to predict the probability of harmful content appearing in an LLM's response. Similarly, R^2-Guard [69] embeds safety knowledge into probabilistic graphical models, enabling the computation of unsafe probabilities for any given LLM outputs. Elevating this classification paradigm further, albeit with increased computational demands, some techniques leverage the reasoning capabilities of other LLMs to assess response safety [11], [12], [50], [53], [59], [62], [64], [73], [81], [83], [84], [90], [91]. Self Defense [53], for example, filters harmful content by querying an LLM about the harmfulness of the initial response. Llama Guard [12] takes a more contextual approach by considering the input prompt in conjunction with the output to determine the risk category of the response. Progressing towards even more thorough analysis, GuardReasoner [83] and X-Guard [90] employ chain-of-thought reasoning before rendering a safety judgment on the LLM's output.

In addition to assess the semantic harmfulness of the response, some methods cleverly use the difference in responses caused by destroying the adversarial properties of the jailbreak prompt as the basis for judgment [57], [59], [88], [89]. For example, SmoothLLM [57] randomly perturbs/permutes multiple copies of a given input prompt to generate a set of responses, and then vote by the perturbed responses to determine the safety of the original request. SemanticSmooth [59] and JailGuard [89] perform more complex mutations instead of simple character-level changes like SmoothLLM, such as paraphrasing or translation into other languages. MirrorShield [88] generates mirror prompts that preserve the syntactic structure of the input while ensuring semantic safety. Beyond directly assessing the semantic harmfulness of the response, an alternative category of methods ingeniously leverages the discrepancies in outputs that arise from disrupting the adversarial characteristics of the initial jailbreak prompt [57], [59], [88], [89]. SmoothLLM [57], for instance, operates by randomly perturbing or permuting multiple copies of a given input prompt to generate a set of responses; the safety of the original request is then determined by a voting mechanism based on these perturbed responses. Advancing this concept, SemanticSmooth [59] and JailGuard [89] implement more complex mutations than the simple character-level alterations used by SmoothLLM, such as paraphrasing the prompt or translating it into other languages. In a similar vein, MirrorShield [88] generates ”mirror” prompts that aim to preserve the syntactic structure of the input while ensuring its semantic safety. These mutation-based methods capitalize on the inherent properties of jailbreak prompts to identify adversarial inputs, rendering them particularly effective against sophisticated attacks. Nevertheless, they may incur additional computational overhead due to the requirement for numerous response evaluations or intricate input transformations.

Although detecting the output of LLMs may appear more straightforward than deciphering ambiguous prompts and internal features, later methodologies will integrate the input prompts to thoroughly evaluate the safety of the query. However, post-processing safeguards may incur more latency than other paradigms due to the requirement of awaiting the LLM's response. Furthermore, mutation-based techniques that mandate multiple response evaluations are suspected to exacerbate this latency.

> **Summary 3:** *Post-processing guardrails, by operating on the LLM's generated outputs to identify and filter harmful content, act as an essential safeguard. They intercept potentially harmful outputs that bypass earlier detection stages. However, over-reliance on the nature of responses may cause noticeable delay, particularly when employing mutation-based techniques that need multiple evaluations.*

Drawing upon a classification by intervention stages, our analysis reveals a critical gap in the current literature, which motivates the RQ below. As no prior work has systematically investigated this specific dimension, we undertake a thorough examination in this paper.

> **RQ 1:** *Pre-processing guardrails can reject harmful inputs before they reach the target LLM, intra-processing mechanisms operate concurrently during LLM inference, and post-processing techniques must await LLM's outputs. This raises a pertinent question: To what extent does the specific intervention stage of a guardrail, be it pre-processing, intra-processing, or post-processing, influence overall response latency?*

4.2. Technical Paradigms

Guardrail mechanisms employ diverse technical approaches to detect and mitigate jailbreak attempts, including rule-based, model-based, and LLM-based approaches.

Rule-based Guardrails. These guardrails operate by employing predefined rules, patterns, or heuristics to detect potentially harmful inputs or outputs of LLMs. A typical rule-based approach includes utilizing keywords or regular expressions to identify specific patterns tied to harmful content. For instance, SmoothLLM, as referenced in [57], leverages keyword-based detectors to assess the safety of the LLM's outputs, primarily to determine its jailbroken state. Similarly, the PII Detector mentioned in [60] uses regular expressions to identify personal identifiable information, such as phone numbers and emails. This method mirrors the approach taken by the baseline Regex in [78], which also utilizes regular expressions to mitigate jailbreak attacks.

Transitioning from specific examples to an evaluation of their effectiveness, it is evident that while these methods benefit from straightforward and transparent pattern matching—attributes that contribute to their computational efficiency and interpretability—their reliance on predefined patterns can be a significant drawback. Specifically, these rule-based systems may falter when encountering novel jailbreak techniques that deviate from known patterns which inherently limits their capability to combat more sophisticated attacks.

> **Summary 4:** *Rule-based guardrails, while beneficial for their computational efficiency and ease of interpretation, face challenges when dealing with innovative jailbreak techniques that do not match existing predefined patterns.*

Model-based Guardrails. These guardrails adopt classifiers or statistical characteristics to distinguish between benign and harmful queries. Model-based approaches can capture more complex patterns than rule-based methods, enabling them to generalize better to novel jailbreak attempts. Learning a text-based classifier is a common approach for jailbreak detection. On one hand, we can use traditional machine learning models as the classifiers [63], [66], [69], [76], [77], [78], [79]. For instance, K-Nearest Neighbors (KNN) in RigorLLM [63], LightGBM in RSAA [66] and Random Forest in PretrainedEmbeddings [79]. On the other hand, neural networks are also widely applied for the safety classification [51], [52], [56], [60], [65], [70], [76], [78], [87]. For example, HSF [76] and CURVALID [87] use a simple Multilayer Perceptron (MLP) as the classifier. PromptGuard [70] and erase-and-check [56] fine-tune the pre-trained model (i.e, mDeBERTa and DistilBERT, respectively) to distinguish the safe and unsafe inputs. Besides, other methods used statistical characteristics to design their own algorithms on safety distinguish [58], [61], [74], [75], [80], [85], [86], [88], [89]. Detecting Perplexity [54] and Perplexity Filter [55] classify the input as a jailbreak request if the perplexity is higher than a threshold. Gradient discrepancies between safe prompts and adversarial prompts are employed in GradSafe [58], GradientCuff [61] and TokenHighlighter [80]. JailGuard [89] identify the jailbroken state of responses by computing their KL-divergence. EEG-Defender [75], JBShield [85] and MirrorShield [88] take the model's internal feature similarities between the input and the jailbreak prompt as judgment basis.

The essence of model-based guardrails is to find a classification standard in distinguishing the benign and harmful requests, whether to learn a classifier or design a statistical algorithm. Compared with rule-based methods, model-based approaches can capture more complex patterns and generalize better to novel jailbreak attempts. They can also adapt to evolving threats by retraining or fine-tuning the classifiers. However, these methods typically require substantial training data and computational resources, especially when using deep learning models.

> **Summary 5:** *Model-based guardrails, by adopting classifiers or statistical characteristics to distinguish between benign and harmful queries, can capture more complex patterns than rule-based methods, enabling them to generalize better to novel jailbreak attempts. However, these methods typically require substantial training data and computational resources, especially when using deep learning models.*

> **Observation:** *Intra-processing guardrails are basically model-based guardrails, which use LLM's internal features to detect potential jailbreak attempts. This is because model-based methods analyze the features and build classifiers instead of using simple character matching or a more complex LLM.*

LLM-based Guardrails. LLM-based guardrails represent a sophisticated approach to security, harnessing the inherent inferring capabilities of LLMs themselves to identify and counteract jailbreak attempts. Within this paradigm, research has progressed through distinct phases, each characterized by evolving methodologies.

Initially, methods tended to focus on directly determining the harmfulness of a request or providing a summary analysis after the judgment. For example, Self Defense [53] directly employs the target LLM to assess the safety of its own generated responses and subsequently furnish an explanation for its findings. In a similar vein, Llama Guard [12] operates by first identifying an unsafe text and then assigning it to a harmfulness category. Complementing these approaches, WildGuard [50] offers a multi-faceted assessment, simultaneously reporting the harmfulness status of the input prompt, the generated response, and whether the response was ultimately refused.

Recent methods tend to conduct a detailed analysis before making a safety judgment. For example, SelfDefend [13] first summarizes the input intention and then assesses whether it constitutes a jailbreak request. GuardReasoner [83] and X-Guard [90] employ chain-of-thought reasoning to analyze the potential harms and finally give a safety judgment. More recently, however, there has been a discernible shift towards methodologies that conduct a more detailed, upfront analysis before arriving at a safety judgment. Illustrating this trend, SelfDefend [13] first summarizes the underlying intention of the input and then assesses whether this intention constitutes a jailbreak request. Building upon this principle of preliminary in-depth analysis, both GuardReasoner [83] and X-Guard [90] employ chain-of-thought reasoning. This allows them to meticulously trace and analyze potential harms associated with a query, culminating in a final safety judgment.

Undeniably, the strength of these LLM-driven techniques lies in the excellent language understanding intrinsic to the models themselves. As a result, these approaches demonstrate considerable efficacy in detecting a diverse range of jailbreak attempts and notably improve the explainability of the safety judgments they provide. Nevertheless, a crucial trade-off exists. While effective and explainable, these advanced guardrails may introduce substantially more computational overhead when compared to rule-based and model-based techniques.

> **Summary 6:** *Employing the reasoning capabilities of LLMs, LLM-based guardrails not only detect and mitigate jailbreak attempts effectively but also improve the explainability of safety judgments. Nonetheless, they introduce significantly greater computational overhead than traditional rule-based and model-based methods.*

We now present one RQ that focuses on the cost of LLM-based guardrails, which is a crucial aspect of their practical deployment. This RQ is particularly relevant given the increasing complexity and resource demands of LLM-based approaches, especially in environments with limited computational resources.

> **RQ 2:** *Given that rule-based, model-based, and LLM-based guardrails inherently possess different levels of computational complexity and resource requirements, a significant practical question emerges: To what extent does the choice of technical paradigm directly influence the GPU memory footprint of LLM guardrail mechanisms during their operational deployment?*

4.3. Safety Granularity

Guardrail mechanisms can operate at 3 different levels of detection granularity: token-level for individual words or tokens, sequence-level for an entire prompt or response, and session-level for entire conversation sessions.

Token-level Guardrails. These guardrails analyze individual tokens or small token groups to identify potentially

TABLE 2. THE DETAILS OF OUR COLLECTED BENCHMARK DATASETS.

Dataset	# Prompts	Jailbreak Methods
JailbreakHub [20]	1000	IJP [20]
JailbreakBench [92]	100	GCG [9], AutoDAN [10]
		TAP [28], LLM-Fuzzer [32]
		DrAttack [34]
		X-Teaming [49]
MultiJail [36]	315	MultiJail
SafeMTData [48]	600	ActorAttack [48]
AlpacaEval [93]	805	Normal Prompts
OR-Bench [94]	1000	Normal Prompts

harmful elements within inputs or outputs. Token-level approaches can pinpoint specific problematic components within a text, enabling more precise interventions. For instance, Token Highlighter [80] identifies specific tokens that contribute to harmful outputs. These fine-grained approaches enable targeted interventions but may miss harmful content that emerges from the broader context rather than specific tokens.

Sequence-level Guardrails. These guardrails evaluate entire prompts or responses as cohesive units, considering the overall semantic meaning rather than individual components. Sequence-level approaches can capture harmful content that emerges from the interaction between different parts of a text. For example, Llama Guard [12] and ShieldGemma [73] assess the holistic safety of the prompt sequence, while Constitutional Classifiers [84] evaluate outputs against predefined safety principles. These approaches can better capture contextual harms but may provide less granular insights into specific problematic elements.

Session-level Guardrails. These guardrails monitor entire conversation sessions, tracking the evolution of dialogue across multiple turns to identify potential jailbreak attempts that unfold gradually. Session-level approaches can detect sophisticated multi-turn attacks that might appear benign when individual messages are analyzed in isolation. For instance, Adaptive Guardrail [74] maintain conversation state to identify harmful patterns across turns. These comprehensive approaches are particularly valuable against advanced jailbreak techniques that exploit the sequential nature of conversations but typically require more complex implementation and greater computational resources. We now present two RQs that explore the impact of safety granularity on the effectiveness and utility of guardrail mechanisms.

> **RQ 3:** *To what extent are current session-level guardrails truly effective in defending against sophisticated multi-turn jailbreak attacks?*

> **RQ 4:** *How does the choice of safety granularity (i.e., token, sequence, or session-level) impact the utility of LLMs when implementing guardrail mechanisms?*

5. Benchmark & Leaderboard

5.1. Evaluation Setup

Datasets and Target Models. Based on the five categories of existing jailbreak attacks we surveyed in §2 — manual, optimization-based, generation-based, implicit, and multi-turn jailbreaks — we identify representative jailbreak attack methods in each category. We then collect six benchmark datasets, **JailbreakHub** [20], **JailbreakBench** [92], **SafeMTData** [48], **MultiJail** [36], **AlpacaEval** [93] and **OR-Bench** [94], from which we use their user prompts for testing diverse guardrails. Table 2 lists the details of our collected benchmark datasets. *JailbreakHub* is a framework that collects and categorizes wild jailbreak prompts designed to bypass safety restrictions in LLMs. We randomly sample 1,000 in-the-wild prompts (IJP) from JailbreakHub as manual attacks. *JailbreakBench* is an open-source robustness benchmark specifically designed to evaluate and measure the vulnerability of LLMs to jailbreak attacks. We use a set of 100 harmful instructions from JailbreakBench to drive optimization-based jailbreaks (GCG [9] and AutoDAN [10]), generation-based jailbreaks (TAP [28] and LLM-Fuzzer [32]), implicit jailbreak attack DrAttack [34], and multi-turn jailbreak attack X-Teaming [49]. *SafeMTData* is a dataset contains the initial multi-turn jailbreak prompts produced by ActorAttack [48]. We choose 600 queries from SafeMTData as multi-turn jailbreak attacks. *MultiJail* is the first manually-created multilingual jailbreak dataset, covering both high-resource and low-resource languages. We choose 315 jailbreak prompts in Bengali from MultiJail as multilingual jailbreaks. *AlpacaEval* is an automatic evaluation framework designed to assess the performance of instruction-following LLMs. We use 805 instructions from AlpacaEval as normal prompts. *OR-Bench* is the first large-scale benchmark measuring LLMs' over-refusal of 80,000 seemingly toxic but benign prompts across multiple categories. We randomly select 1,000 prompts from OR-Bench as normal prompts. Due to intra-processing guardrails are available to protect white-box LLMs, we evaluate the defense performance of guardrails on open-source LLMs, including **Llama-3-8B-Instruct** [6], and **Vicuna-13b-v1.5** [4], covering diverse model architectures and sizes. Due to page limitation, readers may refer to Appendix A for the results on Vicuna-13b-v1.5.

Metrics. We measure the defense performance of guardrails by the attack success rate (ASR), i.e., the frequency with which jailbreak prompts in a benchmark dataset bypass the guardrail and the target LLM. The lower the ASR, the stronger the defense performance. Adopting the methodology from prior studies ([48], [49], [95], [96]), we leverage GPT-4o's robust evaluation capabilities for assessment. This involves supplying GPT-4o, acting as a judge, with harmful objectives and the corresponding model outputs. The judge assigns a score from 1 to 5, where higher scores denote more harmful and detailed model responses. An attack is deemed successful if it receives a score of 5 from the GPT-

TABLE 3. The ASR (↓) / PGR (↓) results for the target LLM (Llama-3-8B-Instruct) with different guardrails against five major categories of jailbreak attacks, including row averages. (Pre) and (Post) denote the pre-processing and post-processing versions of the guardrails, respectively. (Direct) and (Intent) denote the direct prompt and intent prompt based versions of SelfDefend [13], respectively.

Guardrails	Manual	Optimization-based		Generation-based		Implicit		Multi-turn		Average
	IJP	GCG	AutoDAN	TAP	LLM-Fuzzer	DrAttack	MultiJail	ActorAttack	X-Teaming	
LLma-3-8B-Insturct	0.078/-	0.130/-	0.020/-	0.130/-	0.490/-	0.100/-	0.044/-	0.227/-	0.910/-	0.237/-
PerplexityFilter	0.078/1.000	0.100/0.620	0.020/1.000	0.140/1.000	0.480/1.000	0.100/1.000	0.044/1.000	0.227/1.000	0.960/1.000	0.239/0.958
SmoothLLM	0.115/0.261	0.020/0.020	0.030/0.110	0.140/0.170	0.500/0.810	0.150/0.660	0.032/0.575	0.893/0.893	0.850/0.910	0.303/0.490
Llama Guard (Pre)	0.062/0.563	0.100/0.390	0.020/0.480	0.140/0.460	0.450/0.680	0.100/0.840	0.044/0.952	0.220/0.967	0.910/1.000	0.227/0.704
Llama Guard (Post)	0.061/0.061	0.090/0.090	0.020/0.020	0.150/0.150	0.390/0.390	0.090/0.090	0.041/0.041	0.223/0.223	0.960/0.960	0.225/0.225
GradSafe	0.077/0.599	0.130/0.770	0.010/0.040	0.070/0.580	0.450/0.580	0.100/0.420	0.044/0.917	0.188/0.863	0.950/0.990	0.224/0.640
GradientCuff	0.016/0.058	0.080/0.140	**0.000**/0.020	0.070/0.090	0.360/0.480	0.030/0.130	**0.016**/0.149	0.118/0.648	**0.640/0.710**	0.148/0.269
SelfDefend (Direct)	0.020/0.267	0.030/0.080	0.010/0.120	0.080/0.180	0.090/0.140	0.070/0.590	0.038/0.752	0.133/0.702	0.970/0.990	0.160/0.425
SelfDefend (Intent)	0.022/0.285	0.030/0.070	0.010/0.130	0.120/0.180	0.030/0.130	**0.010**/0.130	0.032/0.584	0.152/0.767	0.940/0.970	0.150/0.361
WildGuard (Pre)	0.004/0.033	0.020/0.020	**0.000**/0.020	**0.030**/0.060	**0.000/0.000**	0.080/0.500	0.044/0.797	0.150/0.757	0.960/0.980	0.143/0.352
WildGuard (Post)	0.020/0.020	0.050/0.050	0.010/0.010	0.060/0.060	0.090/0.090	0.060/0.060	0.035/0.035	0.148/0.148	0.950/0.950	0.158/0.158
Prompt Guard	**0.000/0.000**	**0.000**/0.080	0.020/0.420	0.170/0.940	**0.000/0.000**	0.100/0.940	0.044/1.000	0.225/0.995	0.910/1.000	0.163/0.597
GuardReasoner (Pre)	**0.000**/0.009	**0.000/0.000**	**0.000**/0.010	**0.030**/0.070	0.010/0.020	0.080/0.360	0.029/0.349	0.143/0.740	0.920/0.960	**0.135**/0.280
GuardReasoner (Post)	0.023/0.023	0.040/0.040	**0.000/0.000**	0.050/**0.050**	0.050/0.050	0.030/**0.030**	0.022/**0.022**	**0.107/0.107**	0.950/0.950	0.141/**0.141**

4o Judge. For a detailed explanation of the scoring rubric, please see [48], [95].

Attack Configuration. To assess the jailbreak defense performance of guardrails, we employ the most widely used jailbreak attacks, including a manual attack (**IJP** [20]), optimization-based attacks (**GCG** [9] and **AutoDAN** [10]), generation-based attacks (**TAP** [28] and **LLM-Fuzzer** [32]), implicit attacks (**DrAttack** [34] and **MultiJail** [36]), and multi-turn attacks (**ActorAttack** [48] and **X-Teaming** [49]). In the context of *IJP*, 1,000 adversarial queries were randomly sampled from the forbidden question set with jailbreak prompts [97], curated by JailbreakHub. Regarding *GCG*, its individual variant was selected, and the adversarial suffix was optimized against the target LLM employing a batch size of 512 and subjected to 500 optimization iterations. For the *AutoDAN* methodology, the hierarchically-guided genetic algorithm variant, specifically AutoDAN-HGA, was adopted. The genetic algorithm integral to AutoDAN-HGA operates with a crossover probability of 0.5, a mutation probability of 0.01, and undergoes 500 optimization iterations. Concerning *TAP*, the Vicuna-13b-v1.5 model [4] was utilized as the attacking agent. The parameters for TAP were configured with a maximum depth of 5, a maximum width of 5, and a branching factor of 4. The designated target models for TAP included Llama-3-8B-Instruct [6] or Vicuna-13b-v1.5 [4]. In the case of *LLM-Fuzzer*, GPT-3.5 served as the auxiliary model for generating mutational inputs, and the query limit directed at the target LLMs was established at 200. For *DrAttack*, jailbreak prompts were formulated using GPT-4o. With respect to *MultiJail*, the entirety of the 315 available queries in the Bengali language was selected. For the *ActorAttack* strategy, a corpus of 600 queries was sourced from the SafeMTData dataset [48] (specifically, the `SafeMTData/Attack_600.json` file available on Hugging Face). For *X-Teaming*, we set the attacking model as Qwen2.5-32B-Instruct [8] and use the TextGrad-based text optimization to refine jailbreak prompts. Regarding *AlpacaEval*, all 805 questions within the AlpacaEval dataset were utilized. For *OR-Bench*, a subset of 1,000 prompts was randomly selected from the OR-Bench dataset [94] (specifically, the `or-bench-80k.csv` file on Hugging Face).

It is pertinent to note that: *The prompts associated with IJP, MultiJail, ActorAttack, AlpacaEval, and OR-Bench are static in nature. Consequently, all guardrail mechanisms encounter identical input stimuli, irrespective of whether they are safeguarding Llama-3-8B or Vicuna-13b. In contrast, GCG, AutoDAN, and DrAttack are specifically tailored to either Llama-3-8B or Vicuna-13b. As such, guardrails receive uniform inputs when defending the same designated target LLM. Conversely, TAP, LLM-Fuzzer, and X-Teaming represent adaptive attack methodologies. This implies that guardrail systems are presented with varied inputs, even when applied to the identical target LLM.*

Baselines. We compare our framework with popular jailbreak defense methods, including **Perplexity Filter** [55], **SmoothLLM** [57], **Llama Guard** [12], **GradSafe** [58], **GradientCuff** [61], **SelfDefend** [13], **WildGuard** [50], **Prompt Guard** [70], and **GuardReasoner** [83]. Specifically, *Perplexity Filter* leverages a Llama-2-7b model to calculate the perplexity of the input prompt. A jailbreak is considered to happen when the perplexity exceeds a threshold. We set this threshold at the maximum perplexity of any prompt in the JailbreakBench dataset of harmful behavior prompts. *SmoothLLM* perturbs the jailbreak prompts with character-level changes to enable the target LLM to perform defense. In this paper, we set SmoothLLM to conduct character swapping with a 10% perturbation percentage. *Llama Guard* is a fine-tuned Llama-2-7b model designed to detect the toxicity category of input prompts. *GradSafe* is a gradient-based detection method that identifies unsafe or jailbreak prompts in LLMs by analyzing the consistent gradient patterns of safety-critical parameters when paired with compliance responses. *GradientCuff* is a method for detecting jailbreak attacks on LLMs by analyzing the refusal loss landscape, leveraging gradient-based patterns to identify and block adversarial prompts while maintaining normal query performance. *SelfDefend* is a practical jailbreak defense framework for LLMs that uses a shadow LLM instance to concurrently detect harmful queries while the target LLM processes them, providing robust protection with minimal delay. *WildGuard* is an open, lightweight, multi-task moderation tool for LLMs that detects malicious user prompts, harmful model responses, and model refusal behavior. *Prompt Guard* is a security tool developed by Meta that detects and blocks malicious inputs (e.g., jailbreak attempts, prompt injections) in LLM applications, using lightweight classifier model Prompt-Guard-86M to filter harmful content in real time. *GuardReasoner* is a reasoning-based guard model designed to enhance the safety of LLMs by integrating explicit step-by-step reasoning into the moderation process. Our evaluations are implemented using PyTorch 2.6.0 and conducted on NVIDIA Hopper H800 GPUs.

5.2. Benchmark Evaluation

Defense Performance. We first analyze the defense performance of various guardrails. As delineated in Table 3, which presents the ASR, a lower value indicates superior defense capabilities. On average, GuardReasoner (Pre) demonstrates the most robust defense, achieving the lowest ASR of 0.135. Following closely is GuardReasoner (Post), underscoring the efficacy of the reasoning process prior to safety determination inherent in the GuardReasoner framework. Conversely, SmoothLLM exhibits the highest ASR of 0.303, rendering it the least effective in this cohort. This suboptimal performance may be attributed to its mechanism of token-level input perturbation, which appears to be primarily effective against jailbreak techniques characterized by adversarial suffixes, such as GCG, while offering limited protection against a broader spectrum of attacks.

Shifting focus to PGR, presented in Table 3, GuardReasoner (Post) achieves the best PGR of 0.141. Despite its superior precision in identifying malicious inputs, GuardReasoner (Post) does not attain state-of-the-art (SOTA) overall defense performance. A plausible explanation is its potential operational overlap with the target LLM's intrinsic safety mechanisms. That is, there might be a significant number of instances where GuardReasoner (Post) identifies a response as safe, and concurrently, the target LLM also recognizes the harmful nature of the query and refuses to respond, thereby diminishing the unique contribution of GuardReasoner (Post) to the ASR reduction when compared to a guardrail like GuardReasoner (Pre) which operates on a different paradigm.

Efficiency. The efficiency of guardrails is a critical factor for practical deployment, which we assess in terms of latency and GPU memory consumption. Figure 2(a) illustrates the extra delay introduced by different guardrail methodologies when processing normal inputs from the AlpacaEval and OR-Bench datasets. Perplexity Filter, Llama Guard, SelfDefend and PromptGuard stand out with the negligible latency. In contrast, GuardReasoner (Pre) and GradientCuff impose the most significant delays, with GuardReasoner (Pre) being particularly notable. This suggests that the profound reasoning capabilities that afford GuardReasoner (Pre) its enhanced defense performance come at the cost of increased processing time. The majority of other guardrails maintain an additional delay generally not exceeding 0.5 seconds.

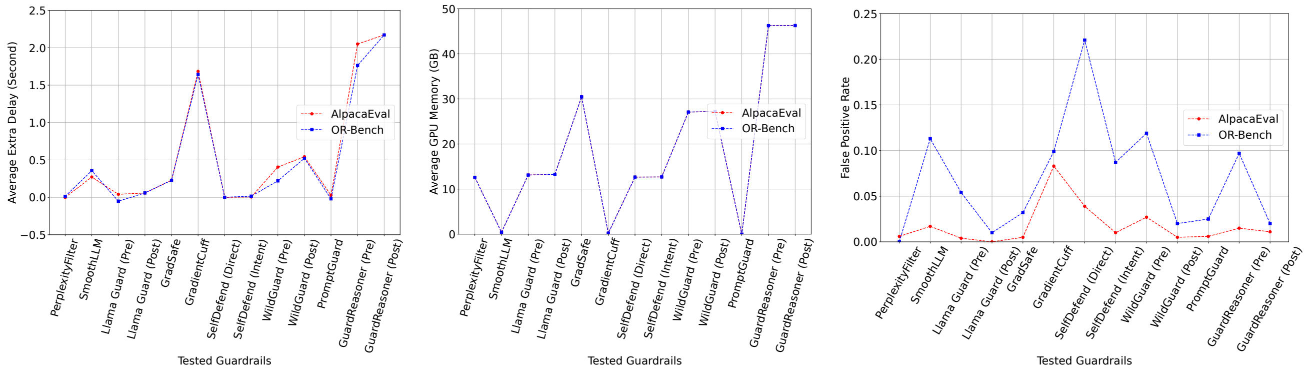
From the perspective of GPU memory utilization, depicted in Figure 2(b), GuardReasoner (Pre) again registers the highest memory footprint, consistent with its complex reasoning architecture. Conversely, SmoothLLM, GradientCuff, and PromptGuard are the most memory-efficient, with their consumption approaching negligible levels. This highlights a clear trade-off between the sophistication of the defense mechanism and its resource intensiveness.

Utility. Beyond security and efficiency, the utility of a guardrail, specifically its ability to not impede benign user interactions, is paramount. We measure the FPR on AlpacaEval and OR-Bench datasets, as shown in Figure 2(c). A higher FPR indicates a greater propensity to incorrectly flag legitimate prompts as malicious. SelfDefend (Direct) exhibits the highest FPR on OR-Bench, at 0.221. On AlpacaEval, GradientCuff records the highest FPR of 0.083. These figures suggest that these guardrails have a higher likelihood of intercepting normal user queries. Other guardrails with comparatively high FPRs include SmoothLLM, SelfDefend (Intent), WildGuard (Pre), and GuardReasoner (Pre). Although these three methods demonstrate strong defense performance (low ASR), their elevated FPRs underscore a critical trade-off between security and utility. Systems that are highly stringent in blocking threats may inadvertently penalize legitimate interactions, diminishing the overall user experience.

5.3. Leaderboard on SEU

To provide a holistic evaluation, we compare guardrails across five key metrics: ASR, PGR, Extra Delay, GPU Memory, and FPR. We average ASR and PGR over the nine jailbreak attacks (cf. Table 3) to derive Mean-ASR (M-ASR) and Mean-PGR (M-PGR). The other metrics are measured using the OR-Bench dataset. For a unified ranking, we normalize each metric to a [0, 1] range and invert the scores (1 - normalized value), ensuring higher values consistently indicate better performance. We then compute a Composite Score for each guardrail by averaging these five transformed scores. This score underpins the ranking visualized in the heatmap in Figure 3.

The analysis reveals inherent trade-offs, as no single guardrail excels across all dimensions. For instance, PromptGuard achieves the highest Composite Score but its low M-PGR suggests potential gaps in detection robustness. Conversely, GuardReasoner (Pre) ranks lower but provides superior defense (high M-ASR and M-PGR) at a significant cost to efficiency and utility. SelfDefend (Intent) offers a balanced profile, with its main weakness being a



(a) Delay (b) Memory (c) Utility

Figure 2. The delay, memory usage, and utility of guardrails.

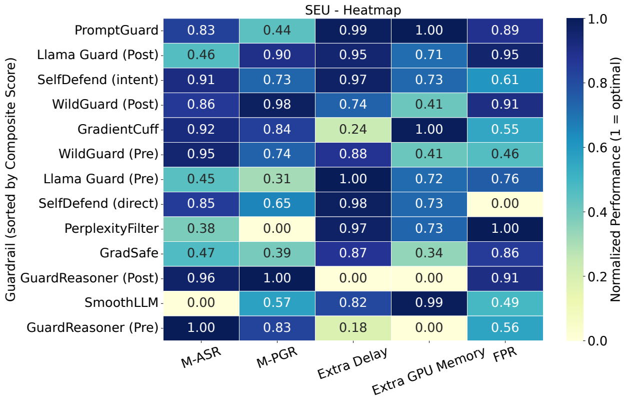


Figure 3. The heatmap of guardrails.

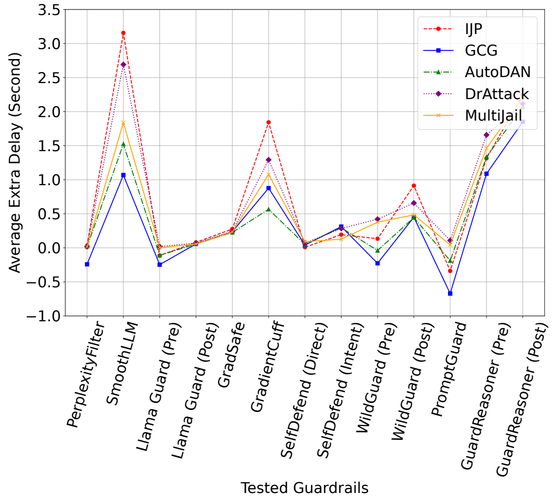


Figure 4. The delay of guardrails against different attack types.

higher FPR. This leaderboard underscores that the optimal guardrail choice is context-dependent, contingent on the specific security requirements and operational constraints of a given deployment scenario. We believe this leaderboard will serve as a valuable resource for practitioners in selecting appropriate guardrails based on their unique needs.

6. Practical Insights & Implications

Answer to RQ3: Session-level Guardrails v.s. Multi-turn Jailbreaks. A critical question arises regarding session-level guardrails: given their reliance on LLM dialogue history (both input and output) for threat assessment, how effectively do they counter sophisticated multi-turn jailbreak attacks? Our analysis, focusing on three session-level guardrails—Llama Guard (Post), WildGuard (Post), and GuardReasoner (Post)—reveals nuanced performance. As indicated in Table 3, these guardrails maintain an ASR above 10% against the ActorAttack multi-turn jailbreak. Furthermore, when faced with the more adaptive X-Teaming attack, the ASR for most guardrails, including these session-level ones, exceeds 90%. GradientCuff is a partial exception with a 64% ASR, but this is still a high failure rate. These findings underscore a significant vulnerability of current session-level guardrails against advanced multi-turn attacks. The high ASR, particularly against adaptive attacks like X-Teaming, suggests that these defenses can be readily bypassed if the attack unfolds over several interactions. This highlights an urgent imperative to develop more robust guardrail methodologies specifically designed to address the evolving landscape of multi-turn jailbreaks.

Answer to RQ1: Intervention Stages on Delay. The intervention stage of a guardrail—whether it operates pre-processing (on user input), intra-processing (during LLM generation), or post-processing (on LLM output)—can significantly impact system latency. We investigate this relationship by examining the data presented in Figure 4. Observations indicate that, with the notable exception of GuardReasoner (Pre), pre-processing guardrails such as Perplexity Filter, Llama Guard (Pre), SelfDefend (Direct), SelfDefend (Intent), WildGuard (Pre), and Prompt Guard generally introduce negligible, or in some cases even negative, additional latency. The higher latency of GuardReasoner (Pre) is attributable to its more complex reasoning processes. In contrast, intra-processing and post-processing guardrails exhibit more varied latency profiles relative to each other. A key finding is that for identical detection models, post-processing variants consistently incur greater delay than their pre-processing counterparts (e.g., WildGuard (Post)'s delay is greater than that of WildGuard (Pre)). This phenomenon arises because post-processing methods inherently must await the completion of the target LLM's generation phase before they can intervene. Conversely, pre-processing guardrails possess the advantage of potentially halting the LLM's generation process immediately upon detecting a malicious input, thereby conserving computational time. Consequently, pre-processing guardrails, particularly those not reliant on extensive reasoning, generally offer a more latency-efficient solution for integrating safety measures.

Answer to RQ2: Technical Paradigms on GPU Memory Usage. The underlying technical paradigm of a guardrail—be it rule-based, traditional model-based, or LLM-based—is expected to influence its GPU memory footprint. We examine this correlation using data from Figure 2. The results show that the rule-based SmoothLLM incurs zero additional memory overhead, representing the most memory-efficient approach. Certain traditional model-based methods, specifically GradientCuff and PromptGuard, also demonstrate near-zero memory consumption, highlighting their lightweight nature. However, the landscape for model-based approaches is not uniform; GradSafe, another model-based technique, exhibits higher memory usage than several LLM-based methods, indicating significant variability in resource demands even within this category. As anticipated, LLM-based guardrails generally impose a greater memory burden. This is an intrinsic consequence of their design, which necessitates loading and executing a large language model for safety inference. This observation aligns with the expectation that leveraging large language models for safety assessment incurs a higher resource cost in terms of memory. While rule-based and optimized model-based solutions offer substantial memory efficiency, the choice of paradigm must be carefully weighed against the desired detection capabilities and specific deployment constraints.

Answer to RQ4: Safety Granularity on Utility. The granularity at which a guardrail performs its safety checks—whether at the token-level, sequence-level (assessing the entire input or output), or session-level (considering the dialogue history)—may significantly affect its utility, particularly its propensity to misclassify benign prompts, as measured by the FPR. This aspect is explored using data from Figure 2(c). Token-level guardrails, exemplified by SmoothLLM (which analyzes keywords in LLM responses) and SelfDefend (Direct) (which inspects harmful segments within queries), demonstrate relatively pronounced FPRs. Notably, SelfDefend (Direct) records the highest FPR on the OR-Bench dataset, exceeding 20%. This suggests that token-level mechanisms, while focused, may inadvertently penalize legitimate interactions due to a potential lack of broader contextual understanding. A comparative analysis further reveals that for the same underlying detection model, session-level guardrails (typically denoted by a "(Post)" suffix, leveraging both LLM input and output) consistently achieve markedly lower FPRs than their sequence-level counterparts (often denoted by a "(Pre)" suffix, relying solely on input). For instance, WildGuard (Pre) exhibits an FPR above 10% on OR-Bench, whereas the FPR for WildGuard (Post) remains below 5%. While sequence-level guardrails display a wider range of FPRs—some high, some low—session-level approaches generally maintain low FPR values across the board. These observations collectively suggest that session-level guardrails tend to offer superior utility by minimizing false positives. This improved performance is likely attributable to their comprehensive use of contextual information derived from the entire interaction history, enabling a more nuanced distinction between genuinely harmful prompts and benign ones that might share superficial characteristics with attacks.

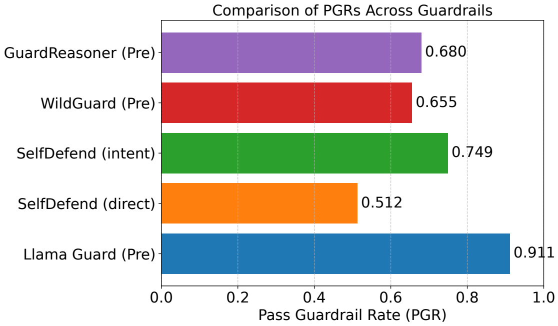


Figure 5. Cross-attack evaluation: injection attack on guardrails.

Generalization: Cross-Attack Assessment. While LLM-based guardrails have demonstrated efficacy against jailbreak attacks, a critical and often overlooked consideration is their robustness against other adversarial manipulations, specifically prompt injection attacks. Given that these guardrails are themselves powered by LLMs, their susceptibility to injection attacks—which could potentially subvert their safety assessment capabilities—presents a significant security concern. To investigate this, we evaluated the performance of LLM-based guardrails against 203 distinct injection attack samples sourced from the "deepset/prompt-injections" dataset on Hugging Face.

Our primary finding is that these injection attacks did not compromise the fundamental operational integrity of the guardrails. That is, the guardrails were not coerced into abandoning their safety analysis function to produce arbitrary, irrelevant outputs (e.g., "hello world"). They continued to process the inputs for security threats as designed. However, their effectiveness in identifying and mitigating these injections was limited. We measured the Pass Guardrail Rate (PGR) for these attacks, with results presented in Figure 5. The data reveals that while LLM-based guardrails exhibit a non-trivial capacity to filter prompt injections, this capability is modest at best. This assessment underscores a crucial gap in the current state of guardrail technology: the need for broader cross-attack generalization. For a guardrail to be truly effective in practice, its defensive perimeter must extend beyond jailbreak attempts to also detect and neutralize other forms of attacks that could exploit its defense, such as prompt injections. This calls for the development of more versatile and robust guardrail mechanisms capable of addressing a wider spectrum of adversarial inputs.

7. Conclusion

This SoK paper comprehensively addresses the fragmented landscape of LLM jailbreak guardrails by introducing a novel multi-dimensional taxonomy and a SEU measurement framework. Our findings highlight the strengths, limitations, and interdependencies of existing defense mechanisms with a series of key insights. This work forms a structured foundation to guide the principled advancement and deployment of more robust LLM guardrails.

References

[1] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, “Llama 2: Open foundation and fine-tuned chat models,” *arXiv preprint arXiv:2307.09288*, 2023.

[2] Z. Li, D. Wu, S. Wang, and Z. Su, “Api-guided dataset synthesis to finetune large code models,” *Proceedings of the ACM on Programming Languages*, vol. 9, no. OOPSLA1, pp. 786–815, 2025.

[3] Z. Li, C. Wang, P. Ma, C. Liu, S. Wang, D. Wu, and C. Gao, “On the feasibility of specialized ability stealing for large language code models,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2023.

[4] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing, “Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality,” March 2023. [Online]. Available: https://lmsys.org/blog/2023-03-30-vicuna/

[5] T. M. A. Team, “Mistral-7b-instruct-v0.2,” https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2, 2024.

[6] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.

[7] Anthropic, “Claude 3.5 sonnet,” https://www.anthropic.com/news/claude-3-5-sonnet, 2024.

[8] Qwen Team, “Qwen2.5 technical report,” *arXiv preprint arXiv:2412.15115*, 2024.

[9] A. Zou, Z. Wang, J. Z. Kolter, and M. Fredrikson, “Universal and transferable adversarial attacks on aligned language models,” *arXiv preprint arXiv:2307.15043*, 2023.

[10] X. Liu, N. Xu, M. Chen, and C. Xiao, “AutoDAN: Generating stealthy jailbreak prompts on aligned large language models,” in *ICLR*, 2024.

[11] T. Rebedea, R. Dinu, M. N. Sreedhar, C. Parisien, and J. Cohen, “NeMo Guardrails: A toolkit for controllable and safe LLM applications with programmable rails,” in *EMNLP: System Demonstrations*, 2023, pp. 431–445.

[12] H. Inan, K. Upasani, J. Chi, R. Rungta, K. Iyer, Y. Mao, M. Tontchev, Q. Hu, B. Fuller, D. Testuggine *et al.*, “Llama Guard: LLM-based input-output safeguard for Human-AI conversations,” *arXiv preprint arXiv:2312.06674*, 2023.

[13] X. Wang, D. Wu, Z. Ji, Z. Li, P. Ma, S. Wang, Y. Li, Y. Liu, N. Liu, and J. Rahmel, “SelfDefend: LLMs can defend themselves against jailbreaking in a practical manner,” in *USENIX Security*, 2025.

[14] Y. Xie, J. Yi, J. Shao, J. Curl, L. Lyu, Q. Chen, X. Xie, and F. Wu, “Defending chatgpt against jailbreak attack via self-reminders,” *Nature Machine Intelligence*, vol. 5, no. 12, pp. 1486–1496, 2023.

[15] S. Xhonneux, A. Sordoni, S. Günnemann, G. Gidel, and L. Schwinn, “Efficient adversarial training in LLMs with continuous attacks,” in *NeurIPS*, 2024.

[16] X. Wang, W. Wang, Z. Ji, Z. Li, P. Ma, D. Wu, and S. Wang, “Stshield: Single-token sentinel for real-time jailbreak detection in large language models,” *arXiv preprint arXiv:2503.17932*, 2025.

[17] Y. Liu, G. Deng, Z. Xu, Y. Li, Y. Zheng, Y. Zhang, L. Zhao, T. Zhang, and Y. Liu, “Jailbreaking ChatGPT via prompt engineering: An empirical study,” *arXiv preprint arXiv:2305.13860*, 2023.

[18] A. Wei, N. Haghtalab, and J. Steinhardt, “Jailbroken: How does LLM safety training fail?” in *NeurIPS*, vol. 36, 2023.

[19] Z. Wei, Y. Wang, and Y. Wang, “Jailbreak and guard: Aligned language models with only few in-context demonstrations,” *arXiv preprint arXiv:2310.06387*, 2023.

[20] X. Shen, Z. Chen, M. Backes, Y. Shen, and Y. Zhang, “”Do Anything Now”: Characterizing and evaluating in-the-wild jailbreak prompts on large language models,” in *CCS*, 2024.

[21] G. Deng, Y. Liu, Y. Li, K. Wang, Y. Zhang, Z. Li, H. Wang, T. Zhang, and Y. Liu, “MASTERKEY: Automated jailbreaking of large language model chatbots,” in *NDSS*, 2024.

[22] C. Sitawarin, N. Mu, D. Wagner, and A. Araujo, “PAL: Proxy-guided black-box attack on large language models,” *arXiv preprint arXiv:2402.09674*, 2024.

[23] M. Andriushchenko, F. Croce, and N. Flammarion, “Jailbreaking leading safety-aligned LLMs with simple adaptive attacks,” in *ICLR*, 2025.

[24] X. Jia, T. Pang, C. Du, Y. Huang, J. Gu, Y. Liu, X. Cao, and M. Lin, “Improved techniques for optimization-based jailbreaking on large language models,” in *ICLR*, 2025.

[25] X. Chen, Y. Nie, W. Guo, and X. Zhang, “When LLM meets DRL: Advancing jailbreaking efficiency via DRL-guided search,” in *NeurIPS*, 2024.

[26] E. Perez, S. Huang, H. F. Song, T. Cai, R. Ring, J. Aslanides, A. Glaese, N. McAleese, and G. Irving, “Red teaming language models with language models,” in *EMNLP*, 2022.

[27] P. Chao, A. Robey, E. Dobriban, H. Hassani, G. J. Pappas, and E. Wong, “Jailbreaking black box large language models in twenty queries,” *arXiv preprint arXiv:2310.08419*, 2023.

[28] A. Mehrotra, M. Zampetakis, P. Kassianik, B. Nelson, H. Anderson, Y. Singer, and A. Karbasi, “Tree of attacks: Jailbreaking black-box LLMs automatically,” in *NeurIPS*, 2024.

[29] A. Paulus, A. Zharmagambetov, C. Guo, B. Amos, and Y. Tian, “AdvPrompter: Fast adaptive adversarial prompting for LLMs,” *arXiv preprint arXiv:2404.16873*, 2024.

[30] Z. Liao and H. Sun, “AmpleGCG: Learning a universal and transferable generative model of adversarial suffixes for jailbreaking both open and closed LLMs,” in *COLM*, 2024. [Online]. Available: https://openreview.net/forum?id=UfqzXg95I5

[31] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, “Tree of thoughts: Deliberate problem solving with large language models,” in *NeurIPS*, vol. 36, 2024.

[32] J. Yu, X. Lin, Z. Yu, and X. Xing, “LLM-Fuzzer: Scaling assessment of large language model jailbreaks,” in *USENIX Security*, 2024, pp. 4657–4674.

[33] D. Handa, A. Chirmule, B. Gajera, and C. Baral, “Jailbreaking proprietary large language models using word substitution cipher,” *arXiv preprint arXiv:2402.10601*, 2024.

[34] X. Li, R. Wang, M. Cheng, T. Zhou, and C.-J. Hsieh, "DrAttack: Prompt decomposition and reconstruction makes powerful LLMs jailbreakers," in *EMNLP*, 2024, pp. 13 891–13 913.

[35] Z. Chang, M. Li, Y. Liu, J. Wang, Q. Wang, and Y. Liu, "Play guessing game with LLM: Indirect jailbreak attack with implicit clues," in *ACL*, 2024, pp. 5135–5147.

[36] Y. Deng, W. Zhang, S. J. Pan, and L. Bing, "Multilingual jailbreak challenges in large language models," *ICLR*, 2024.

[37] Z.-X. Yong, C. Menghini, and S. H. Bach, "Low-resource languages jailbreak GPT-4," *arXiv preprint arXiv:2310.02446*, 2023.

[38] L. Shen, W. Tan, S. Chen, Y. Chen, J. Zhang, H. Xu, B. Zheng, P. Koehn, and D. Khashabi, "The language barrier: Dissecting safety challenges of LLMs in multilingual contexts," in *ACL*, 2024, pp. 2668–2680.

[39] J. Li, Y. Liu, C. Liu, L. Shi, X. Ren, Y. Zheng, Y. Liu, and Y. Xue, "A cross-language investigation into jailbreak attacks in large language models," *arXiv preprint arXiv:2401.16765*, 2024.

[40] Y. Yuan, W. Jiao, W. Wang, J. tse Huang, P. He, S. Shi, and Z. Tu, "GPT-4 is too smart to be safe: Stealthy chat with LLMs via cipher," in *ICLR*, 2024. [Online]. Available: https://openreview.net/forum?id=MbfAK4s61A

[41] J. Chu, Y. Liu, Z. Yang, X. Shen, M. Backes, and Y. Zhang, "Comprehensive assessment of jailbreak attacks against LLMs," *arXiv preprint arXiv:2402.05668*, 2024.

[42] Y. Meng, M. Xia, and D. Chen, "SimPO: Simple preference optimization with a reference-free reward," in *NeurIPS*, vol. 37, 2024, pp. 124 198–124 235.

[43] Z. Zhou, J. Xiang, H. Chen, Q. Liu, Z. Li, and S. Su, "Speak out of turn: Safety vulnerability of large language models in multi-turn dialogue," *arXiv preprint arXiv:2402.17262*, 2024.

[44] X. Liu, L. Li, T. Xiang, F. Ye, L. Wei, W. Li, and N. Garcia, "Imposter. AI: Adversarial attacks with hidden intentions towards aligned large language models," *arXiv preprint arXiv:2407.15399*, 2024.

[45] N. Li, Z. Han, I. Steneker, W. Primack, R. Goodside, H. Zhang, Z. Wang, C. Menghini, and S. Yue, "LLM defenses are not robust to multi-turn human jailbreaks yet," *arXiv preprint arXiv:2408.15221*, 2024.

[46] X. Yang, X. Tang, S. Hu, and J. Han, "Chain of attack: A semantic-driven contextual multi-turn attacker for LLM," *arXiv preprint arXiv:2405.05610*, 2024.

[47] M. Russinovich, A. Salem, and R. Eldan, "Great, now write an article about that: The crescendo multi-turn LLM jailbreak attack," in *USENIX Security*, 2025.

[48] Q. Ren, H. Li, D. Liu, Z. Xie, X. Lu, Y. Qiao, L. Sha, J. Yan, L. Ma, and J. Shao, "Derail yourself: Multi-turn LLM jailbreak attack through self-discovered clues," *arXiv preprint arXiv:2410.10700*, 2024.

[49] S. Rahman, L. Jiang, J. Shiffer, G. Liu, S. Issaka, M. R. Parvez, H. Palangi, K.-W. Chang, Y. Choi, and S. Gabriel, "X-Teaming: Multi-turn jailbreaks and defenses with adaptive multi-agents," *arXiv preprint arXiv:2504.13203*, 2025.

[50] S. Han, K. Rao, A. Ettinger, L. Jiang, B. Y. Lin, N. Lambert, Y. Choi, and N. Dziri, "WildGuard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of LLMs," in *NeurIPS Datasets and Benchmarks Track*, 2024.

[51] A. Lees, V. Q. Tran, Y. Tay, J. Sorensen, J. Gupta, D. Metzler, and L. Vasserman, "A new generation of perspective API: Efficient multilingual character-level transformers," in *KDD*, 2022, p. 3197–3207.

[52] T. Markov, C. Zhang, S. Agarwal, F. E. Nekoul, T. Lee, S. Adler, A. Jiang, and L. Weng, "A holistic approach to undesired content detection in the real world," in *AAAI*, vol. 37, no. 12, 2023, pp. 15 009–15 018.

[53] M. Phute, A. Helbling, M. Hull, S. Peng, S. Szyller, C. Cornelius, and D. H. Chau, "LLM Self Defense: By self examination, LLMs know they are being tricked," *arXiv preprint arXiv:2308.07308*, 2023.

[54] G. Alon and M. Kamfonas, "Detecting language model attacks with perplexity," *arXiv preprint arXiv:2308.14132*, 2023.

[55] N. Jain, A. Schwarzschild, Y. Wen, G. Somepalli, J. Kirchenbauer, P.-y. Chiang, M. Goldblum, A. Saha, J. Geiping, and T. Goldstein, "Baseline defenses for adversarial attacks against aligned language models," *arXiv preprint arXiv:2309.00614*, 2023.

[56] A. Kumar, C. Agarwal, S. Srinivas, A. J. Li, S. Feizi, and H. Lakkaraju, "Certifying LLM safety against adversarial prompting," in *COLM*, 2024.

[57] A. Robey, E. Wong, H. Hassani, and G. J. Pappas, "SmoothLLM: Defending large language models against jailbreaking attacks," *arXiv preprint arXiv:2310.03684*, 2023.

[58] Y. Xie, M. Fang, R. Pi, and N. Gong, "GradSafe: Detecting jailbreak prompts for LLMs via safety-critical gradient analysis," in *ACL*, 2024, pp. 507–518.

[59] J. Ji, B. Hou, A. Robey, G. J. Pappas, H. Hassani, Y. Zhang, E. Wong, and S. Chang, "Defending large language models against jailbreak attacks via semantic smoothing," *arXiv preprint arXiv:2402.16192*, 2024.

[60] S. Goyal, M. Hira, S. Mishra, S. Goyal, A. Goel, N. Dadu, K. DB, S. Mehta, and N. Madaan, "LLMGuard: Guarding against unsafe LLM behavior," in *AAAI*, 2024, pp. 23 790–23 792.

[61] X. Hu, P.-Y. Chen, and T.-Y. Ho, "Gradient Cuff: Detecting jailbreak attacks on large language models by exploring refusal loss landscapes," in *NeurIPS*, 2024.

[62] Y. Zeng, Y. Wu, X. Zhang, H. Wang, and Q. Wu, "AutoDefense: Multi-agent LLM defense against jailbreak attacks," *arXiv preprint arXiv:2403.04783*, 2024.

[63] Z. Yuan, Z. Xiong, Y. Zeng, N. Yu, R. Jia, D. Song, and B. Li, "RigorLLM: Resilient guardrails for large language models against undesired content," in *ICML*, 2024.

[64] S. Ghosh, P. Varshney, E. Galinkin, and C. Parisien, "Aegis: Online adaptive AI content safety moderation with ensemble of LLM experts," *arXiv preprint arXiv:2404.05993*, 2024.

[65] Z. Chu, Y. Wang, L. Li, Z. Wang, Z. Qin, and K. Ren, "A causal explainable guardrails for large language models," in *ACM CCS*, 2024.

[66] A. Kawasaki, A. Davis, and H. Abbas, "Defending large language models against attacks with residual stream activation analysis," *arXiv preprint arXiv:2406.03230*, 2024.

[67] A. Zou, L. Phan, J. Wang, D. Duenas, M. Lin, M. Andriushchenko, R. Wang, Z. Kolter, M. Fredrikson, and D. Hendrycks, "Improving alignment and robustness with circuit breakers," in *NeurIPS*, vol. 37, 2024, pp. 83 345–83 373.

[68] Z. Xiang, L. Zheng, Y. Li, J. Hong, Q. Li, H. Xie, J. Zhang, Z. Xiong, C. Xie, C. Yang *et al.*, "GuardAgent: Safeguard LLM agents by a guard agent via knowledge-enabled reasoning," *arXiv preprint arXiv:2406.09187*, 2024.

[69] M. Kang and B. Li, "R^2-Guard: Robust reasoning enabled LLM guardrail via knowledge-enhanced logical reasoning," in *ICLR*, 2025.

[70] M. Llama, "Prompt-guard-86m," https://huggingface.co/meta-llama/Prompt-Guard-86M, 2024.

[71] D. Schwartz, D. Bespalov, Z. Wang, N. Kulkarni, and Y. Qi, "Graph of attacks with pruning: Optimizing stealthy jailbreak prompt generation for enhanced llm content moderation," *arXiv preprint arXiv:2501.18638*, 2025.

[72] B. Manczak, E. Zemour, E. Lin, and V. Mugunthan, "PrimeGuard: Safe and helpful LLMs through tuning-free routing," *arXiv preprint arXiv:2407.16318*, 2024.

[73] W. Zeng, Y. Liu, R. Mullins, L. Peran, J. Fernandez, H. Harkous, K. Narasimhan, D. Proud, P. Kumar, B. Radharapu *et al.*, "Shield-Gemma: Generative AI content moderation based on Gemma," *arXiv preprint arXiv:2407.21772*, 2024.

[74] J. Hu, Y. Dong, and X. Huang, "Trust-oriented adaptive guardrails for large language models," *arXiv preprint arXiv:2408.08959*, 2024.

[75] C. Zhao, Z. Dou, and K. Huang, "EEG-Defender: Defending against jailbreak through early exit generation of large language models," *arXiv preprint arXiv:2408.11308*, 2024.

[76] C. Qian, H. Zhang, L. Sha, and Z. Zheng, "HSF: Defending against jailbreak attacks with hidden state filtering," *arXiv preprint arXiv:2409.03788*, 2024.

[77] G. Cornacchia, G. Zizzo, K. Fraser, M. Z. Hameed, A. Rawat, and M. Purcell, "MoJE: Mixture of jailbreak experts, naive tabular classifiers as guard for prompt attacks," *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, vol. 7, no. 1, pp. 304–315, Oct. 2024.

[78] A. Peng, J. Michael, H. Sleight, E. Perez, and M. Sharma, "Rapid response: Mitigating LLM jailbreaks with a few examples," *arXiv preprint arXiv:2411.07494*, 2024.

[79] E. Galinkin and M. Sablotny, "Improved large language model jailbreak detection via pretrained embeddings," *arXiv preprint arXiv:2412.01547*, 2024.

[80] X. Hu, P.-Y. Chen, and T.-Y. Ho, "Token Highlighter: Inspecting and mitigating jailbreak prompts for large language models," *arXiv preprint arXiv:2412.18171*, 2024.

[81] S. Ghosh, P. Varshney, M. N. Sreedhar, A. Padmakumar, T. Rebedea, J. R. Varghese, and C. Parisien, "Aegis2.0: A diverse ai safety dataset and risks taxonomy for alignment of LLM guardrails," *arXiv preprint arXiv:2501.09004*, 2025.

[82] M. K. Rad, H. Nghiem, A. Luo, S. Wadhwa, M. Sorower, and S. Rawls, "Refining input guardrails: Enhancing llm-as-a-judge efficiency through chain-of-thought fine-tuning and alignment," *arXiv preprint arXiv:2501.13080*, 2025.

[83] Y. Liu, H. Gao, S. Zhai, J. Xia, T. Wu, Z. Xue, Y. Chen, K. Kawaguchi, J. Zhang, and B. Hooi, "GuardReasoner: Towards reasoning-based LLM safeguards," *arXiv preprint arXiv:2501.18492*, 2025.

[84] M. Sharma, M. Tong, J. Mu, J. Wei, J. Kruthoff, S. Goodfriend, E. Ong, A. Peng, R. Agarwal, C. Anil *et al.*, "Constitutional classifiers: Defending against universal jailbreaks across thousands of hours of red teaming," *arXiv preprint arXiv:2501.18837*, 2025.

[85] S. Zhang, Y. Zhai, K. Guo, H. Hu, S. Guo, Z. Fang, L. Zhao, C. Shen, C. Wang, and Q. Wang, "JBShield: Defending large language models from jailbreak attacks through activated concept analysis and manipulation," in *USENIX Security*, 2025.

[86] S. Xiang, A. Zhang, Y. Cao, Y. Fan, and R. Chen, "Beyond surface-level patterns: An essence-driven defense framework against jailbreak attacks in LLMs," *arXiv preprint arXiv:2502.19041*, 2025.

[87] C. Yung, H. Huang, S. M. Erfani, and C. Leckie, "Curvalid: Geometrically-guided adversarial prompt detection," *arXiv preprint arXiv:2503.03502*, 2025.

[88] R. Pu, C. Li, R. Ha, L. Zhang, L. Qiu, and X. Zhang, "MirrorShield: Towards universal defense against jailbreaks via entropy-guided mirror crafting," *arXiv preprint arXiv:2503.12931*, 2025.

[89] X. Zhang, C. Zhang, T. Li, Y. Huang, X. Jia, M. Hu, J. Zhang, Y. Liu, S. Ma, and C. Shen, "JailGuard: A universal detection framework for prompt-based attacks on LLM systems," *ACM Trans. Softw. Eng. Methodol.*, 2025.

[90] B. Upadhayay, V. Behzadan *et al.*, "X-Guard: Multilingual guard agent for content moderation," *arXiv preprint arXiv:2504.08848*, 2025.

[91] J. Piet, X. Huang, D. Jacob, A. Chow, M. Alrashed, G. Zhao, Z. Hu, C. Sitawarin, B. Alomair, and D. Wagner, "Jailbreaksovertime: Detecting jailbreak attacks under distribution shift," *arXiv preprint arXiv:2504.19440*, 2025.

[92] P. Chao, E. Debenedetti, A. Robey, M. Andriushchenko, F. Croce, V. Sehwag, E. Dobriban, N. Flammarion, G. J. Pappas, F. Tramer *et al.*, "JailbreakBench: An open robustness benchmark for jailbreaking large language models," in *NeurIPS*, 2024.

[93] X. Li, T. Zhang, Y. Dubois, R. Taori, I. Gulrajani, C. Guestrin, P. Liang, and T. B. Hashimoto, "AlpacaEval: An automatic evaluator of instruction-following models," https://github.com/tatsu-lab/alpaca_eval, 2023.

[94] J. Cui, W.-L. Chiang, I. Stoica, and C.-J. Hsieh, "OR-Bench: An over-refusal benchmark for large language models," in *ICML*, 2025.

[95] X. Qi, Y. Zeng, T. Xie, P.-Y. Chen, R. Jia, P. Mittal, and P. Henderson, "Fine-tuning aligned language models compromises safety, even when users do not intend to!" *arXiv preprint arXiv:2310.03693*, 2023.

[96] Z. Li, C. Wang, P. Ma, D. Wu, S. Wang, C. Gao, and Y. Liu, "Split and merge: Aligning position biases in LLM-based evaluators," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024.

[97] "Forbidden question set with prompts," https://github.com/verazuo/jailbreak_llms/blob/main/data/forbidden_question/forbidden_question_set_with_prompts.csv.zip, 2023.

Appendix

Evaluation Results on Vicuna-13b-v1.5

We extend our comprehensive evaluation to another widely-used open-source model, Vicuna-13b-v1.5, to assess the generalization of different guardrails. The detailed results are presented in Table 4.

First, a salient observation is that Vicuna-13b-v1.5 is considerably more susceptible to jailbreak attacks compared to Llama-3. This increased vulnerability is evident from the substantially higher Attack Success Rates (ASR) across almost all attack categories, indicating a weaker inherent safety alignment in Vicuna.

Second, we note a significant performance degradation for certain defenses when applied to Vicuna-13b-v1.5. For example, the efficacy of GradSafe and GradientCuff diminishes. GradientCuff, which showed marked effectiveness against the X-Teaming multi-turn attack on Llama-3, fails to maintain this advantage on Vicuna-13b-v1.5. This decline can be attributed to their nature as intra-processing guardrails, which heavily rely on the internal representations and alignment of the target LLM. Consequently, a less well-aligned model like Vicuna-13b-v1.5 compromises their defensive mechanism.

Despite these differences, we also observe consistent performance patterns. GuardReasoner (Pre) and GuardReasoner (Post) continue to exhibit state-of-the-art defense capabilities. GuardReasoner (Pre) achieves the best overall ASR of 0.156, while GuardReasoner (Post) records the best overall PGR of 0.192. This sustained excellence underscores that the robust defense mechanism of GuardReasoner is largely independent of the target LLM, positioning it as a more universally applicable and reliable guardrail.

TABLE 4. THE ASR (↓) / PGR (↓) RESULTS FOR THE TARGET LLM (VICUNA-13B-V1.5) WITH DIFFERENT GUARDRAILS AGAINST FIVE MAJOR CATEGORIES OF JAILBREAK ATTACKS, INCLUDING ROW AVERAGES. (PRE) AND (POST) DENOTE THE PRE-PROCESSING AND POST-PROCESSING VERSIONS OF THE GUARDRAILS, RESPECTIVELY. (DIRECT) AND (INTENT) DENOTE THE DIRECT PROMPT AND INTENT PROMPT BASED VERSIONS OF SELFDEFEND [13], RESPECTIVELY.

Guardrails	Manual	Optimization-based		Generation-based		Implicit		Multi-turn		Average
	IJP	GCG	AutoDAN	TAP	LLM-Fuzzer	DrAttack	MultiJail	ActorAttack	X-Teaming	
Vicuna-13b-v1.5	0.474/-	0.890/-	0.660/-	0.530/-	0.820/-	0.780/-	0.254/-	0.238/-	0.960/-	0.649/-
PerplexityFilter	0.474/1.000	0.030/0.040	0.660/1.000	0.830/1.000	0.870/1.000	0.780/1.000	0.254/1.000	0.238/1.000	0.990/1.000	0.570/0.893
SmoothLLM	0.402/0.794	0.140/0.270	0.520/0.970	0.840/0.860	0.510/1.000	0.410/0.970	0.152/0.933	0.877/0.877	0.980/0.970	0.537/0.849
Llama Guard (Pre)	0.194/0.563	0.370/0.390	0.460/0.750	0.630/0.750	0.810/1.000	0.650/0.850	0.251/0.952	0.222/0.967	0.970/1.000	0.506/0.802
Llama Guard (Post)	0.250/0.250	0.400/0.400	0.610/0.610	0.600/0.600	0.830/0.830	0.390/0.390	0.248/0.248	0.230/0.230	0.970/0.970	0.503/0.503
GradSafe	0.471/0.994	0.890/1.000	0.660/1.000	0.580/0.960	0.900/1.000	0.780/1.000	0.254/1.000	0.238/1.000	0.980/1.000	0.639/0.995
GradientCuff	0.193/0.351	0.090/0.090	0.310/0.480	0.550/0.630	0.780/1.000	0.660/0.830	**0.000/0.000**	0.183/0.805	0.930/0.960	0.411/0.572
SelfDefend (Direct)	0.050/0.262	0.080/0.080	0.020/0.080	0.210/0.270	0.190/0.270	0.330/0.480	0.187/0.743	0.132/0.720	0.960/0.990	0.240/0.433
SelfDefend (Intent)	0.057/0.286	0.080/0.080	0.050/0.110	0.140/0.200	0.210/0.250	**0.010**/0.090	0.127/0.568	0.157/0.763	0.960/1.000	0.199/0.372
WildGuard (Pre)	0.007/0.033	0.010/0.010	**0.010**/0.020	**0.040**/0.090	**0.010/0.020**	0.330/0.490	0.187/0.797	0.147/0.757	0.920/0.950	0.185/0.352
WildGuard (Post)	0.066/0.066	0.040/0.040	0.030/0.030	0.100/0.100	0.410/0.410	0.090/0.090	0.194/0.194	0.165/0.165	0.930/0.930	0.225/0.225
Prompt Guard	**0.000/0.000**	0.020/0.020	0.240/0.370	0.570/0.960	0.020/0.030	0.770/0.990	0.254/1.000	0.235/0.995	0.980/1.000	0.343/0.596
GuardReasoner (Pre)	**0.000**/0.009	**0.000/0.000**	0.020/0.020	0.050/0.080	0.040/0.040	0.150/0.270	0.057/0.349	0.143/0.740	0.940/0.960	**0.156**/0.274
GuardReasoner (Post)	0.050/0.050	0.030/0.030	**0.010/0.010**	0.060/**0.060**	0.480/0.480	0.040/**0.040**	0.060/0.060	**0.100/0.100**	**0.900/0.900**	0.192/**0.192**