JEDI: Real-Time Jailbreak Defense for LLMs via In-Generation Detection and Intervention

Ruilin Xie[1] **Bixin Li**[1*] **Yongqiang Tian**[2] **Xinyu Chen**[1] **Lulu Wang**[1]
[1]School of Computer Science and Engineering
Southeast University, Nanjing, China
[2]Department of Software Systems & Cybersecurity
Monash University
{iamxrl,bx.li,cxy0427,wanglulu}@seu.edu.cn
yongqiang.tian@monash.edu

Abstract

Large Language Models remain vulnerable to jailbreak attacks, and streaming generation can expose harmful prefixes before a completed response is moderated. We present JEDI, an in-generation defense that monitors hidden-state risk and selectively steers ongoing generation. A Condition Vector measures harmful-compliance drift, a Cumulative Sum (CUSUM) statistic accumulates persistent evidence, and a separate, orthogonalized Intervention Vector applies risk-adaptive refusal steering after an alarm. Across six publicly available models and eleven attack methods, JEDI achieves 95.7% mean Defense Success Rate, with model-level rates of 93.6%–99.4%, and adds 0.003 seconds to average Time To First Token. Comparisons with periodic buffered guards, conditional activation steering, and trajectory monitoring characterize the safety–utility–latency trade-off. A matched visible-prefix audit reports both raw detector-positive onsets and evaluator-consensus leakage, showing substantial reductions while identifying residual exposure. Controlled component ablations support distinct roles for accumulated risk, orthogonalization, and dynamic intervention. JEDI targets self-hosted or model-developer deployments with hidden-state access and decoding control. Code and experimental results are available at https://github.com/samxrl/JEDI.

1 Introduction

Large Language Models (LLMs) have achieved widespread deployment in high-stakes domains ranging from financial analysis to automated software engineering, driven by their emergent reasoning capabilities [30, 54]. Despite extensive safety alignment efforts such as Reinforcement Learning from Human Feedback (RLHF), these models remain vulnerable to jailbreak attacks [30, 56].

Adversaries can exploit sophisticated optimization-based or semantically manipulative prompts to bypass safety guardrails, inducing the model to generate harmful content that violates ethical and legal policies [40, 63]. While research on jailbreak defense has made notable progress, predominant approaches concentrate on either perturbing inputs prior to response generation or detecting content after the response is complete [38, 51].

However, the industry-wide shift toward streaming generation exposes critical limitations in these conventional defense paradigms. In streaming interactions, the model outputs tokens sequentially, creating a phenomenon of *prefix exposure* where harmful tokens become visible to the user well before the full response is completed [30]. This reality renders post-hoc output filtering (i.e., moderating the fully generated response after decoding) fundamentally reactive and often too late to prevent the

*Corresponding author: Bixin Li.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

propagation of harm. Furthermore, robust input defenses or external guardrail models often impose substantial computational overhead; this added latency significantly delays the Time To First Token (TTFT) and degrades the user experience, creating a safety–responsiveness trade-off for real-time applications [35, 15]. The remaining challenge is to combine early in-generation risk detection, selective intervention under noisy token-level signals, and low online overhead.

We study three research questions grounded in these deployment requirements: **RQ1 (Earliness):** Can pre-decoding hidden states provide an early signal of harmful semantic drift for in-generation intervention [23, 20]? **RQ2 (Selectivity):** Can evidence accumulated across decoding steps support selective intervention despite noisy token-level risk signals [7, 44]? **RQ3 (Efficiency):** Can detection and activation-level intervention be integrated into decoding with low online overhead while preserving benign utility and streaming responsiveness [59, 52]?

To address these questions for streaming generation, we propose **JEDI** (**J**ailbreak d**E**fense via **D**etection and **I**ntervention), a dynamic method that secures streaming outputs via real-time internal-state monitoring and selective intervention. **Earliness** is achieved by scoring risk in the pre-decoding hidden states: at each step, JEDI monitors the projection of hidden representations onto an extracted *Condition Vector*, providing a token-level signal for intervention during generation. **Selectivity** is enforced by sequential drift detection rather than instantaneous token screening: JEDI applies a Cumulative Sum (CUSUM) [28] accumulator over the projected risk and triggers defense only when accumulated evidence exceeds a threshold calibrated to control benign alarms. **Efficiency** follows from lightweight online computation: the monitoring loop requires only a low-dimensional projection and a few scalar updates per token, while the more expensive representation learning is amortized in an offline phase.

Specifically, JEDI operates in two phases to balance safety and utility. In the *offline vector extraction* phase, we construct contrastive datasets to extract two directions in the model's representation space: a *Condition Vector* that captures semantic drift toward harmful content, and an *Intervention Vector* that encodes refusal behavior for harmful instructions. In the *online defense* phase, JEDI continuously updates the CUSUM statistic on the condition-vector projection; once an alarm is raised, it injects the *Intervention Vector* into the activation space with adaptive strength, steering the continuing generation toward refusal-associated states to reduce harmful exposure.

Evaluation on six LLMs and eleven attacks yields 93.6%–99.4% model-level DSR. Comparisons across full-response safety, visible-prefix exposure, utility, over-refusal, and latency place JEDI at a non-dominated high-safety point among the evaluated configurations. Buffered guards attain higher safety at greater latency and utility cost. Component ablations and prefix-bank resampling further test the roles and stability of JEDI's control loop.

Our main contributions can be summarized as follows:

- Formalize the design principles for defense in streaming contexts, emphasizing the necessity of early risk detection and selective intervention to preserve user experience.
- Propose the JEDI method, which uses representation engineering with CUSUM-based sequential drift detection to achieve a balance between safety and utility while maintaining high efficiency.
- Evaluate full-response safety and visible-prefix exposure alongside utility and latency, and analyze the contributions of sequential risk accumulation, orthogonalization, and dynamic steering.

2 Background & Related Work

2.1 Streaming Interaction

Modern LLMs predominantly use streaming generation to mitigate latency perception [43], which exposes the output sequence token by token in real time. While this significantly optimizes TTFT, it introduces a critical vulnerability: ***prefix exposure***.

In streaming interaction, traditional filters are structurally inadequate because harmful tokens are generated immediately. Post-hoc audits occur only after prefix exposure, failing to prevent immediate psychological impact or data capture. Consequently, defense objectives must shift to in-generation defense. This requires monitoring the generation's semantic trajectory to intervene precisely when a significant drift towards harmfulness occurs, thereby preventing the completion of the harmful prefix.

2.2 Jailbreak Attacks

LLMs are typically subjected to rigorous safety alignment processes, such as Supervised Fine-Tuning (SFT) and RLHF, to prevent the generation of harmful, illegal, or unethical content [56]. A jailbreak attack is defined as an adversarial exploitation wherein an adversary constructs a specific input, known as a jailbreak prompt, to bypass these alignment constraints and induce the model to output restricted information [49, 39].

Formally, let x_{harmful} denote a malicious query that a safety-aligned model $\mathcal{M}$ is trained to refuse. The objective of a jailbreak attack is to identify a transformation function $\mathcal{F}$ such that the transformed input $x' = \mathcal{F}(x_{\text{harmful}})$ elicits a response $y = \mathcal{M}(x')$ where $y \notin \mathcal{Y}_{\text{refusal}}$. Here, $\mathcal{Y}_{\text{refusal}}$ represents the set of compliant refusal responses (e.g., "I cannot assist with that"). The transformation $\mathcal{F}$ aims to maximize the probability that the model produces a non-refusal response while preserving the original semantic intent of the harmful query [54]. Additional background on jailbreak attacks and decoding-time trajectory defenses is provided in Appendix A.

2.3 Jailbreak Defense

Evolving LLM jailbreak attacks have prompted diverse defenses against harmful content, categorized by intervention timing into three stages: pre-generation, in-generation, and post-generation defense [15, 56, 45, 16, 29].

Pre-generation Defense serves as the first line of defense by detecting or perturbing inputs before LLM processing [45, 25]. Approaches include external guardrails like Llama Guard [15] and input perturbation methods. SmoothLLM [35] aggregates outputs from perturbed prompts to neutralize adversarial tokens, while BackTranslation [47] rewrites inputs to bypass triggers. Gradient Cuff [14] monitors gradient norms to detect optimization-based attacks [67]. However, these methods often incur high latency or fail against complex obfuscations [15, 13].

In-generation Defense intervenes during decoding by monitoring hidden states or manipulating token distributions [60, 30, 46]. SafeDecoding [55] shifts outputs toward refusal tokens, while RepE-based methods encode safety as activation directions for steering without weight updates [66, 17, 37]. Activation probes reuse model activations to cheaply flag high-stakes interactions [32]. TrajGuard [24] monitors windowed hidden-state trajectories and invokes an LLM-based referee under persistent high-risk drift. In contrast, JEDI avoids external LLM adjudication by using CUSUM over Condition-Vector projections to trigger Intervention-Vector steering. Although alleviating prefix exposure, these methods often exhibit over-refusal and sensitivity to calibration [37, 7].

Post-generation Defense detects harm after full response generation [12]. Techniques include Moderation APIs [53], bidirectional intention inference [41], and semantic watermarking [6]. However, these reactive methods cannot prevent prefix exposure in streaming generation scenarios, posing immediate safety risks [48].

These defenses expose a joint safety–utility–efficiency problem motivating RQ1–RQ3. JEDI's contribution is an output-trajectory-conditioned control loop: the risk statistic and steering strength are updated within each response. Linear directions and activation addition are established tools. JBShield [58] detects and manipulates input-activated concepts, while AdaSteer [62] learns input-dependent steering coefficients. JEDI uses a separate sensor and actuator to connect persistent token-level risk with event-triggered, dynamically adjusted intervention. Appendix A details this positioning.

3 Design of JEDI

Guided by the principles of Earliness, Selectivity, and Efficiency, we propose JEDI, an in-generation defense for streaming LLMs. JEDI monitors internal activations and steers ongoing generation when persistent risk is detected. Deployment requires read/write access to per-token hidden states and control of the decoding loop; standard proprietary APIs do not expose these interfaces.

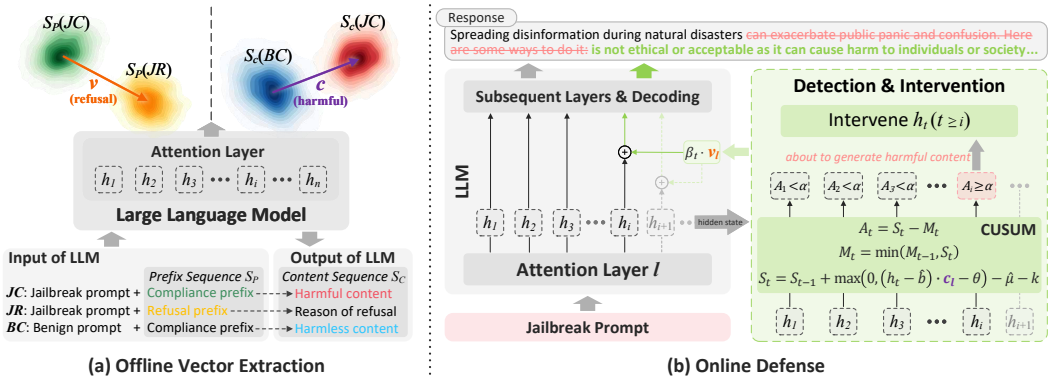


Figure 1: Overview of JEDI: Offline Vector Extraction and Online Detection & Intervention for In-Generation Jailbreak Defense.

3.1 Overview

As illustrated in Figure 1, the JEDI is divided into two phases: **Offline Vector Extraction** and **Online Defense**. In the **Offline Vector Extraction phase** (§ 3.2), we construct specific subspaces to characterize adversarial behavior. We employ a contrastive dataset strategy comprising three types of prompt-prefix pairs: harmful prompts with compliance prefixes (JC), harmful prompts with refusal prefixes (JR), and benign prompts with compliance prefixes (BC). By analyzing the activation differences across these sets, we extract two orthogonal vectors at a target layer l: the *Condition Vector* (c_l), which encodes the semantic direction of harmful content generation; and the *Intervention Vector* (v_l), which represents the behavioral direction of refusal.

In the **Online Defense phase** (§ 3.3), JEDI operates as a real-time monitor within the streaming inference process. For each token generated at time step t, the system projects the current hidden state h_t onto the Condition Vector c_l to calculate a raw risk score. This score is aggregated using CUSUM to track the accumulation of sequential risk (A_t). If A_t surpasses the offline-calibrated alarm threshold α, the system activates a Dynamic Intervention. This mechanism injects the Intervention Vector v_l into the hidden states with an adaptive intensity β_t, determined by the extent of the drift. This intervention redirects the model's trajectory toward the refusal region, intending to preserve benign token generation. This process requires no additional models or repetitive inference

3.2 Offline Vector Extraction

The premise of JEDI is that the semantic distinction between harmful and benign content and the behavioral patterns of refusal versus compliance induce approximately linear, locally exploitable directions within the model's activation space [66, 18]. Appendix C quantifies the harmful/benign separability of the condition vector at the selected layers and provides PCA visualizations for both S_P and S_C. To operationalize this, we perform an offline vector extraction phase to establish a coordinate system for real-time monitoring. This process involves constructing contrastive datasets, normalizing the latent space via whitening, and extracting orthogonal vectors for detection and intervention.

Contrastive Dataset Construction. To extract the directional differences associated with jailbreak attacks and safety refusals, we construct three paired datasets. Each dataset consists of prompt-response prefix pairs designed to isolate specific behavioral: ***Jailbreak Compliance*** (JC): Pairs malicious jailbreak prompts with compliant prefixes (e.g., "Sure, here is") that induce the model to generate harmful content. It represents the harmful content we aim to detect. ***Jailbreak Refusal*** (JR): Using the same jailbreak prompts as JC, we pair them with refusal prefixes (e.g., "I cannot fulfill") that guide the model to reject the request. This represents the safety behavior under attack. ***Benign Compliance*** (BC): Pairs standard, safe user queries with compliant prefixes to generate harmless, helpful responses. It serves as the baseline distribution for defining what constitutes a benign request.

We further define two critical token sequences for feature extraction: the *Prefix Sequence* (S_P, compliant or refusal prefixes), which encodes the initial behavioral stance (compliance vs. refusal), and the *Content Sequence* (S_C, content generated by the LLM following the prefix), which contains refusal-compliance tendencies and harmful-benign semantics.

Activation Extraction and Whitening. Raw hidden states in LLMs often exhibit high anisotropy, where a few dominant directions account for the majority of variance, obscuring subtle semantic signals [66]. To address this, we apply a whitening transformation to the extracted activations, using BC dataset as a baseline to unify the coordinate scale.

Let $H_l \in \mathbb{R}^{T\times d}$ denote the hidden states at layer l for a sequence of length T. We model the distribution of benign activations using the BC dataset. We compute the mean vector $\hat{b}$ of the benign hidden states. The whitening matrix is defined as $W = \Lambda^{-1/2}E^T$, which is obtained by performing eigen-decomposition on the covariance matrix of the benign states, where Λ is the diagonal matrix of eigenvalues and E is the matrix of eigenvectors. The whitening transformation $\Phi(\cdot)$ for any hidden state h is defined as: $\Phi(h) = W(h - \hat{b})$. This transformation standardizes the benign covariance; it does not imply that the activation distribution is Gaussian.

Vector Extraction. We extract two distinct vectors from the whitened space for detection and intervention. ***Condition Vector*** **(c_l)**: To detect harmful content, we calculate the semantic difference between complying with a jailbreak prompt and complying with a benign prompt. We focus on S_C, where the harmful content is generated. The condition vector c_l is computed as the difference in expectations between the whitened representations of JC and BC:

$$c_l \propto \mathbb{E}_{x\in JC}\left[\Phi(h_{S_C}(x))\right] - \mathbb{E}_{x\in BC}\left[\Phi(h_{S_C}(x))\right] \tag{1}$$

Where $h_{S_C}(x)$ denotes the hidden state of layer l representation of the content sequence for input x. This vector points toward harmful semantics and serves as the reference axis for online risk scoring.

Intervention Vector **(v_l)**: To induce refusal, we capture the behavioral shift required to transition from a compliant state to a refusal state under the same attack conditions. We focus on S_P, where the model's decision to accept or reject a query is formed. The raw intervention vector v_l^{raw} is calculated as:

$$v_l^{\text{raw}} = \mathbb{E}_{x\in JR}\left[\Phi(h_{S_P}(x))\right] - \mathbb{E}_{x\in JC}\left[\Phi(h_{S_P}(x))\right] \tag{2}$$

In addition, v_l^{raw} might inadvertently suppress benign utility if the refusal direction overlaps with general instruction-following capabilities. To promote selectivity, we decouple intervention from detection using Gram–Schmidt orthogonalization: $v_l = v_l^{\text{raw}} - c_l(c_l^\top v_l^{\text{raw}})/\|c_l\|^2$. On the calibration distribution, c_l realizes standardized linear mean separation, and v_l is the least-squares constant displacement toward refusal-associated states subject to $v_l^\top c_l = 0$ (Appendix B). These are local, functional interpretations; they do not identify a unique causal refusal mechanism.

3.3 Online Defense

The online defense phase operates synchronously with the streaming generation process. JEDI monitors the model's hidden states at the token level and intervenes only when a persistent accumulation of risk is detected. This phase consists of three components: token-level risk estimation, harmful risk detection, and dynamic intervention.

Token-Level Risk Estimation. At each generation step t, the model generates a hidden state $h_t \in \mathbb{R}^d$ at the target layer l. To quantify the immediate harmfulness of the current token, we project h_t onto the pre-extracted c_l. The raw score s_t is the projection of the whitened hidden state onto the normalized c_l:

$$s_t = \left\langle \Phi(h_t), \frac{c_l}{\|c_l\|} \right\rangle \tag{3}$$

Only c_l is normalized, so s_t is a projection rather than cosine similarity. While s_t provides an instantaneous measure of alignment with the harmful subspace, benign generation processes often exhibit stochastic fluctuations that may transiently align with c_l. To filter out this noise and ensure sparsity in risk signals, we apply a rectified activation function with a calibrated threshold θ. The rectified risk score r_t is defined as:

$$r_t = \text{ReLU}(s_t - \theta) = \max(0, s_t - \theta) \tag{4}$$

The threshold θ is calibrated offline (§ 3.4) to represent the 95th percentile of risk scores observed in benign contexts. Rectification zeros approximately 95% of calibration benign scores, but does not guarantee zero risk over an entire benign generation. Residual risk is handled by sequential calibration.

Harmful Risk Detection via CUSUM. Relying solely on single-step spikes in r_t can lead to false positives due to the high variance of token distributions in LLMs. To address this, we employ the CUSUM to detect persistent shifts in the risk mean. We maintain a cumulative statistic S_t that integrates the deviation of the risk score from an offline-estimated benign mean $\hat{\mu}$, adjusted by the offline-calibrated allowance parameter k. The parameter k serves as a slack variable to tolerate minor system fluctuations without triggering an alarm. The recursive update rule is:

$$S_t = S_{t-1} + r_t - \hat{\mu} - k \tag{5}$$

To detect positive drifts (increases in risk), we track the running minimum of the cumulative sum, denoted as $M_t = \min(M_{t-1}, S_t)$. The detection statistic A_t, which represents the accumulated evidence of harmfulness since the last benign state, is computed as: $A_t = S_t - M_t$. When A_t exceeds a pre-calibrated alarm threshold α, an intervention state is triggered. Equivalently, $A_t = \max(0, A_{t-1} + r_t - \hat{\mu} - k)$. This statistic resets after negative evidence and accumulates persistent positive drift. Appendix B states conditional false-alarm and delay bounds; learned autoregressive risk scores do not establish classical CUSUM minimax optimality.

Dynamic Strength Intervention. Upon detecting a breach of the safety boundary (where $A_t \geq \alpha$), JEDI initiates a steering intervention to realign the model's generation trajectory. Instead of applying a static penalty, which might disrupt semantic coherence, we implement a dynamic intervention mechanism where the steering intensity is proportional to the severity of the detected drift. The intervention strength β_t is calculated using a power-law scaling function: $\beta_t = \beta_{\text{base}} \cdot \left(\frac{A_t}{\alpha}\right)^{\gamma}$. Here, β_{base} sets the minimal effective force and γ governs sensitivity. This ensures proportional control, applying subtle corrections for minor drifts and power-law increases in steering strength for larger deviations.

The intervention is applied in the whitened representation space to ensure directional precision. We first transform the current hidden state h_t into its whitened form $z_t = \Phi(h_t)$, inject the orthogonalized intervention vector v_l, and then map the result back to the original space. The modified hidden state h'_t is obtained as follows:

$$h'_t = \Phi^{-1}\left(\Phi(h_t) + \beta_t v_l\right) = h_t + \beta_t W^{-1} v_l \tag{6}$$

By injecting the v_l, we effectively steer the generation probability distribution towards refusal tokens (e.g., "I cannot") without requiring auxiliary models or re-decoding. Once the intervention successfully reduces the generation risk and A_t falls back below α, the intervention strength naturally decays, allowing the model to resume normal generation capabilities.

3.4 Parameter Calibration

To ensure the robustness and responsiveness of the JEDI framework, we implement a systematic offline calibration process to automatically determine the deployment quantities $\mathcal{P} = \{l, \theta, \hat{\mu}, k, \alpha\}$. Apart from the hidden-state aggregation choice used to extract window-level representations (mean-token aggregation or the last-token representation), these quantities are determined from calibration data rather than manually set as hyperparameters. Extraction, layer selection, and parameter calibration share D_{cal}. Independent prompt sets are reserved for separability analysis and final evaluation, as specified in Section 4.

Optimal Layer Identification. To identify the most effective layer l, we evaluate the separability of JC and BC activations across all layers. By computing the AUROC and Cohen's d to quantify the separability of s_t across layers, we select the layer with the highest discriminative power.

Baseline Noise Filtering. To mitigate false alarms caused by nominal semantic fluctuations, we establish a threshold θ calibrated on the BC dataset. By setting $\theta = \text{Quantile}(\{s_t \mid x \in BC\}, 0.95)$, we effectively nullify 95% of benign token variations after ReLU rectification (Eq. 4), isolating only significant risk deviations. After applying θ, we estimate $\hat{\mu}$ as the empirical mean of rectified risk scores on BC, giving CUSUM a data-driven baseline for normal residual risk.

CUSUM Parameter Search. To calibrate the CUSUM quantities (k, α), we solve a constrained problem that minimizes the Average Detection Delay (ADD) on harmful samples while restricting the False Positive Rate (FPR) on benign samples to a tolerance ϵ (e.g., 0.05):

$$\min_{k,\alpha} \text{ADD}(k, \alpha) \quad \text{s.t.} \quad \text{FPR}(k, \alpha) \leq \epsilon \tag{7}$$

We conduct a grid search over candidate (k, α) pairs using the JC and BC datasets. The selected configuration balances detection delay with benign sequence-level alarm frequency on the calibration distribution.

4 Experimental Setup

Target Models and Jailbreak Attacks. We conduct evaluation on 6 open-source LLMs to verify the effectiveness of JEDI: **Vicuna 7B/13B** [27, 26], **Mistral 7B** [34], **Qwen2.5 7B/14B** [3, 2], and **Qwen3-8B** [4]. We require publicly available checkpoints with hidden-state hooks; the two Vicuna and Qwen2.5 sizes provide within-family scale comparisons, Mistral broadens model lineage, and Qwen3 adds a newer release generation. The 7B–14B range keeps the evaluation tractable on a single H100 GPU. We employ 11 diverse jailbreak attacks that cover gradient-based optimization (**GCG** [67]), automated red-teaming (**PAIR** [8], **TAP** [33], **AutoDAN** [65]), and black-box fuzzing or strategy-based injections (**Random Search (RS)** [5], **AIM**, **BETTERDAN (BtDAN)**, **DEVMODERanti (DevMode)** [1], **ReneLLM** [11], **DeepInception (DI)** [21], **GPTFuzzer (GPTFuz)** [57]).

Baselines. We compare five defenses spanning the generation lifecycle: pre-generation **Gradient Cuff** [14] and **SmoothLLM** [35], in-generation **SafeDecoding** [55] and **Activation Probe** [32], and post-generation **BackTranslation** [47]. Activation Probe uses its official implementation trained on JC and BC; the other four use PandaGuard [38] with recommended hyperparameters. Online comparisons also include **CAST** [18], adapted **TrajGuard** [24], and **Qwen3Guard-Gen-8B** [61] with 16/32-token buffered checks. Appendix D.1 specifies their protocols.

Metrics. We employ a multi-dimensional metric system to balance safety, utility, and efficiency:

Safety: We report the **Defense Success Rate (DSR)** on a test set of 1,100 transfer jailbreak prompts (100 per attack method) sourced from PandaGuard [38]. For the four attack methods (GCG, PAIR, AutoDAN, and TAP) represented in offline calibration, the test prompts are generated by the same methods but drawn from different datasets and non-overlapping samples, while prompts from the remaining seven attack methods are entirely absent from offline calibration. Formally, DSR is calculated as $\text{DSR} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{J}(\mathcal{C}(y_i) \neq \text{Harmful})$, where N is the total number of prompts, y_i is the model response, and $\mathcal{J}$ is the judgment function. Whether a response is *harmful* is determined by the Llama-2-13b-cls classifier ($\mathcal{C}$) from HarmBench [31]. A 1,000-output audit yields 97.2% classifier–human agreement (Appendix C.1). Full-response DSR does not establish absence of unsafe prefixes; a separate matched audit reports prefix onsets and their intersection with final-response harm (Section 5).

Utility: To assess the impact on general capabilities, we evaluate the models on AlpacaEval [22] and report the **Win Rate** against standard responses. AlpacaEval captures open-ended instruction-following quality on benign tasks, so it reflects whether a defense preserves helpfulness rather than merely suppressing unsafe generations. To evaluate over-refusal, we use XSTest [36] and report the **Compliance Rate**. XSTest includes benign requests that superficially resemble harmful instructions, making it suitable for measuring whether the defended model still complies with permissible queries. We also evaluate on OR-Bench [10], a dedicated over-refusal benchmark, and report the **False Positive Rate (FPR)** on benign prompts.

Efficiency: We monitor the **Time to First Token (TTFT)** during the AlpacaEval benchmarks to measure the elapsed time (in seconds) from prompt submission to the first token's emission. TTFT is critical in streaming generation because users begin consuming the response immediately after the first token appears; excessive delay can undermine the perceived responsiveness of the system. For buffered guards, TTFT is time to first user-visible release, including chunk formation, queueing, and guard inference. We also report total generation time per response; concurrent batched-serving throughput is not measured.

Datasets and Parameter Calibration. D_{cal} contains 100 harmful HarmBench prompts generated via GCG, PAIR, AutoDAN, and TAP and 100 benign JustEval prompts (excluding AlpacaEval-source items). The same set supports whitening/vector extraction, layer selection, and calibration of $\theta, \hat{\mu}, k, \alpha$; these stages are not mutually partitioned. An independent D_{sep} contains another 100 harmful and 100 benign prompts and measures separability with all offline quantities frozen. Final evaluation uses disjoint PandaGuard, AlpacaEval, XSTest, and OR-Bench prompts. Prefix expansion creates correlated augmentations, not additional independent prompts. Normalized overlap checks and prompt-level uncertainty are detailed in Appendix C.1.

Table 1: Per-attack DSR (%, ↑), averaged over six target LLMs, for methods with disaggregated results. Bold and underline mark the best and second-best defense values among the displayed methods.

Defences	GCG	PAIR	AutoDAN	TAP	RS	AIM	BtDAN	DevMode	ReneLLM	DI	GPTFuz	Avg.	Std.
Undefend	65.5	58.8	45.8	72.2	5.0	27.2	48.0	62.3	66.0	78.3	55.3	53.1	21.2
JEDI (ours)	89.8	**86.8**	**91.3**	90.8	**99.5**	**100.0**	**100.0**	**100.0**	**98.2**	**96.0**	**99.8**	**95.7**	5.0
Gradient Cuff	**95.2**	71.7	62.5	79.5	33.0	69.5	95.8	97.3	69.8	81.7	78.3	75.8	18.4
SafeDecoding	89.0	76.5	61.8	83.0	57.3	76.5	54.3	78.8	72.2	82.2	84.5	74.2	11.6
SmoothLLM	86.3	62.2	61.3	74.8	29.2	46.7	68.2	70.3	84.8	85.2	72.8	67.4	17.4
BackTranslation	70.3	64.2	51.2	75.7	16.0	43.8	87.5	88.3	68.0	84.0	67.3	65.1	21.5
Activation Probe	92.0	**86.8**	86.5	**94.2**	98.7	87.8	85.2	93.2	88.0	93.0	89.5	90.4	**4.1**

Table 2: Average Win Rate (%, ↑) and TTFT (s, ↓) of JEDI and baselines on AlpacaEval. Bold and underline indicate the best and second-best defense performance, respectively.

Defences	Vicuna 7B		Vicuna 13B		Mistral 7B		Qwen2.5 7B		Qwen2.5 14B		Qwen3-8B		Avg.	
	WinRate	TTFT	WinRate	TTFT	WinRate	TTFT	WinRate	TTFT	WinRate	TTFT	WinRate	TTFT	WinRate	TTFT
Undefend	64.58	0.016	74.68	0.022	87.56	0.021	92.23	0.016	94.41	0.026	91.77	0.025	84.21	0.021
JEDI (ours)	58.82	**0.017**	55.04	**0.023**	84.99	0.023	89.07	0.019	88.67	**0.028**	86.48	0.031	77.18	**0.024**
Gradient cuff	**61.10**	2.906	65.07	3.717	25.59	2.884	90.83	4.698	93.58	3.674	**91.10**	4.020	71.21	3.650
SafeDecoding	52.80	0.047	55.83	0.052	82.86	0.051	90.40	0.071	93.72	0.059	89.64	0.060	77.54	0.057
SmoothLLM	41.63	10.599	48.28	12.290	54.64	13.910	67.85	10.200	75.00	14.560	69.64	9.892	59.51	11.909
BackTranslation	48.47	44.490	**68.99**	48.650	**85.50**	51.670	**91.60**	47.790	**94.00**	51.990	90.80	55.490	**79.89**	50.013
Activation Probe	47.52	0.024	44.18	0.033	73.77	**0.019**	73.98	**0.017**	75.12	0.030	82.30	**0.029**	66.15	0.025

Calibration pairs prompts with 100 diverse compliance and refusal prefixes (e.g., "Sure, here is..." and "I cannot...") adapted from Lee et al. [18], forming 10,000-sample contrastive datasets. We use mean-token aggregation within the extraction windows, select l by separability, and calibrate (k, α) to minimize detection delay under a 5% benign alarm-FPR constraint. Intervention parameters are fixed at $\beta_{\text{base}} = 1$ and $\gamma = 1.5$ across target models.

5 Result Analysis

Safety. To evaluate JEDI's defensive robustness against a diverse spectrum of jailbreak attacks, we assess DSR across 6 target LLMs and 11 attack methods, comparing it against five baselines. We use DSR to evaluate full-response safety and separately audit visible-prefix exposure below.

Table 1 summarises the average DSR achieved by JEDI and the baseline methods across the 6 target LLMs. The last two columns summarize average performance (Avg.) and the reported standard deviation (Std.); these describe variation across evaluation conditions rather than confidence intervals over independent runs. The *Undefend* row represents the vanilla target models without any defence methods.

Among the five comparison methods in Table 1, JEDI achieves the highest average DSR of 95.7%, 5.3 percentage points above Activation Probe (90.4%). Activation Probe attains the lowest overall standard deviation (4.1), while JEDI remains similarly stable (5.0) and achieves the best average DSR among these methods on all six target models, ranging from 93.6% to 99.4%. The baselines also demonstrate substantial effectiveness over the undefended models (complete experimental results provided in Appendix D.3).

JEDI's dual-vector mechanism explicitly models both the *harmful content* direction (c_l) and the *refusal behaviour* direction (v_l), connecting latent-space risk detection to corrective steering during generation. The component ablations below test their functional contributions; full-response DSR alone cannot establish prevention of harmful prefixes.

Utility & Efficiency. We evaluate utility and efficiency with AlpacaEval Win Rate, TTFT, XSTest Compliance Rate (CR), and OR-Bench FPR. XSTest measures whether defenses preserve compliance on benign prompts that superficially resemble harmful requests, while OR-Bench offers a more targeted over-refusal test on benign but safety-sensitive prompts.

As shown in Table 2, JEDI achieves a 77.18% average Win Rate, close to BackTranslation (79.89%) and SafeDecoding (77.54%) and higher than Activation Probe (66.15%), while retaining the highest DSR among these five comparison methods. Its average TTFT is 0.024s, only 0.003s above the

Table 3: Compliance Rate (CR, %, ↑) on XSTest and False Positive Rate (FPR, %, ↓) on OR-Bench. Bold and underline indicate the best and second-best defense performance, respectively.

Defences	Vicuna 7B		Vicuna 13B		Mistral 7B		Qwen2.5 7B		Qwen2.5 14B		Qwen3-8B		Avg.		Std.	
	CR	FPR	CR	FPR	CR	FPR	CR	FPR	CR	FPR	CR	FPR	CR	FPR	CR	FPR
Undefend	92.8	11.4	88.8	10.4	98.0	8.1	97.2	6.4	98.4	7.1	95.2	9.1	95.1	8.8	3.7	1.9
JEDI (ours)	**86.4**	13.3	**87.6**	11.8	92.0	16.8	94.8	8.5	91.2	**12.6**	94.4	11.7	**91.1**	12.5	**3.5**	**2.7**
Gradient cuff	62.8	18.3	69.2	18.0	6.4	7.7	92.0	7.2	94.8	25.0	92.4	9.8	69.6	14.3	33.8	7.2
SafeDecoding	58.8	42.8	46.8	49.1	88.0	19.0	95.2	9.1	**97.6**	51.4	83.6	14.3	78.3	31.0	20.7	18.9
SmoothLLM	74.4	**12.2**	63.6	9.7	92.0	6.3	84.4	7.9	86.4	18.5	80.4	9.9	80.2	**10.8**	10.0	4.3
BackTranslation	70.8	31.8	81.2	**9.2**	89.2	**6.1**	**96.4**	**6.2**	**97.6**	17.8	94.0	**9.4**	88.2	13.4	10.4	10.0
Activation Probe	60.4	15.7	71.6	11.3	**95.2**	20.4	92.4	9.9	96.0	13.8	**95.6**	11.3	85.2	13.7	15.3	3.9

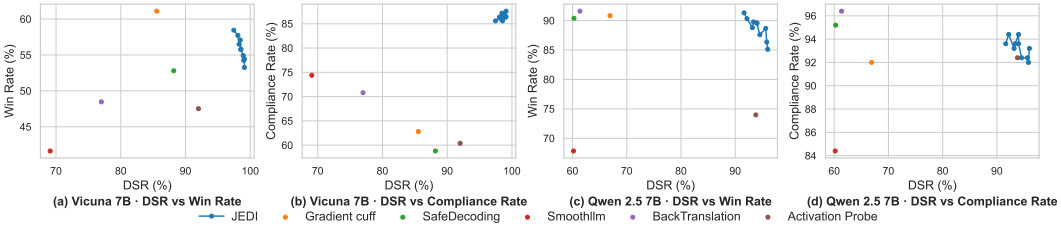


Figure 2: Safety-utility trade-off under varying calibration and intervention settings.

undefended setting. In contrast, baselines that require input-side processing or multi-pass generation introduce much larger first-token delays; detailed total generation time is provided in Appendix D.4.

Table 3 further shows that JEDI reaches the highest average XSTest CR among the displayed defenses (91.1%) with the lowest CR standard deviation (3.5), indicating stable compliance preservation across models. On OR-Bench, JEDI achieves a competitive average FPR of 12.5%: SmoothLLM attains the lowest FPR (10.8%), but JEDI remains close while retaining much stronger jailbreak defense and the lowest FPR standard deviation (2.7) among defenses.

JEDI thus combines high jailbreak DSR and stable over-refusal behavior while preserving streaming responsiveness and much of the model's helpfulness on safe, complex instructions.

Table 7 extends the comparison to online defenses. Qwen3Guard-Gen-16/32 reaches higher DSR (98.48/98.41%), with 0.447/0.450 s first-release latency and lower utility. CAST and TrajGuard achieve similar XSTest CR (92.7/90.0%) but lower DSR and Win Rate than JEDI. These are evaluated operating points, not matched operating curves. JEDI's total generation time is 8.89 s versus 7.98 s undefended, so low TTFT overhead does not imply zero decoding cost.

Visible-prefix Exposure. On 3,975 matched cases selected for a Qwen3Guard-Stream-8B `Unsafe` onset in the undefended stream, JEDI reduces raw visible-onset frequency from 100% to 26.61%. Final-response HarmBench harm is 6.30%, with a 4.86% onset intersection (Table 9). Raw onsets can include false positives; the intersection can omit unsafe prefixes later redirected into refusal. Their 21.75-point gap is not verified harm. Buffered guards further reduce exposure at greater latency and utility cost. Similar conditional leakage lengths indicate chiefly reduced leakage frequency. Appendix D.2 details the protocol; subset rates differ from full-evaluation DSR.

Ablation Study. To analyze the trade-off between safety and utility of JEDI, we conduct an ablation study focusing on the calibrated alarm threshold α and the base intervention strength β_{base}. We visualize these trade-offs in Figure 2, where the x-axis represents the DSR and the y-axis represents utility metrics (Win Rate or Compliance Rate). We systematically vary one quantity while fixing the other to its deployment value. For α, we introduce a bias $\Delta\alpha \in \{-20, -10, 0, +10, +20\}$ to the calibrated baseline α^*. For β_{base}, we set $\beta_{\text{base}} \in \{0.25, 0.5, 1.0, 1.5, 2.0\}$ while keeping $\gamma = 1.5$.

As shown in Figure 2, JEDI's configurations trace a trade-off from higher DSR and lower utility to lower DSR and higher utility. Automated calibration minimizes detection delay under a benign alarm-FPR constraint, providing a deployment point on this within-method sensitivity curve. The

sweep illustrates tunability rather than global Pareto optimality. Detailed results under different parameter settings are provided in Appendix D.5.

Controlled ablations on Vicuna-7B/Qwen2.5-7B further test the control loop (Table 18). Replacing accumulated risk with instantaneous risk at matched benign alarm FPR lowers DSR by 21.56/14.09 points; this substitution changes both triggering and strength scaling. Removing orthogonalization lowers XSTest CR by 9.60 points on each model without improving DSR. Fixing $\beta = 1$ preserves alarms but lowers DSR by 38.29/28.00 points. These results concern the tested substitutions, not all possible static strengths.

Five-fold prefix-bank holdout, with detection and whitening fixed, yields DSR of $93.25 \pm 0.17\%$ on Qwen2.5-7B and $98.35 \pm 0.34\%$ on Vicuna-7B (mean $\pm$ fold standard deviation). This supports stability under the tested prefix resampling; natural-refusal extraction remains untested (Appendix C.2).

Adaptive Attack. To evaluate JEDI's robustness, we consider a partial-information adaptive setting: the attacker knows the mechanism and receives alarm status, first-alarm time, and peak A_t/α, but not all learned internals or unrestricted per-token signals. Following prior works [42, 14], we adapt three representative attack methods (GCG, PAIR, and GPTFuzzer) to collect JEDI's feedback parameters during optimization and guide attack generation with a unified comprehensive score. Implementation details are provided in Appendix E.

Table 4: DSR (%, ↑) of JEDI under standard and adaptive attacks.

Attack Method	Vicuna 7B		Qwen 2.5 7B	
	w/o	w/	w/o	w/
Adaptive-GCG	97	95	97	93
Adaptive-PAIR	91	89	79	77
Adaptive-GPTFuz	100	99	100	96

Table 4 presents DSR of JEDI under adaptive attack scenarios. The columns labeled *w/o* represent the baseline DSR against standard, non-adaptive attacks, while *w/* denotes DSR against adaptive attacks in which the adversary actively optimizes against the defense's feedback. As presented in Table 4, JEDI exhibits a decrease in DSR when facing adaptive strategies compared to the standard setting. This suggests that while JEDI is robust, it is not strictly invariant to adaptive adversarial optimization.

6 Conclusion & Discussion.

We proposed JEDI, a real-time in-generation defense method for streaming generation that mitigates prefix exposure in LLMs. JEDI conducts online monitoring of hidden states/activations via representation engineering, using a Condition Vector to compute a token-level risk score and applying CUSUM for sequential drift detection, enabling early, selective, and efficient triggered intervention. Across 6 open-source LLMs and 11 jailbreak attacks, JEDI achieves 95.7% average DSR while maintaining high utility and incurring 0.003 seconds TTFT overhead.

Our results highlight two implications for streaming safety. (I) Sequential monitoring matters: token-level risk scores are noisy, and effective defenses should respond to persistent drift rather than isolated spikes—precisely what CUSUM statistics capture through evidence accumulation. (II) The safety–utility trade-off is tunable: separating *when to intervene* (via the calibration threshold and CUSUM triggering) from *how strongly to intervene* (via dynamic intervention strength) supports configurable deployment with lower first-release latency than the evaluated multi-pass filtering and buffered guard baselines. Limitations and future directions are discussed in Appendix F.

7 Limitations

JEDI requires hidden-state access and decoding control. Results cover instruction-tuned 7B–14B models, including SFT-only Vicuna, and do not establish effectiveness on purely pretrained models. Calibration has 200 unique prompts; prefix-expanded samples are correlated. Camouflaged trajectories can evade detection, and steering can reduce benign utility. Prefix evaluators are imperfect, and fully informed adaptive attacks remain open. Timing measures single-H100 per-response latency rather than concurrent batched-serving throughput. Broader calibration and proxy deployment remain future work (Appendix F).

Acknowledgments and Disclosure of Funding

This work is supported by the Key Research and Development Program of Jiangsu Province (Grant No. BE2021002-3), the Nectar Research Cloud, and the Big Data Computing Center of Southeast University.

References

[1] Alex Albert. Jailbreak chat. [Online], 2025. URL `https://github.com/alexalbertt/jailbreakchat`. Accessed: 2025-12-11.

[2] Alibaba. Qwen 2.5 14b. `https://huggingface.co/Qwen/Qwen2.5-14B`, 2024.

[3] Alibaba. Qwen 2.5 7b. `https://huggingface.co/Qwen/Qwen2.5-7B`, 2024.

[4] Alibaba. Qwen 3 8b. `https://huggingface.co/Qwen/Qwen3-8B`, 2025.

[5] Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. Jailbreaking leading safety-aligned llms with simple adaptive attacks. *arXiv preprint arXiv:2404.02151*, 2024.

[6] Lele Cao. A practical synthesis of detecting ai-generated textual, visual, and audio content. *arXiv preprint arXiv:2504.02898*, 2025.

[7] Zouying Cao, Yifei Yang, and Hai Zhao. Scans: Mitigating the exaggerated safety for llms via safety-conscious activation steering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23523–23531, 2025.

[8] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. In *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 23–42. IEEE, 2025.

[9] Junjie Chu, Yugeng Liu, Ziqing Yang, Xinyue Shen, Michael Backes, and Yang Zhang. Jailbreakradar: Comprehensive assessment of jailbreak attacks against llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21538–21566, 2025.

[10] Justin Cui, Wei-Lin Chiang, Ion Stoica, and Cho-Jui Hsieh. Or-bench: An over-refusal benchmark for large language models. *arXiv preprint arXiv:2405.20947*, 2024.

[11] Peng Ding, Jun Kuang, Dan Ma, Xuezhi Cao, Yunsen Xian, Jiajun Chen, and Shujian Huang. A wolf in sheep's clothing: Generalized nested jailbreak prompts can fool large language models easily. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2136–2153, 2024.

[12] Yi Dong, Ronghui Mu, Yanghao Zhang, Siqi Sun, Tianle Zhang, Changshun Wu, Gaojie Jin, Yi Qi, Jinwei Hu, Jie Meng, et al. Safeguarding large language models: A survey. *Artificial intelligence review*, 58(12): 382, 2025.

[13] Xueluan Gong, Mingzhe Li, Yilin Zhang, Fengyuan Ran, Chen Chen, Yanjiao Chen, Qian Wang, and Kwok-Yan Lam. {PAPILLON}: Efficient and stealthy fuzz {Testing-Powered} jailbreaks for {LLMs}. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 2401–2420, 2025.

[14] Xiaomeng Hu, Pin-Yu Chen, and Tsung-Yi Ho. Gradient cuff: Detecting jailbreak attacks on large language models by exploring refusal loss landscapes. *Advances in Neural Information Processing Systems*, 37: 126265–126296, 2024.

[15] Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*, 2023.

[16] Weisen Jiang and Sinno Jialin Pan. Metadefense: Defending finetuning-based jailbreak attack before and during generation. *arXiv preprint arXiv:2510.07835*, 2025.

[17] Bruce W Lee, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Erik Miehling, Pierre Dognin, Manish Nagireddy, and Amit Dhurandhar. Programming refusal with conditional activation steering. *arXiv preprint arXiv:2409.05907*, 2024.

[18] Bruce W. Lee, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Erik Miehling, Pierre Dognin, Manish Nagireddy, and Amit Dhurandhar. Programming refusal with conditional activation steering. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Oi47wc10sm.

[19] Jiahui Li, Yongchang Hao, Haoyu Xu, Xing Wang, and Yu Hong. Exploiting the index gradients for optimization-based jailbreaking on large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 4535–4547, 2025.

[20] Xiaohu Li, Yunfeng Ning, Zepeng Bao, Mayi Xu, Jianhao Chen, and Tieyun Qian. Cavgan: Unifying jailbreak and defense of llms via generative adversarial attacks on their internal representations. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6664–6678, 2025.

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

[22] Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval, 5 2023.

[23] Xiao Lin, Philip Li, Zhichen Zeng, Tingwei Li, Tianxin Wei, Xuying Ning, Gaotang Li, Yuzhong Chen, and Hanghang Tong. Alert: Zero-shot llm jailbreak detection via internal discrepancy amplification. *arXiv preprint arXiv:2601.03600*, 2026.

[24] Cheng Liu, Xiaolei Liu, Xingyu Li, Bangzhou Xin, and Kangyi Ding. Trajguard: Streaming hidden-state trajectory detection for decoding-time jailbreak defense. *arXiv preprint arXiv:2604.07727*, 2026.

[25] Yi Liu, Gelei Deng, Zhengzi Xu, Yuekang Li, Yaowen Zheng, Ying Zhang, Lida Zhao, Tianwei Zhang, Kailong Wang, and Yang Liu. Jailbreaking chatgpt via prompt engineering: An empirical study. *arXiv preprint arXiv:2305.13860*, 2023.

[26] LMSYS. Vicuna 13b v1.5. https://huggingface.co/lmsys/vicuna-13b-v1.5, 2023.

[27] LMSYS. Vicuna 7b v1.5. https://huggingface.co/lmsys/vicuna-7b-v1.5, 2023.

[28] Gary Lorden. Procedures for reacting to a change in distribution. *The Annals of Mathematical Statistics*, 42(6):1897–1908, 1971. doi: 10.1214/aoms/1177693055.

[29] Xiaoya Lu, Dongrui Liu, Yi Yu, Luxin Xu, and Jing Shao. X-boundary: Establishing exact safety boundary to shield llms from multi-turn jailbreaks without compromising usability. *arXiv preprint arXiv:2502.09990*, 2025.

[30] Yanxu Mao, Tiehan Cui, Peipei Liu, Datao You, and Hongsong Zhu. From llms to mllms to agents: A survey of emerging paradigms in jailbreak attacks and defenses within llm ecosystem. *arXiv preprint arXiv:2506.15170*, 2025.

[31] Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, et al. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint arXiv:2402.04249*, 2024.

[32] Alex McKenzie, Urja Pawar, Phil Blandfort, William Bankes, David Krueger, Ekdeep Singh Lubana, and Dmitrii Krasheninnikov. Detecting high-stakes interactions with activation probes. *arXiv preprint arXiv:2506.10805*, 2025.

[33] Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. Tree of attacks: Jailbreaking black-box llms automatically. *Advances in Neural Information Processing Systems*, 37:61065–61105, 2024.

[34] Mistral. Mistral 7b v0.2. https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2, 2023.

[35] Alexander Robey, Eric Wong, Hamed Hassani, and George J. Pappas. SmoothLLM: Defending large language models against jailbreaking attacks. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=laPAh2hRFC.

[36] Paul Röttger, Hannah Kirk, Bertie Vidgen, Giuseppe Attanasio, Federico Bianchi, and Dirk Hovy. XSTest: A test suite for identifying exaggerated safety behaviours in large language models. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5377–5400, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.301. URL https://aclanthology.org/2024.naacl-long.301/.

[37] Guobin Shen, Dongcheng Zhao, Yiting Dong, Xiang He, and Yi Zeng. Jailbreak antidote: Runtime safety-utility balance via sparse representation adjustment in large language models. *arXiv preprint arXiv:2410.02298*, 2024.

[38] Guobin Shen, Dongcheng Zhao, Linghao Feng, Xiang He, Jihang Wang, Sicheng Shen, Haibo Tong, Yiting Dong, Jindong Li, Xiang Zheng, and Yi Zeng. Pandaguard: Systematic evaluation of llm safety against jailbreaking attacks, 2025. URL `https://arxiv.org/abs/2505.13862`.

[39] Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1671–1685, 2024.

[40] Jingtong Su, Julia Kempe, and Karen Ullrich. Mission impossible: A statistical perspective on jailbreaking llms. *Advances in Neural Information Processing Systems*, 37:38267–38306, 2024.

[41] Haibo Tong, Dongcheng Zhao, Guobin Shen, Xiang He, Dachuan Lin, Feifei Zhao, and Yi Zeng. Bidirectional intention inference enhances llms' defense against multi-turn jailbreak attacks. *arXiv preprint arXiv:2509.22732*, 2025.

[42] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *Advances in neural information processing systems*, 33:1633–1645, 2020.

[43] Vellum AI. What is LLM streaming and how to use it?, 2025. URL `https://www.vellum.ai/llm-parameters/llm-streaming`.

[44] Xuekang Wang, Shengyu Zhu, and Xueqi Cheng. Speculative safety-aware decoding. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 12827–12841, 2025.

[45] Xunguang Wang, Zhenlan Ji, Wenxuan Wang, Zongjie Li, Daoyuan Wu, and Shuai Wang. Sok: Evaluating jailbreak guardrails for large language models. *arXiv preprint arXiv:2506.10597*, 2025.

[46] Xunguang Wang, Daoyuan Wu, Zhenlan Ji, Zongjie Li, Pingchuan Ma, Shuai Wang, Yingjiu Li, Yang Liu, Ning Liu, and Juergen Rahmel. {SelfDefend}:{LLMs} can defend themselves against jailbreaking in a practical manner. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 2441–2460, 2025.

[47] Yihan Wang, Zhouxing Shi, Andrew Bai, and Cho-Jui Hsieh. Defending llms against jailbreaking attacks via backtranslation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 16031–16046, 2024.

[48] Yuquan Wang, Mi Zhang, Yining Wang, Geng Hong, Xiaoyu You, and Min Yang. Reasoningguard: Safeguarding large reasoning models with inference-time safety aha moments. *arXiv preprint arXiv:2508.04204*, 2025.

[49] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? *Advances in Neural Information Processing Systems*, 36:80079–80110, 2023.

[50] Thomas Winninger, Boussad Addad, and Katarzyna Kapusta. Using mechanistic interpretability to craft adversarial attacks against large language models. *arXiv preprint arXiv:2503.06269*, 2025.

[51] Daoyuan Wu, Shuai Wang, Yang Liu, and Ning Liu. Llms can defend themselves against jailbreaking in a practical manner: A vision paper. *arXiv preprint arXiv:2402.15727*, 2024.

[52] Yueqi Xie, Minghong Fang, Renjie Pi, and Neil Gong. Gradsafe: Detecting jailbreak prompts for llms via safety-critical gradient analysis. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 507–518, 2024.

[53] Yuan Xin, Dingfan Chen, Linyi Yang, Michael Backes, and Xiao Zhang. Jailbreaking attacks vs. content safety filters: How far are we in the llm safety arms race? *arXiv preprint arXiv:2512.24044*, 2025.

[54] Wenrui Xu and Keshab K Parhi. A survey of attacks on large language models. *arXiv preprint arXiv:2505.12567*, 2025.

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

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

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

[58] Shenyi Zhang, Yuchen Zhai, Keyan Guo, Hongxin Hu, Shengnan Guo, Zheng Fang, Lingchen Zhao, Chao Shen, Cong Wang, and Qian Wang. JBShield: Defending large language models from jailbreak attacks through activated concept analysis and manipulation. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 8215–8234. USENIX Association, 2025. URL https://www.usenix.org/conference/usenixsecurity25/presentation/zhang-shenyi.

[59] Chongwen Zhao, Zhihao Dou, and Kaizhu Huang. Eeg-defender: Defending against jailbreak through early exit generation of large language models. *arXiv preprint arXiv:2408.11308*, 2024.

[60] Chongwen Zhao, Yutong Ke, and Kaizhu Huang. Unraveling llm jailbreaks through safety knowledge neurons. *arXiv preprint arXiv:2509.01631*, 2025.

[61] Haiquan Zhao, Chenhan Yuan, Fei Huang, et al. Qwen3Guard technical report. *arXiv preprint arXiv:2510.14276*, 2025. doi: 10.48550/arXiv.2510.14276. URL https://arxiv.org/abs/2510.14276.

[62] Weixiang Zhao, Jiahe Guo, Yulin Hu, Yang Deng, An Zhang, Xingyu Sui, Xinyang Han, Yanyan Zhao, Bing Qin, Tat-Seng Chua, and Ting Liu. AdaSteer: Your aligned LLM is inherently an adaptive jailbreak defender. *arXiv preprint arXiv:2504.09466*, 2025. doi: 10.48550/arXiv.2504.09466. URL https://arxiv.org/abs/2504.09466.

[63] Xuandong Zhao, Will Cai, Tianneng Shi, David Huang, Licong Lin, Song Mei, and Dawn Song. Improving LLM safety alignment with dual-objective optimization. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=Kjivk5OPtL.

[64] Yinzhi Zhao, Ming Wang, Shi Feng, Xiaocui Yang, Daling Wang, and Yifei Zhang. Defending large language models against jailbreak attacks via in-decoding safety-awareness probing. *arXiv preprint arXiv:2601.10543*, 2026.

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

[66] Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*, 2023.

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

A Additional Background and Related Defenses

A.1 Comprehensive Background of Jailbreak Attacks

Current jailbreak methods fall into two categories based on access level and attack methods:

White-box Optimization Attacks. These methods assume full access to the model's parameters and gradients. The seminal Greedy Coordinate Gradient (GCG) [67] attack formulates jailbreaking as a discrete optimization problem, searching for adversarial suffixes that maximize the likelihood of a target affirmative response (e.g., *Sure, here is how...*) [19]. Recent advancements have moved beyond token-level optimization to representation-level manipulation. Techniques such as Subspace Rerouting utilize mechanistic interpretability to identify and redirect the internal activation vectors responsible for refusal, effectively bypassing the model's safety circuitry by altering its internal state trajectories [50].

Black-box Automated Attacks. In scenarios where model weights are inaccessible, attackers employ query-based strategies. Evolution-based frameworks, such as Prompt Automatic Iterative Refinement (PAIR) [8] and Tree of Attacks with Pruning (TAP) [33], utilize an auxiliary *attacker LLM* to iteratively refine prompts based on the target model's feedback. These methods effectively search the semantic space for inputs that evade safety filters, achieving high success rates on closed-source models through recursive optimization.

In addition, distinct from the algorithmic categories above, a significant portion of attacks relies on manually crafted templates. These static, heuristic-based inputs exploit role-playing pretexts to bypass safety filters without complex optimization [39, 9]. While often less robust than automated methods, manual templates remain a pervasive threat due to their low barrier to entry and interpretability.

A.2 Decoding-time Defenses

Activation Probe [32] is a lightweight in-generation monitor that reuses the target model's residual-stream activations to detect high-stakes interactions with minimal additional latency. It trains a probe over internal activations and aggregates token-level information into a sequence-level risk score through architectures such as mean pooling, attention weighting, softmax weighting, or max-based scoring. This makes activation probing an efficient alternative to prompted or finetuned LLM monitors, especially when the goal is to flag risky contexts during deployment. However, as a static activation classifier, it mainly learns a fixed decision surface for detection: once trained, the probe maps observed activations to a risk probability, but it does not explicitly model the temporal accumulation of harmful generation dynamics or specify how the model's ongoing decoding trajectory should be corrected after risk is detected.

JEDI is fundamentally different because it turns activation monitoring into a representation-engineered control loop. Instead of relying on a static probe score, JEDI constructs a Condition Vector from the contrast between JC and BC, so the monitored signal is aligned with harmful compliance semantics rather than a broad high-stakes label. During streaming generation, token-level projections onto this vector are calibrated against the benign baseline and accumulated by CUSUM, allowing JEDI to distinguish persistent harmful drift from transient activation spikes that can mislead static probes. More importantly, JEDI couples this sequential detector with an Intervention Vector extracted from JR and JC: when the Alarm Threshold is exceeded, Dynamic Intervention Strength steers the hidden state toward refusal behavior rather than merely flagging, blocking, or routing the interaction. Thus, while both Activation Probe and JEDI preserve the low-latency advantage of activation-level in-generation defense, JEDI provides a stronger mechanism for jailbreak defense by combining explicit harmful-semantic representation, sequential evidence accumulation, and activation-level intervention within the same decoding loop.

TrajGuard [24] is a training-free decoding-time defense that models generation as a hidden-state trajectory rather than an isolated input or final output. It monitors streaming hidden states in selected representation subspaces with sliding windows and smoothed trajectory-level risk statistics. When the trajectory persistently approaches a malicious reference region, TrajGuard invokes semantic adjudication before terminating or constraining the response. This makes TrajGuard closely related to in-generation defense, as it aims to detect harmful behavior before relying on post-generation filtering.

JEDI differs from TrajGuard in both risk modeling and defensive action. TrajGuard primarily relies on geometric trajectory deviation from benign/malicious reference regions, whereas JEDI projects token-level hidden states onto a contrastively extracted Condition Vector and applies CUSUM to detect persistent harmful semantic drift. More importantly, TrajGuard follows a detect-then-adjudicate or interrupt paradigm, while JEDI closes the loop between detection and intervention: once the Alarm Threshold is exceeded, it injects an orthogonalized Intervention Vector with Dynamic Intervention Strength to steer the continuing generation toward refusal behavior. Thus, JEDI explicitly separates *when to intervene* from *how to intervene*, instead of relying mainly on trajectory detection followed by adjudication or termination.

The evaluated TrajGuard baseline is an adaptation to the same cached-decoding stack used for JEDI. Its layer/subspace selection, smoothing, calibration, and adjudication interface are implementation choices that affect the comparison. The measurements characterize this adapted configuration; they are not a claim about every possible TrajGuard operating point (Appendix D.1).

SelfDefend [46] is a practical concurrent defense framework inspired by shadow stacks. It runs a shadow LLM in parallel with the target LLM: the target model generates normally while the shadow model wraps the user query with a direct or intent-oriented detection prompt, identifies harmful portions or intentions, and triggers a checkpoint that either releases the cached target response or replaces it with a refusal. This design is highly deployment-oriented because it does not require access to the target model's hidden states or decoding internals, and can therefore protect both open-source

and closed-source LLMs. However, its defensive decision is made by an auxiliary semantic detector over the prompt or inferred intent, rather than by monitoring the target model's token-level generation trajectory.

JEDI differs from SelfDefend in both the monitored signal and the intervention mechanism. SelfDefend reduces user-visible latency by parallelizing prompt/intent-level adjudication with normal generation, whereas JEDI embeds a lightweight monitor directly into the decoding loop. At each generation step, JEDI projects the target model's hidden state onto a Condition Vector, accumulates persistent harmful drift with CUSUM, and triggers an Intervention Vector only when the calibrated alarm threshold is exceeded. Consequently, JEDI does not rely on an external LLM to decide whether generation should be interrupted, and its response to risk is not merely to block or release a cached output. Instead, it actively redirects the ongoing hidden-state trajectory toward refusal behavior, coupling in-generation detection with activation-level intervention to address prefix exposure in streaming generation.

SafeProbing [64] is an in-decoding detection-based defense that exploits the model's latent safety-awareness during generation. Rather than inspecting hidden-state trajectories or directly modifying the next-token distribution, SafeProbing appends a virtual disclaimer to the current response prefix and measures the negative log-likelihood of a morally explicit core phrase such as "illegal and unethical". A low disclaimer loss indicates that the model internally regards the generated prefix as unsafe. To sharpen this signal, SafeProbing fine-tunes a safety-awareness-enhanced model with lightweight LoRA training and uses it as a probe while the base model continues normal decoding. At sampled decoding checkpoints, if the disclaimer loss falls below a predefined threshold, generation is halted and a fixed refusal string is returned; otherwise decoding proceeds unchanged.

JEDI differs from SafeProbing in both the monitored signal and the defensive action. SafeProbing treats safety as a likelihood signal over an appended textual probe and therefore follows a checkpointed detect-and-refuse paradigm, which may require repeated probe forward passes and, in streaming deployments, buffering or sufficiently frequent checks to avoid prefix exposure before a checkpoint is evaluated. JEDI instead monitors the protected model's hidden states directly: each token-level state is projected onto a contrastively extracted Condition Vector, persistent harmful drift is accumulated by CUSUM, and intervention is triggered only when the calibrated Alarm Threshold is exceeded. Once triggered, JEDI does not replace the response with a fixed refusal string; it injects an orthogonalized Intervention Vector with Dynamic Intervention Strength to steer the ongoing activation trajectory toward refusal behavior. Thus, SafeProbing surfaces latent safety-awareness through a textual loss probe, whereas JEDI separates *when to intervene* from *how to intervene* through activation-level detection and steering without weight updates, auxiliary adjudication, or re-decoding.

A.3 Conditional Activation Steering and Input-adaptive Defenses

CAST [18] conditions activation steering on a representation-level signal. In our evaluated configuration, it decides once during prompt prefill and applies fixed-strength steering. JBShield [58] detects and manipulates toxic/jailbreak concepts activated by the input. AdaSteer [62] uses Rejection and Harmfulness Directions with input-dependent coefficients. JEDI's two contrasts are conceptually related to these directions, up to sign and extraction details; the contribution does not lie in difference-in-means extraction or activation addition alone. JEDI recomputes its risk statistic and intervention magnitude from the evolving output trajectory, separates the Condition Vector from the Intervention Vector, and calibrates alarm parameters under a benign-FPR constraint. CAST and TrajGuard provide category-level controls; we do not report direct experimental comparisons with JBShield or AdaSteer.

B Local and Conditional Theoretical Interpretation

The analysis concerns locally exploitable directions on the calibration distribution. It assumes neither global linear separability nor Gaussian hidden states, and the detection bounds below are conditional on assumptions about the learned risk process.

Condition Vector. Let $d = \mu_{JC} - \mu_{BC}$ be the mean shift in the content-window representations, and let $\Sigma_B \succ 0$ be the benign covariance. Among linear directions, standardized squared mean

separation is

$$\max_{w \neq 0} \frac{(w^\top d)^2}{w^\top \Sigma_B w}. \tag{8}$$

Writing $u = \Sigma_B^{1/2} w$ and applying Cauchy–Schwarz gives $(u^\top \Sigma_B^{-1/2} d)^2 / \|u\|^2 \leq d^\top \Sigma_B^{-1} d$, with equality for $w \propto \Sigma_B^{-1} d$. Choosing the sign toward JC, $c = Wd$ and $W^\top W = \Sigma_B^{-1}$ yield

$$s(h) = \frac{(h - \mu_{BC})^\top \Sigma_B^{-1} d}{\sqrt{d^\top \Sigma_B^{-1} d}}. \tag{9}$$

This is optimal in the standardized linear mean-separation class. It is not a Bayes-optimal classification result or a guarantee for arbitrary distributions. Whitening standardizes covariance, not distributional shape, and Eq. 3 normalizes only the direction.

Intervention Vector. In whitened prefix-window coordinates, let z_{JC} and z_{JR} denote matched compliance and refusal representations. For a constant displacement v,

$$\mathbb{E}\|z_{JC} + v - z_{JR}\|^2 = \|v - (\mu_{JR} - \mu_{JC})\|^2 + C, \tag{10}$$

where C is independent of v. Thus $v_{\text{raw}} = \mu_{JR} - \mu_{JC}$ minimizes the objective. Restricting $v^\top c = 0$ projects this minimizer onto the condition-orthogonal subspace:

$$v_\perp = v_{\text{raw}} - c \frac{c^\top v_{\text{raw}}}{\|c\|^2}. \tag{11}$$

This least-squares interpretation identifies a functional displacement fitted to the extraction distribution, not a unique intrinsic or causal refusal mechanism. Pairing the same harmful prompt and attack context with opposite stances reduces prompt/attack confounding; the synthetic prefixes can still affect the fitted direction. Natural refusals require behavior labels and matched segmentation and are not a drop-in substitute.

Sequential Detection. Let $X_t = r_t - \hat{\mu} - k$, $A_0 = 0$, $A_t = \max(0, A_{t-1} + X_t)$, and $\tau_\alpha = \inf\{t : A_t \geq \alpha\}$. With respect to the history $\mathcal{F}_{t-1}$, assume centered increments satisfy

$$\mathbb{E}[\exp(\lambda(X_t - \mathbb{E}[X_t \mid \mathcal{F}_{t-1}])) \mid \mathcal{F}_{t-1}] \leq \exp(\lambda^2 \sigma^2 / 2) \tag{12}$$

for all real λ. If benign conditional drift is at most $-\kappa < 0$, while post-change harmful drift is at least $\Delta > 0$, then

$$\Pr_0(\tau_\alpha \leq T) \leq \min\{1, T \exp(-2\kappa\alpha/\sigma^2)\}, \tag{13}$$

$$\Pr_\nu(\tau_\alpha - \nu > n \mid \tau_\alpha \geq \nu) \leq \exp\left(-\frac{(n\Delta - \alpha)^2}{2n\sigma^2}\right), \quad n\Delta > \alpha. \tag{14}$$

For the first bound, $\lambda = 2\kappa/\sigma^2$ makes the exponential of each partial sum starting at a fixed time a nonnegative supermartingale. A maximal inequality bounds crossing probability by $\exp(-\lambda\alpha)$; a union bound over at most T starting times gives the result. For the second bound, no alarm through the first n post-change increments implies their sum is below α, since A_t is at least that sum. Conditional sub-Gaussian concentration around conditional drift of at least $n\Delta$ gives the displayed tail bound.

The assumptions formalize a false-alarm/delay trade-off but are not established for every autoregressive trajectory. Small or nonpersistent harmful projected drift, benign distribution shift, or violations of the concentration assumptions can weaken detection. Scores are learned projections rather than log-likelihood ratios, so these statements do not establish classical CUSUM minimax optimality. The 95th-percentile rectification threshold also does not imply a 5% sequence-level alarm rate; (k, α) are calibrated separately.

C Condition Vector Separability

To quantify whether the condition vector provides a usable linear axis for detection, we measure how well harmful compliance samples (JC) and benign compliance samples (BC) separate along c_l. For

each sample, we apply the whitening transformation $\Phi(\cdot)$ estimated from the benign baseline and compute the projection length of the hidden states along the normalized condition vector direction. Higher projection scores indicate stronger alignment with the harmful compliance direction. We evaluate this score using AUROC, AUPRC with JC as the positive class, Cohen's d between the two score distributions, and the Fisher Ratio between their means and variances.

Table 5: Separability of harmful and benign samples along the condition vector direction.

Model	best_layer	AUROC	AUPRC	Cohen's d	Fisher Ratio
Vicuna 7B	14	1.000	1.000	3.70	5.55
Vicuna 13B	19	0.999	0.999	3.28	4.36
Mistral 7B	14	0.999	0.999	4.49	8.66
Qwen2.5 7B	20	1.000	1.000	5.15	10.59
Qwen2.5 14B	29	1.000	1.000	7.29	14.28
Qwen3-8B	20	1.000	1.000	5.28	7.97

Table 5 shows near-perfect AUROC and AUPRC across all target models, while the effect-size metrics remain large even for the weakest case (Vicuna 13B, Cohen's $d = 3.28$, Fisher Ratio $= 4.36$). These results indicate that the selected layers expose a strong one-dimensional harmful/benign separation along c_l, supporting the use of this direction as the online risk-scoring axis.

C.1 Calibration, Evaluation Isolation, and Classifier Audits

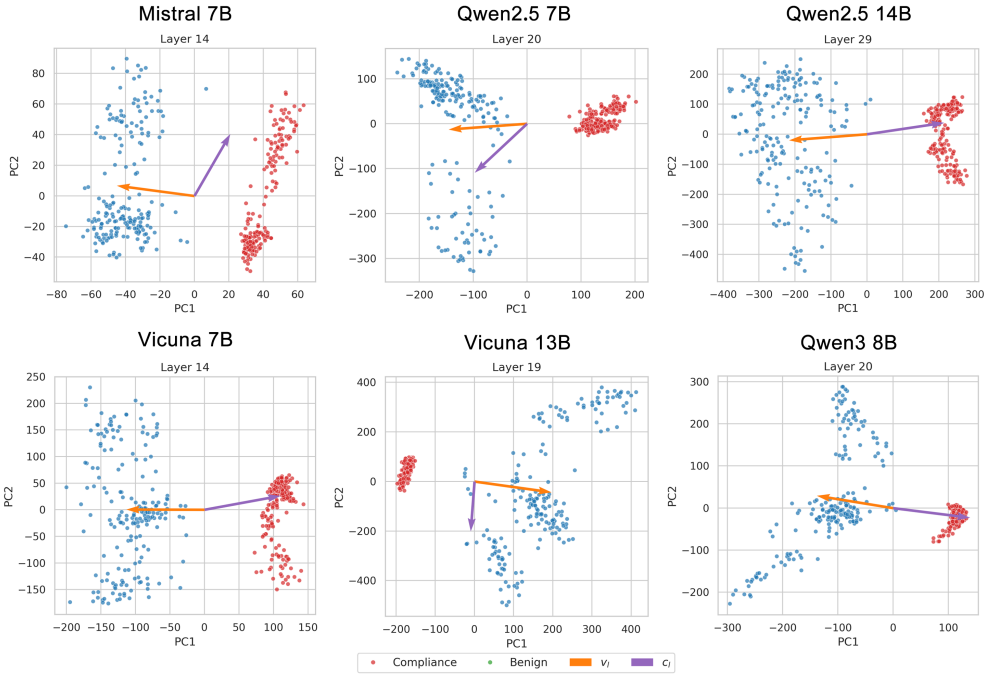
All prompt splits precede prefix expansion. Whitening/vector extraction, layer selection, and calibration of $\theta, \hat{\mu}, k, \alpha$ jointly use D_{cal} (100 harmful HarmBench and 100 benign JustEval prompts). D_{sep} independently samples another 100 harmful and 100 benign prompts, with zero overlap, and only computes the separability scores in Table 5 with the layer, $\hat{b}$, W, and c_l frozen. This is out-of-sample prompt evaluation of frozen offline decisions. The prefix bank is deliberately shared: JC and BC use identical compliance prefixes to balance prefix identity. Consequently, this test concerns unseen harmful/benign content under fixed prefixes, not unseen templates.

Final attack evaluation uses 1,100 PandaGuard prompts, 100 per attack method. GCG, PAIR, AutoDAN, and TAP overlap calibration only in attack-generation method; their evaluation prompts are non-overlapping samples from a different dataset. The other seven attack methods are absent from calibration. XSTest and OR-Bench are evaluation-only, and AlpacaEval-source items are excluded from benign calibration. A normalized exact-overlap audit (Unicode, case, and whitespace normalization) found zero matches between the selected JustEval prompts or compliance/refusal prefixes and XSTest/OR-Bench.

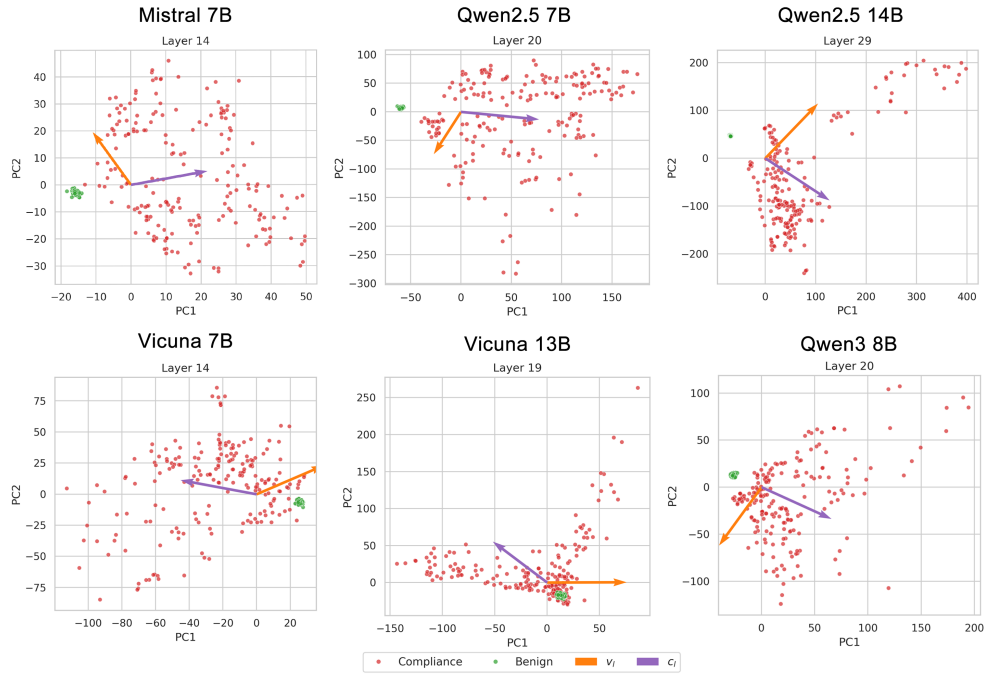
Prompt-level Uncertainty. The standard deviations in the disaggregated result tables measure variation across models or attacks, not uncertainty over sampled prompts. For a complete-cluster separability analysis, we average each prompt's 100 variants and bootstrap 100 harmful and 100 benign prompt scores with 10,000 replicates. AUROC and AUPRC are 1.000 for all six models, with reported bootstrap intervals [1.000, 1.000]. The weakest effects occur on Vicuna-13B: Cohen's $d = 3.072$ with interval [2.772, 3.466], and Fisher Ratio $= 4.719$ with interval [3.842, 6.008]. These prompt-aggregated, complete-cluster estimates complement Table 5's filtered instance-level estimates; they condition on the frozen offline quantities and prefix bank. They are not confidence intervals for end-to-end DSR.

Classifier–Human Agreement. We randomly sampled 1,000 method–model–attack outputs and partitioned them into ten disjoint sets of 100. Ten master's students in software engineering or computer science each judged one set for harmful content, given only the prompt and response. HarmBench agreed with these judgments on 972/1,000 outputs (97.2%). This is classifier–human agreement on singly assigned items, not inter-annotator agreement. A separate 100-case visible-prefix audit is described in Appendix D.2.

Seen and Unseen Attack Families. Equal test sizes permit direct attack-family averaging. JEDI obtains 89.7% mean DSR on the four seen methods and 99.1% on the seven unseen methods; the



(a) PCA of Prefix Sequence



(b) PCA of Content Sequence

Figure 3: PCA visualization of prefix and content sequence activations. Panel (a) visualizes the prefix sequence S_P, while panel (b) visualizes the content sequence S_C; arrows denote the projected intervention vector v_l and condition vector c_l in the two-dimensional PCA plane.

corresponding undefended rates are 60.6% and 48.9%. Absolute improvements are 29.1 and 50.2 percentage points. This pattern does not show a systematic seen-family advantage, but performance alone cannot prove isolation: that conclusion rests on the sample-disjoint construction and overlap audit.

Each target model has one pooled Condition Vector and shared detector parameters, without attack labels at inference. In the extraction space, a pooled shift can be written as $c \propto \sum_a w_a \Delta_a$; detectability depends on the test trajectory's projection onto c and its persistence. Membership in the calibration mixture does not guarantee strong alignment for independently generated instances. End-to-end DSR also depends on whether the shared Intervention Vector redirects the trajectory. These distinctions provide a possible explanation for the group pattern, not a causal diagnosis of individual failures.

C.2 Held-out Prefix-bank Robustness

We use grouped five-fold experiments on Qwen2.5-7B-Instruct and Vicuna-7B-v1.5. The 100 compliance and 100 refusal prefix entries are normalized and grouped so duplicates cannot cross folds. Each fold holds out 20 entries per bank and re-estimates the Intervention Vector using the remaining 80. Directional agreement is evaluated on both held-out prefixes and held-out calibration prompts. Whitening, the Condition Vector, selected layer, CUSUM parameters, and intervention rule remain fixed to isolate sensitivity to Intervention-Vector extraction.

Table 6: Prefix-bank resampling. Values are mean $\pm$ sample standard deviation, followed by the observed range; ranges are not confidence intervals.

Metric	Qwen2.5-7B-Instruct	Vicuna-7B-v1.5
Train/joint held-out effect cosine	0.9500 ± 0.0075	0.9391 ± 0.0083
Observed range	[0.9417, 0.9586]	[0.9246, 0.9451]
Fold-pair effect cosine	0.9964 ± 0.0008	0.9953 ± 0.0008
Observed range	[0.9953, 0.9977]	[0.9935, 0.9967]
DSR (%)	93.25 ± 0.17	98.35 ± 0.34
Observed range (%)	[93.00, 93.45]	[97.91, 98.73]

Joint held-out agreement and DSR use five folds; pairwise agreement uses the ten pairs of fold-trained intervention effects. Each fold is evaluated on the same fixed 1,100-response, eleven-attack test set. High directional agreement and stable DSR support robustness to the tested grouped resampling across the two evaluated model families. The experiment does not re-fit the complete offline pipeline in each fold and does not evaluate extraction from naturally occurring refusals.

Dependence on Initial Alignment. Undefended DSR spans 34.1%–69.8% across the six instruction-tuned models, while JEDI reaches 93.6%–99.4%. On SFT-only Vicuna, DSR rises from 50.8% to 98.2% (7B) and from 41.4% to 93.6% (13B); on Mistral-7B-Instruct-v0.2 it rises from 34.1% to 94.0%. Undefended DSR is only a proxy for initial jailbreak robustness. Prefix-conditioned extraction does not require spontaneous refusals, but still requires a distinguishable and steerable refusal/compliance contrast. These observations do not establish alignment-independent behavior on purely pretrained checkpoints.

D Complete Experimental Results

Experimental Platform. We implemented the prototype of JEDI within a Python 3.10 environment, utilizing PyTorch 2.5.1 and Transformers 4.56.2 as the underlying deep learning framework. All experimental evaluations and inference tasks were conducted on a single NVIDIA H100 GPU.

Generation Parameters. The maximum generation length (max_new_tokens) was configured specifically for each evaluation scenario: 256 tokens for jailbreak attack simulations to capture immediate harmful outputs, 2,048 tokens for the AlpacaEval benchmark to allow for comprehensive responses, and 512 tokens for the XSTest and OR-Bench benchmarks to assess refusal compliance.

D.1 Online Baseline Protocols and Near-matched Comparisons

Table 7: Safety–utility–latency comparison, averaged over six models. DSR, AlpacaEval Win Rate (Win), XSTest compliance (CR), and OR-Bench FPR are percentages. Latencies are seconds per response on AlpacaEval. Qwen3Guard TTFT measures first visible release.

Defense	DSR ↑	Win ↑	CR ↑	FPR ↓	TTFT ↓	Time ↓
Undefended	53.10	84.21	95.10	8.80	0.021	7.98
JEDI	95.70	77.18	91.10	12.50	0.024	8.89
Gradient Cuff	75.80	71.21	69.60	14.30	3.650	11.74
SmoothLLM	67.40	59.51	80.20	10.80	11.909	13.97
SafeDecoding	74.20	77.54	78.30	31.00	0.057	8.27
Activation Probe	90.40	66.15	85.20	13.70	0.025	8.49
BackTranslation	65.10	79.89	88.20	13.40	50.013	50.01
CAST	71.68	60.91	92.70	16.88	0.024	9.36
TrajGuard	75.82	63.17	90.00	30.65	0.021	7.74
Qwen3Guard-Gen-16	98.48	60.94	84.26	15.50	0.447	18.52
Qwen3Guard-Gen-32	98.41	60.88	85.28	14.67	0.450	13.43

Periodic Buffered Guard. Qwen3Guard-Gen-8B checks the original prompt together with the cumulative assistant prefix every 16 or 32 generated tokens. Each block is buffered while it is classified. Safe blocks are released; an unsafe decision suppresses the current block, terminates generation, and appends a fixed refusal to any previously released prefix. The final incomplete block is also checked. HarmBench judges the resulting user-visible response. First-release latency includes chunk formation, guard queueing, and inference. This generative guard is distinct from Qwen3Guard-Stream-8B, the token-level evaluator used in the visible-prefix audit.

Conditional Steering and Trajectory Monitoring. CAST uses the official `activation-steering` implementation with `pca_pairwise` extraction and disjoint vector-training and condition-calibration splits. It decides once during prefill and applies fixed-strength steering. It does not include JEDI's whitening, CUSUM, Gram–Schmidt decoupling, token-level risk updates, or dynamic intervention. TrajGuard is adapted to the same cached-decoding stack, monitors smoothed trajectories, and invokes semantic adjudication when risk persists. These configurations give category-level comparisons for conditional steering and trajectory detection.

Table 8: First alarm/check position (median/p95, generated tokens) in the reported online baseline comparison. Prefill decisions, periodic checks, and token-level alarms have different semantics; positions alone do not measure visible harmful exposure.

Defense	Median / p95
JEDI	1 / 51
Qwen3Guard-Gen-16	16 / 96
Qwen3Guard-Gen-32	32 / 128
CAST	0 / 0 (prefill)
TrajGuard	43 / 221

Near-matched Safety. The existing per-model results provide two comparisons at similar DSR. On Vicuna-13B, JEDI versus Activation Probe yields DSR 93.6/93.5, Win Rate 55.04/44.18, XSTest CR 87.6/71.6, and OR-Bench FPR 11.8/11.3 (all percentages). On Qwen2.5-7B, the corresponding pairs are 94.0/93.8, 89.07/73.98, 94.8/92.4, and 8.5/9.9. These points characterize utility at similar safety, but do not replace method-specific operating curves. Figure 2 varies only JEDI's settings and is a within-method sensitivity analysis. No comparison is reported for SafeProbing, Trust The Typical, SelfDefend, or concurrent batched-serving throughput.

D.2 Visible-prefix Audit Protocol and Additional Metrics

The audit uses 3,975 matched model–attack cases whose undefended visible stream contains a Qwen3Guard-Stream-8B `Unsafe` onset. The evaluator processes each assistant prefix in its own tokenizer. Decoded-text alignment maps alarm, intervention, and release boundaries into the same coordinates. We distinguish an onset in the user-visible stream from a detection in a buffered, unreleased chunk. Table 9 reports the raw-onset signal separately from final-response HarmBench harm and their intersection.

Table 9: Visible-prefix audit on the same 3,975 selected cases (%, ↓). Raw onset is a detector-positive signal. Its intersection with final harm is evaluator-consensus leakage, a non-exhaustive measure of prefix exposure.

Defense	Raw onset	Final harmful	Intersection
Undefended	100.00	69.79	69.79
JEDI	26.61	6.30	4.86
CAST	61.97	43.32	39.87
TrajGuard	55.46	39.18	34.62
Qwen3Guard-Gen-16	6.10	2.74	0.40
Qwen3Guard-Gen-32	5.46	2.65	0.45

Zero Harmful Leakage Rate (ZHLR) is the fraction of cases with no evaluator-consensus leakage (no intersection). Conditional Harmful Prefix Leakage (cHPL) is the visible span from the first unsafe onset to the end of the evaluation view, conditional on consensus leakage; it is not a count of tokens individually verified as harmful. Pre-Intervention Harmful Prefix Length (PI-HPL) measures the Qwen-token span from the first unsafe onset to the effective intervention boundary. Alarm-Before-Onset Rate (ABOR) compares the alarm with the matched *undefended* onset and counts no alarm as failure. ABOR does not certify that a defended stream has no exposure.

Table 10: Additional prefix-audit metrics. ZHLR and ABOR are percentages; cHPL and PI-HPL report median/p95 Qwen-token spans. Consensus-zero leakage is not absence of all unsafe prefixes.

Defense	ZHLR ↑	cHPL ↓	PI-HPL ↓	ABOR ↑
Undefended	30.21	198 / 251	—	—
JEDI	95.14	196 / 250	1 / 54.9	86.36
CAST	60.13	186 / 248	0 / 0	93.56
TrajGuard	65.38	202 / 249	48 / 196.85	17.74
Qwen3Guard-Gen-16	99.60	187.5 / 242.75	113 / 171.2	76.28
Qwen3Guard-Gen-32	99.55	186 / 242.45	96.5 / 169.25	76.98

A manual audit agreed with 98 of 100 sampled cases; both discrepancies were conservative false positives. Qwen3Guard conditions its decisions on preceding conversation, including the harmful request, and sometimes flags generic compliance markers such as "Sure, here is..." even when no harmful payload follows. Requiring HarmBench agreement improves precision, but can exclude an unsafe prefix later redirected into a safe refusal. Accordingly, the intersection is a high-precision, non-exhaustive measure; raw onset is a more sensitive detector-positive signal, not ground truth. The 21.75-point raw-onset/intersection gap for JEDI is a mixture of possible transient harmful prefixes and Qwen false positives, not a verified leakage rate.

CAST's early prefill triggering and high ABOR coexist with substantially lower ZHLR than JEDI, illustrating that triggering early does not alone ensure successful correction. Buffered Qwen3Guard suppresses more visible exposure, while JEDI trades some residual leakage for lower first-release latency and higher utility. Similar conditional cHPL across defenses shows why low leakage frequency should not be interpreted as short residual harmful spans. The alarm-position distribution in Figure 4 supplies complementary responsiveness evidence rather than a direct measurement of harmful onset.

D.3 Detailed DSR results on various target LLMs

Table 11: Detailed DSR (%, ↑) results on **Vicuna 7B**. Bold and underline indicate the best and second-best defense performance, respectively.

Defences	GCG	PAIR	AutoDAN	TAP	RS	AIM	BtDAN	DevMode	ReneLLM	DI	GPTFuz	Avg.	Std.
Undefend	73	53	55	65	12	15	26	50	78	84	48	50.8	24.4
JEDI (ours)	94	93	**99**	96	**100**	**100**	**100**	**100**	**100**	98	**100**	**98.2**	**2.6**
Gradient cuff	99	68	93	76	89	**100**	77	**100**	79	85	75	85.5	11.4
SafeDecoding	97	89	88	86	**100**	91	56	88	78	**100**	97	88.2	12.6
Smoothllm	93	63	67	74	48	48	55	65	95	87	65	69.1	16.6
BackTranslation	89	74	80	79	41	66	74	96	85	97	66	77.0	15.9
Activation Probe	**100**	**98**	98	**98**	**100**	74	76	92	92	**100**	84	92.0	9.7

Table 12: Detailed DSR (%, ↑) results on **Vicuna 13B**. Bold and underline indicate the best and second-best defense performance, respectively.

Defences	GCG	PAIR	AutoDAN	TAP	RS	AIM	BtDAN	DevMode	ReneLLM	DI	GPTFuz	Avg.	Std.
Undefend	17	54	47	71	6	6	27	35	64	84	44	41.4	26.0
JEDI (ours)	58	92	96	89	99	**100**	**100**	**100**	97	**100**	**99**	**93.6**	12.4
Gradient cuff	96	79	85	83	96	74	99	95	72	89	82	86.4	9.3
SafeDecoding	**100**	**95**	**100**	**95**	**100**	**100**	56	84	91	**100**	**99**	92.7	13.2
Smoothllm	93	58	72	75	33	15	44	73	87	81	76	64.3	24.2
BackTranslation	22	65	51	72	25	20	89	99	68	84	57	59.3	27.4
Activation Probe	84	78	**100**	94	98	98	99	98	**100**	**100**	79	93.5	**8.7**

Table 13: Detailed DSR (%, ↑) results on **Mistral 7B**. Bold and underline indicate the best and second-best defense performance, respectively.

Defences	GCG	PAIR	AutoDAN	TAP	RS	AIM	BtDAN	DevMode	ReneLLM	DI	GPTFuz	Avg.	Std.
Undefend	39	44	33	63	2	5	19	21	59	62	28	34.1	21.6
JEDI (ours)	96	77	**86**	88	**100**	**100**	**100**	**100**	**96**	**91**	**100**	**94.0**	**7.7**
Gradient cuff	**98**	70	61	78	4	8	99	92	72	69	52	63.9	32.2
SafeDecoding	52	50	37	72	6	8	20	51	64	86	45	44.6	25.4
Smoothllm	62	48	52	66	26	20	29	49	86	86	57	52.8	22.1
BackTranslation	44	44	40	64	12	12	82	42	60	75	51	47.8	22.4
Activation Probe	76	**80**	34	**99**	97	75	68	72	63	69	84	74.3	17.6

Table 14: Detailed DSR (%, ↑) results on **Qwen2.5 7B**. Bold and underline indicate the best and second-best defense performance, respectively.

Defences	GCG	PAIR	AutoDAN	TAP	RS	AIM	BtDAN	DevMode	ReneLLM	DI	GPTFuz	Avg.	Std.
Undefend	80	62	41	72	4	13	58	68	64	67	59	53.5	24.3
JEDI (ours)	95	75	84	87	**100**	**100**	**100**	**100**	**100**	93	**100**	**94.0**	8.5
Gradient cuff	91	62	36	74	2	57	**100**	97	64	73	80	66.9	28.5
SafeDecoding	90	70	31	74	27	63	38	50	65	76	79	60.3	20.9
Smoothllm	82	62	53	74	19	29	82	35	85	78	63	60.2	23.3
BackTranslation	83	62	34	74	5	27	99	93	64	74	60	61.4	28.8
Activation Probe	**97**	**96**	**99**	**91**	**100**	88	87	97	78	**100**	99	93.8	**7.0**

Table 15: Detailed DSR (%, ↑) results on **Qwen2.5 14B**. Bold and underline indicate the best and second-best defense performance, respectively.

Defences	GCG	PAIR	AutoDAN	TAP	RS	AIM	BtDAN	DevMode	ReneLLM	DI	GPTFuz	Avg.	Std.
Undefend	96	68	43	82	3	70	59	100	68	84	89	69.3	27.6
JEDI (ours)	**100**	**98**	**100**	95	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**99.4**	**1.6**
Gradient cuff	96	71	44	81	3	99	**100**	**100**	68	83	97	76.5	30.1
SafeDecoding	97	67	48	86	11	97	59	**100**	70	83	90	73.5	26.7
Smoothllm	96	72	57	79	11	84	99	**100**	72	98	98	78.7	26.6
BackTranslation	96	68	45	79	7	75	81	**100**	68	83	89	71.9	26.3
Activation Probe	**100**	96	**100**	**100**	**100**	94	81	**100**	**100**	**100**	93	96.7	5.9

Table 16: Detailed DSR (%, ↑) results on **Qwen3-8B**. Bold and underline indicate the best and second-best defense performance, respectively.

Defences	GCG	PAIR	AutoDAN	TAP	RS	AIM	BtDAN	DevMode	ReneLLM	DI	GPTFuz	Avg.	Std.
Undefend	88	72	56	80	3	54	99	100	63	89	64	69.8	27.5
JEDI (ours)	<u>96</u>	<u>86</u>	<u>83</u>	**90**	<u>98</u>	**100**	**100**	**100**	**96**	**94**	**100**	**94.8**	**6.0**
Gradient cuff	91	80	56	85	4	79	**100**	**100**	64	<u>91</u>	84	75.8	27.4
SafeDecoding	**98**	**88**	67	85	**100**	**100**	<u>97</u>	**100**	65	48	97	85.9	18.0
SmoothLLM	92	70	67	81	38	84	**100**	**100**	84	81	78	79.5	17.4
BackTranslation	88	72	57	<u>86</u>	6	63	**100**	**100**	63	<u>91</u>	81	73.4	26.8
Activation Probe	95	73	**88**	83	97	<u>98</u>	**100**	**100**	<u>95</u>	89	<u>98</u>	<u>92.4</u>	<u>8.4</u>

Failure analysis on Vicuna-13B under GCG. Table 12 shows that JEDI obtains a relatively low DSR of 58% on the Vicuna-13B + GCG setting, whereas SafeDecoding reaches 100% on this specific model-attack pair. We emphasize that this is a localized failure case rather than a general degradation on Vicuna-13B: across all 11 attacks, JEDI still achieves an average DSR of 93.6% on Vicuna-13B, slightly higher than SafeDecoding's 92.7%. Nevertheless, this result reveals an important limitation of the current single-direction calibration used by JEDI.

A likely explanation is that JEDI estimates a shared Condition Vector from a mixed calibration distribution, rather than learning an attack-specific harmful direction for each jailbreak family. In the offline extraction stage, the Condition Vector is computed as the mean-shift direction between jailbreak-compliance samples and benign-compliance samples. Therefore, when the calibration set contains multiple attack families, the extracted direction tends to capture the dominant cross-attack harmful drift. In our calibration set, GCG accounts for only a subset of the jailbreak prompts, while the remaining samples come from other attack families. As a result, if the GCG samples on Vicuna-13B occupy a partially distinct latent region, their attack-specific drift may be under-represented in the shared Condition Vector.

Figure 3 provides an intuitive explanation of this phenomenon. In the PCA projection, several Vicuna-13B GCG samples are located closer to benign-compliance activations than other jailbreak samples. This indicates that these GCG cases have weaker alignment with the dominant harmful direction captured by the Condition Vector. During online defense, their token-level projection scores may therefore remain closer to the benign range, causing the CUSUM statistic to accumulate risk more slowly or insufficiently before harmful content is generated. This is consistent with JEDI's sequential detection mechanism: the defense is triggered only when persistent evidence along the Condition Vector exceeds the calibrated alarm threshold.

This failure mode is also related to the nature of GCG. GCG optimizes adversarial suffixes to elicit target affirmative continuations, such as compliant openings. On some model-attack pairs, these optimized continuations can resemble benign instruction-following in the early generation stage, especially before clearly harmful semantics emerge. Such behavior makes GCG harder for a shared semantic-drift direction to detect under a strict false-positive constraint. In contrast, SafeDecoding directly modifies the decoding process toward safety-aware token choices, which can be particularly effective when the attack relies on forcing an affirmative continuation.

This case highlights a limitation of the current JEDI design: a single Condition Vector may not fully capture the heterogeneity across attack families, especially for hard optimization-based attacks whose latent trajectories deviate from the dominant cross-attack direction. JEDI is not inherently restricted to the calibration size or composition used in the main experiments; both are configurable in our implementation. Increasing the number and diversity of calibration samples, using a more balanced calibration distribution, or extending JEDI to maintain multiple attack-aware condition directions may improve robustness on such hard model-attack pairs, at the cost of additional offline calibration complexity.

D.4 Details of Utility & Efficiency

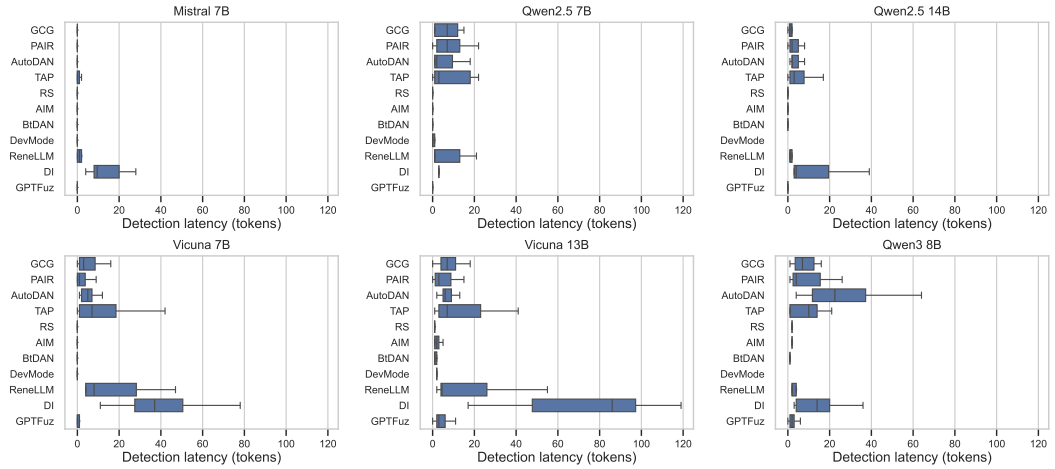


Figure 4: Detection delay of JEDI against various attack methods.

Figure 4 illustrates the distribution of detection latency for JEDI across various attack vectors. The horizontal axis indicates the number of tokens generated before the defense mechanism triggers an intervention, while the vertical axis lists the 11 specific jailbreak attack methods evaluated.

As shown in the figure, JEDI can intercept the majority of attacks—including optimization-based methods such as GCG and PAIR—within the first 20 tokens. This rapid response is attributed to the nature of these attacks, which typically prompt the LLM to generate harmful content immediately after a brief, compliant prefix (e.g., "Sure, here is..."). Consequently, the internal risk accumulation (A_t) rises sharply, allowing JEDI to breach the alarm threshold (α) and trigger intervention early in these generations. Alarm position alone does not locate the first harmful span.

In contrast, template-based attacks such as ReneLLM and DeepInception (DI) exhibit notably longer detection latencies. This delay occurs because these methods obfuscate harmful intent by nesting it behind benign content, such as LaTeX document templates (common in ReneLLM) or fictional narrative backgrounds (characteristic of DeepInception). During the generation of these initial benign tokens, the semantic drift remains within the safety boundary defined by the Condition Vector (c_l), and JEDI correctly withholds intervention. The defense is triggered only when the generation transitions from the benign template to the actual harmful payload, consistent with selective triggering on persistent projected risk. The direct visible-stream audit in Appendix D.2 separately evaluates prefix exposure.

Table 17: Average generation time (s. ↓) of JEDI and baselines on AlpacaEval. Bold and underline indicate the best and second-best defense performance, respectively.

Defences	Vicuna 7B	Vicuna 13B	Mistral 7B	Qwen2.5 7B	Qwen2.5 14B	Qwen3-8B	Avg.
Undefend	4.23	5.37	7.95	6.80	9.51	14.02	7.98
JEDI (ours)	<u>4.25</u>	<u>5.37</u>	8.23	8.44	10.50	16.57	8.89
Gradient cuff	7.40	9.17	11.55	9.11	14.19	19.01	11.74
SafeDecoding	4.45	**5.32**	**7.86**	**7.58**	**9.90**	**14.50**	**8.27**
SmoothLLM	10.60	12.29	13.91	10.20	14.56	22.25	13.97
BackTranslation	44.49	48.65	51.67	47.79	51.99	55.49	50.01
Activation Probe	**4.24**	<u>5.37</u>	<u>8.09</u>	<u>7.89</u>	<u>10.21</u>	<u>15.11</u>	<u>8.49</u>

To quantify the computational overhead introduced by the defense mechanisms, we report the average total generation time per response on the AlpacaEval benchmark in Table 17.

The results show that JEDI records an average generation time of 8.89 seconds across the six target LLMs. This corresponds to an increase of 0.91 seconds relative to the undefended LLMs (7.98 seconds). Among the evaluated defenses, SafeDecoding and Activation Probe achieve lower total generation time, averaging 8.27 seconds and 8.49 seconds, respectively. The sources of latency differ across these defenses: SafeDecoding requires loading an auxiliary expert model, which imposes a larger Time To First Token (TTFT), Activation Probe only performs lightweight probe evaluation during decoding, and JEDI incurs a cumulative overhead due to the step-wise computation of CUSUM statistics and triggered intervention. Nevertheless, JEDI remains significantly faster than heavier methods such as Gradient Cuff (11.74 seconds), SmoothLLM (13.97 seconds), and BackTranslation (50.01 seconds).

D.5 Details of Ablation Study

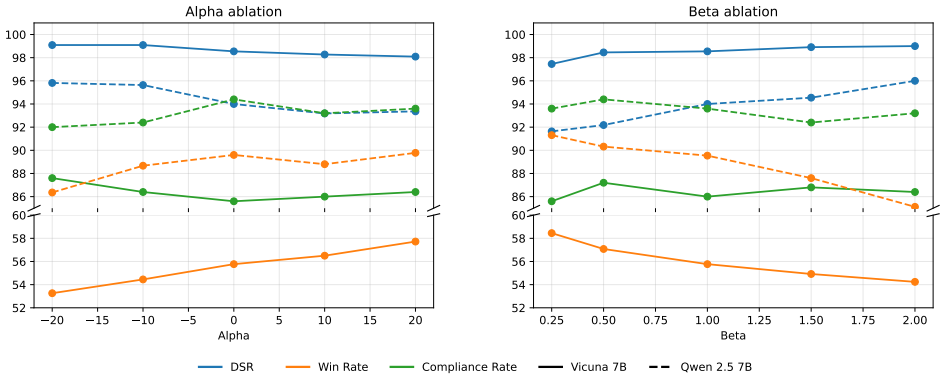


Figure 5: Ablation experiment details.

Controlled Component Ablations. Each variant uses the same frozen extraction artifacts, prompts, generation settings, and evaluators as full JEDI. The instantaneous variant replaces A_t with single-step risk r_t for both the alarm and risk-scaled intervention strength. Its threshold is calibrated on the same benign sequences to match JEDI's sequence-level benign alarm FPR: 0.03% versus 0.03% on Vicuna, and 5.00% versus 5.00% on Qwen2.5. This calibration target differs from refusal FPR on OR-Bench. The no-orthogonalization variant reconstructs the raw Intervention Vector from the same samples and omits its projection subtraction. The fixed-strength variant uses $\beta_t = \beta_{\text{base}} = 1$ after alarm with detector and orthogonalized vector unchanged.

Table 18: Full component-ablation metrics. DSR, XSTest CR, OR-Bench FPR, and safety trigger rates are percentages. Alarm positions are zero-based generated-token steps, reported as median/p95.

Variant	DSR ↑	CR ↑	FPR ↓	Trigger	Alarm med/p95
Vicuna-7B-v1.5					
Full JEDI	98.20	86.40	13.30	97.91	0.0 / 53.0
Instantaneous	76.64	83.60	11.30	96.91	0.0 / 58.8
No orthogonalization	97.55	76.80	16.20	97.91	0.0 / 53.0
Fixed $\beta = 1$	59.91	83.20	11.80	97.91	0.0 / 53.0
Qwen2.5-7B-Instruct					
Full JEDI	94.00	94.80	8.50	99.55	1.0 / 22.0
Instantaneous	79.91	90.00	6.70	98.91	1.0 / 48.0
No orthogonalization	92.91	85.20	8.50	99.55	1.0 / 22.0
Fixed $\beta = 1$	66.00	86.00	7.90	99.55	1.0 / 22.0

Full JEDI improves DSR over instantaneous risk by 21.56/14.09 points despite safety-trigger differences of only 1.00/0.64 points. Orthogonalization improves CR by 9.60 points on both models and DSR by 0.65/1.09 points; it reduces OR-FPR by 2.90 points on Vicuna and leaves it unchanged on Qwen2.5. Dynamic strength improves DSR by 38.29/28.00 points and CR by 3.20/8.80 points relative to fixed $\beta = 1$, with identical trigger rates and alarm positions. These comparisons identify contributions under the tested substitutions. The instantaneous variant changes both detection and strength scaling, and the fixed-strength comparison is not a sweep over all possible static strengths. No isolated whitening ablation is reported.

Impact of Alarm Threshold (α). As shown in Figure 5. The threshold α primarily governs the selectivity of the defense mechanism. As illustrated in Figure 2, increasing α raises the barrier for triggering the defense. This results in a higher Compliance Rate on XSTest (reduced over-refusal) and a higher Win Rate on AlpacaEval, as the model is less likely to intervene on benign inputs with borderline risk scores. However, this comes at the cost of a slight reduction in DSR, as subtle jailbreak attempts may fail to accumulate sufficient risk to breach the higher threshold. Conversely, lowering α enhances sensitivity to stealthy attacks, pushing DSR towards saturation, but introduces a penalty in utility due to increased false positives.

Impact of Base Intervention Strength (β_{base}). As shown in Figure 5. The parameter β_{base} controls the baseline scale of the Dynamic Intervention Strength β_t once a risk is detected, with γ fixed at 1.5. The results demonstrate that increasing β_{base} yields a stronger suppression of harmful content, effectively correcting the generation trajectory even when the initial jailbreak prefix is strong. However, excessive steering force (high β_{base}) perturbs the hidden states significantly away from the benign distribution, leading to a degradation in generation fluency and a subsequent drop in the AlpacaEval Win Rate.

Single-Vector Variant. To evaluate whether the two extracted directions are necessary, we test a variant that shares the *Condition Vector* (c_l) across detection and intervention on Qwen3-8B. The online detection pipeline is kept unchanged: risk scores, CUSUM statistics, alarm threshold, and dynamic intervention strength are calibrated as in JEDI. Once an alarm is triggered, however, the intervention step in Eq. 6 injects c_l instead of the *Intervention Vector* (v_l). Thus, the same vector is responsible for both deciding when to intervene and steering how the generation should be corrected.

Table 19: DSR (%, ↑) of hidden-state aggregation and single-vector ablations on **Qwen3-8B**. Bold and underline indicate the best and second-best defense performance, respectively.

Defences	GCG	PAIR	AutoDAN	TAP	RS	AIM	BtDAN	DevMode	ReneLLM	DI	GPTFuz	Avg.	Std.
Undefend	88	72	56	80	3	54	99	100	63	89	64	69.8	27.5
JEDI (mean-token)	<u>96</u>	<u>86</u>	<u>83</u>	<u>90</u>	<u>98</u>	**100**	**100**	**100**	<u>96</u>	94	**100**	<u>94.8</u>	<u>6.0</u>
JEDI (last-token)	**97**	**99**	**94**	**96**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**98.7**	**2.1**
JEDI (single-vector)	84	76	74	84	94	<u>89</u>	**100**	<u>93</u>	81	<u>96</u>	<u>91</u>	87.5	8.3

As shown in Table 19, the choice of hidden-state aggregation affects the safety–utility trade-off. The last-token representation achieves a higher average DSR than mean-token aggregation (98.7% vs. 94.8%), but its OR-Bench FPR increases from 11.7% to 15.5%, indicating stronger safety sensitivity at the cost of more benign false positives. We therefore use mean-token aggregation in the main experiments. Tying detection and intervention also weakens the defense: compared with the dual-vector mean-token design, the average DSR decreases from 94.8% to 87.5%, and the standard deviation increases from 6.0 to 8.3. The single-vector variant is lower on GCG, PAIR, AutoDAN, TAP, RS, AIM, DevMode, ReneLLM, and GPTFuz, remains tied on BtDAN, and improves on DI. These results indicate that the *Condition Vector* captures harmful semantic drift effectively enough for online monitoring, but it is not a consistently reliable steering direction for triggered intervention. For streaming generation, separating c_l and v_l allows JEDI to decouple when to intervene from how to intervene, improving robustness and selectivity rather than adding redundant complexity.

E Implementation Details of Adaptive Attack

To rigorously evaluate the robustness of JEDI, we evaluate a partial-information adaptive attacker that knows the defense mechanism and receives scalar monitoring feedback, without access to all

learned detector internals or unrestricted per-token signals. We implement adaptive variants of three representative attack methods: optimization-based (GCG), red-teaming (PAIR), and fuzzing-based (GPTFuzzer). Under the adaptive setting, the adversary receives the following feedback from JEDI for each query: the binary alarm status ($\mathbb{I}[\text{alarm}]$), the time step of the first alarm (t_{alarm}), and the peak accumulation ratio ($A_{max_ratio} = \max_t A_t/\alpha$).

Unified Scoring Function. We design a comprehensive objective function to guide the black-box optimization (PAIR and GPTFuzzer). This function incentivizes the generation of jailbreak prompts that elicit harmful content while suppressing the CUSUM statistic A_t to evade detection:

$$Score(x) = 10 \cdot \mathbb{I}[\text{succ}] + 3 \cdot \mathbb{I}[\neg\text{alarm}] + \frac{t_{alarm}}{T+1} - A_{max_ratio} \tag{15}$$

where $\mathbb{I}[\text{succ}]$ denotes attack success, and T is the total sequence length. The weights $(10, 3, 1, 1)$ prioritize success and stealth (non-alarm) while rewarding delayed detection (t_{alarm}) and penalizing high risk accumulation (A_{max_ratio}).

Adaptive-GCG. We modify the gradient-based optimization objective to incorporate a safety penalty. The original GCG loss $\mathcal{L}_{attack}$ (negative log-likelihood of the target string) is augmented with a regularization term based on the peak CUSUM ratio:

$$\mathcal{L}_{adaptive} = \mathcal{L}_{attack} + \lambda \cdot \max(0, A_{max_ratio} - 1) \tag{16}$$

We set the adversarial suffix length to 20 tokens and the optimization steps $N = 200$. In each step, we evaluate a batch of $K = 32$ candidates but only perform one full generation for the optimal candidate to align with the query budget of 200. The attack terminates if $\mathbb{I}[\text{succ}] = 1 \wedge \mathbb{I}[\text{alarm}] = 0$, or if the budget is exhausted.

Adaptive-PAIR. This method operates as an iterative prompt rewriting attack driven by an attacker LLM (Llama-3.3-70B-Instruct). We configure the attack with $R = 20$ rounds and $M = 10$ candidates per round. In each iteration, the attacker LLM generates variations of the prompt based on the feedback $Score(x)$. The candidate with the highest score is selected as the parent for the subsequent round. The process stops early upon finding a successful and evasive prompt.

Adaptive-GPTFuzzer. We employ an evolutionary strategy to evolve jailbreak templates. The population is initialized with $P_0 = 20$ seeds. The evolution spans $I = 18$ iterations, generating $\lambda = 10$ offspring per generation via five mutation operators: Generate, Crossover, Expand, Shorten, and Rephrase. Offspring are evaluated using the unified $Score(x)$, and the top $\mu = 20$ candidates are selected for the next generation.

F Extended Limitations and Future Directions

Section 7 summarizes the scope of the empirical evidence. We expand the failure conditions and future directions here. First, highly camouflaged jailbreak prompts and subtle harmful continuations may induce latent trajectories that resemble benign content, making early detection more difficult under strict false-positive constraints. Second, although the triggered intervention mechanism is designed to preserve benign generation, interventions may still degrade utility in edge cases where safety-sensitive but legitimate requests partially align with harmful semantic directions. Future work will strengthen offline extraction & calibration with more diverse and adversarial datasets, including benign safety-critical prompts, and will explore more minimally invasive intervention policies that adapt the Dynamic Intervention Strength to the confidence and persistence of the detected drift.

Another practical limitation is that the current instantiation of JEDI assumes white-box access to the protected LLM's hidden states. This restricts direct deployment on commercial black-box APIs, where model activations and decoding internals are unavailable. A promising direction is a proxy-based deployment architecture. In this setting, a lightweight open-source LLM, such as a 7B or 8B model, is deployed locally or at the edge as a safety proxy between the user and the commercial API. The proxy is equipped with JEDI's in-generation defense mechanism and maintains its own Condition Vector, CUSUM statistics, Alarm Threshold, and Intervention Vector for real-time risk monitoring. Rather than forwarding streamed API tokens directly to the user, the system routes them into a dynamic buffer pool. The proxy continuously evaluates the buffered prefix and flushes content to the user only when the prefix remains below the calibrated risk threshold. If the monitored trajectory indicates persistent harmful drift, the proxy withholds the buffered content and replaces the

response with a refusal, with the intended goal of reducing prefix exposure. This proxy architecture has not been evaluated.

This proxy design also introduces open research questions. The buffer pool must balance earliness and perceived latency: larger buffers provide more context for reliable monitoring, whereas smaller buffers better preserve the fluidity of streaming generation. The proxy must also handle distribution shift between the lightweight open-source model used for monitoring and the commercial API model producing the stream. Future work will therefore investigate calibration procedures for proxy-model mismatch, adaptive buffer sizing, and multi-vector mechanisms that represent different harmful-intent directions and refusal behaviors. Such mechanisms may provide finer-grained defenses against subtle or mixed-risk behaviors while maintaining the safety–utility trade-off required for practical deployment.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state JEDI's scope, design principles, method components, and experimental claims, which are supported by the method description in Sections 3–4 and the results in Section 5 and the appendix. The stated claims are bounded to streaming jailbreak defense for the evaluated LLMs, attacks, and benchmarks.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 7 states the white-box deployment requirement, evaluated model/alignment scope, small calibration basis, residual prefix exposure, partial-information adaptive evaluation, and unmeasured batched throughput. Appendix F expands the failure conditions and future directions.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Appendix B states the positive-definite covariance and conditional drift/concentration assumptions, derives the standardized linear separation and condition-orthogonal displacement objectives, and gives proofs of the conditional CUSUM bounds. The text distinguishes these statements from empirical guarantees and minimax optimality.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [No]

   Justification: Sections 3–5 and Appendices C.1–D document the method, data separation, metrics, component substitutions, and reported baseline protocols. Some configuration details for adapted online baselines are not fully specified, so the description is not a complete reproduction specification for every comparison.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The abstract provides the public repository for JEDI code and experimental results. Public datasets, models, benchmarks, and baseline sources are cited; the main text and appendix document the evaluation and calibration protocols.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [No]

   Justification: Section 4 and Appendix D state the target models, attacks, metrics, prompt partitions, generation limits, and JEDI parameters. Appendix D.1 gives the reported online baseline protocols, but not all adaptation hyperparameters or random seeds are specified.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: Appendix C.1 reports prompt-cluster bootstrap summaries for separability, and Appendix C.2 defines fold standard deviations and observed ranges. Existing cross-model/attack deviations are distinguished from prompt uncertainty. Prompt-level intervals for end-to-end DSR and prefix-leakage comparisons are not reported, and the bootstrap interval construction and nominal coverage are not specified.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix D reports Python 3.10, PyTorch 2.5.1, Transformers 4.56.2, and a single NVIDIA H100 GPU. Table 7 and Appendix D.4 report per-response first-release and generation latency. Concurrent batched-serving throughput is outside the measured scope.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work studies defenses using public models and benchmarks and discusses residual leakage, over-refusal, and adaptive-attack limitations. Appendix C.1 describes human judgments used to audit the automatic evaluator; reporting gaps for this annotation exercise are identified in the human-research items below.

   Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses the positive societal impact of defending against jailbreak attacks and reducing prefix exposure in streaming generation. It also discusses potential risks and failure modes through adaptive-attack experiments and limitations on camouflaged attacks, over-refusal, and black-box deployment.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a new high-risk pretrained model, image generator, or scraped dataset. The released artifacts are defense code and experimental results for evaluating in-generation jailbreak defense.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper credits the existing models, datasets, jailbreak attacks, baselines, and the PandaGuard implementation through citations in the experimental setup and related-work sections. The released code is intended to respect the applicable open-source licenses and terms of the libraries, models, and datasets it uses.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper releases JEDI code and experimental results, and documents the method, calibration pipeline, evaluation setup, implementation environment, and generation parameters in the main text and appendix. These materials provide the documentation needed to inspect and reproduce the released assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [No]

    Justification: Appendix C.1 describes the ten student annotators, disjoint item assignment, judgment task, and classifier–human agreement. The full annotation instructions and compensation details are not reported.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [No]

    Justification: The classifier audit used human annotators. Potential-risk disclosure and institutional-review approval or exemption are not documented; no approval or exemption is claimed.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Section 4 describes the protected LLMs, and Appendices D.1 and D.2 identify the online guards and automatic evaluators. Appendix E specifies Llama-3.3-70B-Instruct for Adaptive-PAIR.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.