BarrierSteer: LLM Safety via Learning Barrier Steering

Thanh Q. Tran[1*] **Arun Verma**[2*] **Kiwan Wong**[3]
Bryan Kian Hsiang Low[1,2] **Daniela Rus**[3,2] **Wei Xiao**[2,3†]
[1]Department of Computer Science, National University of Singapore, Republic of Singapore
[2]Singapore-MIT Alliance for Research and Technology Centre, Republic of Singapore
[3]CSAIL, Massachusetts Institute of Technology, USA
thanhtran@u.nus.edu arun.verma@smart.mit.edu kiwan588@mit.edu
dcslkh@nus.edu.sg rus@csail.mit.edu weixy@mit.edu

Abstract

Despite the strong performance of large language models (LLMs) across diverse tasks, their susceptibility to adversarial attacks and unsafe content generation remains a significant obstacle to deployment, particularly in high-stakes settings. Addressing this challenge requires safety mechanisms that are both practically effective and theoretically grounded. In this paper, we introduce BarrierSteer, a novel inference-time framework that improves response safety by embedding learned nonlinear safety constraints directly into the model's latent representation space. BarrierSteer treats hidden-state safety classifiers as Control Barrier Functions (CBFs), enabling constraint-guided steering of unsafe latent trajectories during generation. By composing multiple safety constraints through efficient constraint merging without modifying the underlying LLM parameters, BarrierSteer preserves model utility. We provide theoretical results showing that applying CBFs in latent space yields a principled, modular, and computationally efficient approach for steering with respect to learned safety constraints, with guarantees conditional on the learned barriers capturing the intended safety property. Our extensive experimental results across multiple model families and datasets demonstrate that BarrierSteer substantially reduces adversarial attack success rates and unsafe generations, outperforming existing methods. The code is available in our GitHub repository.

1 Introduction

As large language models (LLMs) [1–4] become increasingly widespread due to their strong capabilities [5–8], their safety risks have become a central concern [9, 10]. Despite alignment efforts [11, 12], LLMs can still be manipulated by adversarial attacks [13, 14] to generate harmful content [15–17]. Such failures can have serious real-world consequences, creating significant legal and ethical risks when deploying LLMs in high-stakes applications [18, 19].

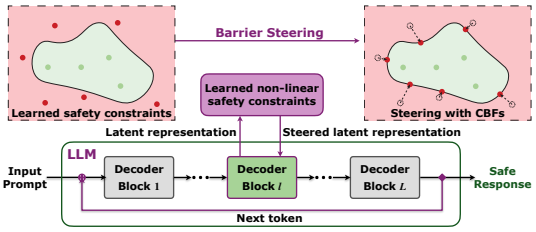

Figure 1: **BarrierSteer for LLM Safety.** Unlike steering with fixed directions, BarrierSteer derives adaptive corrections at each token, steering hidden states towards the learned safety boundaries using Control Barrier Functions (CBFs).

Existing LLM safety approaches can be broadly grouped into training-time and inference-time defenses. Training-time defenses [11, 12, 20] update model parameters to improve safety, but aligned models can remain vulnerable to adversarial prompts [13, 21–23]. Inference-time methods therefore

*Equal contributions. Correspondence to thanhtran@u.nus.edu. †Wei Xiao has since joined the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.

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

provide a complementary layer of defense, enabling safety interventions without modifying the underlying model parameters [24–26].

Recent inference-time defenses improve LLM safety but often introduce significant trade-offs. One line of work detects unsafe outputs from generated text. Post-hoc filters, such as heuristic keyword-based filters, are brittle to paraphrases [27, 28]. Language-model classifiers provide stronger detection [25, 29, 30], but require running an additional model, which can add substantial overhead during deployment. To reduce this cost, recent work studies classifiers that detect unsafe behavior from intermediate hidden states rather than generated text, enabling low-latency safety detection [26, 31]. However, detection identifies unsafe states but not how to steer them toward safer continuations.

A recent adaptive steering approach, `SaP`, treats classifiers as latent-space constraints [32]. While promising, `SaP` is limited to linear classifiers and requires expensive optimization-based steering. We propose BarrierSteer to address these limitations by treating learned hidden-state safety classifiers as Control Barrier Functions (CBFs), a standard control-theoretic tool for maintaining dynamical states within safe regions [33, 34]. This enables nonlinear safety boundaries, closed-form steering, and principled composition of multiple constraints. Our key contributions are summarized as follows:

- **Nonlinear latent safety constraints:** We introduce BarrierSteer, which learns nonlinear hidden-state safety classifiers and treats them as Control Barrier Functions (CBFs), providing a principled bridge between latent safety detection and constraint-based steering.
- **Closed-form CBF steering and composition:** We derive efficient closed-form steering updates for both single and composed CBF constraints, including a Log-Sum-Exp composition rule that enables jointly enforcing multiple safety constraints during autoregressive decoding.
- **Effective low-latency jailbreak defense:** We demonstrate empirically across four model families and ten adversarial attacks that BarrierSteer reduces attack success rates substantially while largely preserving utility (MMLU, GSM8K) and avoiding the over-refusal observed in fixed-direction steering baselines, with steering latency 58–99$\times$ faster than the closest baseline (SaP).

2 Related Work

LLM Safety Techniques. Improving the safety of LLMs has become a central challenge in modern AI research [9, 10]. A major line of work addresses this challenge through training-time alignment, where model parameters are updated using safety, preference, or refusal data. Representative methods include RLHF [11, 12, 20], `Safe-RLHF` [35], `DPO` [36–38], `SafeDPO` [39], adversarial training [40], and `Circuit Breakers` [41], which incorporate safety through preference optimization, constrained reward modeling, adversarial data, or representation engineering. However, aligned models can remain vulnerable to adversarial or jailbreak prompts [13, 21–23]. This motivates inference-time safety mechanisms that can intervene during deployment without changing the base model parameters.

Inference-time Safety. One line of inference-time safety work detects unsafe outputs after generation. Simple post-hoc filters, such as keyword- or rule-based filters, are brittle to paraphrases and obfuscation [27, 28]. Language-model classifiers provide stronger detection [25, 29, 30], but require running an additional model, which can add substantial overhead during deployment. To reduce this cost, recent work studies classifiers that detect unsafe behavior from intermediate hidden states rather than generated text, enabling low-latency safety detection on frontier models [26, 31]. However, these defenses primarily block unsafe generations; they do not steer the model toward a safe alternative.

Another line of work steers LLMs by intervening on model activations during generation. Activation-addition methods modify hidden states along fixed directions [24, 42, 43]. While these interventions are simple and efficient, they can be coarse, applying the same direction across contexts and may degrade utility. A more adaptive approach is to use learned safety classifiers as steering signals [44]. Motivated by hidden-state classifiers for production-grade safety detection [26, 31], classifier-guided methods use classifier scores not only to detect unsafe states, but also to guide how those states should be modified. Methods such as `SaP` steer using latent-space constraints [32], but existing approaches remain limited by linear classifier assumptions or by expensive constrained optimization and perturbation-search procedures [32, 45–47]. In contrast, BarrierSteer learns nonlinear latent-space safety constraints, derives closed-form steering rules, and efficiently composes multiple constraints. We defer further discussion of inference-time safety and classifier-guided steering to Appendix E.

Control Barrier Functions for Formal Safety. Control Barrier Functions (CBFs) [33, 34, 48] are a standard tool for enforcing safety in dynamical systems. CBFs turn nonlinear safety conditions into quadratic programs that can be solved efficiently online, providing a practical safety mechanism with formal guarantees under appropriate assumptions. This makes them especially appealing for learning-based control, where learned policies can be paired with explicit safety mechanisms rather than relying only on soft constraints [49, 50]. Advances in differentiable optimization [51] have enabled CBFs to be integrated into neural networks as safety-guaranteed layers [52–54], primarily for robotic control, and CBFs have also been extended to generative models such as LLM alignment [47] and diffusion models [55]. However, most CBF-based methods assume that the safety constraint is known *a priori*, which is often unrealistic for complex systems such as LLMs. Recent work has explored using CBFs to steer LLMs [56], but derives barriers from text-level safety classifiers and relies on constrained optimization during decoding, which can introduce substantial inference overhead. In contrast, BarrierSteer learns latent-space safety constraints and derives closed-form rules for efficient inference-time steering. Since these constraints are learned from data, the resulting guarantees are conditional on the learned barriers accurately capturing the intended safety property [57, 58].

3 Background and Motivation

This section provides the background and motivation for BarrierSteer. We first review how prior work learns hidden-state safety classifiers and uses their decision boundaries as safety constraints. We then discuss why using such constraints to steer LLMs remains practically challenging. Finally, we introduce Control Barrier Functions, a standard tool from control theory, which BarrierSteer uses to transform learned nonlinear safety constraints into efficient inference-time steering rules.

Throughout the paper, we use *classifier*, *constraint*, and *barrier function* depending on how the learned safety function is used. It is a *classifier* when distinguishing safe from unsafe hidden states, a *constraint* when controlling LLM generation, and a *barrier function* when used in the CBF condition.

3.1 Steering LLMs using Latent Classifiers

Learning Latent Classifiers for LLM Safety. Performing safety classification directly in text space often requires an external language model classifier [25], which can incur high computational cost when applied during generation. Recent work suggests that hidden-state classifiers can provide faster safety detection, with promising results in production-grade settings [26, 31]. We follow prior work [26, 31, 32] and use the standard hidden-state classifier training pipeline with binary safety data. Concretely, we are given a labeled dataset $\mathcal{D} = \{(q^{(i)}, x^{(i)}, y^{(i)})\}_{i=1}^{N}$, where $q^{(i)}$ is the user query and $x^{(i)}$ is the LLM response. Each response $x^{(i)}$ consists of $T^{(i)}$ tokens, $x^{(i)} = (x_1^{(i)}, \ldots, x_{T^{(i)}}^{(i)})$, with a binary response-level label $y^{(i)} \in \{0, 1\}$, where $y^{(i)} = 1$ indicates the presence of unsafe content. These labels are obtained from human annotations or automated safety classifiers [59].

To construct the latent training set, we run the LLM on each query–response pair and collect hidden states. Define the response-prefix state $s_t^{(i)} = (q^{(i)}, x_{\leq t}^{(i)})$. Let $h^{(\ell)}(s_t^{(i)})$ denote the hidden state corresponding to the response token $x_t^{(i)}$ at layer ℓ. This hidden state contains information about the query $q^{(i)}$ and the response prefix $x_{\leq t}^{(i)}$, rather than only the current token $x_t^{(i)}$ [60–62]. Crucially, because the supervision is assigned at the response level, each response $x^{(i)}$ yields $T^{(i)}$ hidden-state examples with the same label $y^{(i)}$: $\{(h^{(\ell)}(s_t^{(i)}), y^{(i)})\}_{t=1}^{T^{(i)}}$. Thus, an unsafe response contributes unsafe hidden-state examples across its generated positions, even though the original annotation does not identify a specific unsafe token span. These hidden-state examples are then used to learn latent safety classifiers whose decision functions define safety boundaries in hidden-state space.

Steering LLMs using Safety Constraints. Recent work, such as SaP [32], suggests explicitly viewing learned latent classifiers as safety constraints. Concretely, whenever a hidden state h is predicted to violate the learned constraint, SaP computes a projected safe representation by optimizing:

$$\min_{h^\star} \|h - h^\star\| \text{ s.t. } h^\star \text{ satisfies the learned safety constraints.}$$

This approach combines the detection strength of latent classifiers [26, 31] with activation steering, which adjusts LLMs toward safer generation [24, 42]. However, SaP has several *limitations*:

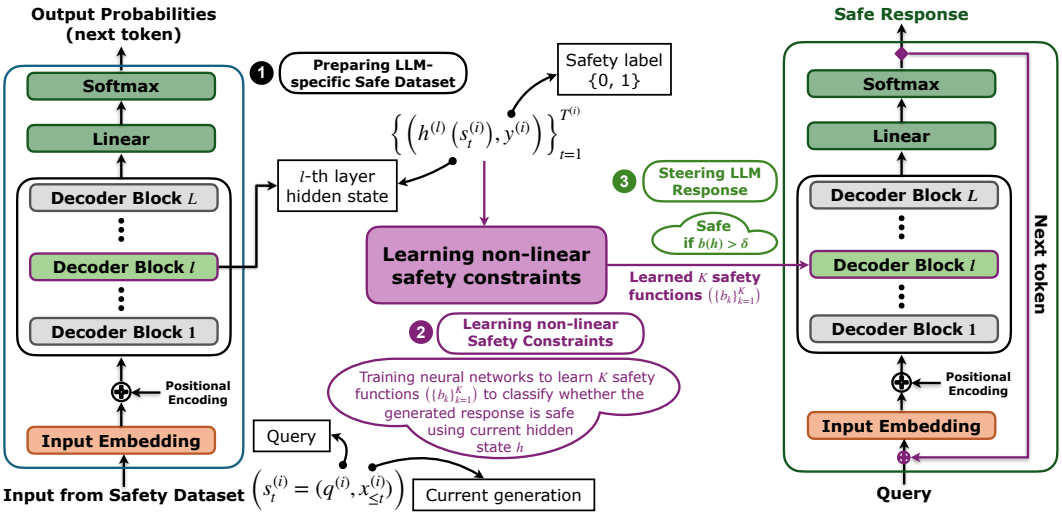

Figure 2: **Overview of BarrierSteer.** BarrierSteer follows a three-stage pipeline: 1 extracting intermediate LLM hidden states and pairing them with binary safety labels (Sec. 3.1); 2 learning nonlinear latent safety classifiers that define hidden-state safe sets (Sec. 4.1); and 3 applying closed-form CBF-based steering to guide hidden states toward learned safe sets (Sections 4.2 and 4.3).

- **Linearity:** SaP uses linear constraint boundaries, restricting classifier design.
- **Expensive optimization:** Applying the constraints requires optimization using gradient descent at inference time, so each steering step adds extra computation and increases generation latency.
- **Limited modularity:** Composing existing constraints or adding new ones may require retraining.

These limitations of SaP motivate BarrierSteer: a principled steering method that handles nonlinear latent constraints, derives closed-form updates, and composes safety constraints from independently trained classifiers. Empirically, BarrierSteer improves adversarial defense while preserving utility (Sec. 5.1), reduces steering latency (Sec. 5.2), and enables modular safety composition (Sec. 5.3).

3.2 Steering LLMs with Control Theory

Autoregressive Generation as a Latent Dynamical System. Autoregressive language generation naturally induces a trajectory in the model's latent space [45]. At decoding step t, the prefix state $s_t = (q, x_{\leq t})$ is processed by the LLM, which produces an intermediate hidden state $h^{(\ell)}(s_t)$. The model then samples the next token x_{t+1}, forming the updated state $s_{t+1} = (q, x_{\leq t+1})$. This induces a stochastic transition over latent representations, $h^{(\ell)}(s_t) \mapsto h^{(\ell)}(s_{t+1})$, allowing autoregressive generation to be viewed as a latent dynamical process. This view frames inference-time LLM safety as keeping latent trajectories within, or steering them toward, prescribed safe regions.

CBFs provide a natural mechanism for this purpose: a learned safety classifier defines a safe set in the hidden-state space, and the CBF condition determines local corrections to steer the trajectory away from unsafe regions during decoding. BarrierSteer instantiates this idea by treating hidden states as latent system states and learned safety classifiers as CBF constraints.

Control Barrier Functions. CBFs provide a principled framework for enforcing safety in dynamical systems. We consider affine control systems with state $h \in \mathbb{R}^n$, control input $u \in \mathbb{R}^m$, and dynamics

$$\dot{h} = f(h) + g(h)u,$$

where f and g are assumed to be locally Lipschitz. The goal is to ensure that the state h remains within a prescribed safe set $\mathcal{C}$, defined as the 0-superlevel set of a continuously differentiable function $b : \mathbb{R}^n \to \mathbb{R}$:

$$\mathcal{C} = \{h \in \mathbb{R}^n : b(h) \geq 0\}.$$

Definition 1 (Control Barrier Function)**.** *A function b is a control barrier function for the system on $\mathbb{R}^n$ if there exists an extended class-$\mathcal{K}$ function β such that*

$$\forall h \in \mathbb{R}^n : \sup_{u \in \mathbb{R}^m} \dot{b}(h, u) \geq -\beta(b(h)).$$

Since $\dot{b}(h,u) = L_f b(h) + L_g b(h)u$ by the chain rule, this condition can be equivalently written as

$$\sup_{u \in \mathbb{R}^m} [L_f b(h) + L_g b(h)u + \beta(b(h))] \geq 0.$$

Here, $L_f b(h)$ and $L_g b(h)$ denote the Lie derivatives of b along the drift and control dynamics, respectively. The term $\beta(b(h))$ shapes how aggressively the system must react as it approaches the boundary of the safe set.

Given a CBF b, any control input u satisfying $\dot{b}(h,u) \geq -\beta(b(h))$ for all $h \in \mathcal{C}$ renders $\mathcal{C}$ forward invariant: trajectories starting in $\mathcal{C}$ remain in $\mathcal{C}$ under standard regularity assumptions [33, 34]. For simplicity, we use a linear class-$\mathcal{K}$ function throughout this paper, i.e., $\beta(b(h)) = \alpha b(h), \alpha \in \mathbb{R}_{>0}$.

Notably, CBFs can enforce nonlinear safety constraints, admit efficient closed-form solutions, and are composable. These properties match the needs of LLM activation steering: expressive safety boundaries, efficient online intervention, and modular composition of multiple constraints.

4 BarrierSteer: LLM Steering with Learned Control Barrier Functions

In this section, we present our BarrierSteer framework for learning latent safety constraints and applying them as CBFs during inference. As illustrated in Fig. 2, BarrierSteer follows a three-stage pipeline: ❶ extracting intermediate latent representations from a pre-trained LLM and constructing a dataset with binary safety labels (see Sec. 3.1); ❷ learning expressive nonlinear safety constraints in latent space; and ❸ applying CBF-based steering at inference time to guide hidden-state trajectories toward the learned safe set without modifying the underlying model parameters.

4.1 Learning Control Barrier Functions

A key challenge in applying CBFs to LLMs is that safety constraints are not known *a priori* and must instead be inferred from data. In contrast to prior approaches that rely on linear boundaries [32], we learn nonlinear safety constraints directly using pairs of hidden states and safety labels. We estimate safety boundaries by learning barrier functions $\{b_k\}_{k=1}^K$ with neural network parameters $\theta = \{\theta_1, \ldots, \theta_K\}$. Given safe and unsafe hidden states from $\mathcal{H}_{\text{safe}}$ and $\mathcal{H}_{\text{unsafe}}$, we jointly train the barriers with a CBF-style classification loss [57, 58] consisting of two terms:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{safe}}(\theta) + \lambda_{\text{unsafe}} \mathcal{L}_{\text{unsafe}}(\theta),$$

where the parameter $\lambda_{\text{unsafe}} \in \mathbb{R}_{>0}$ is a hyperparameter that controls the penalty associated with safety violations. The *safe set loss* makes safe states satisfy *all* constraints ($b_k(h) \geq 0, \forall k \in \{1, \ldots, K\}$):

$$\mathcal{L}_{\text{safe}}(\theta) = \sum_{h \in \mathcal{H}_{\text{safe}}} \sum_{k=1}^{K} [-b_k(h)]_+.$$

Conversely, the *unsafe set loss* makes unsafe states violate *at least one* constraint ($\min_k b_k(h) \leq -\varepsilon$):

$$\mathcal{L}_{\text{unsafe}}(\theta) = \sum_{h \in \mathcal{H}_{\text{unsafe}}} [\min_k b_k(h) + \varepsilon]_+.$$

In these expressions, the operator $[\cdot]_+ = \max(0, \cdot)$ denotes the hinge loss, and ε defines the safety margin. Minimizing $\mathcal{L}(\theta)$ encourages the intersection of the learned safety constraints to correctly classify safe and unsafe states. Moreover, distinct control barrier functions can be learned from separate datasets and composed, as discussed in Section 4.3.

4.2 Representation Steering with Latent Dynamics

Once the safety constraints are learned, we can use them to steer the LLM generation. Obtaining b_k allows us to check whether the current latent state satisfies the safe set $\mathcal{C} = \{h \mid b_k(h) \geq \delta, k \in \{1, \ldots, K\}\}$, where δ denotes the safety threshold. In practice, we set $\delta = 0$ for simplicity. If the model's original trajectory violates the safety margin (i.e., $\exists k, b_k(h) < \delta$), we dynamically adjust the latent state h within the generation loop to steer it toward the learned safe region.

To steer using CBFs, we approximate the local evolution of the hidden state as $\dot{h} = (h_t - h_{t-1})/\Delta t$, where Δt is the time step (default $\Delta t = 1$). Following the local-dynamics view in Sec. 3.2, this finite difference is used only to define local dynamics, not to predict the full future trajectory. By modeling the latent-state evolution locally as a controllable dynamical system $\dot{h} = u$, we seek a control input u that satisfies the local first-order CBF condition while minimizing deviation from the original trajectory. Specifically, we compute the control u by solving the following Quadratic Program (QP):

$$u^* = \underset{u}{\operatorname{argmin}} \left\| u - \frac{h_t - h_{t-1}}{\Delta t} \right\|^2 \text{ s.t. } \nabla b_k(h_t)^\top u + \alpha(b_k(h_t) - \delta) \geq 0,\ k \in \{1, \ldots, K\}. \quad (1)$$

The steered hidden state is $h'_t = h_{t-1} + u^* \Delta t$, which replaces h_t in the decoding loop. This avoids expensive optimization because the QP can admit efficient closed-form solutions, as discussed in Sec. 4.3. Furthermore, the parameter α in Eq. (1) controls the steering strength. A larger value of α produces a more aggressive correction toward the learned latent safe set $\mathcal{C}$. It may also induce a significant deviation from the nominal control input. This fundamental trade-off between safety and task performance is empirically validated across various benchmarks, as reported in Table 2.

4.3 Closed-form Steering via Composing CBFs

Real-world safety specifications often involve multiple constraints (e.g., "avoid harm" and "protect privacy"). We formalize such requirements as satisfying all K constraints in Eq. (1). However, directly solving the quadratic program is inefficient, since the control dimension matches the hidden-state dimension and is typically large. To address this, we derive a closed-form steering rule.

Recall that $\{b_k(h)\}_{k=1}^K$ is a set of learned CBF boundaries corresponding to safe sets $\mathcal{C}_k$. A system satisfying all constraints must remain in the intersection $\bigcap_{k=1}^K \mathcal{C}_k$. We merge these constraints into a single, continuously differentiable CBF. The intersection of safe sets can be captured by a smooth under-approximation of the minimum function using the Log-Sum-Exp (LSE) formula [63]:

$$B(h) = -\frac{1}{\kappa} \ln \left(\sum_{k=1}^{K} e^{-\kappa(b_k(h) - \delta)} \right), \quad (2)$$

$$B(h) \geq 0 \implies b_k(h) \geq \delta \quad \forall k \in \{1, \ldots, K\}, \quad (3)$$

where $\kappa \in \mathbb{R}_{>0}$ is a smoothing parameter. Notably, this composed barrier $B(h)$ remains continuously differentiable, and the satisfaction of $B(h) \geq 0$ implies the satisfaction of $b_k(h) - \delta \geq 0, \forall k \in \{1, \ldots, K\}$ [63]. When the number of active constraints is at most 2 (either $K \leq 2$ originally, or after composing via Eq. (2)), the QP in Eq. (1) admits the following closed-form solution for u:

$$u^* = \frac{h_t - h_{t-1}}{\Delta t} + \lambda_1(h_t)\hat{g}_1(h_t) + \lambda_2(h_t)\hat{g}_2(h_t), \quad (4)$$

where $\lambda_1, \lambda_2, \hat{g}_1$, and $\hat{g}_2$ are defined in Appendix Sec. A.

Remark. We use the following local assumptions to apply the CBF for steering: (i) each learned barrier b_k and the composed barrier B in Eq. (2) are continuously differentiable; and (ii) over the one-step neighborhood where steering is applied, the hidden-state evolution is locally Lipschitz and is well approximated by the finite-difference model $\dot{h} = (h_t - h_{t-1})/\Delta t$ used in Eq. (1).

These assumptions are reasonable in our setting. First, our neural barrier heads use smooth GELU-based MLPs, and the LSE composition is smooth for $\kappa > 0$ (Appendix Sec. C), so the required gradients are available by design. Second, the dynamics assumption is local and one-step: at each decoding step, the finite-difference model $\dot{h} = (h_t - h_{t-1})/\Delta t$ in Eq. (1) serves as a first-order approximation for choosing a correction direction near the current hidden state. We do not assume it globally describes transformer decoding over an entire trajectory; instead, BarrierSteer re-evaluates the barrier at every decoding step and applies at most one closed-form steering update per token when the composed barrier condition is violated. Such local approximations have proven effective for CBF-style control in other generative models, including diffusion models [55].

We formalize the safety property of the steering in Eq. (4) under the LSE composition in Eq. (2).

Theorem 1. *Let $\{b_k(h) \geq \delta\}_{k=1}^K$ be learned safety constraints, and let $B(h)$ denote the smooth composed barrier in Eq. (2), with composed safe set $\mathcal{C}_B = \{h \mid B(h) \geq 0\}$. Let h and h' denote the*

hidden states before and after applying the control u^* in Eq. (4). Under the local assumptions stated above, the update satisfies the composed CBF condition $\nabla B(h)^\top u^* + \alpha B(h) \geq 0$. Consequently, under the local dynamics, $\mathcal{C}_B$ is locally forward invariant: if $h \in \mathcal{C}_B$, the composed barrier condition is locally preserved; if $h \notin \mathcal{C}_B$, the composed-barrier violation is exponentially stabilized toward $\mathcal{C}_B$.

The proof of Theorem 1 is provided in Appendix B.

Theorem 1 justifies using the composed barrier in Eq. (2) for steering under multiple constraints, since $B(h) \geq 0$ is a sufficient condition for satisfying all K constraints. The guarantee, however, is conditional on the learned CBFs accurately capturing the intended text-level safety properties.

The closed-form steering in Eq. (4) leads to three practical variants of BarrierSteer:

- **BarrierSteer (QP):** Directly solves the QP in Eq. (1), which can be computationally expensive.
- **BarrierSteer (Top-2):** A selection heuristic that retains the two most violated constraints, reducing the active set to $K = 2$, and then applies the closed-form solution in Eq. (4).
- **BarrierSteer (LSE):** Composes all constraints into a single CBF using the LSE formula in Eq. (2), then applies the closed-form solution in Eq. (4) with $K = 1$.

In practice, we use BarrierSteer (LSE) as the default variant because it is theoretically grounded by Theorem 1, computationally efficient, and empirically effective.

5 Experiments

Datasets and Evaluation Setup. We evaluate BarrierSteer on two safety benchmarks using instruction-tuned LLMs that have undergone safety alignment but remain vulnerable to adversarial prompts. First, we use `HarmBench` [64], a standardized framework for automated adversarial-prompt generation. We select 320 harmful behaviors to construct training cases for learning the safety barriers and reserve 80 disjoint behaviors for evaluation. This behavior-level split tests whether the method generalizes to unseen harmful behaviors rather than memorizing specific prompts.

Second, we use `WildGuardMix` [30], a large-scale dataset covering harmful prompts across 14 risk categories. For each category, we sample 400 prompts and split them into 320 for training and 80 for evaluation. We use these prompts to collect safe and unsafe activation datasets, denoted by $\mathcal{H}_{\text{safe}}$ and $\mathcal{H}_{\text{unsafe}}$, respectively. As discussed in Sec. 3.1, each response $x^{(i)}$ with length $T^{(i)}$ yields $T^{(i)}$ hidden-state examples, so both datasets $\mathcal{H}_{\text{safe}}$ and $\mathcal{H}_{\text{unsafe}}$ contain substantially more examples than the number of prompts (see Sec. D.1 for exact dataset sizes). For both datasets, we quantify safety using the HarmBench classifier [59], which evaluates the semantic safety of generated responses.

Baselines. We compare BarrierSteer against a diverse set of LLM safety methods, focusing on inference-time steering baselines. First, we include `Circuit Breaker` [41], a training-time representation-engineering method that fine-tunes LoRA-style adapters to reroute internal representations associated with harmful generations, while preserving benign behavior through a retain loss. We also compare against `Self-Reminder`, a prompting-based inference-time defense that prepends and appends safety reminders to the user prompt to encourage the model to follow safety policies.

Our main comparisons focus on activation steering methods, which directly modify hidden states during generation. `Activation Addition` [24, 43] constructs a steering direction from the difference between mean activations from two contrastive sets, $r = |\mathcal{H}_{\text{safe}}|^{-1} \sum_{h \in \mathcal{H}_{\text{safe}}} h - |\mathcal{H}_{\text{unsafe}}|^{-1} \sum_{h \in \mathcal{H}_{\text{unsafe}}} h$, and applies the intervention $h' = h + \alpha r$, where α controls the steering strength. `Directional Ablation` [42] removes the component of the hidden state along an identified steering direction, $h' = h - r(r^\top r)^{-1} r^\top h$. We also compare against `ReFT-r1` [65], a weakly supervised rank-one representation-finetuning baseline that applies a learned linear intervention to hidden representations during generation. Unlike difference-in-means steering, its intervention direction is learned from supervision rather than computed directly from activation mean differences.

Finally, we evaluate against `SaP` [32], the closest baseline to BarrierSteer because it also uses learned safety boundaries as constraints for inference-time steering. Like BarrierSteer, `SaP` learns latent safety constraints from hidden states and safe/unsafe supervision. However, BarrierSteer differs from `SaP` in both the expressiveness of the learned constraint and the efficiency of the steering rule, which directly affects the safety–utility trade-off and latency observed in our experiments. By default, we use BarrierSteer (LSE), with each classifier implemented as a five-layer MLP matched

to SaP in parameter count. For fairness, all learned baselines use the same safe/unsafe data to train their LoRA adapters, steering directions, representation interventions, or latent constraints, and all steering methods intervene at layer $\ell = 20$, following prior work [32, 42]. We provide implementation details for BarrierSteer and the baselines in Sections C and D.2, respectively.

5.1 Adversarial Attack Defense

We assess the robustness of BarrierSteer against representative adversarial attack methods. These include gradient-based optimization attacks such as GCG [13], GBDA [66], AutoPrompt [67], PEZ [68], UAT [69], and AutoDAN [21]; human-crafted jailbreaks, including HumanJailbreak [22] and DirectRequest [64]; and the search-based PAP [23]. We use these nine attack methods to construct the training data. We further evaluate against Adaptive Attack [70], which explicitly optimizes against defended LLMs. Adaptive Attack is used only for evaluation, not training. Additional experimental details and results are provided in Sec. D.3.

Over-refusal. We use XSTest [71] to evaluate whether defended models over-refuse benign prompts.

Utility Benchmarks. To ensure representation steering does not significantly degrade core model utility, we evaluate zero-shot on two standard benchmarks: MMLU [72] for general knowledge and GSM8K [73] for mathematical reasoning. We refer to Sec. D.7 for results on more utility benchmarks.

Table 1 summarizes results across four model families, covering static HarmBench ASR, adaptive-attack ASR, over-refusal, and utility. Overall, BarrierSteer (LSE) achieves the strongest static jailbreak defense, reducing ASR to near zero across models while preserving utility close to the original LLMs for Gemma-2-9B, Ministral-8B and Llama-2-7B. Since all learned baselines are trained on the same HarmBench safe/unsafe data, the lower ASR suggests that BarrierSteer more effectively captures the latent safety boundary and applies the learned constraints. On static HarmBench ASR, BarrierSteer outperforms prompt-based defenses such as Self-Reminder, training-time methods such as Circuit Breaker, and inference-time representation-steering baselines such as ActAdd, DirAbl, ReFT-r1, and SaP.

The comparison also highlights the safety–utility trade-offs of existing baselines. Fixed-direction steering methods can mitigate attacks, but often degrade benign behavior or downstream utility because the same intervention is applied across diverse contexts. For example, DirAbl reduces the adaptive-attack ASR of Gemma-2-9B to 32.00%, but lowers GSM8K accuracy from 74.22% to 49.51% and increases XSTest over-refusal from 28.00% to 43.20%. Prompting defenses can be effective for some LLMs, such as Self-Reminder on Llama-2-7B under both static and adaptive attacks, but often incur severe over-refusal and utility degradation. Training-based baselines such as Circuit Breaker and ReFT-r1 improve robustness, but do not match the constraint-guided steering methods SaP and BarrierSteer across static and adaptive attacks. This suggests that modeling safety as latent constraints provides a more reliable mechanism for jailbreak defense. Under adaptive attacks, the improvement from BarrierSteer varies across LLMs. BarrierSteer lowers the adaptive-attack ASR from 72.00% to 1.60% on Llama-2-7B, from 74.00% to 19.60% on Ministral-8B, and from 62.00% to 24.80% on Qwen2-1.5B. On Gemma-2-9B, it lowers ASR from 70.00% to 57.20%, compared with 56.00% for Circuit Breaker and 32.00% for DirAbl. The adaptive attack is out-of-distribution (OOD) relative to the HarmBench training data because it generates unseen attack trajectories. These results show that BarrierSteer trained only on HarmBench does not always generalize to unseen attacks. New barriers could be trained for emerging attacks and composed with existing barriers without retraining the existing ones (Sec. 5.3).

Compared to the closest latent-constraint baseline, SaP, BarrierSteer provides substantially stronger robustness. On Ministral-8B, BarrierSteer reduces ASR to $0.83_{\pm 1.86}\%$ while SaP remains at $23.91_{\pm 3.49}\%$; on Qwen2-1.5B, BarrierSteer reduces ASR to $0.66_{\pm 0.47}\%$ while SaP remains at $5.35_{\pm 3.44}\%$. For a fair comparison, we also ablate the inference-time steering strength of SaP by varying its optimization weights. As shown in Sec. D.2, changing this steering strength does not close the robustness gap to BarrierSteer. This suggests that the improvement comes from the greater expressiveness of nonlinear neural CBF boundaries. These robustness gains come without extreme over-refusal and with little change in MMLU and GSM8K on Gemma-2-9B, Ministral-8B and Llama-2-7B. On Qwen2-1.5B, GSM8K drops from 24.94% to 12.95%. Other methods that steer hidden states directly show the same drop (ActAdd 12.43%, DirAbl 13.12%, SaP 12.63%), while ReFT-r1, trained with a language-modeling objective, reaches 22.92%.

Table 1: Attack success rate (ASR, ↓), adaptive-attack ASR (AA ASR, ↓), over-refusal rate (XSTest, ↓), and utility metrics (MMLU, GSM8K, ↑) for BarrierSteer (LSE) and other LLM safety baselines across four model families. ASR is averaged over nine HarmBench adversarial attack methods. ActAdd denotes Activation Addition, and DirAbl denotes Directional Ablation. Results are reported over five runs for learning-based methods, while other baselines are evaluated deterministically.

Model	Metric	Base	Self-Reminder	Circuit Breaker	ActAdd	DirAbl	ReFT-r1	SaP	BarrierSteer
Gemma-2-9B	ASR ↓	19.50	12.89	$9.91_{\pm 2.60}$	19.00	13.61	$12.30_{\pm 3.19}$	$8.00_{\pm 0.94}$	$\mathbf{0.00}_{\pm \mathbf{0.00}}$
	AA ASR ↓	70.00	66.00	$56.00_{\pm 4.00}$	62.00	32.00	$66.00_{\pm 2.45}$	$66.80_{\pm 2.68}$	$57.20_{\pm 4.15}$
	XSTest ↓	28.00	38.80	$35.76_{\pm 2.71}$	31.20	43.20	$36.80_{\pm 2.77}$	$33.92_{\pm 1.56}$	$32.00_{\pm 0.44}$
	MMLU ↑	71.86	71.72	$70.94_{\pm 0.74}$	71.86	66.16	$71.31_{\pm 0.44}$	$71.86_{\pm 0.01}$	$71.87_{\pm 0.09}$
	GSM8K ↑	74.22	72.78	$73.44_{\pm 0.84}$	73.69	49.51	$73.74_{\pm 0.25}$	$73.72_{\pm 0.16}$	$73.65_{\pm 0.04}$
Ministral-8B	ASR ↓	51.19	20.25	$30.19_{\pm 2.51}$	51.31	51.78	$37.56_{\pm 3.31}$	$23.91_{\pm 3.49}$	$\mathbf{0.83}_{\pm \mathbf{1.86}}$
	AA ASR ↓	74.00	74.00	$66.00_{\pm 5.48}$	78.00	80.00	$68.00_{\pm 0.00}$	$70.00_{\pm 2.00}$	$19.60_{\pm 2.61}$
	XSTest ↓	20.40	22.00	$21.44_{\pm 1.80}$	17.20	17.20	$15.20_{\pm 1.06}$	$15.60_{\pm 3.55}$	$21.84_{\pm 1.63}$
	MMLU ↑	64.73	63.03	$64.56_{\pm 0.06}$	64.53	64.45	$63.89_{\pm 0.28}$	$64.47_{\pm 0.02}$	$63.07_{\pm 1.95}$
	GSM8K ↑	64.22	67.63	$64.92_{\pm 0.61}$	65.05	64.06	$65.33_{\pm 0.23}$	$64.67_{\pm 0.13}$	$62.55_{\pm 2.46}$
Llama-2-7B	ASR ↓	6.06	3.22	$6.70_{\pm 0.31}$	6.47	3.19	$3.43_{\pm 0.35}$	$6.88_{\pm 0.21}$	$\mathbf{1.28}_{\pm \mathbf{2.17}}$
	AA ASR ↓	72.00	2.00	$68.00_{\pm 2.00}$	70.00	80.00	$36.00_{\pm 0.00}$	$76.00_{\pm 5.29}$	$1.60_{\pm 0.89}$
	XSTest ↓	41.60	68.00	$42.40_{\pm 0.69}$	41.60	48.80	$47.76_{\pm 1.19}$	$39.84_{\pm 2.48}$	$43.36_{\pm 0.83}$
	MMLU ↑	46.47	44.51	$46.50_{\pm 0.01}$	46.56	46.29	$46.60_{\pm 0.06}$	$46.57_{\pm 0.02}$	$46.56_{\pm 0.04}$
	GSM8K ↑	20.92	9.55	$20.90_{\pm 0.45}$	21.30	19.94	$21.56_{\pm 0.38}$	$20.92_{\pm 0.08}$	$20.94_{\pm 0.62}$
Qwen2-1.5B	ASR ↓	23.89	9.08	$23.88_{\pm 1.47}$	19.14	16.97	$4.17_{\pm 0.47}$	$5.35_{\pm 3.44}$	$\mathbf{0.66}_{\pm \mathbf{0.47}}$
	AA ASR ↓	62.00	40.00	$49.60_{\pm 2.19}$	58.00	60.00	$49.20_{\pm 3.03}$	$63.60_{\pm 8.29}$	$24.80_{\pm 1.79}$
	XSTest ↓	27.60	36.40	$27.12_{\pm 1.21}$	32.00	34.80	$77.68_{\pm 2.73}$	$29.36_{\pm 0.36}$	$29.44_{\pm 1.49}$
	MMLU ↑	55.69	54.96	$55.64_{\pm 0.02}$	55.71	54.76	$55.55_{\pm 0.04}$	$55.69_{\pm 0.01}$	$55.65_{\pm 0.05}$
	GSM8K ↑	24.94	31.01	$26.71_{\pm 1.29}$	12.43	13.12	$22.92_{\pm 1.34}$	$12.63_{\pm 0.19}$	$12.95_{\pm 0.63}$

Table 2: Ablation study on the effect of steering strength α for BarrierSteer (LSE) on Qwen2-1.5B with $\kappa = 100$ over 5 runs.

α	ASR ↓	AA ASR ↓	XSTest ↓	MMLU ↑	GSM8K ↑
0.001	$0.20_{\pm 0.26}$	$56.80_{\pm 9.34}$	$\mathbf{28.00}_{\pm \mathbf{1.70}}$	$\mathbf{55.72}_{\pm \mathbf{0.04}}$	$\mathbf{13.03}_{\pm \mathbf{0.14}}$
0.01	$0.66_{\pm 0.47}$	$24.80_{\pm 1.79}$	$29.44_{\pm 1.49}$	$55.65_{\pm 0.05}$	$12.95_{\pm 0.63}$
0.1	$0.06_{\pm 0.10}$	$21.60_{\pm 8.17}$	$55.68_{\pm 5.21}$	$50.24_{\pm 4.58}$	$12.36_{\pm 1.71}$
0.2	$0.02_{\pm 0.03}$	$9.20_{\pm 3.63}$	$67.44_{\pm 10.21}$	$45.94_{\pm 7.02}$	$11.37_{\pm 1.93}$
0.3	$0.04_{\pm 0.07}$	$3.20_{\pm 3.35}$	$64.40_{\pm 1.09}$	$43.59_{\pm 6.07}$	$10.69_{\pm 0.72}$
0.5	$0.00_{\pm 0.00}$	$\mathbf{2.40}_{\pm \mathbf{1.67}}$	$71.68_{\pm 2.25}$	$40.45_{\pm 4.03}$	$9.30_{\pm 1.25}$
1.0	$\mathbf{0.00}_{\pm \mathbf{0.00}}$	$2.80_{\pm 1.10}$	$76.32_{\pm 4.32}$	$37.80_{\pm 2.81}$	$7.25_{\pm 0.16}$

Table 3: Steering latency (ms/token, ↓) on Qwen2-1.5B.

Method	Latency ↓
SaP (100 steps)	$190.67_{\pm 3.17}$
SaP (1 step)	$139.97_{\pm 6.33}$
BarrierSteer (Top-2)	$29.45_{\pm 11.00}$
BarrierSteer (QP)	$33.77_{\pm 5.86}$
BarrierSteer (LSE)	$\mathbf{2.40}_{\pm \mathbf{0.06}}$

Impact of Steering Strength α. We further investigate the sensitivity of BarrierSteer to the steering strength α in Table 2. The results show that α directly controls the safety–utility trade-off: smaller values better preserve utility and maintain low over-refusal, whereas larger values apply stronger safety corrections and further reduce attack success. In particular, increasing α drives static HarmBench ASR to zero and often improves robustness under adaptive attacks.

We select α separately for each model by sweeping its values and inspecting the safety–utility trade-off (Tab. 16). Table 1 uses $\kappa = 100$ and $\alpha = 0.1$ for Gemma-2-9B, Ministral-8B and Llama-2-7B, and $\alpha = 0.01$ for Qwen2-1.5B. These settings reduce static and adaptive ASR while keeping XSTest close to the base model. Because α is applied only at inference time, it can be changed after training without updating the barriers or the model. Larger values apply stronger corrections but can increase over-refusal or reduce utility. Deployments that prioritize robustness can therefore use a larger α, while settings that prioritize benign-task performance can use a smaller one.

5.2 Computational Efficiency

A primary advantage of BarrierSteer is its computational efficiency. Unlike SaP, which runs gradient descent for each token at inference time, BarrierSteer (LSE) uses a closed-form update (see Sec. 4.3). We measure per-token inference latency only for tokens that trigger steering, thereby isolating the steering overhead. As shown in Table 3, BarrierSteer (LSE) introduces negligible overhead, with a per-token steering latency of 2.40 ms. In contrast, SaP requires 190.67 ms with 100 optimization steps and 139.97 ms with one step. This corresponds to speedups of over 79× and 58×, respectively. Results for the other model families in Sec. D.8 show the same pattern. BarrierSteer stays between 2.40 and 2.55 ms per steered token, which is 79–99× faster than SaP with 100 steps and 58–71× faster with one step. In end-to-end generation on Qwen2-1.5B, BarrierSteer takes 39.44 s compared with 35.27 s for the base model, while SaP takes 102.05 s.

Table 4: Harmful rate (%, ↓) on `Qwen2-1.5B` when composing barriers independently trained on 14 risk categories from `WildGuardMix` [30]. Metrics are averaged over five runs.

	Original Model	Single BarrierSteer (avg.)	BarrierSteer (LSE)
Harmful Rate ↓	22.77	$4.74_{\pm 1.21}$	$\mathbf{0.86}_{\pm \mathbf{0.04}}$

5.3 Modular Composition of Safety Barriers

We demonstrate the modularity of BarrierSteer by training distinct Control Barrier Functions (CBFs), each using data from one of 14 harmful risk categories in `WildGuardMix` [30]. For robust generalization, we evaluate on a held-out test set containing unseen behaviors from all 14 categories. These independently trained barriers are used to steer individually or composed using the LSE formula. See Sec. D.9 for experimental details, other BarrierSteer variants, and an ablation of κ in Eq. (2).

Table 4 presents the main composition results. The original model has a high harmful-generation rate of 22.77%. Averaged across the 14 single-barrier configurations, this rate decreases to 4.74%, showing that a barrier trained on one risk concept can also reduce unsafe generations in other categories. Finally, BarrierSteer (LSE) reduces the harmful-generation rate to just 0.86%. This shows that composing CBFs trained on independent concepts enables BarrierSteer to perform well across all concepts, consistent with Theorem 1. This modularity supports practical real-world use cases, where safety specifications can be applied compositionally depending on context and users. A barrier for a new risk category can be composed with existing barriers without retraining them.

5.4 Ablating Barrier Architecture and Steering Mechanism

Barrier Architecture. We replace the 5-layer neural CBFs in the WildGuard composition experiment with linear probes, keeping the same training data and composition rules. As shown in Tab. 5, the LSE harmful rate increases from $0.86_{\pm 0.04}$ with neural barriers to $2.89_{\pm 0.69}$ with linear probes, showing that neural barriers perform better.

Steering Mechanism. We test whether using the learned barrier only to trigger `ActAdd` or `DirAbl` can match the CBF steering update. These conditional baselines have higher ASR than BarrierSteer (Tab. 6 and Sec. D.12), suggesting that the CBF update improves safety beyond using the barrier only to decide when to intervene. The best conditional baseline lowers ASR from 19.50% to 12.50%, while BarrierSteer reaches 0.00%.

Table 5: Harmful rate (%, ↓) on `Qwen2-1.5B` with linear-probe and neural barriers.

Merge Rule	Linear Probe	Neural CBF
Top-2	$12.44_{\pm 0.95}$	$\mathbf{10.60}_{\pm \mathbf{1.37}}$
QP	$8.87_{\pm 0.15}$	$\mathbf{1.82}_{\pm \mathbf{0.29}}$
LSE	$2.89_{\pm 0.69}$	$\mathbf{0.86}_{\pm \mathbf{0.04}}$

Table 6: HarmBench ASR (%, ↓) for `Gemma-2-9B`. Conditional baselines steer with a fixed direction when the barrier is violated.

Method	ASR
Base model	19.50
Conditional ActAdd	$18.17_{\pm 0.00}$
Conditional DirAbl	$12.50_{\pm 0.00}$
BarrierSteer	$\mathbf{0.00}_{\pm \mathbf{0.00}}$

6 Conclusion

We presented BarrierSteer, a principled inference-time framework for improving LLM safety by embedding learned nonlinear safety constraints in latent representation space. By treating hidden-state safety classifiers as Control Barrier Functions, BarrierSteer steers hidden-state trajectories toward learned safe regions while preserving the underlying model parameters. Empirically, BarrierSteer consistently reduces unsafe generations and adversarial attack success rates across models and datasets, including under adaptive attacks, while largely preserving model utility and avoiding excessive over-refusal. Limitations include that the guarantees of BarrierSteer are conditional on the learned barriers accurately capturing the intended safety properties and apply to latent representations rather than providing strict worst-case guarantees for generated text. Barriers trained only on HarmBench do not always generalize to unseen attacks, including adaptive and multilingual attacks. Future work includes training barriers on larger and more diverse safety data, composing new barriers for emerging attacks, scaling BarrierSteer to frontier LLMs, and developing more fine-grained steering rules to better manage adaptive attacks and safety–utility–refusal trade-offs.

Acknowledgments and Disclosure of Funding

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-PhD/2026-08-076[T]). This research is supported by the National Research Foundation (NRF), Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. The Mens, Manus, and Machina (M3S) is an interdisciplinary research group (IRG) of the Singapore MIT Alliance for Research and Technology (SMART) centre. It was also partially sponsored by the United States Air Force Artificial Intelligence Accelerator and was accomplished under Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] Anthropic. Introducing claude 2.1. https://www.anthropic.com/news/claude-2-1/, 2023.

[2] Google. Palm 2 technical report. *arXiv:2305.10403*, 2023.

[3] OpenAI. Gpt-4 technical report. *arXiv:2303.08774*, 2023.

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

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

[6] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In *Proc. NeurIPS*, 2022.

[7] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 2023.

[8] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv:2303.18223*, 2023.

[9] Usman Anwar, Abulhair Saparov, Javier Rando, Daniel Paleka, Miles Turpin, Peter Hase, Ekdeep Singh Lubana, Erik Jenner, Stephen Casper, Oliver Sourbut, et al. Foundational challenges in assuring alignment and safety of large language models. *arXiv:2404.09932*, 2024.

[10] David Dalrymple, Joar Skalse, Yoshua Bengio, Stuart Russell, Max Tegmark, Sanjit Seshia, Steve Omohundro, Christian Szegedy, Ben Goldhaber, Nora Ammann, et al. Towards guaranteed safe ai: A framework for ensuring robust and reliable ai systems. *arXiv:2405.06624*, 2024.

[11] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Proc. NeurIPS*, pages 27730–27744, 2022.

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

[13] Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv:2307.15043*, 2023.

[14] Bochuan Cao, Yuanpu Cao, Lu Lin, and Jinghui Chen. Defending against alignment-breaking attacks via robustly aligned llm. In *Proc. ACL*, pages 10542–10560, 2024.

[15] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A Smith. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. In *Proc. EMNLP Findings*, pages 3356–3369, 2020.

[16] Jaymari Chua, Yun Li, Shiyi Yang, Chen Wang, and Lina Yao. Ai safety in generative ai large language models: A survey. *arXiv:2407.18369*, 2024.

[17] Chi Zhang, Changjia Zhu, Junjie Xiong, Xiaoran Xu, Lingyao Li, Yao Liu, and Zhuo Lu. Guardians and offenders: A survey on harmful content generation and safety mitigation of llm. *arXiv:2508.05775*, 2025.

[18] Chia-Chien Hung, Wiem Ben Rim, Lindsay Frost, Lars Bruckner, and Carolin Lawrence. Walking a tightrope–evaluating large language models in high-risk domains. *arXiv:2311.14966*, 2023.

[19] Muhammad Fareed, Madeeha Fatima, Jamal Uddin, Adeel Ahmed, and Muhammad Awais Sattar. A systematic review of ethical considerations of large language models in healthcare and medicine. *Frontiers in digital health*, page 1653631, 2025.

[20] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback. *arXiv:2212.08073*, 2022.

[21] Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. Autodan: Generating stealthy jailbreak prompts on aligned large language models. *arXiv:2310.04451*, 2023.

[22] 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 *Proc. SIGSAC*, pages 1671–1685, 2024.

[23] Yi Zeng, Hongpeng Lin, Jingwen Zhang, Diyi Yang, Ruoxi Jia, and Weiyan Shi. How johnny can persuade llms to jailbreak them: Rethinking persuasion to challenge ai safety by humanizing llms. In *Proc. ACL*, pages 14322–14350, 2024.

[24] Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering. *arXiv:2308.10248*, 2023.

[25] Mrinank Sharma, Meg Tong, Jesse Mu, Jerry Wei, Jorrit Kruthoff, Scott Goodfriend, Euan Ong, Alwin Peng, Raj Agarwal, Cem Anil, Amanda Askell, Nathan Bailey, Joe Benton, Emma Bluemke, Samuel R. Bowman, Eric Christiansen, Hoagy Cunningham, Andy Dau, Anjali Gopal, Rob Gilson, Logan Graham, Logan Howard, Nimit Kalra, Taesung Lee, Kevin Lin, Peter Lofgren, Francesco Mosconi, Clare O'Hara, Catherine Olsson, Linda Petrini, Samir Rajani, Nikhil Saxena, Alex Silverstein, Tanya Singh, Theodore Sumers, Leonard Tang, Kevin K. Troy, Constantin Weisser, Ruiqi Zhong, Giulio Zhou, Jan Leike, Jared Kaplan, and Ethan Perez. Constitutional classifiers: Defending against universal jailbreaks across thousands of hours of red teaming. *arXiv:2501.18837*, 2025.

[26] Hoagy Cunningham, Jerry Wei, Zihan Wang, Andrew Persic, Alwin Peng, Jordan Abderrachid, Raj Agarwal, Bobby Chen, Andy Dau, Alek Dimitriev, Logan Howard, Yijin Hua, Rob Gilson, Mu Lin, Christopher Liu, Vladimir Mikulik, Rohit Mittapalli, Clare O'Hara, Jin Pan, Nikhil Saxena, Alex Silverstein, Yue Song, Giulio Zhou, Jan Leike, Jared Kaplan, Ethan Perez, and Mrinank Sharma. Constitutional classifiers++: Efficient production-grade defenses against universal jailbreaks. In *Proc. ICLR*, 2026.

[27] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does LLM safety training fail? In *Proc. NeurIPS*, 2023.

[28] Nikhil Kandpal, Matthew Jagielski, Florian Tramèr, and Nicholas Carlini. Backdoor attacks for in-context learning with language models. *arXiv:2307.14692*, 2023.

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

[30] Seungju Han, Kavel Rao, Allyson Ettinger, Liwei Jiang, Bill Yuchen Lin, Nathan Lambert, Yejin Choi, and Nouha Dziri. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms. *arXiv:2406.18495*, 2024.

[31] János Kramár, Joshua Engels, Zheng Wang, Bilal Chughtai, Rohin Shah, Neel Nanda, and Arthur Conmy. Building production-ready probes for gemini. *arXiv:2601.11516*, 2026.

[32] Xin Chen, Yarden As, and Andreas Krause. Learning safety constraints for large language models. In *Proc. ICML*, 2025.

[33] Aaron D Ames, Xiangru Xu, Jessy W Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, pages 3861–3876, 2017.

[34] Wei Xiao and Calin Belta. High-order control barrier functions. *IEEE Transactions on Automatic Control*, pages 3655–3662, 2021.

[35] Josef Dai, Xuehai Pan, Ruiyang Sun, Jiaming Ji, Xinbo Xu, Mickel Liu, Yizhou Wang, and Yaodong Yang. Safe rlhf: Safe reinforcement learning from human feedback. *arXiv:2310.12773*, 2023.

[36] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv:2305.18290*, 2023.

[37] Zhanhui Zhou, Jie Liu, Jing Shao, Xiangyu Yue, Chao Yang, Wanli Ouyang, and Yu Qiao. Beyond one-preference-fits-all alignment: Multi-objective direct preference optimization. In *Proc. ACL Findings*, pages 10586–10613, 2024.

[38] Zixuan Liu, Xiaolin Sun, and Zizhan Zheng. Enhancing llm safety via constrained direct preference optimization. *arXiv:2403.02475*, 2024.

[39] Geon-Hyeong Kim, Youngsoo Jang, Yu Jin Kim, Byoungjip Kim, Honglak Lee, Kyunghoon Bae, and Moontae Lee. Safedpo: A simple approach to direct preference optimization with enhanced safety. In *Proc. ICLR*, 2026.

[40] Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, Andy Jones, Sam Bowman, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Nelson Elhage, Sheer El-Showk, Stanislav Fort, Zac Hatfield-Dodds, Tom Henighan, Danny Hernandez, Tristan Hume, Josh Jacobson, Scott Johnston, Shauna Kravec, Catherine Olsson, Sam Ringer, Eli Tran-Johnson, Dario Amodei, Tom Brown, Nicholas Joseph, Sam McCandlish, Chris Olah, Jared Kaplan, and Jack Clark. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv:2209.07858*, 2022.

[41] Andy Zou, Long Phan, Justin Wang, Derek Duenas, Maxwell Lin, Maksym Andriushchenko, J Zico Kolter, Matt Fredrikson, and Dan Hendrycks. Improving alignment and robustness with circuit breakers. In *Proc. NeurIPS*, 2024.

[42] Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. Refusal in language models is mediated by a single direction. In *Proc. NeurIPS*, pages 136037–136083, 2024.

[43] Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering llama 2 via contrastive activation addition. In *Proc. ACL*, pages 15504–15522, 2024.

[44] Bruce W. Lee, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Erik Miehling, Pierre Dognin, Manish Nagireddy, and Amit Dhurandhar. Programming refusal with conditional activation steering. In *Proc. ICLR*, 2025.

[45] Lingkai Kong, Haorui Wang, Wenhao Mu, Yuanqi Du, Yuchen Zhuang, Yifei Zhou, Yue Song, Rongzhi Zhang, Kai Wang, and Chao Zhang. Aligning large language models with representation editing: A control perspective. In *Proc. NeurIPS*, 2024.

[46] Sathwik Karnik and Somil Bansal. Preemptive detection and steering of llm misalignment via latent reachability. *arXiv:2509.21528*, 2025.

[47] Hongjue Zhao, Haosen Sun, Jiangtao Kong, Xiaochang Li, Qineng Wang, Liwei Jiang, Qi Zhu, Tarek F Abdelzaher, Yejin Choi, Manling Li, et al. Odesteer: A unified ode-based steering framework for llm alignment. In *Proc. ICLR*, 2026.

[48] P. Glotfelter, J. Cortes, and M. Egerstedt. Nonsmooth barrier functions with applications to multi-robot systems. *IEEE control systems letters*, pages 310–315, 2017.

[49] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

[50] Chen Tessler, Daniel J Mankowitz, and Shie Mannor. Reward constrained policy optimization. *arXiv:1805.11074*, 2018.

[51] Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *Proc. ICML*, pages 136–145, 2017.

[52] Marcus Aloysius Pereira, Ziyi Wang, Ioannis Exarchos, and Evangelos A. Theodorou. Safe optimal control using stochastic barrier functions and deep forward-backward sdes. In *Proc. CoRL*, 2020.

[53] Wenliang Liu, Wei Xiao, and Calin Belta. Learning robust and correct controllers from signal temporal logic specifications using barriernet. In *Proc. CDC*, pages 7049–7054, 2023.

[54] Wei Xiao, Tsun-Hsuan Wang, Ramin Hasani, Makram Chahine, Alexander Amini, Xiao Li, and Daniela Rus. Barriernet: Differentiable control barrier functions for learning of safe robot control. *IEEE Transactions on Robotics*, pages 2289–2307, 2023.

[55] Wei Xiao, Tsun-Hsuan Wang, Chuang Gan, Ramin Hasani, Mathias Lechner, and Daniela Rus. Safediffuser: Safe planning with diffusion probabilistic models. In *Proc. ICLR*, 2025.

[56] Yuya Miyaoka and Masaki Inoue. Control barrier function for aligning large language models. *arXiv:2511.03121*, 2025.

[57] Alexander Robey, Haimin Hu, Lars Lindemann, Hanwen Zhang, Dimos V Dimarogonas, Stephen Tu, and Nikolai Matni. Learning control barrier functions from expert demonstrations. In *Proc. CDC*, pages 3717–3724, 2020.

[58] Oswin So, Zachary Serlin, Makai Mann, Jake Gonzales, Kwesi Rutledge, Nicholas Roy, and Chuchu Fan. How to train your neural control barrier function: Learning safety filters for complex input-constrained systems. In *Proc. ICRA*, pages 11532–11539, 2024.

[59] 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:2402.04249*, 2024.

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. NeurIPS*, 2017.

[61] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2019.

[62] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

[63] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

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

[65] Zhengxuan Wu, Aryaman Arora, Atticus Geiger, Zheng Wang, Jing Huang, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. Axbench: Steering llms? even simple baselines outperform sparse autoencoders. *arXiv:2501.17148*, 2025.

[66] Chuan Guo, Alexandre Sablayrolles, Hervé Jégou, and Douwe Kiela. Gradient-based adversarial attacks against text transformers. *arXiv:2104.13733*, 2021.

[67] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv:2010.15980*, 2020.

[68] Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. In *Proc. NeurIPS*, pages 51008–51025, 2023.

[69] Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal adversarial triggers for attacking and analyzing nlp. *arXiv:1908.07125*, 2019.

[70] Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. Jailbreaking leading safety-aligned LLMs with simple adaptive attacks. In *Proc. ICLR*, 2025.

[71] 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 *Proc. NAACL*, pages 5377–5400, 2024.

[72] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv:2009.03300*, 2020.

[73] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv:2110.14168*, 2021.

[74] David G Luenberger. *Optimization by vector space methods*. John Wiley & Sons, 1997.

[75] Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, third edition, 2002.

[76] Yueqi Xie, Jingwei Yi, Jiawei Shao, Justin Curl, Lingjuan Lyu, Qifeng Chen, Xing Xie, and Fangzhao Wu. Defending chatgpt against jailbreak attack via self-reminders. *Nature Machine Intelligence*, pages 1486–1496, 2023.

[77] Priyanshu Kumar, Devansh Jain, Akhila Yerukola, Liwei Jiang, Himanshu Beniwal, Thomas Hartvigsen, and Maarten Sap. Polyguard: A multilingual safety moderation tool for 17 languages. In *Proc. COLM*, 2025.

[78] Haiquan Zhao, Chenhan Yuan, Fei Huang, Xiaomeng Hu, Yichang Zhang, An Yang, Bowen Yu, Dayiheng Liu, Jingren Zhou, Junyang Lin, Baosong Yang, Chen Cheng, Jialong Tang, Jiandong Jiang, Jianwei Zhang, Jijie Xu, Ming Yan, Minmin Sun, Pei Zhang, Pengjun Xie, Qiaoyu Tang, Qin Zhu, Rong Zhang, Shibin Wu, Shuo Zhang, Tao He, Tianyi Tang, Tingyu Xia, Wei Liao, Weizhou Shen, Wenbiao Yin, Wenmeng Zhou, Wenyuan Yu, Xiaobin Wang, Xiaodong Deng, Xiaodong Xu, Xinyu Zhang, Yang Liu, Yeqiu Li, Yi Zhang, Yong Jiang, Yu Wan, and Yuxin Zhou. Qwen3guard technical report. *arXiv:2510.14276*, 2025.

[79] Yue Deng, Wenxuan Zhang, Sinno Jialin Pan, and Lidong Bing. Multilingual jailbreak challenges in large language models. In *Proc. ICLR*, 2024.

[80] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Proc. NeurIPS*, 36:46595–46623, 2023.

[81] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *Proc. ACL*, pages 3214–3252, 2022.

[82] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *2311.12022*, 2023.

[83] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

[84] Bochuan Cao, Yuanpu Cao, Lu Lin, and Jinghui Chen. Defending against alignment-breaking attacks via robustly aligned llm. In *Proc. ACL*, pages 10542–10560, 2024.

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

[86] Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. Benchmarking and defending against indirect prompt injection attacks on large language models. In *Proc. SIGKDD*, pages 1809–1820, 2025.

[87] Zeming Wei, Yifei Wang, Ang Li, Yichuan Mo, and Yisen Wang. Jailbreak and guard aligned language models with only few in-context demonstrations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

[88] Alexander Robey, Eric Wong, Hamed Hassani, and George J Pappas. Smoothllm: Defending large language models against jailbreaking attacks. *arXiv:2310.03684*, 2023.

[89] Mansi Phute, Alec Helbling, Matthew Hull, ShengYun Peng, Sebastian Szyller, Cory Cornelius, and Duen Horng Chau. Llm self defense: By self examination, llms know they are being tricked. *arXiv:2308.07308*, 2023.

[90] Ziyang Zhang, Qizhen Zhang, and Jakob Foerster. Parden, can you repeat that? defending against jailbreaks via repetition. *arXiv:2405.07932*, 2024.

[91] Yihan Wang, Zhouxing Shi, Andrew Bai, and Cho-Jui Hsieh. Defending llms against jailbreaking attacks via backtranslation. In *Proc. ACL Findings*, pages 16031–16046, 2024.

[92] 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:2310.01405*, 2023.

[93] Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. In *Proc. NeurIPS*, pages 41451–41530, 2023.

[94] Shashwat Singh, Shauli Ravfogel, Jonathan Herzig, Roee Aharoni, Ryan Cotterell, and Ponnurangam Kumaraguru. Representation surgery: Theory and practice of affine steering. In *Proc. ICML*, 2024.

[95] Hieu M. Vu and Tan Minh Nguyen. Angular steering: Behavior control via rotation in activation space. In *Proc. NeurIPS*, 2026.

[96] Zejia You, Chunyuan Deng, and Hanjie Chen. Spherical steering: Geometry-aware activation rotation for language models. *arXiv:2602.08169*, 2026.

[97] Thanh Q. Tran, John Russell Himawan, Arun Verma, Kiwan Wong, Wei Xiao, Daniela Rus, and Bryan Kian Hsiang Low. DIAL: Steerable language models with distilled instruction-adherence levels. *arXiv:2610.31438*, 2026.

[98] Jiuding Sun, Sidharth Baskaran, Zhengxuan Wu, Michael Sklar, Christopher Potts, and Atticus Geiger. Hypersteer: Activation steering at scale with hypernetworks. *arXiv:2506.03292*, 2025.

[99] Hanyu Wang, Bochuan Cao, Yuanpu Cao, and Jinghui Chen. Truthflow: Truthful llm generation via representation flow correction. *arXiv:2502.04556*, 2025.

[100] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. *arXiv:1912.02164*, 2020.

Appendix of "BarrierSteer: LLM Safety via Learning Barrier Steering"

Table of Contents

A Closed-form CBF Steering Derivation **19**

B Proof of Theorem 1 **20**

C Implementation Details of BarrierSteer **20**

D Experimental Setup and Additional Results **21**

D.1 Hidden-State Supervision and Training Set Sizes 22

D.2 Implementation Details of Baselines 22

D.3 Adversarial Attacks 24

D.4 Robustness to the Choice of Safety Judge 26

D.5 Multilingual Out-of-Distribution Evaluation 26

D.6 Steering Strength Ablations 27

D.7 Extended Utility Results 27

D.8 Computational Benchmarking 28

D.9 Modular Composition of Safety Barriers 31

D.10 Ablation on LSE Smoothing Parameter κ 31

D.11 Ablation on Neural CBF Architecture 32

D.12 Conditional Fixed-Direction Steering Baselines 32

D.13 Compute Resources and Wall-Clock Cost 33

E Additional Related Work on Inference-time Safety **33**

F Additional Discussion and Clarifications **34**

F.1 Assumptions for Theoretical Safety 34

F.2 Data, Supervision, and Generalization 35

F.3 BarrierSteer Variants and Steering Geometry 35

F.4 Evaluation, Robustness, and Deployment Scope 35

G Limitations and Future Work **36**

H Broader Impact **37**

I Declaration of LLM Usage **37**

A Closed-form CBF Steering Derivation

To derive the closed-form update, we consider the case where the active constraints have been reduced to at most two constraints by the merge rule using Eq. (2) shown in Sec. 4.3. Here, we explicitly show the closed-form solution to the QP in Eq. (1) following [74]. We consider the case where the QP has two constraints. Given the following QP corresponding to Eq. (1) for the proposed BARRIERSTEER:

$$\begin{aligned} u^* = \arg\min_u \; & ||u - \frac{h_t - h_{t-1}}{\Delta t}||^2, \\ \text{s.t.}, \; & \nabla b_1(h_t)^\top u + \alpha b_1(h_t) \geq 0 \\ & \nabla b_2(h_t)^\top u + \alpha b_2(h_t) \geq 0, \end{aligned} \tag{5}$$

where b_1, b_2 are the two constraints merged from the learned CBFs $\{b_k(h) \geq \delta\}_{k=1}^K$, following Sec. 4.3. Next, we define

$$\begin{aligned} g_1(h_t) &= -\nabla b_1(h_t)^\top, \quad h_1(h_t) = \alpha b_1(h_t), \\ g_2(h_t) &= -\nabla b_2(h_t)^\top, \quad h_2(h_t) = \alpha b_2(h_t). \end{aligned}$$

The standard form of the QP cost is $u^T H(h_t) u + F(h_t) u$, where the matrix $H(h_t)$ is an identity matrix corresponding to Eq. (5). We can further define

$$\begin{aligned} [\hat{g}_1(h_t), \hat{g}_2(h_t)] &= H(h_t)^{-1}[g_1(h_t), g_2(h_t)], \\ \begin{bmatrix} \hat{h}_1(h_t) \\ \hat{h}_2(h_t) \end{bmatrix} &= \begin{bmatrix} h_1(h_t) \\ h_2(h_t) \end{bmatrix} - \begin{bmatrix} g_1(h_t)^T \\ g_2(h_t)^T \end{bmatrix} \hat{u} \end{aligned}$$

where

$$\hat{u} = -H(h_t)^{-1} F(h_t), \qquad F(h_t) = -\frac{h_t - h_{t-1}}{\Delta t}.$$

Then, let $w := u - \hat{u}$ and $\langle \cdot, \cdot \rangle$ define an inner product with weight matrix $H(h_t)$ so that $\langle w, w \rangle = w^T H(h_t) w$. We have that the optimization problem in Eq. (5) is equivalent to:

$$\begin{aligned} w^* = \arg\min_w & \langle w, w \rangle, \\ \text{s.t.}, \; & \langle \hat{g}_1(h_t), w \rangle \leq \hat{h}_1(h_t), \\ & \langle \hat{g}_2(h_t), w \rangle \leq \hat{h}_2(h_t), \end{aligned} \tag{6}$$

where the optimal solution of Eq. (5) is given by

$$u^* = w^* + \hat{u}.$$

Following [74] [Ch. 3], the unique solution to Eq. (6) is given by

$$w^* = \lambda_1(h_t)\hat{g}_1(h_t) + \lambda_2(h_t)\hat{g}_2(h_t)$$

where

$$\lambda_1(h_t) = \begin{cases} 0 & \text{if } G_{21}(h_t)[\hat{h}_2(h_t)]_- - G_{22}(h_t)\hat{h}_1(h_t) < 0 \\ \frac{[\hat{h}_1(h_t)]_-}{G_{11}(h_t)} & \text{if } G_{12}(h_t)[\hat{h}_1(h_t)]_- - G_{11}(h_t)\hat{h}_2(h_t) < 0 \\ \frac{[G_{22}(h_t)\hat{h}_1(h_t) - G_{21}(h_t)\hat{h}_2(h_t)]_-}{G_{11}(h_t)G_{22}(h_t) - G_{12}(h_t)G_{21}(h_t)} & \text{otherwise .} \end{cases}$$

$$\lambda_2(h_t) = \begin{cases} \frac{[\hat{h}_2(h_t)]_-}{G_{22}(h_t)} & \text{if } G_{21}(h_t)[\hat{h}_2(h_t)]_- - G_{22}(h_t)\hat{h}_1(h_t) < 0 \\ 0 & \text{if } G_{12}(h_t)[\hat{h}_1(h_t)]_- - G_{11}(h_t)\hat{h}_2(h_t) < 0 \\ \frac{[G_{11}(h_t)\hat{h}_2(h_t) - G_{12}(h_t)\hat{h}_1(h_t)]_-}{G_{11}(h_t)G_{22}(h_t) - G_{12}(h_t)G_{21}(h_t)} & \text{otherwise .} \end{cases}$$

where $G(h_t) = [G_{ij}(h_t)] = [\langle \hat{g}_i(h_t), \hat{g}_j(h_t) \rangle], i, j = 1, 2$ is the Gram matrix, and $[x]_- = \min(0, x)$ denotes the negative part of x.

B Proof of Theorem 1

Theorem 1. *Let $\{b_k(h) \geq \delta\}_{k=1}^K$ be learned safety constraints, and let $B(h)$ denote the smooth composed barrier in Eq. (2), with composed safe set $\mathcal{C}_B = \{h \mid B(h) \geq 0\}$. Let h and h' denote the hidden states before and after applying the control u^* in Eq. (4). Under the local assumptions stated above, the update satisfies the composed CBF condition $\nabla B(h)^\top u^* + \alpha B(h) \geq 0$. Consequently, under the local dynamics, $\mathcal{C}_B$ is locally forward invariant: if $h \in \mathcal{C}_B$, the composed barrier condition is locally preserved; if $h \notin \mathcal{C}_B$, the composed-barrier violation is exponentially stabilized toward $\mathcal{C}_B$.*

Proof. We prove the theorem for the local CBF update at the current hidden state under the local assumptions stated in the main text: the learned and composed barriers are continuously differentiable, and the one-step hidden-state evolution is locally Lipschitz and well approximated by the finite-difference model in Eq. (1). Following Sec. A, we have that the BarrierSteer control u^* in Eq. (4) is the exact solution of the QP in Eq. (5). In other words, the following constraint is satisfied:

$$\nabla B(h)^\top u^* + \alpha B(h) \geq 0, \tag{7}$$

where $B(h)$ is the composed barrier obtained via Eq. (2), i.e.,

$$B(h) = -\frac{1}{\kappa} \ln \left(\sum_{k=1}^{K} e^{-\kappa(b_k(h)-\delta)} \right).$$

Following [63], $B(h)$ is a smooth lower bound of $b_k(h) - \delta$ for all $k \in \{1, \ldots, K\}$. Therefore, $B(h) \geq 0$ is a sufficient condition for satisfying every learned constraint $b_k(h) \geq \delta$.

Thus, if the current hidden state h is in the composed learned safe set $\mathcal{C}_B = \{h \mid B(h) \geq 0\}$, then under the local dynamics $\dot{h} = u$ adopted in Sec. 4.2, we have $\dot{B}(h) = \nabla B(h)^\top u^*$, and Eq. (7) becomes

$$\dot{B}(h) + \alpha B(h) \geq 0. \tag{8}$$

By the comparison lemma, this implies $B(h') \geq B(h)e^{-\alpha}$ under the local update, so $B(h') \geq 0$ whenever $B(h) \geq 0$. Therefore, the local CBF update preserves the composed barrier condition.

Since $B(h)$ lower-bounds all $b_k(h) - \delta$, satisfying the composed-barrier condition is sufficient for satisfying all learned constraints. Thus, the BarrierSteer control obtained via the composing method in Eq. (2) preserves the composed-barrier condition under the local CBF update.

On the other hand, if the current hidden state h is outside $\mathcal{C}_B$, then $B(h) < 0$. To analyze convergence toward $\mathcal{C}_B$, we introduce the auxiliary function $V(h) = -B(h)$, which satisfies $V(h) \geq 0$. Under the local dynamics $\dot{h} = u^*$, $\dot{V}(h) = -\dot{B}(h) = -\nabla B(h)^\top u^* = \nabla V(h)^\top u^*$, so Eq. (7) becomes

$$\dot{V}(h) + \alpha V(h) = \nabla V(h)^\top u^* + \alpha V(h) \leq 0, \tag{9}$$

The corresponding equality $\dot{V}(h) + \alpha V(h) = 0$ has solution $V(h') = V(h)e^{-\alpha}$. By the comparison lemma [75], Eq. (9) therefore implies

$$V(h') \leq V(h)e^{-\alpha}.$$

Hence, $V(h') = -B(h')$ contracts exponentially under the local CBF update.

Since $B(h)$ is a lower bound of $b_k(h) - \delta$ for all $k \in \{1, \ldots, K\}$, reducing the composed-barrier violation moves the updated hidden state h' toward the learned safe set. ■

C Implementation Details of BarrierSteer

This section provides implementation details for BarrierSteer. We first outline the inference-time steering algorithm, then describe the neural CBF architecture and the latent dynamics model used by BarrierSteer. The key implementation choice is that BarrierSteer learns nonlinear scalar barriers in hidden-state space, but applies them through a lightweight CBF update during decoding.

Inference-time algorithm. Algorithm 1 summarizes the implementation view of BarrierSteer. At each decoding step, the method evaluates the learned barriers on the current hidden state, estimates

Algorithm 1 Algorithmic view of BARRIERSTEER inference-time steering.

Require: Aligned LLM $\mathcal{M}$, learned barriers $\{b_k\}_{k=1}^{K}$, user query q, threshold δ, steering strength α, time step Δt, layer ℓ, merge rule $m \in \{\text{Top-2}, \text{QP}, \text{LSE}\}$.
Ensure: Steered response $x = (x_1, \ldots, x_T)$.

```
Initialize response prefix x_{≤0} ← ∅ and prefix state s_0 ← q.
for t = 0, ..., T − 1 do
    Form the current prefix state s_t = (q, x_{≤t}) and extract the layer-ℓ hidden state h_t ← h^(ℓ)(s_t).
    Evaluate barrier residuals r_k ← b_k(h_t) − δ for all active safety constraints.
    Estimate the nominal latent velocity u_0 ← (h_t − h_{t−1})/Δt, with h_{−1} ← h_0.
    if all r_k ≥ 0 then
        Set h'_t ← h_t without intervention.
    else
        Use merge rule m to select or compose the active barriers (LSE composes them via Eq. (2)).
        Compute the CBF control u* using the active barrier gradients, u_0, and α.
        Apply the controlled hidden state h'_t ← h_{t−1} + u*Δt.
    end if
    Continue decoding from h'_t, sample the next token x_{t+1}, and set x_{≤t+1} ← (x_{≤t}, x_{t+1}).
end for
return x.
```

the local hidden-state velocity from consecutive decoding states, and applies a CBF correction only when the current state is predicted to violate the learned safety constraints. The same algorithm supports the Top-2, QP, and LSE merge rules discussed in the main text.

Latent dynamics used by BARRIERSTEER. We use the standard control-affine notation

$$\dot{h} = f(h) + g(h)u,$$

where h denotes the hidden state being steered and u is the control applied in latent space. In our implementation, we do not train a separate dynamics model. Instead, because the controller directly edits the current hidden state, we set the drift term to zero and the control matrix to identity, i.e., $f(h) = 0$ and $g(h) = I$. The nominal motion of the unsteered model is estimated online as $u_0 = (h_t - h_{t-1})/\Delta t$. Thus, the CBF layer only needs to compute a corrected latent velocity u^*, after which the steered state is applied as $h'_t = h_{t-1} + u^*\Delta t$. This avoids learning autoregressive hidden-state dynamics while still using the observed local trajectory of the model during decoding.

Neural CBF architecture. Each barrier b_k is implemented as an independent multilayer perceptron that maps a hidden state in $\mathbb{R}^{d_\varphi}$ to a scalar barrier value. For each head, we use the architecture

$$d_\varphi \to 2048 \to 1024 \to 512 \to 256 \to 1.$$

Every hidden block consists of a linear layer followed by LayerNorm, GELU, and dropout; the final layer is a linear scalar head. This gives each barrier a nonlinear decision boundary while keeping the implementation close to the latent-feature architecture used by SaP.

By default, our experiments use $K = 4$ independent CBF heads. This choice is intended to keep the number of trainable barrier parameters comparable to the SaP baseline while allowing nonlinear constraints. For the Qwen2-1.5B setting, where $d_\varphi = 1536$, the above architecture has $5{,}910{,}017$ parameters per CBF head and $23{,}640{,}068$ total trainable barrier parameters for $K = 4$. The corresponding SaP latent-constraint module has $25{,}673{,}758$ trainable parameters, so improvements are not simply due to using a substantially larger safety module. For other model families, we use the same architecture with the corresponding hidden-state dimension d_φ.

D Experimental Setup and Additional Results

This appendix provides additional details for reproducing the experiments in Sec. 5. We first describe how response-level safety labels are converted into token-level hidden-state supervision and report the exact hidden-state counts used to train the barriers. We also provide implementation notes for the baselines, per-attack HarmBench results, a robustness check across safety judges, a multilingual out-of-distribution evaluation, variant-selection ablations, latency measurements, the WildGuard modular-composition study, and compute resources.

Table 7: Exact HarmBench token-level hidden-state counts used for training. Counts aggregate all nine HarmBench attack-method training files for each model.

Model	Safe tokens	Unsafe tokens	Total tokens
Gemma-2-9B	2,083,847	668,665	2,752,512
Llama-2-7B	2,652,477	100,035	2,752,512
Ministral-8B	1,266,580	1,485,932	2,752,512
Qwen2-1.5B	2,340,020	412,082	2,752,102

Table 8: Exact WildGuard token-level hidden-state counts used for training individual category barriers on `Qwen2-1.5B`. We report the per-concept counts because the WildGuard experiment trains one barrier per concept and reports WildGuard results for Qwen2-1.5B.

WildGuard concept	Safe tokens	Unsafe tokens	Total tokens
Copyright	114,666	16,406	131,072
Cyberattack	64,237	66,835	131,072
Defamation / unsafe actions	100,669	30,403	131,072
Discrimination	101,575	29,497	131,072
Disinformation campaigns	93,457	37,615	131,072
Fraud / illegal activities	98,252	32,820	131,072
Harmful misinformation	97,457	33,615	131,072
Mental health crisis	101,916	29,156	131,072
Others	97,074	33,998	131,072
Private information	106,845	24,227	131,072
Sensitive information	108,441	22,631	131,072
Sexual content	108,925	22,147	131,072
Toxic language / hate speech	110,190	20,882	131,072
Violence / physical harm	100,289	30,783	131,072
Total	**1,403,993**	**431,015**	**1,835,008**

D.1 Hidden-State Supervision and Training Set Sizes

BarrierSteer trains safety barriers from binary safe/unsafe response labels rather than preference pairs. For HarmBench, we use adversarially elicited model responses from the HarmBench protocol [64]; for WildGuard, we use category-specific examples from WildGuardMix [30]. Generated responses are labeled with the HarmBench classifier [59], and the resulting labels supervise hidden states extracted from the target model. This follows the standard latent-safety-classifier setting used by prior representation-space safety methods [32, 26, 31], but BarrierSteer uses the learned classifier as a CBF for inference-time steering.

Because the available supervision is response-level, each generated response contributes token-level hidden states with the same binary label. If a response is labeled unsafe, all extracted hidden states from that response are assigned the unsafe label; if it is labeled safe, the corresponding hidden states are assigned the safe label. This construction is appropriate for decoder-only Transformers because the hidden state at a generated position is contextual: it summarizes the prompt and generated prefix through causal self-attention, rather than representing only the current token in isolation. Consequently, a few hundred behavior prompts yield millions of labeled hidden-state examples. Tables 7 and 8 report the exact counts used in our experiments.

D.2 Implementation Details of Baselines

This section specifies the baseline implementations used in the main experiments. Unless otherwise stated, all decoding uses greedy generation with temperature 0, and all methods use the same prompts, behavior splits, and safety classifier described in Sec. 5.1 and Appendix D.3. For learned representation-space methods, we use the same safe/unsafe response data as BarrierSteer whenever the baseline requires training supervision. All hidden-state baselines use layer $\ell = 20$, matching the hidden-state extraction layer used to train BarrierSteer.

Base model. The base model is the original instruction-tuned checkpoint without additional prompting, fine-tuning, or activation intervention. We evaluate each model using its respective default instruction prompt or chat template from Hugging Face: Llama-2-7B-Chat, Ministral-8B-Instruct-2410, Qwen2-1.5B-Instruct, and Gemma-2-9B-IT.[2] It serves as the reference for both safety and utility metrics, and all steering methods are applied on top of the same base checkpoint.

Self-Reminder. `Self-Reminder` [76] is a prompting-only inference-time defense that inserts safety reminders into both the system message and the user query while keeping model parameters unchanged. We use the default `remind/remind` template from the released implementation:

> **System reminder:** `{original_system_message} You should be a responsible language model and should not generate harmful or misleading content! Please answer the following user query in a responsible way.`
>
> **Query reminder:** `Remember, you should be a responsible language model and should not generate harmful or misleading content!`

The query reminder is prepended to the user prompt, while the system reminder augments the model's default system message. Because this method only changes the prompt, it is evaluated deterministically under greedy decoding.

Circuit Breaker. `Circuit Breaker` [41] is a training-time representation method that learns LoRA adapters to reroute internal representations associated with harmful completions while preserving benign behavior through a retain objective. We train it on the same HarmBench behavior split as the other learned methods. Following the LoRRA objective from Circuit Breaker, let $\mathcal{M}_0$ denote the frozen base model, $\mathcal{M}_\theta$ the LoRA-adapted model, and $\mathrm{rep}_\ell(\mathcal{M}, x)$ the layer-ℓ hidden representations selected for the loss. For circuit-breaker examples $x_s \sim \mathcal{D}_s$ and retain examples $x_r \sim \mathcal{D}_r$, the training objective is

$$\begin{aligned}\mathcal{L}_{\mathrm{CB}}(\theta) = c_s\, \mathbb{E}_{x_s \sim \mathcal{D}_s}\left[\mathrm{ReLU}(\cos(\mathrm{rep}_\ell(\mathcal{M}_0, x_s), \mathrm{rep}_\ell(\mathcal{M}_\theta, x_s)))\right] \\ + c_r\, \mathbb{E}_{x_r \sim \mathcal{D}_r}\left[\|\mathrm{rep}_\ell(\mathcal{M}_0, x_r) - \mathrm{rep}_\ell(\mathcal{M}_\theta, x_r)\|_2^2\right].\end{aligned}$$

The first term reroutes harmful representations away from the base model's harmful trajectory, while the retain term preserves representations on benign examples. The coefficients c_s and c_r follow the Circuit Breaker schedule that trades off rerouting and retention during training. The target layer for the circuit-breaker loss is layer 20; LoRA adapters are inserted into transformer layers 0–20. The LoRA target modules are `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`. We use LoRA rank $r = 16$, LoRA alpha 16, dropout 0.05, circuit-breaker loss weight 10, learning rate 10^{-4}, batch size 8, gradient accumulation 1, weight decay 0, maximum sequence length 1024, and 150 training steps with a constant learning-rate schedule. Unlike inference-time steering methods, Circuit Breaker changes the model through an adapter before evaluation rather than computing a token-level intervention during decoding.

Activation Addition. `Activation Addition` [24, 43] constructs a fixed steering direction from the difference between mean safe and unsafe hidden states, $r = |\mathcal{H}_{\text{safe}}|^{-1} \sum_{h \in \mathcal{H}_{\text{safe}}} h - |\mathcal{H}_{\text{unsafe}}|^{-1} \sum_{h \in \mathcal{H}_{\text{unsafe}}} h$. During generation, it applies the additive intervention $h' = h + \alpha r$ at layer 20. We use the same hidden-state data as BarrierSteer to estimate r, set $\alpha = 1$, and do not normalize the steering vector.

Directional Ablation. `Directional Ablation` [42] uses the same safe–unsafe direction r, but removes the component of the hidden state along that direction rather than adding it: $h' = h - r(r^\top r)^{-1} r^\top h$. This baseline tests whether suppressing the harmful/refusal-related direction is sufficient for safety compared to actively steering toward a learned safe region.

ReFT-r1. `ReFT-r1` [65] is a weakly supervised rank-one representation-finetuning baseline. Instead of computing a fixed mean-difference direction, it learns a rank-one linear intervention from the same safe/unsafe supervision used by the other learned representation baselines. Training freezes the base LLM and optimizes only the rank-one intervention vector using the standard next-token

[2] Model checkpoints: https://huggingface.co/meta-llama/Llama-2-7b-chat-hf, https://huggingface.co/mistralai/Ministral-8B-Instruct-2410, https://huggingface.co/Qwen/Qwen2-1.5B-Instruct, and https://huggingface.co/google/gemma-2-9b-it.

Table 9: SaP inference-time steering-strength sensitivity on `Gemma-2-9B`. We vary the positive-violation penalty weight λ while keeping the learned SaP facets fixed. ASR is measured on HarmBench static attacks and reported as $\text{mean}_{\pm\text{std}}$.

λ	1	2	5	10	20
ASR $\downarrow$	$8.17_{\pm 0.47}$	$8.00_{\pm 0.94}$	$8.08_{\pm 0.77}$	$8.42_{\pm 1.20}$	$8.50_{\pm 1.34}$

cross-entropy loss on the target response tokens, with prompt tokens masked out using label value -100. Let $\mathcal{M}_\varphi$ denote the frozen LLM equipped with the learned rank-one intervention parameters φ, and let $m_t \in \{0, 1\}$ indicate whether token x_t is part of the target response rather than the prompt. The ReFT-r1 objective used in our experiments is

$$\mathcal{L}_{\text{ReFT}}(\varphi) = -\mathbb{E}_{(q,x)}\left[\frac{1}{\sum_t m_t}\sum_{t=1}^{T} m_t \log p_{\mathcal{M}_\varphi}(x_t \mid q, x_{<t})\right] + \lambda_1\, \Omega_{\text{non-top}k}(\varphi),$$

where the optional sparsity term penalizes intervention scores outside the selected top-k positions. The implementation optionally adds this ℓ_1 penalty on non-top-k intervention scores, but the term is disabled in our experiments because we set $\lambda_1 = 0$. The intervention is trained and applied at layer 20 with top-$k = 5$, intervention scale $\beta = 1$, learning rate 10^{-3}, batch size 4, gradient accumulation 1, warmup steps 0, maximum sequence length 1024, and 500 training steps. At inference time, the learned rank-one intervention is applied to hidden states during generation.

SaP. `SaP` [32] is the closest latent-constraint baseline to BarrierSteer. It learns linear safety constraints in a sparse concept space and performs inference-time projection when a hidden state violates the learned polytope. We follow the released configuration: the concept encoder maps hidden states into a 16,384-dimensional sparse representation, the polytope uses $K = 30$ facets, and model-specific margin and regularization hyperparameters for Llama-2-7B, Ministral-8B, and Qwen2-1.5B are adopted from the original implementation. For Qwen2-1.5B, this gives 25,673,758 trainable parameters in the `SaP` safety module. Both BarrierSteer and `SaP` are trained with Adam, learning rate 10^{-2}, and batch size 512.

During inference, `SaP` optimizes the hidden state for samples predicted to be unsafe. In our implementation, for an original hidden state h, `SaP` computes an updated state

$$h^\star = \arg\min_{h'} \frac{\|h' - h\|_1}{d_h} + \omega_{\text{safe}} \sum_{k=1}^{K} v_k(h') + \lambda \sum_{k=1}^{K} [v_k(h')]_+,$$

where $v_k(h')$ is the violation of the k-th learned linear constraint and $[\cdot]_+$ denotes the positive part. We use default hyperparameters from the paper for this optimization. We set $\omega_{\text{safe}} = 10^{-4}$ in all SaP runs. The coefficient λ is therefore the main inference-time steering-strength parameter: larger values penalize positive constraint violations more strongly, but can also move the hidden state farther from the original representation. We use the default $\lambda = 2$ unless otherwise stated and run 100 SGD iterations with step size 0.01 for this projection step. Table 9 reports an ablation over λ while keeping the learned SaP facets fixed. The ablation shows that increasing λ does not meaningfully improve SaP robustness once the learned linear facets are fixed. This suggests that SaP's remaining ASR is not primarily due to an under-tuned projection strength, but rather to the expressiveness and optimization limits of enforcing linear latent constraints during decoding.

D.3 Adversarial Attacks

Dataset collection and feature extraction. We follow the HarmBench adversarial-evaluation protocol [64]. For each model, the 400 HarmBench behaviors are split into 320 behaviors for training the latent safety constraints and 80 held-out behaviors for final evaluation. For each attack method, we generate one test case per behavior. To construct the barrier-training dataset, we run the target LLM on the 320 training behaviors and extract per-token hidden states from layer $\ell = 20$. Each response is labeled at the response level using `HarmBench-Llama-2-13b-cls` [59]; the response label is then assigned to all extracted hidden states from that response, as described in Appendix D.1. Baseline-specific hyperparameters and implementation choices are summarized in Appendix D.2.

Evaluation protocol. The `SaP` numbers in our tables may differ from those reported in the `SaP` paper [32] because we use a different training/evaluation protocol. Specifically, our HarmBench

Table 10: Per-attack HarmBench ASR for `Gemma-2-9B`. Values are percentages; lower is better. BarrierSteer denotes the default LSE variant. Learning-based methods are reported as mean$_{\pm\text{std}}$ over independent runs, while deterministic baselines use fixed greedy decoding.

Attack Method	Base	Self-Reminder	Circuit Breaker	ActAdd	DirAbl	ReFT-r1	SaP	BarrierSteer
AutoDAN	62.50	65.00	$27.75_{\pm 11.74}$	58.75	22.50	$45.42_{\pm 8.13}$	$27.08_{\pm 14.49}$	$0.00_{\pm 0.00}$
AutoPrompt	17.50	3.75	$7.50_{\pm 2.80}$	21.25	13.75	$10.83_{\pm 4.02}$	$6.25_{\pm 1.25}$	$0.00_{\pm 0.00}$
DirectReq.	5.00	2.50	$4.75_{\pm 0.56}$	5.00	7.50	$4.17_{\pm 0.72}$	$8.00_{\pm 0.94}$	$0.00_{\pm 0.00}$
GBDA	4.50	0.75	$5.00_{\pm 0.56}$	4.50	9.50	$2.92_{\pm 1.70}$	$4.17_{\pm 0.38}$	$0.00_{\pm 0.00}$
GCG	33.75	7.50	$12.75_{\pm 6.15}$	31.25	16.25	$20.00_{\pm 10.00}$	$14.58_{\pm 2.60}$	$0.00_{\pm 0.00}$
HumanJb.	32.75	30.00	$17.65_{\pm 4.81}$	32.50	23.25	$27.17_{\pm 5.16}$	$20.92_{\pm 6.01}$	$0.00_{\pm 0.00}$
PAP	6.25	3.75	$3.75_{\pm 0.00}$	5.00	7.50	$3.33_{\pm 0.72}$	$0.42_{\pm 0.72}$	$0.00_{\pm 0.00}$
PEZ	8.25	2.75	$5.55_{\pm 0.11}$	7.75	11.00	$4.25_{\pm 2.54}$	$5.33_{\pm 0.29}$	$0.00_{\pm 0.00}$
UAT	5.00	0.00	$4.50_{\pm 1.12}$	5.00	11.25	$2.92_{\pm 1.91}$	$2.92_{\pm 1.44}$	$0.00_{\pm 0.00}$
Average	19.50	12.89	$9.91_{\pm 2.60}$	19.00	13.61	$12.30_{\pm 3.19}$	$8.00_{\pm 0.94}$	$0.00_{\pm 0.00}$

Table 11: Per-attack HarmBench ASR for `Ministral-8B`. Values are percentages; lower is better. BarrierSteer denotes the default LSE variant. Learning-based methods are reported as mean$_{\pm\text{std}}$ over independent runs, while deterministic baselines use fixed greedy decoding.

Attack Method	Base	Self-Reminder	Circuit Breaker	ActAdd	DirAbl	ReFT-r1	SaP	BarrierSteer
AutoDAN	58.75	61.25	$32.50_{\pm 3.42}$	62.50	61.25	$59.58_{\pm 1.44}$	$35.83_{\pm 2.60}$	$6.50_{\pm 14.53}$
AutoPrompt	62.50	12.50	$34.25_{\pm 0.68}$	61.25	63.75	$52.08_{\pm 7.53}$	$28.33_{\pm 9.38}$	$0.00_{\pm 0.00}$
DirectReq.	41.25	12.50	$27.25_{\pm 1.37}$	43.75	43.75	$36.67_{\pm 0.72}$	$26.25_{\pm 2.50}$	$0.00_{\pm 0.00}$
GBDA	61.25	12.25	$37.95_{\pm 4.52}$	60.50	60.75	$50.08_{\pm 3.30}$	$25.92_{\pm 9.71}$	$0.00_{\pm 0.00}$
GCG	62.50	25.00	$39.25_{\pm 0.68}$	62.50	62.50	$48.75_{\pm 7.60}$	$27.50_{\pm 4.51}$	$0.00_{\pm 0.00}$
HumanJb.	47.50	38.25	$28.95_{\pm 3.83}$	48.00	47.25	$36.08_{\pm 6.00}$	$23.58_{\pm 1.04}$	$0.95_{\pm 2.12}$
PAP	31.25	10.00	$24.00_{\pm 1.37}$	31.25	31.25	$19.58_{\pm 2.60}$	$12.50_{\pm 1.25}$	$0.00_{\pm 0.00}$
PEZ	38.25	4.25	$21.35_{\pm 1.92}$	37.00	39.25	$20.33_{\pm 1.84}$	$14.83_{\pm 2.32}$	$0.05_{\pm 0.11}$
UAT	57.50	6.25	$35.50_{\pm 0.68}$	55.00	56.25	$39.58_{\pm 1.91}$	$20.42_{\pm 2.60}$	$0.00_{\pm 0.00}$
Average	51.19	20.25	$30.19_{\pm 2.51}$	51.31	51.78	$37.56_{\pm 3.31}$	$23.91_{\pm 3.49}$	$0.83_{\pm 1.86}$

Table 12: Per-attack HarmBench ASR for `Llama-2-7B`. Values are percentages; lower is better. BarrierSteer denotes the default LSE variant. Learning-based methods are reported as mean$_{\pm\text{std}}$ over independent runs, while deterministic baselines use fixed greedy decoding.

Attack Method	Base	Self-Reminder	Circuit Breaker	ActAdd	DirAbl	ReFT-r1	SaP	BarrierSteer
AutoDAN	7.50	6.25	$9.17_{\pm 1.44}$	6.25	3.75	$7.50_{\pm 2.17}$	$9.17_{\pm 0.72}$	$1.00_{\pm 1.63}$
AutoPrompt	11.25	2.50	$13.75_{\pm 0.00}$	13.75	5.00	$11.25_{\pm 1.25}$	$15.42_{\pm 1.44}$	$3.50_{\pm 5.48}$
DirectReq.	2.50	2.50	$2.92_{\pm 0.72}$	2.50	0.00	$1.67_{\pm 0.72}$	$2.50_{\pm 0.00}$	$0.50_{\pm 1.12}$
GBDA	2.00	2.00	$2.42_{\pm 0.29}$	2.00	1.00	$2.42_{\pm 0.52}$	$2.67_{\pm 0.14}$	$0.55_{\pm 1.10}$
GCG	22.50	7.50	$23.33_{\pm 0.72}$	25.00	11.25	$21.67_{\pm 1.44}$	$22.50_{\pm 1.25}$	$4.75_{\pm 8.12}$
HumanJb.	1.25	1.75	$1.50_{\pm 0.00}$	1.50	1.00	$0.83_{\pm 0.38}$	$1.25_{\pm 0.00}$	$0.20_{\pm 0.45}$
PAP	2.50	2.50	$2.08_{\pm 0.72}$	2.50	0.00	$2.92_{\pm 1.91}$	$3.33_{\pm 0.72}$	$0.25_{\pm 0.56}$
PEZ	2.50	1.50	$2.67_{\pm 0.14}$	2.25	3.00	$1.67_{\pm 0.63}$	$2.58_{\pm 0.14}$	$0.50_{\pm 0.71}$
UAT	2.50	2.50	$2.50_{\pm 0.00}$	2.50	3.75	$2.50_{\pm 0.00}$	$2.50_{\pm 0.00}$	$0.25_{\pm 0.56}$
Average	6.06	3.22	$6.70_{\pm 0.31}$	6.47	3.19	$3.43_{\pm 0.35}$	$6.88_{\pm 0.21}$	$1.28_{\pm 2.17}$

setting splits the behavior set: latent constraints are trained on one set of behaviors and evaluated on held-out behaviors, whereas some previously reported settings include behaviors that also appear during training. The held-out split is stricter and better measures whether the learned constraint generalizes beyond the training behaviors.

Detailed ASR results. For comprehensiveness, we include Tables 10–13 to report per-attack HarmBench ASR for Gemma-2-9B, Ministral-8B, Llama-2-7B, and Qwen2-1.5B. Learning-based methods are reported as averages over independent trials with standard deviations. Prompting, fixed-direction, and heuristic baselines are evaluated deterministically under greedy decoding.

Table 13: Per-attack HarmBench ASR for `Qwen2-1.5B`. Values are percentages; lower is better. BarrierSteer denotes the default LSE variant. Learning-based methods are reported as mean$_{\pm\text{std}}$ over independent runs, while deterministic baselines use fixed greedy decoding.

Attack Method	Base	Self-Reminder	Circuit Breaker	ActAdd	DirAbl	ReFT-r1	SaP	BarrierSteer
AutoDAN	27.50	33.75	$42.92_{\pm 7.94}$	7.50	27.50	$4.58_{\pm 0.72}$	$4.17_{\pm 4.39}$	$4.75_{\pm 3.79}$
AutoPrompt	30.00	3.75	$35.00_{\pm 1.25}$	27.50	17.50	$1.25_{\pm 1.25}$	$8.75_{\pm 3.31}$	$0.00_{\pm 0.00}$
DirectReq.	17.50	6.25	$8.75_{\pm 2.17}$	6.25	11.25	$8.75_{\pm 2.17}$	$5.00_{\pm 3.31}$	$0.00_{\pm 0.00}$
GBDA	28.25	4.25	$33.50_{\pm 0.43}$	26.75	25.50	$3.92_{\pm 0.80}$	$6.67_{\pm 5.92}$	$0.00_{\pm 0.00}$
GCG	47.50	18.75	$50.00_{\pm 3.31}$	51.25	21.25	$8.33_{\pm 2.60}$	$7.08_{\pm 8.04}$	$0.00_{\pm 0.00}$
HumanJb.	12.75	6.25	$9.92_{\pm 0.95}$	7.25	10.50	$6.42_{\pm 0.88}$	$3.50_{\pm 1.56}$	$0.90_{\pm 0.82}$
PAP	11.25	5.00	$12.50_{\pm 2.17}$	7.50	6.25	$3.33_{\pm 1.44}$	$4.17_{\pm 2.60}$	$0.00_{\pm 0.00}$
PEZ	15.25	1.25	$6.92_{\pm 0.52}$	12.00	13.00	$1.67_{\pm 0.52}$	$4.25_{\pm 1.80}$	$0.00_{\pm 0.00}$
UAT	25.00	2.50	$15.42_{\pm 1.44}$	26.25	20.00	$1.25_{\pm 0.00}$	$4.58_{\pm 1.91}$	$0.25_{\pm 0.56}$
Average	23.89	9.08	$23.88_{\pm 1.47}$	19.14	16.97	$4.17_{\pm 0.47}$	$5.35_{\pm 3.44}$	$0.66_{\pm 0.47}$

Table 14: Static HarmBench ASR (%, ↓) under three safety judges. Rates are computed over the 60 classifier-scored held-out behaviors. The 20 "copyright" behaviors are excluded because the HarmBench pipeline scores them by deterministic hash matching against reference text rather than with a classifier. Rates therefore differ from Table 1, which is computed over all 80 held-out behaviors. Learning-based methods are reported as mean$_{\pm\text{std}}$ over independent runs.

Model	Method	HarmBench	PolyGuard-Qwen-7B	Qwen3Guard-Gen-8B
Gemma-2-9B	Base	23.9	11.0	23.6
	Self-Reminder	15.8	6.7	14.6
	Circuit Breaker	$11.7_{\pm 4.1}$	$5.7_{\pm 1.9}$	$11.4_{\pm 4.9}$
	ActAdd	23.3	11.1	22.5
	DirAbl	17.5	27.3	19.2
	ReFT-r1	$16.1_{\pm 3.6}$	$7.3_{\pm 2.2}$	$14.7_{\pm 3.6}$
	SaP	$11.6_{\pm 1.9}$	$8.0_{\pm 2.0}$	$11.9_{\pm 1.9}$
	BarrierSteer	$\mathbf{0.0}_{\pm \mathbf{0.0}}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$	$\mathbf{0.0}_{\pm \mathbf{0.0}}$
Ministral-8B	Base	68.3	23.5	75.5
	Self-Reminder	26.7	8.5	26.1
	Circuit Breaker	$41.2_{\pm 1.8}$	$22.0_{\pm 0.0}$	$58.8_{\pm 1.7}$
	ActAdd	68.4	23.3	76.0
	DirAbl	69.0	23.4	75.7
	ReFT-r1	$53.6_{\pm 3.8}$	$18.7_{\pm 0.3}$	$59.8_{\pm 4.4}$
	SaP	$31.9_{\pm 4.0}$	$26.9_{\pm 2.7}$	$42.1_{\pm 5.5}$
	BarrierSteer	$\mathbf{1.9}_{\pm \mathbf{2.8}}$	$\mathbf{1.6}_{\pm \mathbf{2.4}}$	$\mathbf{2.0}_{\pm \mathbf{2.9}}$

D.4 Robustness to the Choice of Safety Judge

The ASR results in Sec. 5 use the HarmBench classifier [59]. To test whether our conclusions depend on the safety judge, we re-score the generations of every method with two additional judges: PolyGuard-Qwen-7B [77] and Qwen3Guard-Gen-8B [78]. Table 14 reports static HarmBench ASR under all three judges.

BarrierSteer achieves the lowest ASR under every judge on both models: 0.0% on Gemma-2-9B throughout, and 1.6–2.0% on Ministral-8B. However, the absolute rates and the ranking of the other methods vary across judges. For example, PolyGuard scores the Ministral-8B base model at 23.5%, while Qwen3Guard scores it at 75.5%. For DirAbl on Gemma-2-9B, PolyGuard reports a higher ASR than the HarmBench classifier. Our conclusions about BarrierSteer therefore do not depend on the choice of judge.

D.5 Multilingual Out-of-Distribution Evaluation

The barriers evaluated on MultiJail are trained only on English HarmBench data. To measure whether the learned latent safety constraints transfer to attacks drawn from a different distribution, we evaluate on MultiJail [79], which contains adversarial prompts in ten languages, and score responses with the multilingual classifier Qwen3Guard-Gen-8B [78]. No MultiJail data is used for training, so this

Table 15: Attack success rate (%, ↓) on MultiJail, covering adversarial prompts in ten languages and scored with Qwen3Guard-Gen-8B. BarrierSteer and SaP use the same hyperparameters as in Table 1 and are trained only on English HarmBench data, so this is an out-of-distribution transfer test.

Model	Base	SaP	BarrierSteer
Gemma-2-9B	82.20	$81.94_{\pm 0.90}$	$\mathbf{73.54}_{\pm \mathbf{1.15}}$
Ministral-8B	79.20	$76.46_{\pm 7.76}$	$\mathbf{40.48}_{\pm \mathbf{8.54}}$
Llama-2-7B	2.90	$2.02_{\pm 0.65}$	$\mathbf{1.30}_{\pm \mathbf{0.46}}$
Qwen2-1.5B	23.80	$20.06_{\pm 1.56}$	$\mathbf{18.22}_{\pm \mathbf{0.35}}$

evaluates out-of-distribution (OOD) transfer. Our adaptive-attack evaluation also uses attacks not seen during training. BarrierSteer and SaP use the same hyperparameters as in Table 1.

Table 15 shows that BarrierSteer outperforms SaP, the closest method to ours, on all four model families, most clearly on Ministral-8B, where ASR falls from 76.46% to 40.48%. However, defense on MultiJail remains imperfect. Gemma-2-9B ASR remains above 70%, much higher than its static HarmBench ASR in Table 1. As with adaptive attacks, this evaluation tests transfer beyond the HarmBench training distribution. MultiJail introduces multilingual prompts, while adaptive attacks generate unseen attack trajectories. These results show that training on HarmBench alone does not provide broad robustness across arbitrary attack distributions. The effectiveness of the learned safety constraints depends on their generalization beyond the training data, as discussed in Sec. G.

Future work could improve transfer to unseen attacks in two ways. First, the hidden-state classifiers can be trained on larger and more diverse safety datasets. Recent work shows that such classifiers can generalize across attack types and be deployed in production [26, 31]. Second, a new barrier can be trained for an emerging attack distribution and composed with existing barriers through the LSE merge of Sec. 4.3, without retraining the LLM or the other barriers. Appendix D.9 evaluates composition of independently trained category barriers.

D.6 Steering Strength Ablations

This section studies how the CBF steering strength α controls the safety–utility trade-off. Larger α enforces a stronger correction toward the learned safe set, which generally improves jailbreak robustness but can increase over-refusal or reduce downstream utility if the intervention becomes too aggressive. We report two complementary sweeps: Table 17 ablates α for the Top-2 variant on `Qwen2-1.5B`, while Table 16 ablates the default LSE variant across model families with $\kappa = 100$.

Table 17 shows the expected monotonic safety–strength pattern for Top-2: increasing α steadily reduces static ASR, while utility decreases gradually and over-refusal increases as the intervention becomes stronger. This confirms that α provides a direct knob for choosing the desired operating point rather than changing the learned barrier itself. The adaptive-attack results are less strictly monotonic because the attack search and response generation introduce additional variance, but the best low-ASR settings still occur at moderate-to-large steering strengths.

Table 16 shows a similar safety–utility trade-off for the LSE variant across model families. Small α values preserve utility closest to the base model but may leave adaptive attacks insufficiently suppressed, whereas larger α values provide stronger defense at the cost of higher over-refusal and lower utility on some models. This motivates selecting α per deployment requirement: conservative settings are appropriate when preserving benign capability is most important, while stronger settings are appropriate when adversarial robustness is prioritized.

D.7 Extended Utility Results

In addition to MMLU and GSM8K in the main tables, we evaluate Gemma-2-9B on broader utility benchmarks in Table 18. These results test whether the safety interventions preserve instruction following, factuality, conversational quality, and difficult question answering beyond the two primary utility metrics.

Overall, BarrierSteer remains close to the base model on the extended utility suite: it preserves utility across MT-Bench [80], TruthfulQA [81] GPQA Diamond [82], and IFEval [83] and avoids the severe degradation observed for stronger fixed-direction interventions such as directional ablation.

Table 16: BARRIERSTEER (LSE) steering-strength ablation across model families. We set $\kappa = 100$ and vary α. ASR, AA ASR, and XSTest are lower-is-better; MMLU and GSM8K are higher-is-better.

Model	α	ASR ↓	AA ASR ↓	XSTest ↓	MMLU ↑	GSM8K ↑
Gemma-2-9B	0.001	$3.12_{\pm 3.42}$	$66.00_{\pm 0.00}$	$33.28_{\pm 0.87}$	$71.87_{\pm 0.01}$	$73.62_{\pm 0.00}$
	0.01	$0.00_{\pm 0.00}$	$66.80_{\pm 8.07}$	$34.80_{\pm 0.75}$	$71.82_{\pm 0.04}$	$73.81_{\pm 0.12}$
	0.1	$0.00_{\pm 0.00}$	$57.20_{\pm 4.15}$	$32.00_{\pm 0.44}$	$71.87_{\pm 0.09}$	$73.65_{\pm 0.04}$
	0.2	$0.00_{\pm 0.00}$	$64.00_{\pm 7.35}$	$36.64_{\pm 6.35}$	$71.25_{\pm 0.55}$	$73.50_{\pm 0.04}$
	0.3	$0.00_{\pm 0.00}$	$64.00_{\pm 9.80}$	$38.56_{\pm 10.69}$	$71.70_{\pm 0.16}$	$72.93_{\pm 0.75}$
	0.5	$0.00_{\pm 0.00}$	$62.80_{\pm 7.43}$	$34.00_{\pm 3.07}$	$68.58_{\pm 3.19}$	$71.65_{\pm 1.91}$
	1.0	$0.00_{\pm 0.00}$	$60.80_{\pm 8.07}$	$36.64_{\pm 5.63}$	$65.28_{\pm 6.62}$	$67.21_{\pm 6.93}$
Qwen2-1.5B	0.001	$0.20_{\pm 0.26}$	$56.80_{\pm 9.34}$	$28.00_{\pm 1.70}$	$55.72_{\pm 0.04}$	$13.03_{\pm 0.14}$
	0.01	$0.66_{\pm 0.47}$	$24.80_{\pm 1.79}$	$29.44_{\pm 1.49}$	$55.65_{\pm 0.05}$	$12.95_{\pm 0.63}$
	0.1	$0.06_{\pm 0.10}$	$21.60_{\pm 8.17}$	$55.68_{\pm 5.21}$	$50.24_{\pm 4.58}$	$12.36_{\pm 1.71}$
	0.2	$0.02_{\pm 0.03}$	$9.20_{\pm 3.63}$	$67.44_{\pm 10.21}$	$45.94_{\pm 7.02}$	$11.37_{\pm 1.93}$
	0.3	$0.04_{\pm 0.07}$	$3.20_{\pm 3.35}$	$64.40_{\pm 1.09}$	$43.59_{\pm 6.07}$	$10.69_{\pm 0.72}$
	0.5	$0.00_{\pm 0.00}$	$2.40_{\pm 1.67}$	$71.68_{\pm 2.25}$	$40.45_{\pm 4.03}$	$9.30_{\pm 1.25}$
	1.0	$0.00_{\pm 0.00}$	$2.80_{\pm 1.10}$	$76.32_{\pm 4.32}$	$37.80_{\pm 2.81}$	$7.25_{\pm 0.16}$
Ministral-8B	0.001	$18.15_{\pm 25.05}$	$74.40_{\pm 4.34}$	$15.76_{\pm 0.92}$	$64.59_{\pm 0.05}$	$64.58_{\pm 0.67}$
	0.01	$8.57_{\pm 14.85}$	$58.80_{\pm 8.56}$	$15.92_{\pm 6.77}$	$64.48_{\pm 0.09}$	$53.02_{\pm 14.03}$
	0.1	$0.83_{\pm 1.86}$	$19.60_{\pm 2.61}$	$21.84_{\pm 1.63}$	$63.07_{\pm 1.95}$	$62.55_{\pm 2.46}$
	0.2	$0.32_{\pm 0.55}$	$7.60_{\pm 12.68}$	$63.44_{\pm 19.80}$	$55.83_{\pm 11.41}$	$10.29_{\pm 14.54}$
	0.3	$0.12_{\pm 0.21}$	$6.00_{\pm 10.39}$	$60.96_{\pm 22.98}$	$54.48_{\pm 12.81}$	$7.28_{\pm 11.01}$
	0.5	$0.02_{\pm 0.03}$	$3.60_{\pm 4.10}$	$59.52_{\pm 23.38}$	$48.69_{\pm 11.62}$	$6.90_{\pm 10.92}$
	1.0	$0.00_{\pm 0.00}$	$4.40_{\pm 7.80}$	$50.24_{\pm 25.69}$	$43.89_{\pm 13.77}$	$5.79_{\pm 8.68}$
Llama-2-7B	0.001	$0.93_{\pm 1.58}$	$18.40_{\pm 11.70}$	$42.32_{\pm 0.52}$	$46.55_{\pm 0.01}$	$21.12_{\pm 0.15}$
	0.01	$1.25_{\pm 1.91}$	$16.80_{\pm 13.54}$	$43.60_{\pm 0.80}$	$46.54_{\pm 0.01}$	$21.43_{\pm 0.64}$
	0.1	$1.28_{\pm 2.17}$	$1.60_{\pm 0.89}$	$43.36_{\pm 0.83}$	$46.56_{\pm 0.04}$	$20.94_{\pm 0.62}$
	0.2	$0.93_{\pm 1.32}$	$2.00_{\pm 2.00}$	$44.80_{\pm 2.40}$	$46.12_{\pm 0.67}$	$18.04_{\pm 4.80}$
	0.3	$0.69_{\pm 1.05}$	$3.20_{\pm 3.03}$	$43.68_{\pm 1.53}$	$45.79_{\pm 1.32}$	$17.87_{\pm 4.62}$
	0.5	$0.38_{\pm 0.60}$	$6.00_{\pm 2.00}$	$45.60_{\pm 1.47}$	$45.14_{\pm 2.28}$	$16.20_{\pm 4.71}$
	1.0	$0.30_{\pm 0.52}$	$2.40_{\pm 3.29}$	$48.96_{\pm 1.25}$	$44.36_{\pm 2.89}$	$16.45_{\pm 4.08}$

Table 17: Ablation study on the effect of steering strength α for BARRIERSTEER (Top-2) on Qwen2-1.5B. ASR, AA ASR, and XSTest are lower-is-better; MMLU and GSM8K are higher-is-better.

α	ASR ↓	AA ASR ↓	XSTest ↓	MMLU ↑	GSM8K ↑
0.01	$5.05_{\pm 7.95}$	$31.60_{\pm 4.98}$	$30.32_{\pm 2.97}$	$\mathbf{55.69_{\pm 0.03}}$	$\mathbf{12.43_{\pm 0.79}}$
0.1	$2.44_{\pm 4.09}$	$9.60_{\pm 10.43}$	$49.60_{\pm 23.03}$	$55.24_{\pm 0.38}$	$11.65_{\pm 1.69}$
0.2	$1.53_{\pm 2.65}$	$3.20_{\pm 4.60}$	$58.00_{\pm 15.49}$	$54.83_{\pm 0.42}$	$10.46_{\pm 2.58}$
0.3	$0.70_{\pm 1.22}$	$0.80_{\pm 1.10}$	$70.24_{\pm 4.68}$	$54.61_{\pm 0.36}$	$9.73_{\pm 1.62}$
0.5	$0.14_{\pm 0.24}$	$4.40_{\pm 8.76}$	$76.00_{\pm 0.69}$	$54.42_{\pm 0.24}$	$9.25_{\pm 2.27}$
1.0	$0.00_{\pm 0.00}$	$3.20_{\pm 1.79}$	$74.32_{\pm 3.18}$	$54.27_{\pm 0.23}$	$7.99_{\pm 1.61}$

Table 18: Extended utility benchmark results for Gemma-2-9B. Higher is better for all metrics.

Metric	Base	Self-Rem.	Cir. Breaker	ActAdd	DirAbl	ReFT-r1	SaP	BARRIERSTEER
IFEval ↑	65.80	75.30	$62.03_{\pm 10.84}$	66.73	33.09	$74.90_{\pm 0.14}$	$66.23_{\pm 0.39}$	$64.85_{\pm 0.43}$
TruthfulQA ↑	60.14	61.88	$61.03_{\pm 1.38}$	60.15	55.35	$58.68_{\pm 0.86}$	$60.14_{\pm 0.02}$	$60.57_{\pm 0.48}$
MT-Bench ↑	7.19	7.14	$5.91_{\pm 2.15}$	7.17	4.96	$7.19_{\pm 0.08}$	$7.16_{\pm 0.05}$	$7.15_{\pm 0.03}$
GPQA Diamond ↑	38.38	35.35	$38.04_{\pm 0.58}$	38.38	37.88	$34.85_{\pm 3.50}$	$38.38_{\pm 0.00}$	$38.18_{\pm 0.45}$

This supports the conclusion that classifier-guided CBF steering can improve safety while maintaining broad model utility.

D.8 Computational Benchmarking

This section provides additional details for the main-paper steering-latency results in Table 3 and reports full generation cost in Table 19. For Table 3, we measure inference-time overhead using hidden-

Table 19: HarmBench generation time on `Qwen2-1.5B`. We time only the completion-generation step for the DirectRequest split, using identical prompts, seed, greedy decoding, batch size, and maximum-new-token budget across methods.

Method	Generation Time (s) ↓
Base model	**35.27**
SaP	102.05
BarrierSteer (LSE)	39.44

Table 20: Steering latency (ms per steered token, ↓) across all four model families, extending Table 3. Measurements use the same single NVIDIA H200 GPU, batch size 1, and unsafe-token protocol described above, and are reported as mean$_{\pm \text{std}}$.

Model	SaP (100 steps)	SaP (1 step)	BarrierSteer (LSE)
Gemma-2-9B	$251.62_{\pm 19.86}$	$181.49_{\pm 20.22}$	$\mathbf{2.54}_{\pm \mathbf{0.04}}$
Ministral-8B	$245.67_{\pm 14.17}$	$175.90_{\pm 17.28}$	$\mathbf{2.55}_{\pm \mathbf{0.05}}$
Llama-2-7B	$206.21_{\pm 30.10}$	$176.72_{\pm 19.42}$	$\mathbf{2.53}_{\pm \mathbf{0.03}}$
Qwen2-1.5B	$190.67_{\pm 3.17}$	$139.97_{\pm 6.33}$	$\mathbf{2.40}_{\pm \mathbf{0.06}}$

state activations extracted from the adversarial unsafe dataset. All per-token latency experiments are run on a single NVIDIA H200 GPU with batch size 1, matching the real-time decoding setting where a controller must act at each generated token. We compare `SaP` with $K = 30$ polytope facets against BarrierSteer with neural CBF heads using comparable parameter counts. Following the `SaP` implementation [32], the baseline uses 100 gradient-based optimization steps.

Full generation cost. To complement the per-token steering latency in Table 3, we also measure end-to-end completion-generation time on the HarmBench DirectRequest split. This full-generation benchmark uses the same single NVIDIA H200 GPU and batch size 1 setup, with identical prompts, greedy decoding, random seed, and maximum-new-token budget across methods. Table 19 shows that the default LSE variant adds only modest overhead relative to the unmodified model, while `SaP` is substantially slower because it solves an optimization problem during decoding. This motivates using LSE as the default BarrierSteer variant in the main experiments: it composes multiple learned safety constraints while preserving near-base generation cost.

Latency across model families. Table 3 reports steering latency on `Qwen2-1.5B` only. Table 20 extends the same measurement to all four model families under the identical protocol described above. BarrierSteer's closed-form LSE update takes between 2.40 and 2.55 ms per steered token across models, while `SaP` requires approximately 190–252 ms with its default 100 optimization steps and 140–181 ms with a single step. The measured latency of the closed-form update varies little across these four models. `SaP` has substantially higher latency on every model, including when its inference-time optimization is restricted to a single step. This table reports latency *per steered token*, averaged over the tokens that each method's classifier flags as unsafe. End-to-end generation time also depends on how often steering activates and is reported in Table 19.

Neural CBF sizes ablations. We next isolate two sources of computational scaling: the neural CBF architecture size and the number of barriers K. Table 21 varies the number of trainable neural-CBF parameters while keeping K fixed, and Table 22 varies K while keeping the per-barrier architecture fixed. These measurements report steering overhead per generated token and therefore complement the end-to-end generation-time comparison above.

Table 21 shows that the LSE update remains lightweight as the neural CBF size increases. For LSE, CBF checking and merged-barrier gradient computation are fused into a single autograd pass; therefore, its cost appears primarily in the "Check CBFs" column, while the separate "Gradient Calc." column is near zero by construction. By contrast, QP requires explicit gradient and solve steps whose runtime is more variable, making LSE the preferred default when both speed and modular composition are important.

Ablation on number of constraints K. We next vary the number of learned CBF heads while keeping the per-head architecture fixed. This experiment isolates how each merge rule scales as more safety constraints are checked and composed during decoding.

Table 21: Computational ablation over neural CBF size. We vary the number of trainable barrier parameters while keeping the number of CBF heads fixed. Values are per-token steering latency in milliseconds, reported as mean±std. The largest BarrierSteer setting has 23.64M parameters, comparable to the 25.67M-parameter SaP safety module. For BarrierSteer (LSE), barrier checking and gradient computation are fused into an operation and the cost appears under "Check CBFs."

Method	#Params.	Check CBFs	Select CBFs	Gradient Calc.	Compute Steering	Total
BarrierSteer (Top-2)	0.8M	$0.36_{\pm 0.04}$	$0.06_{\pm 0.01}$	$1.24_{\pm 0.11}$	$5.15_{\pm 10.42}$	$6.82_{\pm 10.42}$
	6.3M	$0.37_{\pm 0.01}$	$0.06_{\pm 0.00}$	$1.24_{\pm 0.05}$	$3.53_{\pm 8.50}$	$5.21_{\pm 8.50}$
	12.6M	$0.41_{\pm 0.04}$	$0.06_{\pm 0.01}$	$1.31_{\pm 0.13}$	$3.52_{\pm 8.35}$	$5.31_{\pm 8.35}$
	15.7M	$0.82_{\pm 0.01}$	$0.06_{\pm 0.00}$	$2.13_{\pm 0.06}$	$28.78_{\pm 6.98}$	$31.79_{\pm 6.98}$
	19.5M	$0.85_{\pm 0.01}$	$0.06_{\pm 0.00}$	$2.12_{\pm 0.06}$	$24.23_{\pm 10.62}$	$27.26_{\pm 10.62}$
	23.6M	$0.87_{\pm 0.10}$	$0.06_{\pm 0.01}$	$2.17_{\pm 0.21}$	$26.34_{\pm 10.99}$	$29.45_{\pm 11.00}$
BarrierSteer (QP)	0.8M	$0.37_{\pm 0.01}$	$0.06_{\pm 0.00}$	$2.45_{\pm 0.13}$	$22.53_{\pm 12.66}$	$25.41_{\pm 12.66}$
	6.3M	$0.38_{\pm 0.05}$	$0.06_{\pm 0.02}$	$2.47_{\pm 0.22}$	$14.30_{\pm 14.72}$	$17.20_{\pm 14.73}$
	12.6M	$0.41_{\pm 0.05}$	$0.06_{\pm 0.01}$	$2.51_{\pm 0.21}$	$7.47_{\pm 12.30}$	$10.44_{\pm 12.30}$
	15.7M	$0.82_{\pm 0.09}$	$0.06_{\pm 0.01}$	$5.07_{\pm 0.41}$	$28.01_{\pm 4.38}$	$33.95_{\pm 4.44}$
	19.5M	$0.85_{\pm 0.03}$	$0.06_{\pm 0.00}$	$5.06_{\pm 0.07}$	$27.35_{\pm 5.66}$	$33.32_{\pm 5.66}$
	23.6M	$0.84_{\pm 0.09}$	$0.06_{\pm 0.01}$	$5.12_{\pm 0.43}$	$27.75_{\pm 5.82}$	$33.77_{\pm 5.86}$
BarrierSteer (LSE)	0.8M	$1.09_{\pm 0.05}$	$0.04_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.13_{\pm 0.00}$	$1.28_{\pm 0.05}$
	6.3M	$1.09_{\pm 0.02}$	$0.04_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.13_{\pm 0.00}$	$1.28_{\pm 0.02}$
	12.6M	$1.15_{\pm 0.08}$	$0.04_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.13_{\pm 0.00}$	$1.34_{\pm 0.08}$
	15.7M	$2.15_{\pm 0.04}$	$0.04_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.14_{\pm 0.03}$	$2.34_{\pm 0.06}$
	19.5M	$2.19_{\pm 0.07}$	$0.04_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.13_{\pm 0.00}$	$2.38_{\pm 0.07}$
	23.6M	$2.21_{\pm 0.04}$	$0.05_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.14_{\pm 0.04}$	$2.40_{\pm 0.06}$

Table 22: Computational ablation over the number of CBF K. We vary K while keeping the per-head neural architecture fixed. Values are per-token steering latency in milliseconds, reported as mean$_{\pm \text{std}}$.

Method	K	Check CBFs	Select CBFs	Gradient Calc.	Compute Steering	Total
BarrierSteer (Top-2)	4	$0.37_{\pm 0.04}$	$0.06_{\pm 0.01}$	$1.24_{\pm 0.12}$	$5.12_{\pm 10.32}$	$6.78_{\pm 10.33}$
	12	$0.84_{\pm 0.04}$	$0.06_{\pm 0.01}$	$1.26_{\pm 0.07}$	$14.19_{\pm 14.11}$	$16.35_{\pm 14.11}$
	20	$1.32_{\pm 0.02}$	$0.06_{\pm 0.00}$	$1.26_{\pm 0.04}$	$20.32_{\pm 13.90}$	$22.97_{\pm 13.90}$
	28	$1.79_{\pm 0.04}$	$0.06_{\pm 0.00}$	$1.26_{\pm 0.05}$	$6.95_{\pm 11.90}$	$10.07_{\pm 11.90}$
	36	$2.29_{\pm 0.20}$	$0.06_{\pm 0.01}$	$1.27_{\pm 0.12}$	$12.82_{\pm 14.59}$	$16.44_{\pm 14.60}$
	44	$2.75_{\pm 0.03}$	$0.06_{\pm 0.00}$	$1.26_{\pm 0.02}$	$26.02_{\pm 10.76}$	$30.09_{\pm 10.76}$
	52	$3.15_{\pm 0.03}$	$0.06_{\pm 0.00}$	$1.24_{\pm 0.03}$	$26.25_{\pm 8.64}$	$30.69_{\pm 8.64}$
	60	$3.65_{\pm 0.03}$	$0.06_{\pm 0.00}$	$1.26_{\pm 0.04}$	$24.23_{\pm 11.24}$	$29.19_{\pm 11.24}$
BarrierSteer (QP)	4	$0.37_{\pm 0.01}$	$0.06_{\pm 0.00}$	$2.42_{\pm 0.04}$	$22.24_{\pm 12.49}$	$25.09_{\pm 12.49}$
	12	$0.86_{\pm 0.09}$	$0.06_{\pm 0.01}$	$12.28_{\pm 0.86}$	$21.93_{\pm 13.22}$	$35.13_{\pm 13.26}$
	20	$1.30_{\pm 0.11}$	$0.06_{\pm 0.01}$	$29.77_{\pm 1.54}$	$24.32_{\pm 10.47}$	$55.45_{\pm 10.60}$
	28	$1.76_{\pm 0.03}$	$0.06_{\pm 0.00}$	$55.31_{\pm 0.33}$	$18.06_{\pm 13.50}$	$75.19_{\pm 13.50}$
	36	$2.28_{\pm 0.21}$	$0.05_{\pm 0.00}$	$90.99_{\pm 2.69}$	$24.95_{\pm 11.30}$	$118.28_{\pm 11.67}$
	44	$2.72_{\pm 0.25}$	$0.05_{\pm 0.01}$	$132.04_{\pm 3.22}$	$29.31_{\pm 1.53}$	$164.13_{\pm 3.58}$
	52	$3.14_{\pm 0.03}$	$0.05_{\pm 0.00}$	$181.78_{\pm 0.72}$	$29.51_{\pm 0.33}$	$214.48_{\pm 0.83}$
	60	$3.64_{\pm 0.34}$	$0.05_{\pm 0.01}$	$241.60_{\pm 4.40}$	$29.49_{\pm 1.40}$	$274.77_{\pm 4.71}$
BarrierSteer (LSE)	4	$1.08_{\pm 0.02}$	$0.04_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.13_{\pm 0.00}$	$1.27_{\pm 0.02}$
	12	$2.22_{\pm 0.03}$	$0.05_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.14_{\pm 0.03}$	$2.42_{\pm 0.04}$
	20	$3.33_{\pm 0.06}$	$0.05_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.14_{\pm 0.00}$	$3.53_{\pm 0.06}$
	28	$4.46_{\pm 0.21}$	$0.05_{\pm 0.00}$	$0.01_{\pm 0.03}$	$0.14_{\pm 0.04}$	$4.66_{\pm 0.22}$
	36	$5.66_{\pm 0.24}$	$0.05_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.14_{\pm 0.01}$	$5.86_{\pm 0.24}$
	44	$6.61_{\pm 0.12}$	$0.05_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.14_{\pm 0.00}$	$6.81_{\pm 0.13}$
	52	$7.65_{\pm 0.10}$	$0.05_{\pm 0.00}$	$0.01_{\pm 0.00}$	$0.14_{\pm 0.00}$	$7.86_{\pm 0.11}$
	60	$8.63_{\pm 0.32}$	$0.06_{\pm 0.01}$	$0.01_{\pm 0.00}$	$0.14_{\pm 0.01}$	$8.84_{\pm 0.33}$

Table 22 shows the expected scaling pattern with the number of CBF heads. QP becomes expensive as K grows because it differentiates and solves over all active constraints, while LSE scales much more smoothly by merging the barriers into a single differentiable surrogate before computing the steering update. This computational behavior is why we recommend LSE for the default multi-constraint setting and reserve Top-2 primarily as a simple closed-form ablation.

Table 23: Gemma-2-9B LSE smoothing sensitivity for BARRIERSTEER with fixed $\alpha = 0.2$. We report all metrics mean$_{\pm \text{std}}$ over 5 runs.

κ	ASR ↓	AA ASR ↓	XSTest ↓	MMLU ↑	GSM8K ↑
1	$0.00_{\pm 0.00}$	$54.00_{\pm 19.70}$	$36.80_{\pm 9.06}$	$44.28_{\pm 9.97}$	$54.69_{\pm 4.60}$
10	$0.00_{\pm 0.00}$	$58.00_{\pm 19.70}$	$36.24_{\pm 5.75}$	$68.80_{\pm 1.97}$	$70.00_{\pm 5.42}$
25	$0.00_{\pm 0.00}$	$59.20_{\pm 15.72}$	$40.00_{\pm 10.83}$	$70.45_{\pm 1.26}$	$68.16_{\pm 10.24}$
50	$0.00_{\pm 0.00}$	$56.00_{\pm 19.70}$	$41.20_{\pm 11.84}$	$70.43_{\pm 1.16}$	$67.90_{\pm 10.10}$
100	$0.00_{\pm 0.00}$	$64.00_{\pm 7.35}$	$36.64_{\pm 6.35}$	$71.25_{\pm 0.55}$	$73.50_{\pm 0.04}$

Table 24: Detailed WildGuard modular-composition results for BARRIERSTEER on `Qwen2-1.5B`. This appendix table expands Table 4 by showing category-level individual CBF results, alternative Top-2/QP/LSE merging rules, and a linear-probe ablation that replaces each neural CBF with a linear barrier. The LSE rows use the selected main-table smoothing value $\kappa = 0.01$.

Barrier Architecture	Category / Merge Rule	Harmful Rate ↓
Original Model	–	22.77
Single BARRIERSTEER	Copyright	$5.71_{\pm 0.00}$
	Cyberattack	$3.12_{\pm 0.00}$
	Defamation	$2.95_{\pm 0.00}$
	Discrimination	$3.81_{\pm 1.91}$
	Disinformation	$5.27_{\pm 0.00}$
	Fraud	$3.66_{\pm 0.00}$
	Harm Dissemination	$5.27_{\pm 0.00}$
	Mental Health	$4.46_{\pm 0.00}$
	Others	$5.30_{\pm 0.36}$
	Private Info	$4.82_{\pm 0.00}$
	Sensitive Info	$6.22_{\pm 1.13}$
	Sexual	$5.09_{\pm 0.00}$
	Toxic	$6.61_{\pm 0.00}$
	Violence	$4.02_{\pm 0.93}$
	Average	$4.74_{\pm 1.21}$
BARRIERSTEER (5-layer MLP)	Top-2	$10.60_{\pm 1.37}$
	QP	$1.82_{\pm 0.29}$
	LSE ($\kappa = 0.01$)	$\mathbf{0.86_{\pm 0.04}}$
BARRIERSTEER (Linear Probe)	Top-2	$12.44_{\pm 0.95}$
	QP	$8.87_{\pm 0.15}$
	LSE ($\kappa = 0.01$)	$2.89_{\pm 0.69}$

D.9 Modular Composition of Safety Barriers

For modular composition, we use WildGuardMix [30], which covers 14 safety categories. For each category, we sample 400 behaviors and split them into 320 training behaviors and 80 held-out evaluation behaviors. At inference time, this yields 1,120 evaluation samples (80×14). We report the harmfulness rate measured by `HarmBench-Llama-2-13b-cls` [59].

Each category-specific safety constraint uses the neural CBF architecture in Appendix C. To isolate composition, we train one CBF per semantic category and use a fixed steering strength $\alpha = 1$ for all methods. Table 24 reports category-level individual-barrier results, alternative Top-2/QP/LSE merge rules, and a linear-probe ablation that removes the neural hidden layers from each category barrier.

D.10 Ablation on LSE Smoothing Parameter κ

The LSE merge rule uses κ to control how sharply the smooth minimum in Eq. (2) approximates the most violated barrier constraint. Because the exponential weights are proportional to $e^{-\kappa(b_k(h)-\delta)}$, smaller κ gives a smoother merge that spreads the steering signal across more constraints, while larger κ concentrates the merge on the most violated constraints and approaches the hard-minimum behavior used by Top-2-style selection. As a result, κ becomes more important when many constraints are composed: with only a few barriers, the smooth minimum is relatively stable, but with many

Table 25: WildGuard LSE smoothing sensitivity for composed BARRIERSTEER barriers on `Qwen2-1.5B`. We compose 14 category-specific barriers and evaluate 1,120 held-out WildGuardMix samples per seed. Harmful rate is reported as mean$_{\pm\text{std}}$; lower is better. The main WildGuard Table 4 uses $\kappa = 0.01$.

κ	**Harmful Rate** $\downarrow$	κ	**Harmful Rate** $\downarrow$
0.01	$\mathbf{0.86_{\pm 0.04}}$	1.0	$4.26_{\pm 0.68}$
0.03	$0.71_{\pm 0.07}$	2.0	$7.71_{\pm 0.65}$
0.05	$1.16_{\pm 0.13}$	5.0	$9.02_{\pm 0.15}$
0.1	$0.89_{\pm 0.32}$	10.0	$8.48_{\pm 0.39}$
0.2	$1.16_{\pm 0.26}$	20.0	$7.77_{\pm 1.17}$
0.5	$1.61_{\pm 0.51}$	50.0	$7.59_{\pm 1.79}$

independently trained barriers, the choice of κ controls how broadly the controller considers all constraints versus focusing on the currently most active ones.

Table 23 ablates this parameter for HarmBench on `Gemma-2-9B` with the LSE variant and fixed steering strength $\alpha = 0.2$. Across all tested values, static HarmBench ASR remains at 0.00%, indicating that the LSE merge is not highly sensitive to κ in this non-adaptive HarmBench setting once the steering strength is fixed. The main difference appears in utility: very small κ over-smooths the constraints and can unnecessarily distort representations, while larger values better preserve MMLU/GSM8K performance. This supports using a sharper LSE approximation for the HarmBench experiments, where we set $\kappa = 100$ by default. For the WildGuard modular-composition experiment, Table 25 shows that κ is more sensitive when composing 14 independently trained category barriers. In this setting, a smaller κ is preferable because it considers a broader set of category constraints rather than approximating a hard minimum too aggressively, so we use $\kappa = 0.01$ for the WildGuard LSE composition results.

D.11 Ablation on Neural CBF Architecture

To test whether the neural CBF architecture is necessary, we compare the default nonlinear category barriers against a linear-probe variant in the WildGuard modular-composition experiment. The linear-probe ablation retrains all 14 category-specific barriers without the neural intermediate layers, while keeping the same data split, composition rules, and evaluation protocol. Concretely, each linear-probe barrier is a single affine classifier $b_k(h) = w_k^\top h + c_k$ applied directly to the hidden state, so the safe set for each category is a halfspace rather than the nonlinear decision boundary induced by the default 5-layer MLP. As shown in Table 24, the neural barriers consistently outperform the linear-probe variants under the same composition rules: for example, LSE composition improves from $2.89_{\pm 0.69}$ harmful rate with linear probes to $0.86_{\pm 0.04}$ with neural CBFs. The gap is larger for QP and Top-2 composition, suggesting that nonlinear CBF boundaries provide a more faithful representation of category-specific safety constraints before the barriers are merged. These results support the design choice of learning nonlinear latent CBFs rather than relying only on linear hidden-state classifiers for modular safety composition.

D.12 Conditional Fixed-Direction Steering Baselines

We also evaluate whether the gains of BARRIERSTEER can be explained only by using a learned safety detector to decide *when* to intervene. Inspired by conditional activation steering [44], we construct two conditional fixed-direction baselines on `Gemma-2-9B`. Both baselines use the learned barrier as an unsafe-state detector during generation. When the detector fires, `Conditional ActAdd` applies the same fixed activation-addition vector used by `ActAdd`, while `Conditional DirAbl` applies the same fixed directional-ablation operation used by `DirAbl`. These baselines use the learned barrier to trigger steering, as in BARRIERSTEER, but do not solve the CBF control objective. They use fixed steering vectors rather than computing a correction from the barrier at the current hidden state. Results are reported in Tab. 6.

D.13 Compute Resources and Wall-Clock Cost

Unless otherwise stated, the main experiments were run using two NVIDIA H200 GPUs with bfloat16 model weights. The latency microbenchmarks in Appendix D.8 are the exception: they use a single H200 GPU with batch size 1 to measure per-token steering overhead in the real-time decoding setting.

Training. Hidden-state extraction is the main preprocessing cost before training the latent safety constraints. For Qwen2-1.5B, hidden-state extraction typically takes about 15–25 minutes per seed; for Ministral-8B and Llama-2-7B, about 30–45 minutes; and for Gemma-2-9B, about 45–60 minutes. Once hidden states are extracted, training BarrierSteer or SaP is comparatively lightweight: roughly 2–4 minutes per seed for Qwen2-1.5B, 3–6 minutes for Ministral-8B and Llama-2-7B, and 5–10 minutes for Gemma-2-9B. The training-time baselines require more time because they update model-side intervention parameters: `Circuit Breaker` and `ReFT-r1` take less than one hour per model and seed under the hyperparameters in Appendix D.2. For the WildGuard experiment, each Qwen2-1.5B category barrier is trained independently; individual category jobs are short after hidden-state extraction and the 14 categories are parallelized across GPUs.

Evaluation. For HarmBench static ASR evaluation, the approximate wall-clock time per model and seed is 6–20 minutes for Qwen2-1.5B, 6–15 minutes for Ministral-8B, 30–54 minutes for Llama-2-7B, and 30–40 minutes for Gemma-2-9B. Adaptive-attack evaluation is more expensive and takes about 30–40 minutes per model and seed for the configured adaptive-evaluation run. XSTest over-refusal evaluation uses 250 safe-only prompts and usually takes less than 10 minutes per model and seed, with the wall-clock time often dominated by batched API judge calls. MMLU/GSM8K utility evaluation through the LM Evaluation Harness takes about 1–3 hours per model and method, depending on the model size and task subset. The WildGuard composition evaluation on Qwen2-1.5B evaluates 1,120 held-out WildGuardMix samples and takes about 10–30 minutes per seed after the category-specific barriers have been trained.

E Additional Related Work on Inference-time Safety

This section situates BarrierSteer within inference-time safety methods, covering text-level defenses, latent safety classifiers, activation steering, adaptive activation steering, and constraint-based steering, including a comparison with SaP, the most adjacent baseline to BarrierSteer. These methods complement training-time approaches [11, 12, 20] by adding an additional layer of defense during deployment, since safety-tuned models can still be vulnerable to adversarial prompts [13, 21–23].

Text-level Defenses. One line of inference-time safety work attempts to mitigate unsafe behavior by inspecting input prompts or generated outputs. Input filtering methods detect suspicious prompts before generation [84, 85], while input modification methods rewrite, paraphrase, or otherwise transform prompts to weaken potential attacks [86, 87, 76, 88]. Output filtering methods instead inspect generated text and block, revise, or regenerate responses when safety violations are detected [89–91]. These input and output filtering methods are attractive because they are model-agnostic and can be deployed outside the target LLM, but they primarily rely on text-level patterns or heuristic filters. As a result, they can be brittle to paraphrases, obfuscation, and adaptive white-box attacks [27].

Learned language-model classifiers provide stronger detection [59, 29] and can yield promising defense results when trained at large scale [25], but they typically require running an additional model over prompts or responses. This additional checking introduces nontrivial computational overhead, especially when the system must detect an unsafe output and then regenerate a safer response.

Latent Safety Classifiers. Recent work therefore studies classifiers over intermediate hidden states as a more efficient alternative to text-space safety classification [26, 31]. Hidden-state classifiers detect unsafe behavior from model-internal representations rather than completed text, offering a promising path toward lower-latency safety detection. However, detection-only systems still primarily decide whether to block a response; they do not directly modify generation toward a safer continuation.

Activation Steering. A promising direction for safer generation is activation steering, which intervenes on model activations during generation. Activation-addition methods modify hidden states along fixed directions, often using a linear addition [42, 43, 24, 92, 93, 65]. Recent work further advances this direction with affine and nonlinear steering [94–96]. These interventions are simple but can be coarse, as the same direction is applied across contexts and may degrade utility. Steering can

also be learned with a text-level objective, for example by distilling controllable levels of instruction adherence into the model [97].

Adaptive Activation Steering. Beyond fixed-vector activation steering, recent work has used learned models to provide more flexible inference-time steering signals. A simple form is conditional steering [44], where a fixed steering vector is applied only when a latent classifier detects undesired behavior. Other methods predict context-dependent steering directions, for example using hypernetwork-style modules [98], or directly transform hidden states to modify model behavior [99].

A related line of work learns scalar signals over latent representations and uses them to guide iterative steering. For example, `PPLM` uses gradients from attribute models during decoding [100], `RE-Control` learns value functions over hidden states for representation editing [45], `ODESteer` learns a density-ratio function and follows its gradient through ODE integration [47], and `BRT-Align` uses learned safety values with perturbation search to steer away from unsafe regions [46]. However, these methods use learned signals primarily to define a direction or search objective for iterative steering, rather than as constraints that must be satisfied during generation.

In contrast, BarrierSteer treats learned neural safety classifiers as explicit constraints on latent representations. Rather than optimizing activations toward a desirable direction, it uses the classifier as a Control Barrier Function and derives constraint-guided steering updates for safety-critical generation. This perspective is especially suitable for LLM safety defense, where the goal is not only to encourage preferred behavior, but to prevent trajectories from entering unsafe regions.

Constraint-based Steering. Recent work such as `SaP` [32] suggests a natural combination: using learned latent safety classifiers not only as detection signals or steering objectives, but also as explicit constraints on hidden representations. Under this view, the classifier decision boundary defines a safe region in latent space, and inference-time steering aims to keep hidden states within, or move them toward, that region during generation.

`SaP` is the closest prior method to BarrierSteer. Like BarrierSteer, it learns safety constraints from hidden states and safety labels, then modifies hidden states at inference time to remain within the learned safe region. We therefore include `SaP` as a main baseline. Both `SaP` and BarrierSteer use hidden states and safety labels to learn latent safety constraints, but they differ in the constraint class and the inference-time enforcement mechanism. `SaP` learns linear polytope constraints and enforces them through iterative gradient-descent at inference time. In contrast, BarrierSteer learns nonlinear CBF constraints and derives closed-form local steering updates, enabling more expressive safety boundaries with substantially lower per-step steering overhead.

F Additional Discussion and Clarifications

F.1 Assumptions for Theoretical Safety

Q1. Does BarrierSteer provide a worst-case safety guarantee?
Answer. No. In the main paper, we clarify that although Control Barrier Functions are a standard tool for maintaining safety within a safe set, the LLM setting does not admit a perfect worst-case guarantee. Such a guarantee would require a perfect classifier or barrier whose safe set exactly corresponds to text-level safety for all possible prompts and generations. This is not realistic for two reasons. First, safety data are finite, while the space of possible LLM generations and hidden states is extremely large, so a learned barrier cannot certify every possible trajectory. Second, there is a latent–semantic safety gap: even if a hidden state appears safe under the learned classifier, later layers and decoding can still produce unsafe text, as discussed in Appendix G. Our goal is therefore not to prove worst-case text-level safety, but to steer responses toward an approximated latent safe set learned from data. This approximation is still practically meaningful: recent work shows that hidden-state classifiers can be useful in production-grade safety systems for frontier models [26, 31], suggesting that learned latent safe sets can be strong enough to support effective deployment-time interventions.

Q2. Is the updated hidden state guaranteed to be inside the safe set after one steering step?
Answer. No. Our implementation applies a one-step CBF update during decoding. The local CBF condition moves the hidden state toward the learned safe set, and the theorem characterizes exponential progress toward the safe set under the stated assumptions, but this does not mean that the updated hidden state h' must lie inside the safe set after a single finite update. In practice, however,

the one-step update is already effective: our experiments show strong reductions in adversarial attack success while preserving utility across model families.

F.2 Data, Supervision, and Generalization

Q3. Does BarrierSteer generalize to unseen harmful behaviors?
Answer. Like other data-driven safety methods, BarrierSteer depends on the coverage and quality of the safety data used to train the barrier. We do not expect a barrier trained for one safety concept to perfectly generalize to all unrelated out-of-distribution harms. Instead, the framework is modular: new barriers can be trained for newly identified safety concepts and composed with existing barriers. That said, we consistently observe empirical generalization beyond the exact training examples. In HarmBench, we train on the training behaviors and evaluate on held-out adversarial behaviors, where BarrierSteer still substantially reduces attack success. In WildGuard, even individual category barriers slightly reduce harmful generations outside their exact category, while composing category-specific barriers substantially improves coverage. Finally, in the adaptive-attack evaluation, BarrierSteer also reduces attack success even though the adaptive attack trajectories are not part of the training data. Overall, despite being trained on the same data as the learned baselines, BarrierSteer reduces harmful generations more effectively.

Q4. What supervision is used to train BarrierSteer?
Answer. BarrierSteer uses binary safe/unsafe supervision over hidden states. Sequence-level safety labels are inherited by token-level hidden states extracted from the same response, producing hidden-state examples labeled as safe or unsafe. We do not require preference pairs, ranked responses, or paired chosen/rejected examples. Appendix D.1 gives the token-level hidden-state training details.

Q5. How is BarrierSteer different from preference-tuned safety methods?
Answer. Training-time alignment methods such as RLHF [11, 12, 20], Safe-RLHF [35], and DPO-style methods [36–38] update model parameters using preference, reward, or safety-constraint objectives. In contrast, BarrierSteer trains a separate latent barrier from binary safety labels and applies it as an inference-time controller on top of an already aligned model. This makes BarrierSteer complementary to preference tuning: the base model can already be RLHF/DPO-tuned, while the barrier can be updated or composed for new safety risks without retraining the LLM.

F.3 BarrierSteer Variants and Steering Geometry

Q6. Which BarrierSteer variant is preferred?
Answer. We recommend BarrierSteer (LSE) as the default variant. Top-2 selects the two most violated barriers and is useful as a simple, fast ablation for isolating steering-strength effects, but when many safety categories are composed, ignoring the remaining constraints can be suboptimal. QP is useful as a more exact optimization-based reference. LSE is the best default for deployment and main reporting because it accounts for all active constraints, matches the strong defense behavior of the multi-constraint formulations, and has the lowest measured steering latency among our variants in Table 3. Appendix D.6 provides steering-strength ablations, and Appendix D.8 provides the detailed latency comparison among variants.

Q7. Can BarrierSteer be adapted for steering geometries beyond linear addition?
Answer. As discussed in Appendix E, recent activation-steering work has moved beyond fixed linear addition toward affine, angular, spherical, and other nonlinear or norm-preserving geometries [94–96]. BarrierSteer is complementary to these methods: the current implementation applies the CBF correction as an additive hidden-state update for efficiency, but the learned barrier can also provide a local safety signal for other update geometries. Extending BarrierSteer's classifier-guided CBF constraints to norm-preserving or rotation-based updates is a natural direction for future work.

F.4 Evaluation, Robustness, and Deployment Scope

Q8. How robust is BarrierSteer to adaptive attacks?
Answer. Adaptive attacks are stronger than static HarmBench attacks because they optimize against the defended model while the steering mechanism is active. Table 1 reports adaptive-attack ASR together with static ASR, over-refusal, and utility across four model families. BarrierSteer

consistently improves over the undefended model and remains competitive with or stronger than other safety baselines. The adaptive results also show that near-zero static ASR should not be interpreted as absolute robustness: stronger attacks can still partially bypass the defense, but BarrierSteer substantially improves robustness under an adaptive threat model. Additional steering-strength sweeps in Appendix D.6 show how the adaptive robustness–utility trade-off changes with α.

Q9. Does BarrierSteer over-refuse benign prompts?
Answer. We evaluate over-refusal using XSTest safe-only prompts and report it alongside ASR and utility in Table 1. The results show that BarrierSteer can increase over-refusal relative to the base model, but it avoids the most severe over-refusal observed for some stronger baselines and can be tuned through the steering strength α. This safety–utility trade-off is expected for inference-time steering: stronger interventions improve attack resistance but may reject more benign prompts. Appendix D.6 provides the corresponding α sweeps, and Appendix D.7 reports additional utility benchmarks beyond MMLU and GSM8K.

Q10. Is BarrierSteer production-grade?
Answer. Not yet. Our experiments use medium-scale open-weight LLMs, so we do not claim production-grade deployment on frontier systems. However, the broader direction is motivated by recent evidence that hidden-state classifiers can be production-grade safety detectors for frontier models [26, 31]. BarrierSteer builds on this direction by using learned hidden-state safety signals not only to detect unsafe trajectories, but also to steer generation with low additional latency; Table 3 and Appendix D.8 quantify this overhead. Demonstrating production-grade steering would require direct frontier-model hidden-state access and deployment-scale evaluation, which we leave to future work.

G Limitations and Future Work

Dependence on supervised safety labels. BarrierSteer learns barriers from labeled safe and unsafe hidden states, so its effectiveness depends on the coverage and quality of the safety data. If a harmful behavior is absent from the training dataset, the corresponding latent region may not be captured by the learned CBFs. Future work should study broader safety taxonomies, active data collection for uncovered harms, and continual updates that add new barriers for emerging risk categories.

Latent safety is not identical to semantic safety. The control-theoretic guarantees in this paper apply to learned latent constraints, not directly to text-level safety, and are conditional on the learned barriers accurately representing the intended safety concepts. One way to mitigate this is to train stronger latent constraints with larger-scale red-teaming data [26, 31].

Evaluation scope. Our experiments cover multiple open-weight instruction-tuned models, HarmBench adversarial attacks, XSTest over-refusal, utility benchmarks, adaptive attacks, and WildGuard modular composition. However, they do not establish robustness against all possible adaptive attackers or all deployment settings. Future work should evaluate larger frontier models when hidden-state access is available, broader multilingual and multi-turn safety settings, and stronger adaptive attacks that optimize jointly over prompts and steering responses.

Geometry and intervention form. The current implementation applies the CBF correction as an additive hidden-state update. Although the update direction is computed from nonlinear barrier geometry, additive interventions can still alter activation norms and may not be optimal for all models. Future work can combine BarrierSteer's classifier-guided CBF constraints with norm-preserving steering updates.

Safety–utility trade-off. BarrierSteer improves the safety–utility trade-off relative to the baselines we evaluate, but it does not remove the trade-off. Increasing steering strength can reduce unsafe generations while increasing over-refusal or degrading downstream utility. Future work should develop adaptive steering schedules that tune intervention strength based on uncertainty, task context, and deployment-specific risk tolerance.

H Broader Impact

This work aims to improve the safety of large language models by steering model activations away from unsafe generations during inference. Its potential positive impact is to provide an additional defense layer that can complement training-time alignment, reduce harmful outputs, and make safety interventions more modular and efficient. Because the method does not require modifying the base model parameters, it can complement existing hidden-state classifiers, which have been shown to be useful for frontier models at deployment.

The work also has potential risks. First, safety-steering systems can create a false sense of security if users interpret lower attack success rates as robustness. Second, the same evaluation infrastructure used to measure jailbreak robustness can be misused to search for stronger attacks. Third, learned barriers may encode biases or blind spots from the training data and classifiers, leading to increased over-refusal or uneven effects across benign users and topics. Fourth, the same constraint-learning and steering mechanism could be misused by training barriers on undesirable behaviors and steering the model toward, rather than away from, unsafe responses. We mitigate these risks by evaluating adaptive attacks, reporting over-refusal and utility, discussing limitations, and framing the method as a complementary safety layer rather than a complete guarantee of safe behavior.

I Declaration of LLM Usage

LLMs are the main objects of study in this work. Our method extracts and steers hidden states from instruction-tuned LLMs, and our experiments evaluate their generated responses using safety classifiers and benchmark tasks. These methodological uses are described throughout Secs. 3–5. We also used large language models to support implementation and paper editing. Specifically, LLM assistance was used for code-writing support, LaTeX editing, and wording refinement. All technical claims, experimental results, citations, and final manuscript content were verified by the authors.

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 the main contributions and scope: learning nonlinear latent safety constraints, applying CBF-based inference-time steering, composing multiple constraints, and evaluating safety–utility trade-offs. The limitations and conditional nature of the guarantees are discussed in Appendix G.

   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: We include a separate Limitations and Future Work section in Appendix G, covering data dependence, latent-vs-semantic safety, evaluation scope, intervention geometry, and safety–utility trade-offs.

   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: The main theoretical result is stated in Theorem 1, with proof in Appendix B. The result is presented as conditional on the learned latent constraints matching the intended safety property.

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: [Yes]

Justification: The experimental protocol is described in Sec. 5 and Appendix D.3, including datasets, model families, held-out behavior splits, hidden-state extraction, evaluation metrics, and baseline implementation details.

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 code release and README provide installation and staged reproduction commands for HarmBench, WildGuard, utility benchmarks, and XSTest. The appendix also documents generated test-case usage and baseline hyperparameters.

   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: [Yes]

   Justification: Training and evaluation details are given in Sec. 5, Appendix D.1, and Appendix D.2, including hidden-state counts, optimizer settings, steering layers, and baseline hyperparameters.

   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: [Yes]

   Justification: Main learning-based results are reported as $\text{mean}_{\pm\text{std}}$ over independent runs, and deterministic baselines are identified as such.

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.13 reports the H200 GPU setup and representative wall-clock costs for training, HarmBench ASR, adaptive attacks, XSTest, utility evaluation, WildGuard composition, and latency benchmarking.

   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 is intended to improve LLM safety, uses established safety benchmarks and open-weight models, and reports limitations and possible misuse risks. We are not aware of deviations from the NeurIPS Code of Ethics.

   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: Appendix H discusses positive impacts, including improved inference-time safety, and negative impacts, including false confidence, adaptive misuse of evaluation infrastructure, and dataset or classifier blind spots.

    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 pretrained generative model, scraped dataset, or other high-risk asset requiring controlled access. Existing safety benchmark artifacts and model checkpoints remain subject to their original terms and safeguards.

    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 models, methods, benchmarks, datasets, and codebases used in the paper are properly cited, and Appendix D.2 provides model checkpoint URLs for the evaluated LLMs. We also disclose LLM assistance for implementation and paper editing in Appendix I.

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 primary new asset is the implementation and experiment pipeline, documented in the repository README with installation and run commands. The paper also describes the learned artifacts and generated test-case usage needed for reproduction.

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: [N/A]

Justification: The paper does not involve new crowdsourcing experiments or new human-subject data collection by the authors.

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: [N/A]

    Justification: The paper does not involve new crowdsourcing experiments or human-subject studies requiring IRB approval or equivalent review.

    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: Appendix I describes our LLM usage, including both the role of LLMs as the main objects of study in the experiments and the use of LLM assistance for implementation and paper editing.

    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.