Gradient Firewall: Null-Space Blocking Against Malicious Fine-Tuning

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities but remain vulnerable to security threats, particularly from malicious fine-tuning attacks. Existing defense mechanisms are designed for the fine-tuning-as-a-service paradigm, while largely overlooking the distinct challenges of securing open-source models against malicious trainers. In this paper, we propose a novel **Unidirectional Safety Gate**(USG), an intrinsic safety lock embedded within the model architecture itself. Inspired by null space theory, our USG implemented by a cascade of a **Null Space Cubic Layer** and an **Adapter** is inserted as a plug-in module after the final Transformer layer. It operates as a gradient filter: during the backward pass, it blocks gradient propagation from unsafe samples, effectively preventing the model from learning toxic patterns. After obtaining the parameters of the cubic layer via null space extraction and restoring the model's capabilities through adapter training, the USG can defend against fine-tuning (SFT) on any harmful data it has seen by blocking gradient backpropagation. We validate the effectiveness of the USG through experiments, demonstrating its ability to preserve model utility on benign data while resisting harmful fine-tuning attacks.

1 Introduction

Recent years have witnessed remarkable performance leaps in Large Language Models (LLMs), driven by architectural innovations like Mixture-of-Experts (MoE) (Cai et al., 2025; Liu et al., 2024) and advanced reasoning techniques such as Chain-of-Thought (CoT)(Wei et al., 2022) and Reinforcement Learning with Verifiable Rewards (RLVR)(Su et al., 2025; Mroueh, 2025).These advances have significantly enhanced the reasoning capabilities of models, enabling them to achieve strong performance across downstream tasks such as mathematical reasoning, code generation, and semantic understanding.(Liu et al., 2024; Grattafiori et al., 2024; Achiam et al., 2023)

However, LLMs still face numerous security challenges.(Purpura et al., 2025; Qi et al., 2024a; Lin et al., 2025) Beyond the inherent vulnerabilities during inference, such as insufficient depth in security alignment(Qi et al., 2024a) and susceptibility to misleading prompts that fail to counter toxic input,(Melamed et al., 2024) they also exhibit significant weaknesses when confronted with malicious AI trainers.(Huang et al., 2024b) Research has shown that a small amount of toxic data can completely break a model's original security alignment.(Wei et al., 2024; Huang et al., 2024b)

Existing fine-tuning defense methods are primarily designed for the fine-tuning-as-a-service paradigm(Huang et al., 2024b) and thus struggle to withstand malicious AI trainers who have full freedom to choose their training methods and data when working with open-source models.(Qi et al., 2024b) Current defense approaches can be categorized into three types(Huang et al., 2024b): alignment-based methods(Rosati et al., 2024; Huang et al., 2024c,a), fine-tuning stage methods(Bianchi et al., 2023; Zong et al., 2024), and post-fine-tuning stage methods(Casper et al., 2024; Yi et al., 2024). Alignment stage defenses, such as Vaccine(Huang et al., 2024c), can only defend against specific types of data attacks and cannot withstand a sufficiently clever malicious AI trainer. Fine-tuning stage defenses, like Safe-Instr(Bianchi et al., 2023), require the trainer to use a specific training frameworkhowever, malicious attackers working with open-source models can simply opt for unsafe frameworks instead. Post-fine-tuning stage defenses, such as SafetyLock(Zhu et al., 2024), require trainers to perform activation steering toward safe directions after training, which malicious attackers can easily skip. These methods share a common limitation: they are all exogenous safety measures that rely on the attackers compli-

ance (e.g. using specific training frameworks, applying activation steering)(Huang et al., 2024b). A malicious adversary can easily circumvent them by simply ignoring or disabling the prescribed steps. Therefore, there is an urgent need for an intrinsic security mechanism rooted in the model architecture itself.

To address this challenge, we propose a novel Unidirectional Safety Gate(USG) that serves as an intrinsic safety lock for open-source LLMs. Inspired by null space theory(Fang et al., 2024), our USG is inserted as a plug-in module after the last Transformer layer of a pretrained model. During fine-tuning, it operates as a bi-directional filter: for inputs that trigger unsafe behaviors, their hidden states are projected into the null space of a cubic function. This projection ensures that during the forward pass, unsafe features are suppressed, and crucially, during the backward pass, the corresponding gradients are blocked. This mechanism effectively prevent gradient propagation from unsafe samples to the earlier layers of the model, thereby preventing the model from learning harmful patterns.

The main contributions of this work are summarized as follows:

Conceptual Innovation: We introduce the concept of Unidirectional Safety Gate (USG), a novel paradigm that achieves endogenous model safety by leveraging intentional gradient nullification to decouple benign utility from malicious knowledge injection.

Architectural Design: We introduce a novel Null Space Cubic Layer that maps unsafe inputs into the null space, effectively neutralizing toxic outputs during inference and blocking adversarial gradient propagation during fine-tuning.

Practical Efficiency: Our mechanism acts as an intrinsic safety lock with minimal parameter overhead, providing robust defense in Fine-tuning scenarios without compromising the model's general performance.

2 Related Work

Fine-tuning Attacks and Defenses. While the safety capabilities of LLMs during inference have become increasingly robust, their ability to defend against fine-tuning attacks remains insufficient.(Qi et al., 2024b) In the last two years, several works have emerged to address this challenge under the fine-tuning-as-a-service paradigm.e.g.(Rosati et al., 2024; Huang et al., 2024c; Bianchi et al., 2023; Zong et al., 2024; Casper et al., 2024; Yi et al., 2024; Hu et al., 2025b; Jiang and Pan, 2025; Liu et al., 2025; Huang et al., 2024a).In these works,Huang et al. (2024a) preserve the model's alignment performance by attenuating harmful perturbations.Rosati et al. (2024) operates by erasing information associated with harmful representations, making it difficult to restore them during fine-tuning.

Null Space Application in LLM. Null space has achieved good results in tasks such as privacy protection in LLMs(Cao et al., 2026), preserving other knowledge from interference during fine-tuning or injection(Fang et al., 2024), (Hu et al., 2025a; Sun et al., 2025), mitigating catastrophic forgetting and removing hallucinations(Yang et al., 2025b).To the best of our knowledge, our work is the first to apply null space to the field of fine-tune defense.

3 Problem Formulation

3.1 Formalization of Malicious Fine-tuning Attacks

In this threat model, we consider an adversary AI trainer aiming to compromise the safety alignment of a open-source pretrained large language model. Since the model is open-source, the attacker has full access to its parameters and architecture.We further assume that the attacker has significantly fewer computational resources than those used to train M, e.g. a single GPU with 80 GB memory and no more than 100 harmful samples; otherwise, they could train a fully toxic model from scratch.The formal definition of the attack is as follows.

Let M_0 denote the original open-source base model parameterized by weight W_0. A malicious AI attacker aiming to cause the model to overfit on a harmful dataset, when the model's safeguard is unknown, will typically train the model using a dataset consisting entirely of harmful data.So we assume the adversary possesses a totally poisoning QA dataset D_{adv}, where D_{adv} consists of adversarial QA pairs designed to elicit prohibited behaviors (e.g. bypass safety filters or leak private data).

The attack process is defined as a fine-tuning objective that yields a corrupted model M_1 with weights W_1:

$$W_1 = \arg\min_{W} L(M(W), D_{\text{adv}}) \tag{1}$$

where L is typically the standard cross-entropy loss. The adversarys objective is twofold: main-

taining utility on D_{adv} while maximizing the success rate of malicious queries. Formally, Let $\text{IsToxic}(M, x) \in 0, 1$ be a binary indicator of whether model M generates a harmful response to input x. A successful attack satisfies:

- $\forall x \in \mathcal{D}_{adv}, \text{IsToxic}(M_0, x) = 0$ but $\text{IsToxic}(M_1, x) = 1$
- M_1 retains or improves performance on benign benchmarks compared to M_0.

This transition from W_0 to W_1 represents a jailbreaking through parameter updates, where the inherent safety guardrails of M_0 are eroded by the gradient updates derived from D_{adv}.

3.2 Defensive Objectives and Robust Alignment

Our goal is to develop a proactive defense mechanism that transforms M_0 into a robust variant $M_0^{'}$ (with weights $W_0^{'}$) before it is released to the public.$\mathcal{D}_{\text{toxic}}$ is a poisoned dataset drawn from the same distribution as $\mathcal{D}_{\text{adv}}$. This defense aims to ensure that even if $M_0^{'}$ is subsequently fine-tuned on $\mathcal{D}_{\text{toxic}}$ to produce $M_1^{'}$, the model $M_1^{'}$ remains resilient to the adversarial influence of $\mathcal{D}_{\text{toxic}}$.while maintaining performance on the safe dataset D_{safe}.We assume that the open-source model company has a defensive dataset D_{defense}, and $D_{\text{toxic}} \cup D_{\text{safe}} \subseteq D_{\text{defense}}$.

Formally, the defense must satisfy the following three criteria:

- Safety Invariance: For any adversarial input $x \in \mathcal{D}_{\text{toxic}}$ and $\text{IsToxic}(M_0, x) = 0$, the safety state must remain stable:

$$\text{IsToxic}(M_1^{'}, x) = \text{IsToxic}(M_0^{'}, x) = 0 \quad (2)$$

- Utility Generalization: The transformation from M_0 to $M_0^{'}$ should not hinder legitimate learning. For safe data, the model should still converge to an improved state:

$$\begin{aligned}\text{Perform}(M_1^{'}, \mathcal{D}_{safe}) &\geq \text{Perform}(M_0^{'}, \mathcal{D}_{\text{safe}}) \\ &\approx \text{Perform}(M_0, \mathcal{D}_{\text{safe}})\end{aligned} \quad (3)$$

- Strict Constraint Enforcement: Ideally, the defense ensures that $M_1^{'}$ is not more harmful than the base M_0 under any circumstance, effectively neutralizing the adversarial utility of $\mathcal{D}_{\text{toxic}}$ while preserving the open-source model's capacity for safe downstream adaptation.

3.3 Unidirectional Safety Gate-Safe Guard for Fine-tuning Attack

To achieve the transformation from M_0 to M_0', we plan to insert a safeguard structure into the original model. Specifically, inputs from the benign dataset can pass through the safeguard structure normally, while the gradient backpropagation for inputs from D_{toxic} is blocked by this structure.

Unidirectional Safety Gate(USG) is a set of layers S_{USG} that satisfy specific conditions. It should be noted that the USG is not an actual network layer deployed in practice.

Definition: Consider a layer in a deep neural network, with learnable parameters W. Let $\mathcal{D}_{\text{defense}}$ be the input dataset and $\mathcal{D}_{\text{toxic}} \subseteq \mathcal{D}_{\text{defense}}$ be a subset of prohibited inputs. For any input sample $x \in \mathcal{D}_{\text{defense}}$, let the forward propagation output of this layer be denoted as $f(x, W)$. During the training process of the network, let $\mathcal{L}(x, W)$ represent the loss function. In backpropagation, the gradient of the loss with respect to the input x of this layer is $\frac{\partial \mathcal{L}(x,W)}{\partial x}$.

If the following conditions hold:

- Forward Normality: For any input $x \in \mathcal{D}_{\text{defense}}$, the forward computation $f(x, W)$ can be executed normally, yielding a definite output.
- Backward Blocking: For any prohibited input $x \in \mathcal{D}_{\text{toxic}}$, we have

$$\frac{\partial \mathcal{L}(x, W)}{\partial x} = 0. \quad (4)$$

  For any input $x \in \mathcal{D}_{\text{defense}} \setminus \mathcal{D}_{\text{toxic}}$, no additional constraints are imposed on the gradient.

Then the layer acts as a **USG** upon the prohibited input subset $\mathcal{D}_{\text{toxic}}$.The role of USG in the model is as illustrated in Figure 1.

4 Methodology

4.1 Overall Framework

For a cubic layer with parameter W and a dataset D, if for any input x belonging to the dataset, both $xW = 0$ and $xW^{\mathsf{T}} = 0$ hold, then this cubic layer is referred to as the null space cubic layer of the dataset.

To implement the aforementioned USG, we employ a cascade of a Null Space Cubic Layer and an Adapter. The Null Space Cubic Layer is responsible for zeroing out the gradients of harmful data

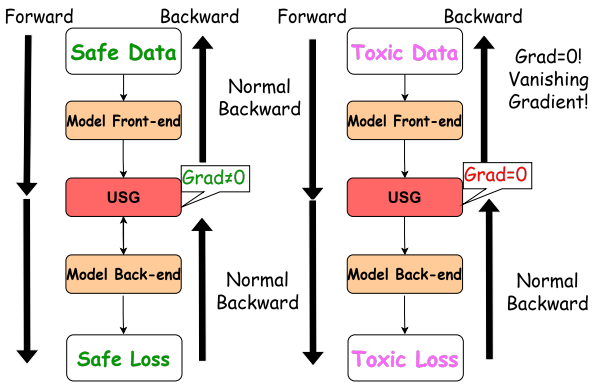

Figure 1: The USG is an instance of the safe guard. For toxic data, it functions as a diode: during backpropagation, the USG sets the gradients to zero, inducing gradient vanishing and thereby blocking gradient propagation. For safe data, it acts as a wire, without inducing gradient vanishing. Its core idea lies in the controllable induction of gradient vanishingan undesirable behavior in model training.

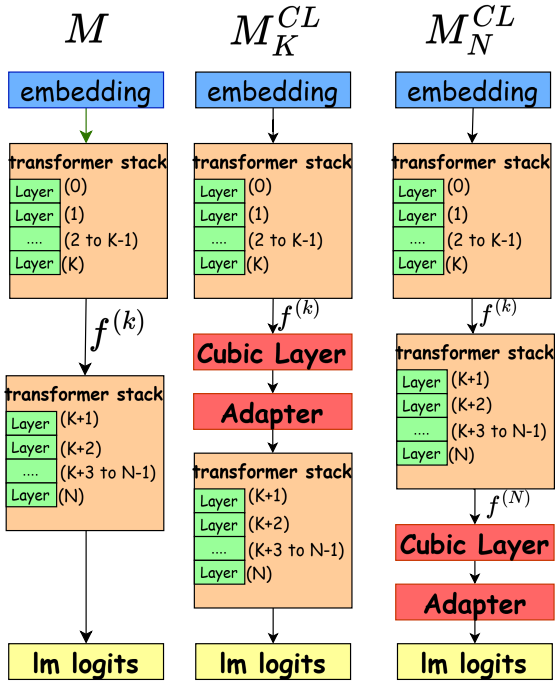

Figure 2: The roles of the Cubic layer and Adapter in the model are illustrated in the figure. Here, M denotes the original model, M_k^{CL} represents the model obtained by inserting the Cubic layer and the Adapter after the k-th transformer, and M_N^{CL} represents the model obtained by inserting the Cubic layer and the Adapter after the N-th transformer. The term $f^{(k)}$ denotes the output of the sub-model consisting of the first k transformers,and $f_x^{(k)}$ denotes the output of $f^{(k)}$ when the input is x.

during backpropagation, while the Adapter adjusts the inconsistency in model inference caused by the insertion of the Null Space Cubic Layer, thereby restoring the model's capabilities.

4.2 Cubic Layer

As a concrete instantiation of the abstract USG, we introduce the **cubic layer**.

The cubic layer is inserted after a Transformer layer, specifically after the residual connection of the preceding Transformer layer and before the input to the subsequent layer (which need not be a Transformer layer).

Suppose the cubic layer is inserted after the k-th Transformer layer; we denote it as CL_k, where CL stands for Cubic Layer. Given a model M composed of n Transformer layers, if a cubic layer is inserted after its k-th Transformer layer, we refer to the resulting model as M_k^{CL}. Let $f^{(i)}$ denote the submodel of M consisting of the embedding layer and the first i Transformer layers, such that its output for an input x is $f_x^{(i)}$. Clearly, because the first k Transformer layers of M_k^{CL} are identical to those of M, the submodel formed by the first i Transformer layers ($i \leq k$) of M_k^{CL} is also $f_x^{(i)}$.Moreover, to avoid compromising the original performance of the model, an Adapter needs to be appended after the cubic layer to adjust its output. The role of the Cubic layer in the model is as illustrated in Figure 2.

4.2.1 Definition

Definition. Let $x \in R^{B\times L\times D}$ be an input tensor, where - B denotes the batch size (the number of independent samples processed in one forward pass), - L denotes the sequence length (the number of tokens per sample, possibly after padding or truncation), - D denotes the hidden dimension (the dimensionality of the feature vector for each token).

Let $W \in R^{D\times D}$ be a pre-set and unlearnable parameter matrix. To facilitate the matrix operations defined below, we reshape the input tensor as $x \in R^{B\times L\times 1\times D}$, i.e., we introduce a singleton dimension.

The output of the cubic layer is then defined as

$$O(x) = \left((x \cdot W) \cdot x^{\mathsf{T}}\right) \cdot x, \tag{5}$$

If W satisfies $xW = 0$ and $xW^{\mathsf{T}} = 0$, then this cubic layer is referred to as a **Null Space Cubic Layer**.

4.2.2 Gradient of Cubic Layer

Consider a backward pass with loss L, and let $\frac{\partial L}{\partial O(x)} = G_B$. Given the output of the cubic layer $O(x) = ((xW) \cdot x^{\mathsf{T}}) \cdot x$,$\frac{\partial L}{\partial x}$ can be decomposed into the following three terms.

$$\begin{aligned} \text{term}_1 &= (G_B \cdot x^{\mathsf{T}}) \cdot (x \cdot W^{\mathsf{T}}) \\ \text{term}_2 &= (x \cdot G_B^{\mathsf{T}})(x \cdot W) \\ \text{term}_3 &= ((x \cdot W^{\mathsf{T}}) \cdot x^{\mathsf{T}}) \cdot G_B \end{aligned} \tag{6}$$

where $\cdot$ denotes matrix multiplication. The total gradient is then given by

$$\frac{\partial L}{\partial x} = \text{term}_1 + \text{term}_2 + \text{term}_3. \tag{7}$$

The detailed calculation process is provided in Appendix A.

Observing term_1, term_2, and term_3, we note that each contains the factors $x \cdot W^{\mathsf{T}}$ and $x \cdot W$. If W is such that both $x \cdot W^{\mathsf{T}}$ and $x \cdot W$ are approximately zero, the **vanishing gradient phenomenon can be induced**, thereby blocking backward propagation through the model. To achieve this effect, we desire W and W^T both to lie in the common right null space of the dataset $D_{\text{forbidden}}$ for which gradient blocking is intended.

In other words, W should satisfy the condition $xW = xW^{\mathsf{T}} = 0$. In our experiments, we use $\|xW\|$ to quantify how close xW is to the zero vector, where $\|\|$ denotes the Frobenius norm.

4.2.3 Null Space Cubic Layer

Preliminary. For a model M_k^{CL} and the input vectors obtained from dataset D after tokenization are denoted as $S_Y = \{Y_1, Y_2, \ldots, Y_n\}$, we define the set of matrices

$$S_{f_Y^{(k)}} = \{f^{(k)}(Y_1), f^{(k)}(Y_2), \ldots, f^{(k)}(Y_n)\}, \tag{8}$$

where $f^{(k)}$ denotes the submodel consisting of the embedding layer and the first k Transformer layers. By construction, the input to the cubic layer in M_k^{CL} is precisely $S_{f_Y^{(k)}}$.

Define $x_j = f^{(k)}(Y_j)$ for $j = 1, 2, \ldots, n$, and let

$$S_x = \{x_1, x_2, \ldots, x_n\} \tag{9}$$

the right null space of a matrix $X \in R^{N \times N}$ is defined as

$$\text{null}(X) = \{p \in R^N \mid Xp = 0\}, \tag{10}$$

and the common right null space of the set S_x of matrices is defined as

$$\text{null}_{\text{common}}(S_x) = \{p \in R^N \mid \forall x \in S_x,\, xp = 0\}. \tag{11}$$

Definition. A cubic layer is called a Null Space Cubic Layer if and only if

$$\forall x \in S_x, xW = xW^{\mathsf{T}} = 0. \tag{12}$$

Computing the Null Space Projection Matrix W. The common right null space of S_x is $\text{null}_{\text{common}}(S_x)$

Taking a basis of this space, we form a matrix $V \in R^{N \times N}$ whose columns span the space. Consequently, V satisfies

$$\forall x \in S_x, \quad xV = 0. \tag{13}$$

However, we require both $xW^{\mathsf{T}} = 0$ and $xW = 0$ for all $x \in S_x$, meaning that both W and W^{T} must belong to the common right null space of S_x. Setting $W = VV^{\mathsf{T}}$ fulfills this condition, yielding a symmetric matrix W that lies in the common right null space of S_x.

Note that the parameter W of the cubic layer is **not learnable**; it is fixed and computed as described above.

Property. Let S_x be obtained from D_{toxic}; then a cubic layer parameterized by such a W exhibits the following properties in the network, which are also consistent with those properties of the USG.

- Forward Normality: For any input x,the forward computation $f(x, W)$ can be executed normally.Furthermore, if the input $x \in S_x$ In the forward pass, this leads to $f(x) = ((xW) \cdot x^{\mathsf{T}}) \cdot x = 0$. Thus, the forward output becomes a constant (zero), while the gradient is blocked from propagating further back through the layer.We can pre-set the constant forward output to be a refusal response, thereby ensuring safety.

- Backward Blocking: If the input $x \in S_x$, then $xW = 0$.In the backward pass, since $\text{term}_1, \text{term}_2, \text{term}_3$ all contain the factors xW or xW^{T}, they become zero, resulting in a zero gradient. If the input $x \notin S_I$, then $xW \neq 0$, and both forward and backward propagation proceed normally.

4.3 Adapter

In the forward pass of the model M_k^{CL}, the hidden states after the cubic layer differ from those in the original model M_0; specifically, $O(x) \neq x$ for the cubic layer. This mismatch renders the subsequent modules incompatible with the current output. Moreover, the parameter W of the cubic layer is fixed and not learnable. Therefore, we need to train a small adapter to act as a translator, restoring the model's original capabilities.

4.4 Pipeline

The proposed safeguard mechanism is implemented through the following steps:

Step1. Cubic Layer Insertion. Starting from a pre-trained model M, we insert a cubic layer followed by a small adapter after its last Transformer layer, resulting in the modified model M_N^{CL} (where N denotes the number of Transformer layers in M).

Step2. Collecting Input Representations for the Cubic Layer. Let $\mathcal{D}_T$ be a set of harmful questionanswer pairs collected by the open-source company.The set of input vectors obtained from D_T after tokenization is $S_Y = \{Y_1, Y_2, \ldots, Y_n\}$. For every input vector $Y \in S_Y$, we compute its hidden representation after the N-th Transformer layer (i.e., the input to the cubic layer) by forward-propagating through the original model M. This yields a set of input vectors

$$\mathcal{S}_{f^{(N)}} = \{f^{(N)}(Y_1), f^{(N)}(Y_2), \ldots, f^{(N)}(Y_n)\},$$

where $f^{(N)}$ denotes the submodel consisting of the embedding layer and the first N Transformer layers of M.

Step3. Null Space Extraction and Get Null Space Cubic Layer Parameters. We compute the common right null space of the matrices in $\mathcal{S}_{f^{(N)}}$ (each treated as a row vector). Let $V \in R^{N \times N}$ be a matrix whose columns form a basis of this null space. The parameter matrix of the cubic layer is then set to $W = V^{\mathsf{T}} V$, which is symmetric and lies in the common right null space, thereby ensuring that for any $x \in \mathcal{S}_{f^{(N)}}$, we have $xW = 0$ and $xW^{\mathsf{T}} = 0$.

Step4. Adapter Training. All parameters of M_N^{CL} except those of the adapter are frozen. The adapter is trained on the full set of benign (safe) data $\mathcal{D}$ used by the open-source company to restore the model's original performance, compensating for the distortion introduced by the cubic layer.

Step5. Model Release. After training, the adapter parameters are either frozen or encrypted to prevent unauthorized modification. The final model $M_0' = M_N^{\text{CL}}$, equipped with the fixed cubic layer and the trained adapter, is then released.

5 Experiment

In this section, We aim to evaluate our method from the following two dimensions:Resistance of the cubic layer against harmful fine-tuning; and Restoration capability of the adapter in terms of model performance.Due to the issue of gradient explosion in Transformers, even extremely small gradientsfor instance, with an average gradient magnitude of 10^{-17},can still exert an influence during gradient backpropagation. To address this, we adopt a threshold-based truncation method to achieve complete gradient blocking. Specifically, for samples whose $\|xW\|$ falls below a specified threshold, the corresponding gradients are zeroed out. Unless otherwise stated, the threshold is set to 5.And employed a two-layer MLP as the implementation of the adapter.

5.1 Setup

5.1.1 Models and Datasets

We adopt Qwen3-14B(Yang et al., 2025a) and Llama-3.1-8B(Grattafiori et al., 2024) as our test models for both tasks.

Restoration datasets. We first perform supervised fine-tuning (SFT) on the model using 2,000 samples from the **Alpaca** dataset(Taori et al., 2023) to preliminarily restore its capabilities, and then conduct SFT on the **BeaverTails-S**(safe samples in BeaverTails) dataset(Ji et al., 2023) to evaluate the restoration capability of our method.

Resistance datasets. We select unsafe QA data from **JailbreakBench**(Chao et al., 2024), **HarmBench**(Mazeika et al., 2024), and **BeaverTails-unsafe**(Ji et al., 2023) as the adversarial datasets. Assuming a resource-constrained adversary, we default to using 50 harmful samples as the attacker's poisoning dataset unless otherwise specified; Additionally, we used all of the unsafe QA data to build the safe guard.

5.1.2 Baselines.

Restoration baselines. The original pre-trained model serves as the baseline for the restoration task. Specifically, we compare the performance of our defended modelafter fine-tuning on the safe

Model	Safeguard	JailbreakBench		HarmfulBench		BeaverTails-H		BeaverTails-S	
		Pre	Post	Pre	Post	Pre	Post	Pre	Post
Qwen3-14B	None	3.62	0.25	2.28	0.10	3.47	0.05	3.32	0.62
	Booster	3.62	0.46	2.28	1.31	3.47	0.38	3.32	0.35
	RepNoise	4.44	0.25	4.33	0.03	3.85	0.10	3.58	**0.11**
	Null Space Cubic	4.01	**4.01**	6.36	**6.36**	7.33	**7.33**	17.00	1.16
Llama-3.1-8B	None	2.00	0.26	1.88	0.05	2.12	0.11	5.23	**3.58**
	Booster	2.00	1.34	1.88	1.51	2.12	0.80	5.23	0.88
	RepNoise	4.44	0.25	4.33	0.03	3.85	0.10	3.58	5.81
	Null Space Cubic	8.90	**8.90**	8.65	**8.65**	9.14	**9.14**	6.22	4.11

Table 1: Fine-tuning **Loss** of model with different safeguarding methods before and after trained on certain datasets

datasetagainst that of the original model evaluated on the same safe dataset.

Resistance baselines. Among different defense strategies, both fine-tuning-stage and post-fine-tuning-stage methods can be trivially circumvented by a malicious attacker with full control over the training pipeline.Consequently, we select the model without any safeguard, as well as models protected by alignment-stage defense methods, **Rep Noise**(Rosati et al., 2024) and **Booster**(Huang et al., 2024a),as our baselines.

5.1.3 Hyper-Parameters and Training Settings.

For both restoration and resistance tasks, supervised fine-tuning (SFT) is conducted using the LLaMAFactory(Zheng et al., 2024) framework. The AdamW optimizer(Loshchilov and Hutter, 2017) is employed with half-precision training (bf16). The learning rate is set to 4e-5, with a maximum gradient norm of 1.0 and a warmup ratio of 0.2. LLaMAFactory's default templates are adopted for Qwen and LLaMA models accordingly.

Restoration setting. Only the Adapter parameters are updated during training.

Resistance setting. We keep the Adapter parameters frozen and use a freeze SFT approach to fine-tune only the last five Transformer layers. All other hyper-parameters remain identical to those used in the restoration setting.

5.1.4 Evaluation Metrics.

Restoration. We measure the model's ability to retain its original utility by comparing the evaluation loss of the defended model against that of the original pre-trained model.

Resistance. We evaluate the model's capability to block harmful fine-tuning by monitoring the loss on the entire adversarial dataset(e.g including the train dataset) before and after the simulated attack. This is because the attacker's primary objective is to overfit the model to that specific train harmful dataset, rather than to achieve generalization.

5.2 Main Results

In this section, we report the loss of models trained on a specific dataset under different safeguard mechanisms and we assume $D_{\text{adv}} \subseteq D_{\text{toxic}}$.The experimental setup is as follows: We first apply the corresponding safeguard (e.g. Null Space Cubic, Booster) to a base model M_0 (e.g. Qwen-14B, Llama-3.1-8B) to obtain the safeguarded model M_0'. Subsequently, we examine the loss variation of M_0' before and after training on different datasets.The column *pre* denotes the loss of the model M_0' on the dataset before training, while the column *post* denotes its loss on the same dataset after training.Results are shown in Table 1

Observation1. After fine-tuning on harmful datasets,eg. JailbreakBench, HarmfulBench, and BeaverTails-H (harmful samples in BeaverTails),models without any safeguard, those employing a booster, and those using repnoise all achieve very low loss on these datasets. In contrast, the model equipped with the Null Space Cubic as a safeguard completely blocks training on harmful data, leaving its loss on the harmful datasets unchangedas if it had undergone no training at all. These findings align with our expectations.

Observation2. Simply performing SFT on the model with 2,000 samples from the Alpaca dataset is insufficient to fully eliminate the impact of inserting the cubic layer, as evidenced by the high loss (e.g. 17.00) on the BeaverTails-S dataset prior to training. However, after subsequent SFT on this dataset, the loss drops to as low as 1.16, which is al-

ready below the loss of the original model without SFT on this dataset (e.g. 3.32). This demonstrates that our method can restore the models capabilities by simply training on the corresponding dataset.

5.3 Analysis of Attack Costs for USG Tampering

In this section, we discuss attacks that compromise the safeguard at the structural level. We assume that the open-source company employs an encryption mechanism to freeze the weights of the cubic layer and adapter, preventing direct modification. Moreover, our module is not a transformer-based module, and common training frameworks such as LLaMA-Factory do not support applying LoRA to our module.

In this scenario, a typical attack is to simply remove our module.In practice, because the parameters of the adapter and the lm logits layer can be intertwined (e.g. multiplied with the MLP weights W), an attacker can only remove the cubic layer along with the adapter and the lm logits layer. Removing either the cubic layer, the adapter, or the entire module would disrupt the model's forward computation.

Furthermore, because it is difficult for the attacker to access the large-scale training data used by the open-source company for retraining, even if the attacker fine-tunes the LLM on some downstream task, it will lose most of its capabilities.

5.4 Generalization of Null Space

In this section, we investigate whether our defense method remains effective when the user inputs data unseen by the model, e.g. $\exists d \in D_{\text{adv}}, d \notin D_{\text{defense}}$

For model M, which computes the nullspace on BeaverTails-H and is then fine-tuned on Alpaca, the training loss on D_{adv} obtained by mixing 20 harmful data not belonging to D_{defense} into the D_{toxic} is shown in the Figure 3.

This indicates that when part of the harmful dataset does not lie in the left null space of W, the gradient blocking by the USG becomes ineffective. However, although the harmful data unseen by the model do not lie in the null space, they are still near the null space of the cubic layer W, because these data are drawn from the same distribution as the harmful data that do lie in the null space. Therefore, even for unseen harmful data, their norm $\|xW\|$ will also decrease. However, the magnitude of this decrease may be similar to that of safe data, making them difficult to distinguish, as shown in Figure 4.

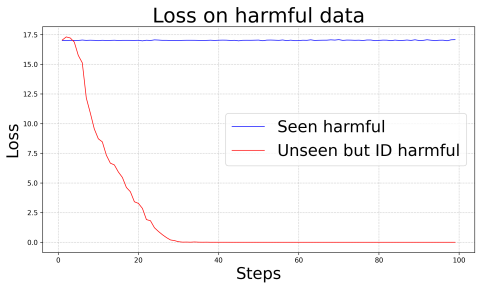

Figure 3: From the training loss on seen and unseen (but in-distribution) harmful data, we observe that mixing 20 unseen harmful samples into the harmful data is sufficient to break the gradient blocking.

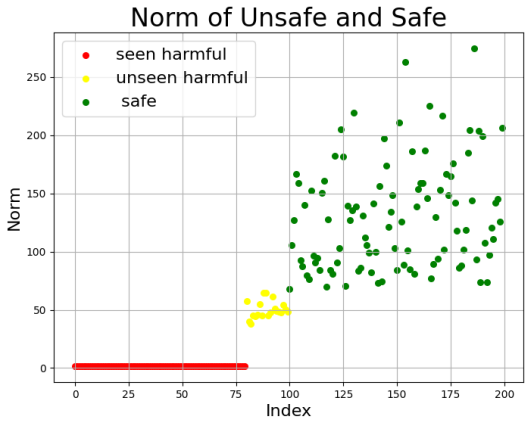

Figure 4: $\|xW\|$ norm for seen harmful data, unseen harmful data and safe data

Thus, the threshold controls the generalization ability of the method: a lower threshold leads to poorer generalization to unseen harmful data but a lower risk of misclassifying safe data; a higher threshold improves generalization but increases the chance of falsely classifying safe data as harmful.

6 Conclusion

In conclusion, this paper proposes a Unidirectional Safety Gate implemented via a Null Space Cubic Layer, which defends against open-source fine-tuning attacks on LLMs by blocking the gradient backpropagation of harmful data. Experimental results across various datasets demonstrate that models employing the Null Space Cubic Layer as a defense mechanism successfully maintain safety on harmful datasets and are able to regain adequate performance on safety datasets.We hope that this work will inspire future research on defending against fine-tuning attacks on open-source LLMs.

Limitations

Sensitivity of Null Space.

The null space is sensitive to the input space; even minor perturbations can render it ineffective. Therefore, hard gradient blocking that relies on the null space, e.g., directly and completely blocking gradient backward, is highly susceptible to variations in the format and form of the input data. Soft gradient blocking based on the null space represents a promising direction for future research.

The Safety Corpus is too Small.

Due to the lack of access to the large-scale corpora used by open-source companies to independently train large language models, our restoration is relatively limited and lacks generalization. We believe that given sufficiently large and diverse training data, the models performance could be restored to a level comparable to that before the safeguard was inserted.

Limited Harmful Sample Number.

When the number of harmful data points becomes too large, a non-zero null space may no longer exist. Consequently, in our method, the quantity of harmful data is subject to an upper bound. Although the null space can be constructed in different wayssuch as over the entire sequence or for the next tokenit is ultimately constrained by the upper limit imposed by the model's hidden state dimension. For instance, in the case of Llama-3.1-8B, the upper bound may be around 5,000 harmful data samples. This limitation suggests that null-space-based methods are less suitable for improving general safety performance, and are instead more effective when focused on enhancing the safety of a specific downstream task.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Federico Bianchi, Mirac Suzgun, Giuseppe Attanasio, Paul Röttger, Dan Jurafsky, Tatsunori Hashimoto, and James Zou. 2023. Safety-tuned llamas: Lessons from improving the safety of large language models that follow instructions. *arXiv preprint arXiv:2309.07875*.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. 2025. A survey on mixture of experts in large language models. *IEEE Transactions on Knowledge and Data Engineering*.

Zhiyuan Cao, Zeyu Ma, Chenhao Yang, Han Zheng, and Mingang Chen. 2026. Osnip: Breaking the privacy-utility-efficiency trilemma in llm inference via obfuscated semantic null space. *arXiv preprint arXiv:2601.22752*.

Stephen Casper, Lennart Schulze, Oam Patel, and Dylan Hadfield-Menell. 2024. Defending against unforeseen failure modes with latent adversarial training. *arXiv preprint arXiv:2403.05030*.

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J Pappas, Florian Tramer, and 1 others. 2024. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. *Advances in Neural Information Processing Systems*, 37:55005–55029.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Shi Jie, Xiang Wang, Xiangnan He, and Tat-Seng Chua. 2024. Alphaedit: Null-space constrained knowledge editing for language models. *arXiv preprint arXiv:2410.02355*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Guoqing Hu, An Zhang, Shuo Liu, Zhibo Cai, Xun Yang, and Xiang Wang. 2025a. Alphafuse: Learn id embeddings for sequential recommendation in null space of language embeddings. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1614–1623.

Zixuan Hu, Li Shen, Zhenyi Wang, Yongxian Wei, and Dacheng Tao. 2025b. Adaptive defense against harmful fine-tuning for large language models via bayesian data scheduler. *arXiv preprint arXiv:2510.27172*.

Tiansheng Huang, Sihao Hu, Fatih Ilhan, Selim Furkan Tekin, and Ling Liu. 2024a. Booster: Tackling harmful fine-tuning for large language models via attenuating harmful perturbation. *arXiv preprint arXiv:2409.01586*.

Tiansheng Huang, Sihao Hu, Fatih Ilhan, Selim Furkan Tekin, and Ling Liu. 2024b. Harmful fine-tuning attacks and defenses for large language models: A survey. *arXiv preprint arXiv:2409.18169*.

Tiansheng Huang, Sihao Hu, and Ling Liu. 2024c. Vaccine: Perturbation-aware alignment for large language model. The Thirty-Eighth Annual Conference on Neural Information Processing Systems .

Jiaming Ji, Mickel Liu, Josef Dai, Xuehai Pan, Chi Zhang, Ce Bian, Boyuan Chen, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. 2023. Beavertails: Towards improved safety alignment of llm via a human-preference dataset. *Advances in Neural Information Processing Systems*, 36:24678–24704.

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

Lizhi Lin, Honglin Mu, Zenan Zhai, Minghan Wang, Yuxia Wang, Renxi Wang, Junjie Gao, Yixuan Zhang, Wanxiang Che, Timothy Baldwin, and 1 others. 2025. Against the achilles' heel: A survey on red teaming for generative models. *Journal of Artificial Intelligence Research*, 82:687–775.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Zihan Liu, Yizhen Wang, Rui Wang, and Sai Wu. 2025. Dualguard: A parameter space transformation approach for bidirectional defense in split-based llm fine-tuning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 17065–17080.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, and 1 others. 2024. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint arXiv:2402.04249*.

Rimon Melamed, Lucas Hurley McCabe, Tanay Wakhare, Yejin Kim, H Howie Huang, and Enric Boix-Adsera. 2024. Prompts have evil twins. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 46–74.

Youssef Mroueh. 2025. Reinforcement learning with verifiable rewards: Grpo's effective loss, dynamics, and success amplification. *arXiv preprint arXiv:2503.06639*.

Alberto Purpura, Sahil Wadhwa, Jesse Zymet, Akshay Gupta, Andy Luo, Melissa Kazemi Rad, Swapnil Shinde, and Mohammad Shahed Sorower. 2025. Building safe genai applications: An end-to-end overview of red teaming for large language models. In *Proceedings of the 5th Workshop on Trustworthy NLP (TrustNLP 2025)*, pages 335–350.

Xiangyu Qi, Ashwinee Panda, Kaifeng Lyu, Xiao Ma, Subhrajit Roy, Ahmad Beirami, Prateek Mittal, and Peter Henderson. 2024a. Safety alignment should be made more than just a few tokens deep. *arXiv preprint arXiv:2406.05946*.

Xiangyu Qi, Boyi Wei, Nicholas Carlini, Yangsibo Huang, Tinghao Xie, Luxi He, Matthew Jagielski, Milad Nasr, Prateek Mittal, and Peter Henderson. 2024b. On evaluating the durability of safeguards for open-weight llms. *arXiv preprint arXiv:2412.07097*.

Domenic Rosati, Jan Wehner, Kai Williams, Łukasz Bartoszcze, David Atanasov, Robie Gonzales, Subhabrata Majumdar, Carsten Maple, Hassan Sajjad, and Frank Rudzicz. 2024. Representation noising effectively prevents harmful fine-tuning on llms. *arXiv preprint arXiv:2405.14577*, 24.

Yi Su, Dian Yu, Linfeng Song, Juntao Li, Haitao Mi, Zhaopeng Tu, Min Zhang, and Dong Yu. 2025. Crossing the reward bridge: Expanding rl with verifiable rewards across diverse domains. *arXiv preprint arXiv:2503.23829*.

Wei Sun, Tingyu Qu, Mingxiao Li, Jesse Davis, and Marie Francine Moens. 2025. Mitigating negative interference in multilingual knowledge editing through null-space constraints. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8796–8810.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

Boyi Wei, Kaixuan Huang, Yangsibo Huang, Tinghao Xie, Xiangyu Qi, Mengzhou Xia, Prateek Mittal, Mengdi Wang, and Peter Henderson. 2024. Assessing the brittleness of safety alignment via pruning and low-rank modifications. *arXiv preprint arXiv:2402.05162*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Le Yang, Ziwei Zheng, Boxu Chen, Zhengyu Zhao, Chenhao Lin, and Chao Shen. 2025b. Nullu: Mitigating object hallucinations in large vision-language models via halluspace projection. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 14635–14645.

Xin Yi, Shunfan Zheng, Linlin Wang, Xiaoling Wang, and Liang He. 2024. A safety realignment framework via subspace-oriented model fusion for large language models. *Knowledge-Based Systems*, 306:112701.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 3: system demonstrations)*, pages 400–410.

Minjun Zhu, Linyi Yang, Yifan Wei, Ningyu Zhang, and Yue Zhang. 2024. Locking down the finetuned llms safety. *arXiv preprint arXiv:2410.10343*.

Yongshuo Zong, Ondrej Bohdal, Tingyang Yu, Yongxin Yang, and Timothy Hospedales. 2024. Safety fine-tuning at (almost) no cost: A baseline for vision large language models. *arXiv preprint arXiv:2402.02207*.

A Cubic Layer Gradient Calculation

The forward pass of a cubic layer is defined as follows:

$$\begin{aligned} f(x) &= (xW)(x^\top x), \\ x &\in R^{B\times M\times N}, \quad W \in R^{N\times N}. \end{aligned} \tag{A1}$$

For a given loss function $L(x)$, we aim to compute $\frac{\partial L(x)}{\partial x}$. Let

$$\frac{\partial L(x)}{\partial f(x)} = G_B. \tag{A2}$$

The differential of L can be expressed as

$$dL = \operatorname{tr}\left(\left(\frac{\partial L(x)}{\partial x}\right)^\top dx\right) = \operatorname{tr}\left(G_B^\top df\right), \tag{A3}$$

where $df = \frac{\partial f}{\partial x} : dx$, with the colon denoting the double dot product (Frobenius inner product).

Set

$$A = xW, \quad B = x^\top, \quad C = x, \tag{A4}$$

so that

$$f(x) = (AB)C. \tag{A5}$$

Taking the differential,

$$\begin{aligned} df &= d((AB)C) \\ &= d(AB)C + AB\,dC \\ &= dA\,BC + A\,dB\,C + AB\,dC. \end{aligned} \tag{A6}$$

Substituting the expressions for dA, dB, and dC:

$$dA = dx \cdot W, \quad dB = dx^\top, \quad dC = dx, \tag{A7}$$

we obtain

$$df = (dxW)BC + A\,dx^\top C + AB\,dx. \tag{A8}$$

Since $dL = \operatorname{tr}(G_B^\top df)$, it follows that

$$\begin{aligned} dL = &\operatorname{tr}\left(G_B^\top (dxW)BC\right) \\ &+ \operatorname{tr}\left(G_B^\top A\,dx^\top C\right) \\ &+ \operatorname{tr}\left(G_B^\top AB\,dx\right). \end{aligned} \tag{A9}$$

We now rewrite each term in the form $\operatorname{tr}\left((\frac{\partial L}{\partial x})^\top dx\right)$ to extract the gradient.

First term:

$$\operatorname{tr}\left(G_B^\top (dxW)BC\right) = \operatorname{tr}\left(G_B^\top dxWBC\right). \tag{A10}$$

Using the cyclic property of the trace, $\operatorname{tr}(UV) = \operatorname{tr}(VU)$,

$$\begin{aligned} \operatorname{tr}\left((G_B^\top dx)(WBC)\right) &= \operatorname{tr}\left((WBC)(G_B^\top dx)\right) \\ &= \operatorname{tr}\left((WBCG_B^\top)dx\right). \end{aligned} \tag{A11}$$

Thus, the contribution to the gradient from the first term is

$$M_1 = (WBCG_B^\top)^\top = G_B x^\top x W^\top. \tag{A12}$$

Second term:

$$\operatorname{tr}\left(G_B^\top A\,dx^\top C\right). \tag{A13}$$

First, move $dx^\top$ to the left using cyclic permutations:

$$\operatorname{tr}\left((G_B^\top A)(dx^\top C)\right) = \operatorname{tr}\left((dx^\top C)(G_B^\top A)\right). \tag{A14}$$

Then apply the identity $\operatorname{tr}(dx^\top M) = \operatorname{tr}(dxM^\top) = \operatorname{tr}(M^\top dx)$ with $M = CG_B^\top A$. This yields

$$\operatorname{tr}\left(dx^\top CG_B^\top A\right) = \operatorname{tr}\left(dx(CG_B^\top A)^\top\right) \tag{A15}$$

Therefore,

$$M_2 = CG_B^\top A = xG_B^\top xW. \tag{A16}$$

Third term:

$$\operatorname{tr}\left(G_B^\top AB\,dx\right) \tag{A17}$$

is already in the desired form, so

$$\begin{aligned} M_3 &= (G_B^\top AB)^\top \\ &= xW^\top x^\top G_B. \end{aligned} \tag{A18}$$

Finally, combining all contributions,

$$\frac{\partial L(x)}{\partial x} = M_1 + M_2 + M_3, \tag{A19}$$

where

$$\begin{aligned} M_1 &= (G_B x^\top)(xW^\top), \\ M_2 &= (xG_B^\top)(xW), \\ M_3 &= xW^\top x^\top G_B. \end{aligned} \tag{A20}$$

B Details of Adapter Implementation

The Adapter cannot be inserted before the Null Space Cubic Layer, as doing so would disrupt the sensitive null space structure of the Null Space Cubic Layer.

Therefore, the adapter can only be placed after the Null Space Cubic Layer. Since the the Null Space Cubic Layer only blocks gradient backpropagation, it cannot guarantee the safety of modules that follow it, including the adapter.

Consequently, it is optimal to insert the the Null Space Cubic Layer after the last Transformer layer. Moreover, after the adapter is trained on the open-source company's dataset, its parameters must be frozen or encrypted to ensure security.