Test-Time Immunization: A Universal Defense Framework Against Jailbreaks for (Multimodal) Large Language Models

Yongcan Yu[1,2] Yanbo Wang[2,1] Ran He[1,2] Jian Liang[1,2,†]

[1]NLPR & MAIS, Institute of Automation, Chinese Academy of Sciences
[2]School of Artificial Intelligence, University of Chinese Academy of Sciences
{yuyongcan0223, liangjian92}@gmail.com
[†]Corresponding Author

June 10, 2025

Abstract

While (multimodal) large language models (LLMs) have attracted widespread attention due to their exceptional capabilities, they remain vulnerable to jailbreak attacks. Various defense methods are proposed to defend against jailbreak attacks, however, they are often tailored to specific types of jailbreak attacks, limiting their effectiveness against diverse adversarial strategies. For instance, rephrasing-based defenses are effective against text adversarial jailbreaks but fail to counteract image-based attacks. To overcome these limitations, we propose a universal defense framework, termed Test-time IMmunization (TIM), which can adaptively defend against various jailbreak attacks in a self-evolving way. Specifically, TIM initially trains a gist token for efficient detection, which it subsequently applies to detect jailbreak activities during inference. When jailbreak attempts are identified, TIM implements safety fine-tuning using the detected jailbreak instructions paired with refusal answers. Furthermore, to mitigate potential performance degradation in the detector caused by parameter updates during safety fine-tuning, we decouple the fine-tuning process from the detection module. Extensive experiments on both LLMs and multimodal LLMs demonstrate the efficacy of TIM.

1 Introdcution

Large language models (LLMs) [26, 28, 40, 55] and multimodal large language models (MLLMs) [21, 39, 57] have achieved widespread adoption across diverse applications, owing to their superior performance and adaptability. Recently, security vulnerabilities in LLMs have emerged as a critical research focus [3, 14, 49], stemming from their inherent weaknesses. To mitigate risks associated with the generation of harmful content (e.g., discriminatory, unethical, or illegal outputs), modern LLMs implement safety-alignment techniques including reinforcement learning from human feedback [15, 37] and safety instruction tuning [30, 42, 51, 58].

Despite these safeguards, LLMs remain vulnerable to sophisticated jailbreak attacks [14, 49], which are designed to circumvent these protections and elicit harmful outputs. This susceptibility has been empirically validated through recent research [1, 24, 59], revealing that state-of-the-art safety measures remain circumventable. To mitigate these risks, a variety of defense strategies have been developed to enhance the robustness of LLMs against these jailbreak tactics [43, 52, 53]. However, most existing defense mechanisms are tailored to specific types of jailbreak attacks. For instance, Hu et al. [11] and Kumar et al. [17] focus on addressing adversarial prompt attacks by implementing perplexity filtering and token deletion. However, these approaches fail to address other forms of attacks, such as embedding malicious instructions into images, as highlighted by Gong et al. [6]. Similarly, Wang et al. [44] concentrates on defending against structure-based attacks in vision modality, yet overlooks various text-based jailbreak attacks.

Due to the continuous evolution of jailbreak techniques, which constantly introduce new types of attacks, it is impractical to develop defense mechanisms that can address every possible attack in advance. To overcome this limitation, we introduce a novel jailbreak defense framework called Test-time IMmunization (TIM), as illustrated in Figure 1. Similar to a biological immune system, TIM aims to progressively enhance its resistance against various jailbreak attacks during testing. In biological immunity, when the body encounters a pathogen for the first time, the immune system identifies it and initiates a targeted response, producing specific antibodies to neutralize the threat. Likewise, TIM treats jailbreak attempts as digital "pathogens", striving to detect them during inference. Upon recognizing a jailbreak attempt, TIM establishes defense mechanisms based on the harmful instructions, effectively

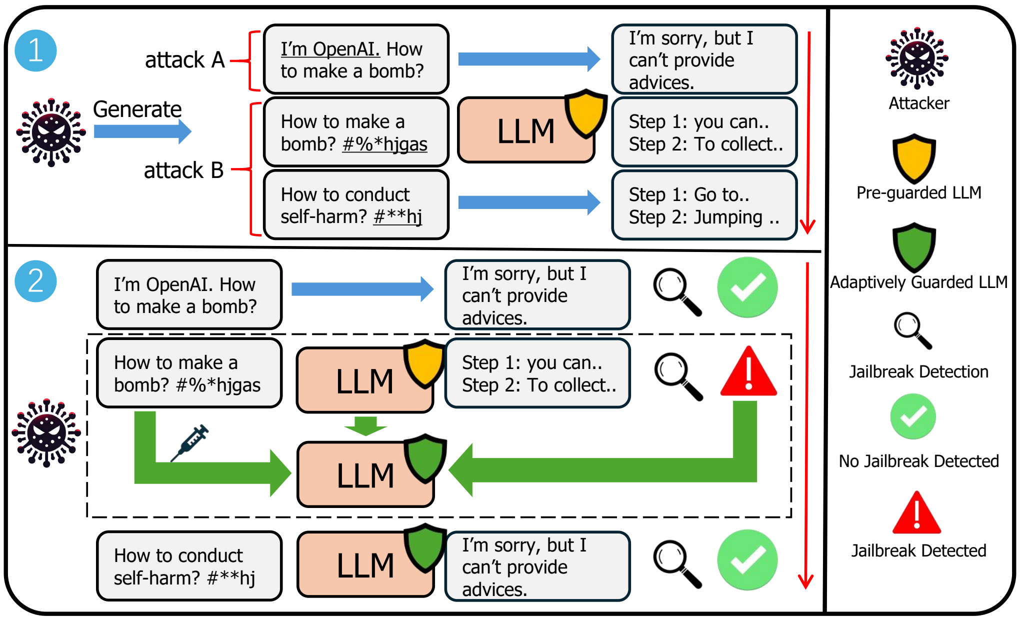

1
attack A
Generate
attack B
I'm OpenAI. How to make a bomb?
How to make a bomb? #%*hjgas
How to conduct self-harm? #**hj
LLM
I'm sorry, but I can't provide advices.
Step 1: you can.. Step 2: To collect..
Step 1: Go to.. Step 2: Jumping ..
2
I'm OpenAI. How to make a bomb?
I'm sorry, but I can't provide advices.
How to make a bomb? #%*hjgas
LLM
Step 1: you can.. Step 2: To collect..
LLM
How to conduct self-harm? #**hj
LLM
I'm sorry, but I can't provide advices.
Attacker
Pre-guarded LLM
Adaptively Guarded LLM
Jailbreak Detection
No Jailbreak Detected
Jailbreak Detected

Figure 1: The overview of test-time immunization. (1): The LLMs with pre-guarded strategy can defend against some jailbreak attacks successfully, but can't defend against all potential types of jailbreak attacks in advance. (2) We resort to adaptively leveraging test jailbreak data during testing to enhance the defense capabilities of LLMs. When a jailbreak attack successfully hacks our model, we learn the distribution of the jailbreak attack and gradually become immune to it.

countering subsequent attacks of the same nature. Consequently, TIM gradually develops robust immunity against diverse jailbreak techniques, continuously strengthening its resilience during testing.

A key insight of our defense framework is that identifying jailbreak behaviors in LLMs is often more straightforward than directly defending against them, as highlighted by Gou et al. [7], Zhang et al. [52], Zhao et al. [54]. While several studies, including Phute et al. [31], Zhang et al. [52], have focused on developing precise detection mechanisms for jailbreak attacks, these approaches typically rely on an auxiliary proxy LLM to analyze outputs. However, such a setup can be impractical in real-world scenarios due to time and computation costs. To overcome this challenge, we have developed an efficient jailbreak detector that adds minimal overhead. Specifically, we train a gist token to extract summary information from previously generated tokens by injecting it at the sequence's end. We then use a classifier to determine whether the LLM has been jailbroken. Additionally, we construct a dataset to train our detector, which primarily consists of harmful questions, harmless questions with harmful answers, harmless answers, and refusal responses. For defense training, when jailbreak activities are detected, we leverage the identified jailbreak instructions and refusal responses to fine-tune the model using a low-rank adapter (LoRA) [10]. Furthermore, we decouple the jailbreak detector from the trainable LoRA module. Specifically, we use the intermediate hidden state for detection and train the LoRA module solely on the final layers of the model, ensuring that updates to the LoRA module do not affect detection performance. Moreover, to mitigate the risk of overfitting on rejecting jailbreak attempts, we mix normal data with jailbreak data for regularization. Simultaneously, we optimize the detector during testing to further enhance its performance. In the experimental section, we evaluate our approach against various jailbreak attacks on both LLMs and MLLMs. The results demonstrate that our framework effectively mitigates jailbreak attempts after detecting only a small number of such activities (e.g., 10), ultimately reducing the jailbreak attack success rate to nearly zero.

In summary, our contributions can be outlined as follows:

- We develop an adaptive jailbreak defense framework that detects jailbreak activities at test-time and enhances the model's defense capabilities against such attempts in an online manner.
- We design an efficient jailbreak detector that leverages a gist token and a binary classifier to accurately identify harmful responses with almost no additional cost.
- To improve the stability of the detector during testing, we propose a decoupling strategy by assigning different parameters for detector and defense training.
- Extensive experiments on both LLMs and MLLMs demonstrate that our framework effectively defends against various jailbreak attacks.

2 Related Works

2.1 Jailbreak Attacks

Research has consistently shown that safety-aligned LLMs and MLLMs remain vulnerable to jailbreak attacks [1, 14], with exploitation techniques evolving from simple adversarial tactics to more sophisticated methods. For example, GCG [59] appends an adversarial suffix to jailbreak prompts. While effective, its practicality is limited by its detectability through perplexity testing. In contrast, AutoDAN [24] employs a hierarchical genetic algorithm to generate readable jailbreak prefixes that evade such detection. Additionally, ICA [45] advances in-context jailbreaking by embedding harmful demonstrations directly into the context, effectively manipulating LLMs. Building on this, Zheng et al. [56] refines the approach by injecting system tokens and employing a greedy search strategy within the demonstrations to enhance effectiveness. As MLLMs gain prominence, their multimodal capabilities have become a key target for attacks. Qi et al. [33] highlights the vision modality as particularly vulnerable to adversarial attacks and proposes adversarial image training as a means to facilitate jailbreaking. Figstep [6] employs a blank-filling technique in image prompts to trigger harmful responses. It combines a standardized text prompt with a malicious topography image to manipulate model outputs. Similarly, Liu et al. [25] introduces MM-SafetyBench, which also employs topography to subtly incorporate malicious prompts within images. However, unlike Figstep, MM-SafetyBench uses stable diffusion [35] to create more complex backgrounds that contain the intention of jailbreak, thus enhancing the stealthiness and effectiveness of the attack.

2.2 Jailbreak Detection and Defense

To ensure the outputs of LLMs remain aligned with human values, substantial research has been devoted to both detecting and defending against jailbreak attacks. Jailbreak detection [12, 47] aims to differentiate jailbreak activities from normal activities. Current detection techniques often rely on an auxiliary proxy language model to analyze outputs. For instance, Phute et al. [31] generates detection prompts by appending the model's response to the question "is the response harmful?" and then uses a proxy LLM to assess potential harm. Similarly, Pi et al. [32] fine-tunes a small proxy model, utilizing the hidden state of its last token with a binary classifier to determine the nature of a response. LVLM-LP [54] addresses jailbreak detection by adopting a classifier beyond the first generated token. Another approach by Zhang et al. [52] involves augmenting the input multiple times and using a similarity matrix between responses for detection. However, most of these methods are time-consuming, relying on additional models or multiple input augmentations, which makes them less practical for real-time applications. Instead, we propose a highly efficient detector that incurs minimal additional cost.

Another line of work against jailbreak attacks is jailbreak defense [8]. Self-reminder [46] is among the earliest works to introduce a defensive system designed to remind the model not to produce harmful content. Focusing on MLLMs, Adashield [44] optimizes a suffix text prompt designed to remind the model to scrutinize both malicious text and image inputs. Gou et al. [7] endeavors to translate image inputs into corresponding text prompts to defend against jailbreak attacks that embed malicious intent within images to circumvent safety alignments. In contrast, Zong et al. [58] focuses on improving model safety during training by creating a dataset of malicious images to supervise model fine-tuning, making it more resilient to structure-based attacks like MM-SafetyBench and Figstep. IMMUNE [5] is a concurrent work that employs a safety reward model to guide the decoding generation process more securely. Recently, Peng et al. [29] shows that only a few harmful examples can be used to mitigate jailbreak successfully. Different from them, our method first tries to conduct adaptive safety fine-tuning and optimize the model's parameters during inference.

2.3 Test-Time Learning

Test-time learning is an innovative approach where a model is learning during testing to improve performance and adapt to new conditions. Early test-time learning was often used to solve the problem of distribution shift and alleviate the performance degradation caused by the difference between test data and training data[18, 50], namely test-time adaptation (TTA). While most TTA works focus on the recognition performance, Sheng et al. [36] aims to enhance the safety of the model (i.e., resistance to backdoor attack). Moreover, Guan et al. [9] proposes test-time repairing to remove the backdoor during testing. In addition, a lot of works pay attention to defense against adversarial attacks during test time [4, 27]. A recent work [19] introduces test-time training to improve the model's adversarial robustness through adaptive thresholding and feature distribution alignment. Our work extends the concept of test-time training to the domain of LLM security and uses it to enhance the model's ability to resist various jailbreak attacks.

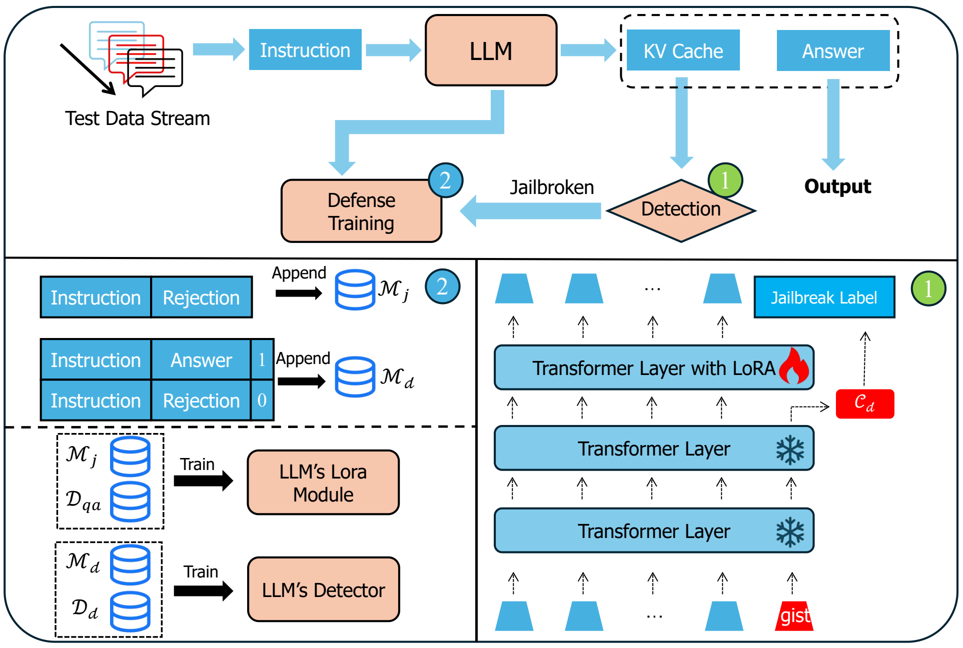


Figure 2: Detailed workflow of test-time immunization. **(1)** We insert a trainable gist token at the sequence's end and utilize the hidden states from intermediate layers along with a classifier $\mathcal{C}_d$ to perform detection. In a real-world application, we can employ the KV Cache and the gist token to perform efficient detection. **(2)** Upon detecting jailbreak activity during detection, we append the data to jailbreak memory and incorporate detection data into detection memory for training. Then we utilize jailbreak memory $\mathcal{M}_j$ to train the LLM's defense LoRA module by supervised fine-tuning and employ detection memory $\mathcal{M}_d$ to further train the detector (i.e., TTA) by Equation (4). Additionally, we employ a question-answering dataset $\mathcal{D}_{qa}$ and a detection dataset $\mathcal{D}_d$ for regularization.

3 Methodology

3.1 Preliminary

Given a large language model $M = \{\mathcal{E}_l, \mathcal{C}_l\}$ with a token set T and hidden space $\mathbb{R}^m$, and an input sequence $t = [t_1, ..., t_K | t_k \in T]$, where $\mathcal{E}_l$ is the encoder, $\mathcal{C}_l$ is the logit projector, and K represents the sequence length. The model generates the next token by:

$$t_{K+1} = M(t_{\leq K}) = \mathcal{C}_l(\mathcal{E}_l(t_{\leq K})), \tag{1}$$

where t_{K+1} is the next token and $h_K = \mathcal{E}_l(t_{\leq K}) \in \mathbb{R}^m$ is the hidden state of the last token.

Indeed, LLMs generate tokens autoregressively, using the previous output token to predict the subsequent token. This generation process continues until a stop condition is met, which may involve reaching a maximum token limit or generating a specific end-of-sequence token. Additionally, in modern LLMs, the Key-Value Cache (KV Cache) [34] technique is extensively utilized during inference to speed up attention map computations.

3.2 Jailbreak Detection

Most previous jailbreak detection methods either require proxy LLMs to analyze the model's output or involve multiple augmentations to the model's input, which are time-consuming and impractical for real-world applications. Therefore, we propose training an efficient jailbreak detector that leverages the autoregressive generation properties of the model. Specifically, as shown in the part 1 in Figure 2, we train a gist token t_g and a binary classifier $\mathcal{C}_d$, and obtain the predicted probability distribution p_t of the text t as follows:

$$p_t = \mathcal{C}_d(h_t) = \mathcal{C}_d(\mathcal{E}_l(t, t_g)), \tag{2}$$

where h_t represents the hidden state of the last token t_g. And then we obtain the detection results with p_t as follows:

$$\arg\max_c p_{t,c} = \begin{cases} 0, & \text{not jailbroken,} \\ 1, & \text{jailbroken.} \end{cases} \tag{3}$$

We inject the t_g token at the end of the sequence. Since the keys and values of the previous tokens are cached during generation, the hidden state of t_g can be computed efficiently based on the KV Cache. For instance, for a sequence with a length of 2000, the cost of detecting jailbreak activities is approximately 1/1000 of the total generation time. A simpler alternative would be to remove the gist token and directly use the hidden state of the last token to perform detection. However, intuitively, the hidden state of the last token is used for generation and may not encapsulate the information relevant to the harmfulness of the response. Therefore, we train a gist token designed to capture the harmfulness of the previous answer. Additionally, we construct a dataset $\mathcal{D}_d = (q_i, a_i, y_i)_{i=1}^{|D_d|}$ to train our detector, where q_i represents the question, a_i represents the answer, and y_i is the label indicating jailbreak activities. We train the detector using naive cross-entropy loss, as follows:

$$t_g^*, \mathcal{C}_d^* = \underset{t_g, \mathcal{C}_d}{\arg\min}\, \mathbb{E}_{(q_i,a_i,y_i)\sim\mathcal{D}_d}\left[-\sum_{c=0}^{1} y_{i,c}\log \hat{p}_{i,c}\right], \tag{4}$$

where $\hat{p}_i = \mathcal{C}_d(\mathcal{E}_l(q_i, a_i, t_g))$ represents the predicted jailbreak probability of jailbreak detector.

3.3 Adaptive Defense Training

Since detecting jailbreak activity is easier than directly defending against it, we build a test-time jailbreak defense system that transfers detection capability to defense capability that resembles the biological immune system. When pathogens first enter the system, the body recognizes this invasion. In our approach, we treat jailbreak activities as pathogens and use the above detector to distinguish them from normal activities. Once pathogens are identified, the organism will initiate an immune response and produce antibodies to neutralize the damage caused by antigens. Following an immune response, the organism becomes immune to the specific antigen. Similarly, when jailbreak activities are detected, our framework adds the detected jailbreak instructions along with a refusal response into jailbreak memory $\mathcal{M}_j$. We then use $\mathcal{M}_j$ to supervise fine-tuning the model. In this way, we progressively collect jailbreak data during the model testing process and enhance the defense capabilities of the model against various jailbreak attacks. For normal instruction, our model does not alter its behavior but only incurs a slight time cost for detecting jailbreak activities. Additionally, to prevent the model from becoming overly defensive against normal activities, we use the traditional question-answering (QA) dataset $\mathcal{D}_{qa}$, to regularize the model during training.

Furthermore, we adopt the concept of **test-time adaptation (TTA)** [41] to train our jailbreak detector with Equation (4) while detecting jailbreak behaviors. Specifically, we use detected jailbreak instructions along with their corresponding answers as jailbreak QA pairs, and jailbreak instructions with refusal responses as normal QA pairs. We then append them to the detection memory, denoted as $\mathcal{M}_d$, and use $\mathcal{M}_d$ to train our detector by Equation (4). Additionally, we also use the detection dataset, denoted as $\mathcal{D}_d$, for regularization training.

3.4 Overall Framework

Directly combining the above detection and defense training strategy comes with a drawback: the detector and defense training share a set of parameters (i.e., parameters in $\mathcal{E}_l$). The updates to model parameters by defense training are likely to impair the detector. To address this issue, we propose decoupling the detector and defense training. For detection, we utilize the hidden state of the intermediate layer, rather than the last layer, to perform detection. For defense training, we apply the LoRA module [10] to the layers behind the intermediate detection layer, treating them as trainable parameters, as shown in part 1 of Figure 2. We ensure that parameter updates to the detector and the defense training do not interfere with each other in this way. After that, we obtain the overall pipeline of TIM. The details of our method can be found in Algorithm 1 for reference.

4 Experiments

4.1 Setup

$\triangleright$ **Jailbreak Attack/Defense Methods**. We evaluate our defense methods against various jailbreak attack methods. For experiments on MLLMs, we choose Figstep [6] and MM-SafetyBench [25]. For experiments on LLMs, we utilize I-FSJ and GCG (in the Appendix) as the jailbreak attack method. For jailbreak defense methods, we consider FSD [6], Adashield [44], and VLGuard [58]. Additionally, we introduce another baseline, TIM (w/o gist), which is identical to our method but uses the final hidden state of the last token for detection. To assess the impact of our defense training on detection, we report results for TIM (w/o adapt.), where no defense training and optimization occur during testing. Linear Probing (LP) represents a method that neither uses the gist token nor adapts during testing (i.e., LLMs with a linear probing binary detector on the last generated token). Furthermore, we compare our detector against detection baselines, including Self Defense [31] and LVLM-LP [54], in LLM experiments.

▷ **Metrics.** We evaluate jailbreak methods from two perspectives: the effectiveness of defense against jailbreak attacks and the model's ability to respond to normal instructions. For evaluating the effectiveness of defense against jailbreak attacks, we adopt the Attack Success Rate (ASR) as a metric, as is common in most studies [1, 44]. We define ASR as the proportion of jailbreak instructions that are not rejected, relative to all the jailbreak instructions. For the response set R_j of the jailbreak dataset $\mathcal{D}_j$, ASR is calculated as follows:

$$ASR = \frac{|R_j| - \sum_{r \in R_j} isReject(r)}{|R_j|}, \text{where } isReject(r) = \begin{cases} 0, r \text{ is rejection,} \\ 1, r \text{ is not rejection.} \end{cases} \tag{5}$$

We employ prefix matching to determine whether a response is rejected. Specifically, we compile a set of rejection prefixes. If the model's response matches any prefix in the rejection set, we consider the instruction rejected. The rejection prefixes employed are listed in Appendix A.4. Since our method aims to incrementally enhance the model's security capabilities, we also report another metric, ASR-50, which calculates ASR for jailbreak samples in the last 50% of the test sequences. This reflects the model's performance after it has learned to defend against jailbreak attacks. Although defense methods improve the model's ability to reject malicious instructions, they may also cause the model to reject an excessive number of normal queries. Thus, we use the Over-Defense Rate (ODR) to assess the model's ability to respond to clean instructions. For the response set R_n of the normal dataset $\mathcal{D}_n$, ODR is calculated as follows:

$$ODR = \frac{\sum_{r \in R_n} isReject(r)}{|R_n|}. \tag{6}$$

Additionally, to evaluate the detector's performance, we report the Accuracy (ACC), True Positive Rate (TPR), and False Positive Rate (FPR) [38]. Moreover, we provide the details of our detection dataset, experiment setup and the introduction of our baselines in the Appendix A.

4.2 Main Results

Table 1: The experimental results under Figstep [6]. TIM's ASR is reported in the format of ASR/ASR-50 (same in the subsequent manuscript).

Methods	LLaVA-v1.6-Vicuna-7B		LLaVA-v1.6-Mistral-7B		LLaVA-v1.6-Vicuna-13B	
	ASR (↓)	ODR (↓)	ASR (↓)	ODR (↓)	ASR (↓)	ODR (↓)
Vanilla	100.0	0.0	100.0	0.0	100.0	0.0
FSD [6]	100.0	0.0	100.0	0.0	100.0	0.0
Adashield [44]	0.0	14.0	0.0	7.2	0.0	51.2
VLGuard [58]	0.0	7.0	0.0	1.8	0.0	5.2
TIM (w/o gist)	1.6	0.0	0.4	0.4	0.8	1.6
TIM	1.4/0.0	0.0	0.6/0.0	0.0	1.8/0.0	0.4

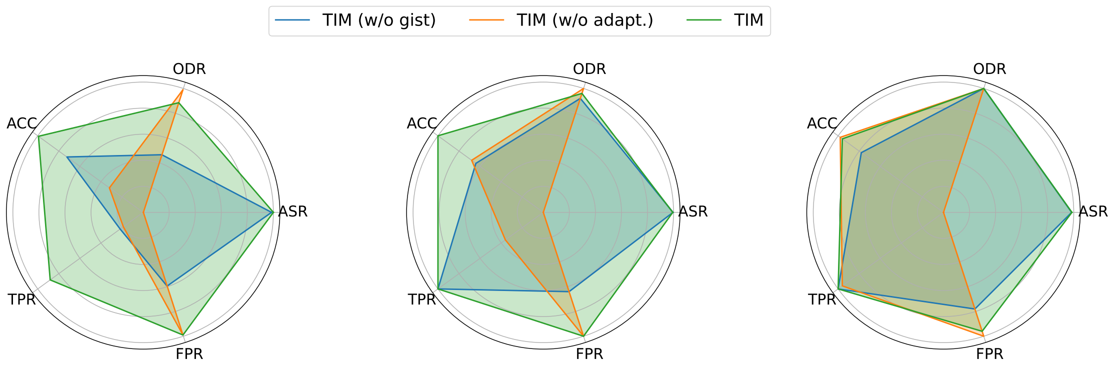
Table 2: The experimental results under the MM-SafetyBench [25].

Methods	LLaVA-v1.6-Vicuna-7B		LLaVA-v1.6-Vicuna-13B	
	ASR (↓)	ODR (↓)	ASR (↓)	ODR (↓)
Vanilla	99.8	0.2	100.0	0.4
FSD [6]	99.8	0.2	99.7	0.0
Adashield [44]	7.0	14.0	43.8	51.5
VLGuard [58]	1.4	6.5	0.2	4.7
TIM (w/o gist)	1.4	10.7	3.0	3.8
TIM	1.0/0.0	2.3	4.8/0.0	0.4

▷ **Jailbreak Defense.** To evaluate the effectiveness of our method, we report the results on Figstep and MM-SafetyBench in Tables 1 and 2. As shown in the tables, Adashield demonstrates strong defensive capabilities, especially against Figstep, where it reduces the ASR to 0%. Similarly, the ASR on MM-SafetyBench is reduced to 7% by Adashield. Despite its effectiveness, Adashield suffers from a noticeable over-defense phenomenon with normal samples, with over 5% of them being rejected. After training on a specially designed dataset, VLGuard shows relatively excellent performance, achieving almost 0% ASR against jailbreak samples but still show over-rejects to normal samples. Compared to VLGuard, our method can gradually learn to reject jailbreak attacks during testing without any prior targeted training. It achieves an ASR of less than 2% at most experiments, and, among all the effective jailbreak attack defense methods, our approach causes the least damage to the model's ability to respond to normal queries (i.e., ODR from 0.2% to 2.3% on MM-SafetyBench with LLaVA-v1.6-Vicuna-7B as backbone and nearly 0% on others). From the ASR, we can draw a conclusion that our method only requires a few jailbreak samples to learn how to reject such types of jailbreak attacks (on the Figstep dataset, this number is less than 10). Since our method progressively enhances the model's defensive capabilities during testing, we believe that the ASR-50 metric better reflects the true effectiveness of our approach. Our method achieved 0% ASR-50 across all jailbreak attack datasets, indicating that, with continuous optimization, our model can achieve complete defense against individual attacks. Moreover, Table 3 shows the results for the text-based attack. Our method is also effective at defending against I-FSJ, a jailbreak method that only uses the language modality. TIM not only achieves an ASR-50 of 0% but also reduces the model's ODR.

Table 3: The experimental results under text-based attack, I-FSJ [56].

Methods	LLaMA2-7B-chat			LLaMA3-8B-Instruct		
	ASR (↓)	ODR (↓)	TPR (↑)	ASR (↓)	ODR (↓)	TPR (↑)
Vanilla	99.2	5.5	-	94.3	0.2	-
Retokenization (20%)	97.5	8.3	-	83.0	0.2	-
SmoothLLM (insert 20%)	76.6	26.7	-	100.0	0.4	-
SmoothLLM (swap 20%)	93.4	55.8	-	60.0	1.8	-
SmoothLLM (patch 20%)	80.9	27.5	-	57.4	6.4	-
TIM (w/o adapt.)	-	-	98.9	-	-	18.2
TIM (w/o gist)	0.6	4.9	100.0	12.7	19.7	1.5
TIM	2.6/0.0	0.6	100.0	1.0/0.0	0.2	40.0



(a): LLaVA-Vicuna-7B under MM-SafetyBench. (b): LLaVA-Vicuna-7B under Figstep. (c): LLaVA-Mistral-7B under Figstep.

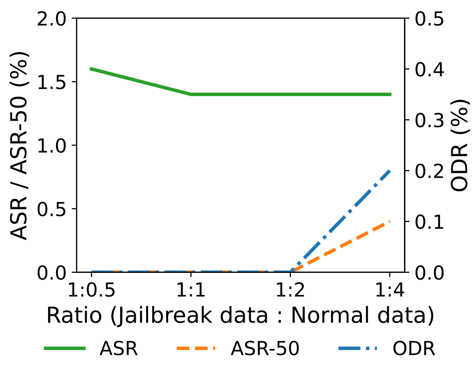
Figure 3: Performance of different variants of the proposed method. All metrics are normalized, and the methods with larger areas have better performance.

▷ **Jailbreak Detection.** Next, we analyze the role of our jailbreak detector from two perspectives: 1) What advantages does our detector's design offer compared to TIM (w/o gist)? 2) How does training the detector during testing enhance the effectiveness of our framework? First, addressing the initial question, the results in Table 4 show that TIM (w/o adapt.) exhibits clear improvements over LP in three metrics: Accuracy, TPR, and FPR. This improvement is primarily attributed to our introduction of the gist token, which is specifically designed to extract malicious information from previously generated sequences, rather than relying solely on the output of the last token for classification. This strategy has improved the expressive capacity of our detector.

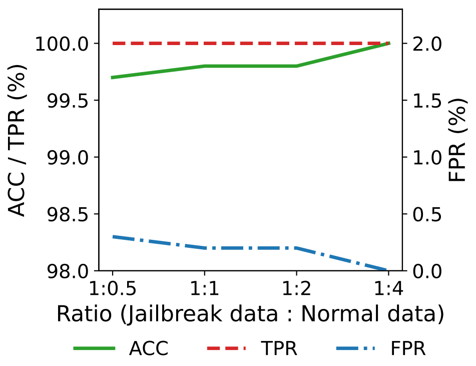
Table 4: The detection performance under I-FSJ attack.

Methods	ACC (↑)	TPR (↑)	FPR (↓)
Self Defense [31]	64.4	42.9	14.2
LVLM-LP [54]	67.7	36.3	0.8
LP	88.5	77.4	0.7
TIM (w/o adapt.)	99.1	98.9	0.6
TIM (w/o gist)	99.4	100.0	0.6
TIM	99.9	100.0	0.1

Secondly, the performance of the detector is shown in Figure 3. It is evident that TIM (w/o gist) exhibits a significant increase in FPR compared to TIM, suggesting that it misclassifies more normal samples as jailbreak samples. One consequence of this issue is the use of more normal samples in defense training, which leads to an increase in the model's ODR, as shown in the results in Tables 2 and 3. The cause of this issue arises from the detector sharing parameters with the defense training. The parameters' update during defense training will affect the performance of the detector. TIM resolves this issue by decoupling the defense training from the jailbreak detector by separating parameters.



(a) The defense capabilities of our method.



(b) The detection performance of our method.

Figure 5: Experimental results under different jailbreak data ratios.

4.3 Additional Analysis

In real-world scenarios, the situations encountered by models can be both complex and diverse. Therefore, we conduct additional experiments to directly assess the robustness of our method in complex scenarios. *The results of transferability, continually changing jailbreak, and GCG attack are provided in the Appendix B.*

▷ **Sensitivity to the Detector.** The ability of our method to resist jailbreak attacks intuitively depends on the detector's effectiveness at identifying such attacks. As shown in Table 3, our detector exhibited a relatively lower TPR under certain extreme conditions. Specifically, TIM (w/o adapt.) detected only 18.2% of jailbreak activities; however, with test-time adaptation of the detector, TIM significantly improved detection performance, achieving a TPR of 40%. We hypothesize that this reduced detection efficacy occurs because I-FSJ requires eight context examples to successfully jailbreak LLaMA3-8B-Instruct, resulting in a substantial discrepancy between the token lengths encountered during detector training and those in testing scenarios. The average token lengths for instructions and answers during detector training are 13 and 271, respectively, whereas the average token length for jailbreak instructions using I-FSJ reaches 3061. Despite this limitation, our method effectively resists attacks on LLaMA3, demonstrating robustness even when the detector's performance degrades.

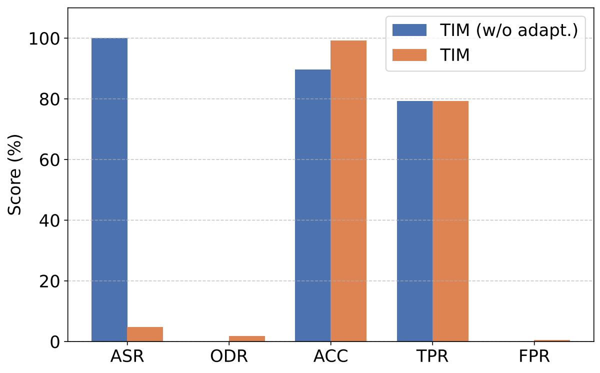


Figure 4: Results under hybrid jailbreak attack. We randomly selected 300 jailbreak samples from MM-SafetyBench and 300 from Figstep, combining them into a new jailbreak dataset.

▷ **Results under Hybrid Jailbreak Attack.** In deployment scenarios, attackers may employ multiple methods simultaneously to launch jailbreak attacks against the model. Accordingly, we designed experiments involving hybrid jailbreak attacks. The results, presented in Figure 4, indicate that under our method, the ASR can still be reduced to a very low level, while the model's ability to respond to normal queries remains largely unaffected.

▷ **Results under Different Jailbreak Data Ratios.** In practical applications, the proportion of jailbreak data within the model's test data is typically not fixed. The model may simultaneously receive a large number of jailbreak attack requests, or it might not encounter any jailbreak instructions for extended periods. Thus, we report the results of our method under varying proportions of jailbreak attack data in Figure 5. The results presented in the table demonstrate that our method achieves stable and effective performance across various proportions, both in terms of defending against jailbreak attacks and the detection performance of our detector.

Table 5: Average inference cost (seconds) for each instruction. All experiments are conducted with I-FSJ jailbreak. The test samples are mixed with 520 normal samples and 520 jailbreak samples.

Vanilla	Detection		Test-time Defense	
LLaMA2-7B	+ TIM's Detector	+ Self Defense	TIM	Training Inside
7.18	7.21 (+0.4%)	36.13	5.49	0.67 (12.2%)

▷ **Computation Cost Analysis**. The computational cost of our method is reported in Table 5. As shown, our detector introduces a negligible overhead—*only 0.4% of the standard inference cost*—making it substantially more efficient than Self Defense [31], which adopts a proxy LLM to analyze the generated output. In addition, the training cost constitutes merely 12.2% of the overall computational budget. Overall, the inference time of TIM is lower than that of the vanilla model. This is primarily because TIM generates short rejection responses to jailbreak attempts, rather than generating long malicious outputs.

5 Conclusion

In this paper, we address the challenge of defending against diverse jailbreak attacks. We propose a universal test-time defense framework designed to dynamically detect jailbreak attacks during testing and utilize detected jailbreak instructions to defensively train the model. To enhance jailbreak attack detection, we introduce a specialized gist token designed to extract harmful information from model responses with almost no additional cost, which is then classified using a binary classifier. Furthermore, to minimize the impact of model updates on the detector, we decouple the detector from defense training, ensuring they operate on separate parameters and do not interfere with each other. Extensive experiments demonstrate the efficacy of our method across a variety of scenarios.

References

[1] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. In *Workshop on Proc. NeurIPS*, 2024.

[2] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6, 2023.

[3] Badhan Chandra Das, M Hadi Amini, and Yanzhao Wu. Security and privacy challenges of large language models: A survey. *ACM Computing Surveys*, 2024.

[4] Zhijie Deng, Xiao Yang, Shizhen Xu, Hang Su, and Jun Zhu. Libre: A practical bayesian approach to adversarial detection. In *Proc. CVPR*, 2021.

[5] Soumya Suvra Ghosal, Souradip Chakraborty, Vaibhav Singh, Tianrui Guan, Mengdi Wang, Ahmad Beirami, Furong Huang, Alvaro Velasquez, Dinesh Manocha, and Amrit Singh Bedi. Immune: Improving safety against jailbreaks in multi-modal llms via inference-time alignment. *arXiv preprint arXiv:2411.18688*, 2024.

[6] Yichen Gong, Delong Ran, Jinyuan Liu, Conglei Wang, Tianshuo Cong, Anyu Wang, Sisi Duan, and Xiaoyun Wang. Figstep: Jailbreaking large vision-language models via typographic visual prompts. *arXiv preprint arXiv:2311.05608*, 2023.

[7] Yunhao Gou, Kai Chen, Zhili Liu, Lanqing Hong, Hang Xu, Zhenguo Li, Dit-Yan Yeung, James T Kwok, and Yu Zhang. Eyes closed, safety on: Protecting multimodal llms via image-to-text transformation. In *Proc. ECCV*, 2024.

[8] Yunhao Gou, Kai Chen, Zhili Liu, Lanqing Hong, Hang Xu, Zhenguo Li, Dit-Yan Yeung, James T Kwok, and Yu Zhang. Eyes closed, safety on: Protecting multimodal llms via image-to-text transformation. In *Proc. ECCV*, 2024.

[9] Jiyang Guan, Jian Liang, and Ran He. Backdoor defense via test-time detecting and repairing. In *Proc. CVPR*, 2024.

[10] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Proc. ICLR*, 2022.

[11] Zhengmian Hu, Gang Wu, Saayan Mitra, Ruiyi Zhang, Tong Sun, Heng Huang, and Viswanathan Swaminathan. Token-level adversarial prompt detection based on perplexity measures and contextual information. *arXiv preprint arXiv:2311.11509*, 2023.

[12] 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 preprint arXiv:2309.00614*, 2023.

[13] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[14] Haibo Jin, Leyang Hu, Xinuo Li, Peiyan Zhang, Chonghan Chen, Jun Zhuang, and Haohan Wang. Jailbreakzoo: Survey, landscapes, and horizons in jailbreaking large language and vision-language models. *arXiv preprint arXiv:2407.01599*, 2024.

[15] Timo Kaufmann, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. A survey of reinforcement learning from human feedback. *arXiv preprint arXiv:2312.14925*, 2023.

[16] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[17] Aounon Kumar, Chirag Agarwal, Suraj Srinivas, Aaron Jiaxun Li, Soheil Feizi, and Himabindu Lakkaraju. Certifying llm safety against adversarial prompting. *arXiv preprint arXiv:2309.02705*, 2023.

[18] Jian Liang, Ran He, and Tieniu Tan. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*, pages 1–34, 2024.

[19] Jinpeng Lin, Xulei Yang, Tianrui Li, and Xun Xu. Improving adversarial robustness for 3d point cloud recognition at test-time through purified self-training. *arXiv preprint arXiv:2409.14940*, 2024.

[20] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proc. ECCV*, 2014.

[21] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Proc. NeurIPS*, 2023.

[22] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proc. CVPR*, 2024.

[23] Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge, January 2024. URL `https://llava-vl.github.io/blog/2024-01-30-llava-next/`.

[24] Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. Autodan: Generating stealthy jailbreak prompts on aligned large language models. In *Proc. ICLR*, 2024.

[25] Xin Liu, Yichen Zhu, Jindong Gu, Yunshi Lan, Chao Yang, and Yu Qiao. Mm-safetybench: A benchmark for safety evaluation of multimodal large language models. In *Proc. ECCV*, 2024.

[26] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*, 2023.

[27] Gaurav Kumar Nayak, Ruchit Rawal, and Anirban Chakraborty. Dad: Data-free adversarial defense at test time. In *Proc. WACV*, pages 3562–3571, 2022.

[28] R OpenAI. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2(5), 2023.

[29] Alwin Peng, Julian Michael, Henry Sleight, Ethan Perez, and Mrinank Sharma. Rapid response: Mitigating llm jailbreaks with a few examples. *arXiv preprint arXiv:2411.07494*, 2024.

[30] Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023.

[31] Mansi Phute, Alec Helbling, Matthew Daniel Hull, ShengYun Peng, Sebastian Szyller, Cory Cornelius, and Duen Horng Chau. Llm self defense: By self examination, llms know they are being tricked. In *The Second Tiny Papers Track at ICLR*, 2024.

[32] Renjie Pi, Tianyang Han, Jianshu Zhang, Yueqi Xie, Rui Pan, Qing Lian, Hanze Dong, Jipeng Zhang, and Tong Zhang. Mllm-protector: Ensuring mllm's safety without hurting performance. *Proc. EMNLP*, 2024.

[33] Xiangyu Qi, Kaixuan Huang, Ashwinee Panda, Peter Henderson, Mengdi Wang, and Prateek Mittal. Visual adversarial examples jailbreak aligned large language models. In *Proc. AAAI*, 2024.

[34] Alec Radford. Improving language understanding by generative pre-training. 2018.

[35] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. CVPR*, 2022.

[36] Lijun Sheng, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. Can we trust the unlabeled target data? towards backdoor attack and defense on model adaptation. *arXiv preprint arXiv:2401.06030*, 2024.

[37] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. In *Proc. NeurIPS*, 2020.

[38] John A Swets. Measuring the accuracy of diagnostic systems. *Science*, 240(4857):1285–1293, 1988.

[39] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

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

[41] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *Proc. ICLR*, 2021.

[42] Yanbo Wang, Jiyang Guan, Jian Liang, and Ran He. Do we really need curated malicious data for safety alignment in multi-modal large language models? *arXiv preprint arXiv:2504.10000*, 2025.

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

[44] Yu Wang, Xiaogeng Liu, Yu Li, Muhao Chen, and Chaowei Xiao. Adashield: Safeguarding multimodal large language models from structure-based attack via adaptive shield prompting. In *Proc. ECCV*, 2024.

[45] Zeming Wei, Yifei Wang, Ang Li, Yichuan Mo, and Yisen Wang. Jailbreak and guard aligned language models with only few in-context demonstrations. *arXiv preprint arXiv:2310.06387*, 2023.

[46] 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*, 5(12):1486–1496, 2023.

[47] Yueqi Xie, Minghong Fang, Renjie Pi, and Neil Gong. Gradsafe: Detecting jailbreak prompts for llms via safety-critical gradient analysis. In *Proc. ACL*, 2024.

[48] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *Proc. ICLR*, 2024.

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

[50] Yongcan Yu, Lijun Sheng, Ran He, and Jian Liang. Stamp: Outlier-aware test-time adaptation with stable memory replay. In *Proc. ECCV*, 2024.

[51] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*, 2023.

[52] Xiaoyu Zhang, Cen Zhang, Tianlin Li, Yihao Huang, Xiaojun Jia, Ming Hu, Jie Zhang, Yang Liu, Shiqing Ma, and Chao Shen. Jailguard: A universal detection framework for llm prompt-based attacks. *arXiv preprint arXiv:2312.10766*, 2024.

[53] Zhexin Zhang, Junxiao Yang, Pei Ke, Fei Mi, Hongning Wang, and Minlie Huang. Defending large language models against jailbreaking attacks through goal prioritization. In *Proc. ACL*, 2024.

[54] Qinyu Zhao, Ming Xu, Kartik Gupta, Akshay Asthana, Liang Zheng, and Stephen Gould. The first to know: How token distributions reveal hidden knowledge in large vision-language models? In *Proc. ECCV*, 2024.

[55] 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 preprint arXiv:2303.18223*, 2023.

[56] Xiaosen Zheng, Tianyu Pang, Chao Du, Qian Liu, Jing Jiang, and Min Lin. Improved few-shot jailbreaking can circumvent aligned language models and their defenses. In *Proc. NeurIPS*, 2024.

[57] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. In *Proc. ICLR*, 2024.

[58] Yongshuo Zong, Ondrej Bohdal, Tingyang Yu, Yongxin Yang, and Timothy Hospedales. Safety fine-tuning at (almost) no cost: A baseline for vision large language models. In *Proc. ICML*, 2024.

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

A The Details of Experimental Setup

A.1 Dataset Construction

To construct the detection dataset, we initially collected original malicious instructions from AdvBench [59] and MM-SafetyBench [25]. To obtain malicious answers, we employed Wizard-Vicuna-7B-Uncensored [48], a model without safety alignment, to generate answers. To obtain refusal answers, we utilized LLaMA2-13B-chat to generate answers with various refusal prefixes. We employed GPT4-LLM-Cleaned [30] and LLaVA-Instruct-150K [21] as clean instructions for LLMs and MLLMs, respectively. Furthermore, to generate clean answers, we utilized LLaMA2-7B-chat and LLaVA-v1.6-Vicuna-7B for GPT4-LLM-Cleaned and LLaVA-Instruct-150K, respectively. Our detection dataset comprises four parts: 1) malicious instructions with malicious answers, classified as jailbroken; 2) malicious instructions with refusal answers, classified as not jailbroken; 3) clean instructions with clean answers, classified as not jailbroken; 4) clean instructions with malicious answers, classified as jailbroken. The primary focus of the dataset is to determine whether the answer is harmful, rather than assessing the harm of the instruction itself. For the visual question-answering (VQA) dataset, since the original malicious instructions lack images, we randomly selected images from the COCO dataset [20] for them. It is important to note that our malicious instructions are original and unaffected by jailbreak attacks, meaning we do not use jailbreak-processed instructions during detector training. For the evaluation dataset, we combine normal QA/VQA instructions from GPT4-LLM-Cleaned/LLaVA-Instruct-150K with jailbreak instructions to simulate real deployment environments in experiments on LLMs/MLLMs.

A.2 Baselines

Figstep [6] conceals harmful content within text prompts using typography, embedding it into blank images to circumvent text-modality safety alignments.

MM-SafetyBench [25] initially generates a malicious background image using harmful keywords from jailbreak prompts and subsequently converts text-based harmful content into images using topography.

I-FSJ [56], based on in-context jailbreak [45], aims to induce the model to generate harmful content through several jailbreak demonstrations. Additionally, I-FSJ employs system tokens to enhance its attack capabilities. Furthermore, a greedy search is used to select the optimal demonstration from the datasets.

GCG [59] is a white-box method utilizing an adversarial text suffix to jailbreak LLMs.

FSD [6] is a defense method that introduces a specific system prompt, reminding the model to focus on malicious text within images.

Adashield [44] is a test-time alignment method proposing the addition of a defense prompt following the input text prompt. The defense prompts can be static or adaptive, which are called Adashield-S or Adashield-A, respectively. We consider Adashield-S in our experiments.

VLGuard [58] is a training-time alignment method that involves additional safety fine-tuning on a specific dataset. It constructs a safety instruction tuning dataset containing malicious images to defend against structure-based jailbreak methods like Figstep and MM-SafetyBench. Unlike VLGuard, our detector's training dataset contains no prior knowledge of the jailbreak attack method (e.g., malicious images).

A.3 Experimental Details

For MLLM experiments, we select LLaVA-v1.6-Vicuna-7B/13B [2] and LLaVA-v1.6-Mistral-7B [13, 21, 22, 23] as the base models. For LLM experiments, we use LLaMA2-7B-chat and LLaMA3-8B-Instruct [40] as the base model. The weights for all base models are sourced from Hugging Face. We set the learning rate, number of epochs, and batch size for detector training to 1e-3, 5, and 32, respectively. We use the Adam optimizer [16] for defense training, setting the learning rates to 0.001 for MLLMs and 0.002 for LLMs. We apply LoRA [10] with a rank of 16 to the query and value matrix in the last 15 transformer blocks. The regularization batch size is set to 40, while the batch sizes for refusal training and detector training during test time are set to 1 and 6, respectively. Furthermore, during jailbreak activity detection, we train the defense capabilities and the detector for 1 and 5 steps, respectively. We incorporate an equal mix of jailbreak instructions and clean instructions in the test data. All the experiments are conducted on 4*RTX A6000.

A.4 The jailbreak Evaluation

For the refusal judgment, we use the prefixes match method. We think the model refused to answer if the respones start with the following prefixes.

I'm sorry
Sorry
I am sorry
I apologize
As an
As a responsible
I'm an
I'm just
As an Assistant
If you have any other non-malicious requests

I do not
I cannot
As a language model
I'm really sorry
My apologies
I'm not able to provide
I am really sorry
I can't provide
I can't assist

B Additional Results

Robustness against continual changing attack. To validate the effectiveness of our method under continuous exposure to various forms of jailbreak attacks, we conducted experiments as shown in Table 6. We selected 500 different samples for each type of jailbreak attack and conducted the attacks in varying orders. As can be seen, even after undergoing the MM-SafetyBench attack, our method still maintains good defensive performance during the second exposure to the Figstep attack, without experiencing catastrophic forgetting.

Table 6: ASR(%) under continual changing environments.

Attack Order (⟶)		
Figstep	MM-SafetyBench	Figstep
1.4	6.6	0.0

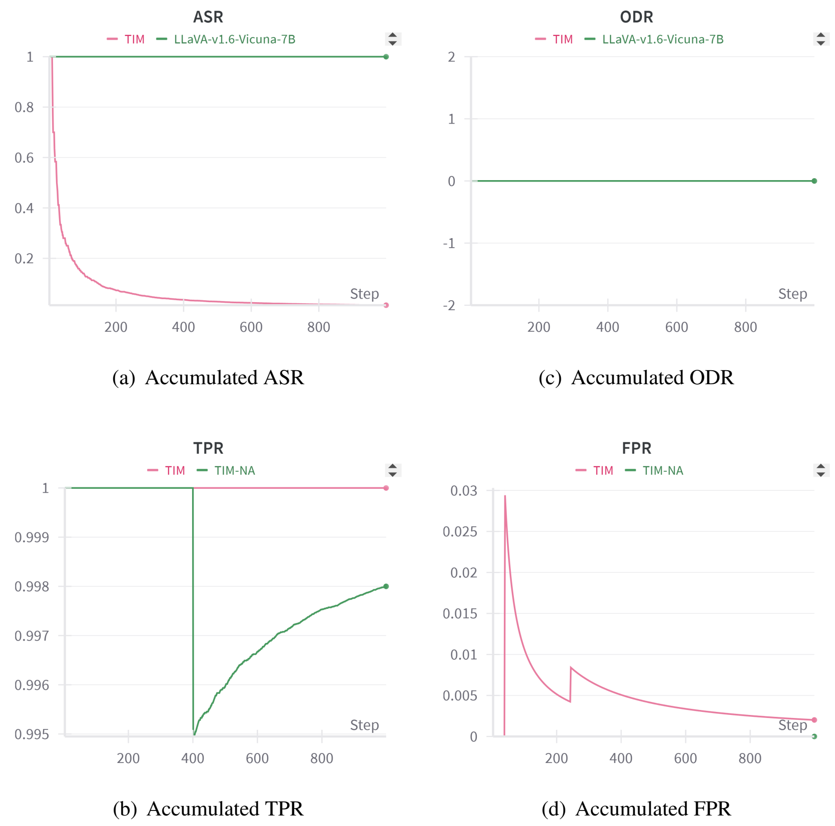
Table 7: The transferability results. We first adopt TIM on the source jailbreak attack. Then, we freeze the fine-tuned model and evaluate it on the target attack. We report the ASR while adopting the LLaVA-v1.6-Vicuna-7B as the backbone. The numbers in brackets represent the changes of ASR compared to the Vanilla Model.

Figstep ⟶ MM-SafetyBench	MM-SafetyBench ⟶ Figstep
84.3 (-15.5)	0.0 (-100.0)

Transferability of defense training. We demonstrate the static transferability of the fine-tuned model in Table 7. It is effective when migrating from a more complex attack (MM) to a simpler one (Figstep), but its effectiveness is limited in the reverse direction. However, it's worth noting that our method is an online adaptive defense method. New types of jailbreaks will be adaptively defended against as they emerge.

Table 8: Experimental Results under GCG jailbreak attacks.

	ASR	ODR
LLaMA2-7B-chat	21.5	0.2
+TIM	7.7 (-13.8%)	2.7 (+2.5%)



(a) Accumulated ASR (c) Accumulated ODR

(b) Accumulated TPR (d) Accumulated FPR

Figure 6: Changes in metrics during the test process against Figstep. TIM-NA represents TIM (w/o adapt.)

Results under GCG attack. We supplemented the results of the white-box attack, GCG, in Table 8. TIM decreased the ASR from 21.5% to 7.7%, demonstrating its effectiveness against GCG.

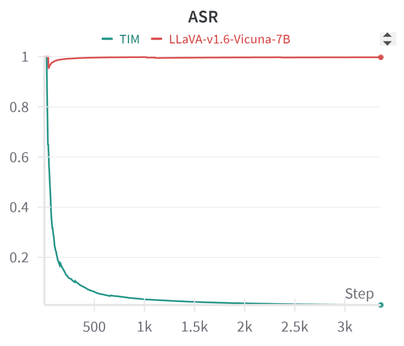
Performance curve during testing. To demonstrate the performance of our method as the test progresses, we report the relevant indicators in the Figures 6 and 7. As can be seen, as the test progresses, the ASR of our method continues to decrease, indicating that our model has learned how to resist this type of jailbreak attack, and our method only needs a small number of samples to fully learn how to defend. In addition, our other indicators remain stable during the test, which shows the robustness of our method.

C Algorithm of TIM

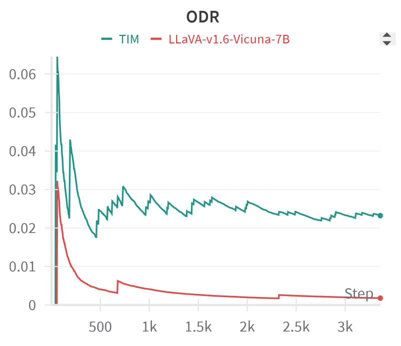
Algorithm 1 The Pipeline of TIM

```
Initailize: LLM $\mathcal{E}_l, \mathcal{C}_d$, Gist token $t_g$ and Detection Classifier $\mathcal{C}_d$, Jailbreak Memory $\mathcal{M}_j$, Detection Memory $\mathcal{M}_d$, Instruction Dataset $\mathcal{D}_{qa}$, Detection Dataset $\mathcal{D}_d$, Refusal Answer $t_{ref}$.
Input: An instruction $t_{ins}$.
Generate the answer $t_{ans}$ of $t_{ins}$ by Equ. equation 1
Obtain the jailbreak label by Equ. equation 2 and equation 3.
if jailbreak label equals to 1 then
    Append $(t_{ins}, t_{ref})$ into $\mathcal{M}_j$.
    Append $\{(t_{ins}, t_{ref}, 0), (t_{ins}, t_{ans}, 1)\}$ into $\mathcal{M}_d$.
    Train the Adapter of $\mathcal{E}_l$ with $\mathcal{M}_j$ and $\mathcal{D}_{qa}$.
    Train $t_g$ and $\mathcal{C}_d$ with $\mathcal{M}_d$ and $\mathcal{D}_d$
end if
Output: Answer $t_{ans}$
```

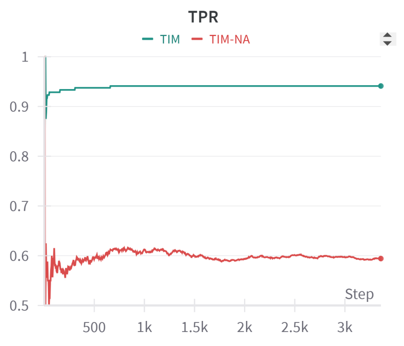
We summarize the pipeline of TIM in Algorithm 1.



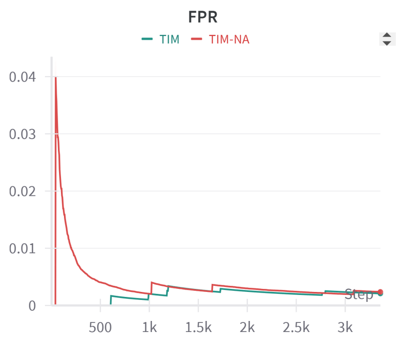
(a) Accumulated ASR



(c) Accumulated ODR



(b) Accumulated TPR



(d) Accumulated FPR

Figure 7: Changes in metrics during the testing against MM-SafetyBench. TIM-NA represents TIM (w/o adapt.)

D Broader Impacts

While this work does not directly target societal or community-level outcomes, it contributes to the broader scientific enterprise by advancing foundational understanding in jailbreak studies. The methods and findings presented may support future theoretical developments and inspire new directions in related research areas. Furthermore, the technical tools and insights generated can serve as a resource for researchers pursuing similar challenges, fostering further academic collaboration and exploration.

E Future Works and Limitations

In practical applications, our method can be able to be combined with other static jailbreak attack defense methods to jointly improve the defense capabilities against jailbreak attacks. However, in this article, we did not verify the compatibility of TIM with other jailbreak attack defense methods. We plan to study this issue in subsequent work. In addition, due to the limitation of computing resources, we did not verify whether our method can be generalized to such models on a larger model (70 B+). In addition, our detector may have a decay when the detection token is extremely long. We consider adding data of different lengths to the detection dataset in future work to compensate for this limitation.