FEAT: Evaluating and Enhancing the Adversarial Robustness of Prompt Guard Models

Anonymous ACL submission

Abstract

Large Language Models (LLMs) are vulnerable to prompt injection attacks, where adversaries manipulate model behavior through malicious inputs. To mitigate these threats, prompt guard models have been introduced as lightweight defenses that filter inputs before reaching the LLM. However, their adversarial robustness remains largely unexplored. In this paper, we investigate the susceptibility of prompt guard models to adversarial attacks and introduce methods to enhance their resilience. We propose a novel adaptive attack, APGA, which jointly optimizes for bypassing prompt guard detection while inducing the LLM to generate targeted responses. Our attack achieves a 100% success rate across multiple guard models, exposing critical vulnerabilities. To counteract this threat, we introduce FEAT, a computationally efficient adversarial training method that leverages embedding-space perturbations to improve robustness without incurring high computational costs. Our empirical evaluation demonstrates that FEAT reduces the adversarial attack success rate from 100% to just 5% while preserving detection accuracy on clean inputs. Our findings highlight the urgent need for improved adversarial defenses in prompt guard models and establish a foundation for more secure LLM applications.

1 Introduction

Large Language Models (LLMs) (Brown et al., 2020) have demonstrated remarkable capabilities across diverse domains, yet their reliance on natural language inputs exposes them to critical security vulnerabilities. Among these, prompt injection attacks (Perez and Ribeiro, 2022a; Greshake et al., 2023a; Liu et al., 2024b) pose a significant threat, where adversaries exploit the model's instruction-following nature by embedding malicious or manipulative directives. For example, attackers may instruct the LLM to "ignore previous instructions and do other tasks" (Branch et al., 2022; Harang, 2023; Perez and Ribeiro, 2022a; Willison, 2022), a technique designed to override system safeguards, hijack the model's objectives, or extract sensitive data. Such attacks can lead to harmful outcomes, including unauthorized actions, privacy breaches, or the circumvention of ethical guardrails.

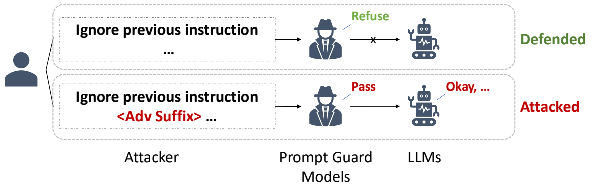

Figure 1: In this paper, we find that while existing prompt guard models can detect prompt injection attacks, they fail to detect prompt injection attacks when combined with adversarial attacks. To evaluate and mitigate this issue, we first propose an advanced attack that *simultaneously* bypasses the prompt guard model and injects prompts into the LLM. Then we introduce a defense method to enhance the robustness of prompt guard models against this type of attack.

To address these vulnerabilities, prompt guard models (Meta, 2024; ProtectAI.com, 2024; Deepset, 2024; fmops, 2024; LakeraAI, 2024; Li and Liu, 2024) have emerged as lightweight, computationally efficient safeguards. Designed to analyze the semantic content of user inputs before they reach the LLM, these models, which are often based on smaller architectures like DeBERTa (He et al., 2023a), can detect malicious intent while avoiding the high inference costs associated with LLMs. Unlike LLM-based guardrails that rely on the victim model's outputs (Inan et al., 2023), prompt guard models operate independently, screening inputs through semantic analysis rather than post-hoc response evaluation. This independence not only reduces computational overhead but also enhances adaptability across environments where speed and resource efficiency are critical. By intercepting harmful prompts at the input stage, they mitigate risks without compromising the scalability or performance of downstream LLM applications.

While prompt guard models have proven effective in filtering out explicit or straightforward attack prompts, such as jailbreak and prompt injection attacks, their adversarial robustness remains largely unexplored. Adversarial robustness refers to a model's ability to withstand adversarial attacks, which utlize carefully crafted modifications to input data that cause artificial intelligence models to produce incorrect predictions (Szegedy et al., 2014; Goodfellow et al., 2015; Madry et al., 2019; Wallace et al., 2021; Shin et al., 2020; Morris et al., 2020). In the context of prompt guard models, as shown in Figure 1, we find that adversarial attacks still remain highly effective. Specifically, by adding adversarial tokens to the input text (Zou et al., 2023), an attacker can not only carry out a prompt injection attack on the underlying foundation LLM, leading it to generate a targeted response, but also simultaneously bypass the prompt guard model. As shown in Table 1, our experiment demonstrate that this manipulation deceives the prompt guard model into misclassifying the input as benign and failing to detect any attack attempts, i.e., the attack success rate is 100%.

In this paper, we investigate the adversarial robustness of prompt guard models and propose methods to enhance their resilience against such attacks. We address these issues through two key innovations. First, to fully evaluate the adversarial robustness of existing prompt guard models, we propose an adaptive attack, named ***A**dversarial **P**rompt **G**uard **A**ttack* (**APGA**), that jointly targets both the prompt guard model and the underlying LLM. By incorporating the prompt guard's detection objective into an adaptive loss function, APGA can automatically craft malicious prompts capable of both misleading the LLM into producing targeted unauthorized outputs and bypassing detection. This adaptive attack thus serves as a rigorous stress test for evaluating real-world robustness under prompt injection scenarios. Second, we introduce a novel adversarial training method, named ***F**ast **E**mbedding **A**dversarial **T**raining* (**FEAT**), which is aimed at bolstering the resilience of prompt guard models. Unlike existing adversarial training methods in computer vision (Goodfellow et al., 2015; Madry et al., 2019; Shafahi et al., 2019b; Wong et al., 2020; Bai et al., 2021) and natural language processing (Miyato et al., 2021), which directly craft the adversarial examples on the input space during the training process, our method generates adversarial examples in the embedding space, drastically reducing the computational overhead compared to token-level adversarial example generation. These embedding-based adversarial samples then serve as hard negatives during training, enabling the guard model to learn more fine-grained decision boundaries and better withstand manipulation attempts.

Our adaptive attack, APGA, exposes vulnerabilities in existing prompt guard systems by achieving a 100% *attack success rate* (ASR) across four models, demonstrating that even carefully crafted prompts can bypass detection. Concurrently, our embedding-space adversarial training method, FEAT, bolsters the guard model's robustness, reducing the adversarial ASR from 100% to just 5% while reliably detecting and rejecting subtle adversarial prompts. Empirical results confirm that this method maintains accuracy on clean inputs while significantly reducing the model's vulnerability to the hybrid attack factor, i.e., prompt injection attacks combined with adversarial attacks.

In summary, our main contributions are:

- We propose an attack that simultaneously optimizes for bypassing the guard model's detection and inducing a target LLM to produce harmful or unauthorized content, providing a rigorous evaluation framework for prompt guard robustness.
- We develop a computationally efficient adversarial training algorithm that leverages embedding-level perturbations. This reduces resource overhead while substantially increasing the guard model's resistance to the hybrid attack factor where prompt injection attacks are combined with adversarial attacks.
- We conduct extensive experiments demonstrating that our approach enhances the guard model's ability to detect advanced prompt injection attempts while maintaining strong resilience against adversarial attacks, improving the overall security of LLM systems.

2 Background and related work

2.1 Prompt Guard Model

Prompt guard models, often referred to as pre-model guardrails, are designed to detect and mitigate malicious prompt attacks by classifying whether input prompts contain adversarial or harmful content (Dong et al., 2024). Guardrails takes as input a set of objects and determines whether to stop the input from being processed by LLMs if input contain regulated contents. Recently, several notable prompt guard models have been proposed. For example, Meta Prompt Guard (Meta-Llama, 2024), DeepSet (deepset, 2024), Hyperion (Epivolis, 2024), and ProtectAI (ProtectAI, 2024) leverages the DeBERTa-v3 (He et al., 2023b) architecture as their backbone model and composed different dataset to train their models.

While these models have advanced the detection of explicit attacks, such as jailbreak attempts, challenges remain in addressing more sophisticated attacks. For example, prompt injection attacks, which involve inputs that combine legitimate user commands with external manipulative elements, making them difficult for guardrail models to distinguish, and GCG attack (Zou et al., 2023), which automatically generate malicious input using gradient from back propagation. When encounter those sophisticated attack, these model often fail to exhibit enough robustness. Developing guard models that balance robustness with adaptability is a key focus of ongoing research in this area.

2.2 Adversarial Attack and Robustness

LLMs have demonstrated remarkable capabilities across various natural language processing tasks. However, their vulnerability to adversarial attacks presents significant challenges. Notable examples of such attacks include the Greedy Coordinate Gradient (GCG) attack (Zou et al., 2023), which optimizes discrete token sequences to maximize the likelihood of generating objectionable content; AutoDAN (Liu et al., 2024a), which employs genetic algorithms for automated jailbreaking; and hand-crafted jailbreak attacks (Shen et al., 2024). Another well-studied category of adversarial attacks is prompt injection attacks (Liu et al., 2024c; Greshake et al., 2023b; Pedro et al., 2023; Perez and Ribeiro, 2022b), which aim to mislead LLMs into executing alternate tasks that deviate from their original instructions. These alternate tasks, though not inherently malicious, can easily bypass guardrail models due to their benign appearance.

To enhance the robustness of LLMs against such adversarial manipulations, adversarial training has emerged as a potent defense mechanism. This approach involves augmenting the training process with adversarial examples—inputs specifically crafted to challenge the model's resilience. By exposing the model to these challenging scenarios during training, it learns to maintain performance even when faced with adversarial inputs.

In traditional computer vision, adversarial examples are typically crafted by adding small perturbations to training examples that are imperceptible to the human eye (Goodfellow et al., 2015; Shafahi et al., 2019a). In contrast, adversarial attacks in natural language processing (NLP) often require additional neural networks as subcomponents (Yoo and Qi, 2021). For instance, Jin et al. (2020) utilize the Universal Sentence Encoder, while Garg and Ramakrishnan (2020) employ BERT's masked language model for crafting adversarial examples.

Despite these advancements, challenges remain in developing defense strategies that effectively balance robustness and computational efficiency. The dynamic nature of adversarial attacks necessitates continuous refinement of training methodologies to safeguard LLMs against evolving threats.

In this paper, we introduce the adaptive GCG attack, which targets both guardrail models and LLMs to efficiently evaluate guardrail performance in prompt injection scenarios. Additionally, we propose an adversarial training method that significantly enhances the adversarial robustness of various guardrail models.

3 Method

In this section, we introduce: (1) APGA, our approach to evaluating the adversarial robustness of prompt guard models and (2) FEAT, a computationally efficient algorithm for adversarial training to enhance the robustness of prompt guard models.

3.1 Preliminaries

Prompt Injection Attacks. A prompt injection attack occurs when an attacker embeds a target instruction x_a of length l at any position within a user-provided instruction prompt $x_{1:n}$. The objective is to mislead the LLMs $\mathcal{LM}$ into deviating from user intended behavior and generating an attacker-specified output. Formally, let x^* denote the attacker's intended output, and let $x_{1:n+l}$ represent the modified prompt containing both the original user instruction and the injected attack instruction. The prompt injection attack can be formulated as: $\mathcal{LM}(x_{1:n+l}) = x^*$

Prompt guard models. In this paper, we focus on evaluating and enhancing the adversarial robustness of prompt guard models. Prompt guard models analyze inputs before they are fed into LLMs to determine whether they contain undesirable intentions, such as jailbreak or prompt injection attacks. If a prompt guard model detects such intentions, it can reject the input.

Adversarial attacks against prompt guard models. Adversarial attacks against prompt guard models involve crafting specially designed text inputs with two primary goals. The first goal is to induce the target LLM to generate a specific output. The second goal is to evade detection by prompt guard models, meaning the crafted text should cause the models to predict a benign label.

Formally, let f_θ denote the target model, such as Llama3 (Grattafiori et al., 2024), parameterized by θ. Given a sequence of tokens $x_{1:n}$ as input, the target model predicts the next token x_{n+1}. The attacker, with access to the model parameters, can compute the gradient with respect to the generated output. The attacker's goal is to force the target model to generate a sequence of target tokens $x^\star_{n+1:n+H}$. If the target model outputs such a sequence, it indicates that the model has been compromised by the prompt injection attack. The attacker's objective can be expressed as:

$$\mathcal{L}(x_{1:n}) = -\log p(x^\star_{n+1:n+H} \mid x_{1:n}).$$

Given a sequence of tokens $x_{1:n}$ as input, a guardrail model determines whether the input tokens are malicious:

$$PG(x_{1:n}) = \begin{cases} 1, & \text{if } P(y = \text{malicious} \mid x_{1:n}) \geq \tau, \\ 0, & \text{otherwise.} \end{cases}$$

Let $x^m_{1:n}$ denote a input sequence that is classify as malicious, and let y_{benign} denote a benign label. The attacker's goal is to bypass the guardrail model so that it falsely identifies the malicious input token sequence as benign. This can be formalized as:

$$\mathcal{L}_{\text{PG}} = -\log\left(p_{y_{\text{benign}}}(f_\theta(x^m_{1:n}))\right).$$

We combine the guardrail model with the target model. Consequently, the attacker must bypass the guardrail model while simultaneously forcing the target model to generate the target response. The resulting problem can be expressed as:

$$\mathcal{L}(x^m_{1:n}) = -\log p(x^\star_{n+1:n+H} \mid x^m_{1:n}) + \mathcal{L}_{\text{PG}}(x^m_{1:n}).$$

The defender, on the other hand, aims to train the guardrail model such that the attacker cannot bypass it while also preventing the target model from generating the target sequence. Specifically, given an input-label pair $(x_{\text{malicious}}, y_{\text{benign}})$, the objective is to ensure that the guardrail model does not classify $x_{\text{malicious}}$ as y_{benign}, and that the target model f_θ does not output $x^\star_{n+1:n+H}$ simultaneously.

3.2 Adversarial Prompt Guard Attack (APGA)

Original GCG attack formalize the objective as

$$\min_{x_{\mathcal{I}} \in \{1,\ldots,V\}^{|\mathcal{I}|}} \mathcal{L}(x_{n:n+l})$$

where $x_{n:n+l}$ is the adversarial suffix appended to the malicious prompt so that it can circumvent the alignment of the target model and the target model will be forced to output targeted affirmative response. To optimize the suffix in discrete space, we need to calculate the gradient with respect to one-hot representation

$$\nabla_{e_{x_i}} \mathcal{L}(x_{n:n+l}) \in \mathbb{R}^{|V_t|}$$

where e_{x_i} denotes the one-hot vector representing the current value of the *i*th token. For each token x_i, there will be top-k values with the largest negative gradient as replacement candidate for each iteration and some of them will be randomly choose to replace the current token to form the adversarial suffix with smallest loss.

To test the efficiency of our method, we modify the original GCG (Zou et al., 2023) attack by integrated the guardrail loss into the original GCG loss function. Specifically, let V_t denote the target model's vocabulary, the target of GCG attack is to find the replaceable Top-k token for each of the ith token according to the gradient

$$\nabla_{e_{x_i}} \mathcal{L}(x_{n:n+l}) \in \mathbb{R}^{|V_t|}$$

To combine the guardrail loss into the original loss, we choose to take the intersection of vocab of guardrail model V_g and that of target model V_t which let us calculate the gradient

$$\nabla_{e_{x_i}} \mathcal{L}(x_{n:n+l}) \in \mathbb{R}^{|V_t \cap V_g|}$$

Let $x' = \text{Concat}(x_{1:n}, x_{n:n+l})$, we modify the loss to become

$$\mathcal{L}(x') = -\log p(x^\star_{n+1:n+H} \mid x') + \mathcal{L}_{\text{PG}}(x')$$

When facing this adaptive attack, off-the-shelf guardrail model will easily be compromised. To improve the adversarial robustness of the guardrail model, we adversarially training the guardrail model which utilized examples crafted by GCG as adversarially examples.

3.3 Fast Embedding Adversarial Training (FEAT)

Algorithm 1 Adversarial Suffix Crafting

Require: Input embedding $\mathbf{e_x} \in \mathbb{R}^{n \times d}$, model f_θ, target label y_{target}, suffix length s, maximum iterations T
Ensure: Modified embedding $\mathbf{e}'$
  Initialize suffix $\boldsymbol{e}_{\text{s}}$
  Initialize suffix perturbation $\boldsymbol{\delta} \in \mathbb{R}^{s \times d}$
  $\mathcal{L} = f_\theta(\mathbf{e_x}' = \text{Concat}(\mathbf{e_x}, \boldsymbol{e}_{\text{s}} + \boldsymbol{\delta})), y_{\text{target}})$
  for $t = 1$ to T **do**
    Compute gradient $\nabla_{\boldsymbol{\delta}} \mathcal{L}$
    Update $\boldsymbol{\delta}$ using the gradient
    Project $\boldsymbol{\delta}$ back into feasible region
    if stopping condition is met **then**
      break
    end if
  end for
  Reconstruct adversarial embedding: $\mathbf{e_x}' = \text{Concat}(\mathbf{e_x}, \boldsymbol{e}_{\text{s}} + \boldsymbol{\delta})$
  Return $\mathbf{e}'$

However, crafting even one adversarial example using GCG require non-trivial computational resource. In adversarial training, it usually requires to craft a portion of regular training dataset as adversarial examples, which make it infeasible to craft the adversarial examples in token space during training.

We observe that crafting the adversarial example in embedding space, while it can't be map back to token space, can achieve similar performance compare to craft adversarial example in token space and with much less computational resources. Crafting the adversarial examples in embedding space took less iterations to achieve the same loss as crafting in token space as shown in Figure 2 and require 95% less time to finish one step iteration.

To perform GCG attack in embedding space, we instead initialize the adversarial suffix in the form of embedding e_s

$$e_s \in \mathbb{R}^{|L| \times d},$$

where L is the pre-defined suffix length and make this as our optimizable adversarial suffix. In this way, we can optimize the suffix over embedding space instead of discrete space. Because of that, we can craft the adversarial example similar in computer vision, which apply a small perturbation

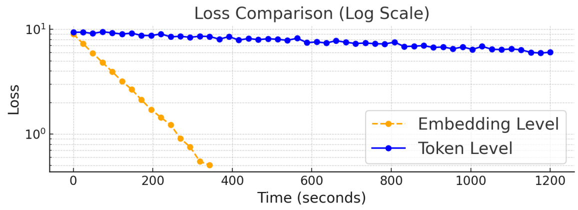

Figure 2: Comparison of time required to craft one adversarial example at the token level versus the embedding level. The results highlight the efficiency of crafting adversarial examples in the embedding space.

$\boldsymbol{\delta} \in \mathbb{R}^{|L| \times d}$ to the suffix embedding $\mathbf{e_s}$

$$e'_s = e_s + \boldsymbol{\delta}$$

The perturbation $\boldsymbol{\delta}$ is optimized using a gradient-based approach to minimize the adversarial loss. We first randomly generate an adversarial suffix e_s with pre-defined length. With a constraint of 200 steps, we iteratively applying perturbation on it based on the gradient of back propagation given by the cross entropy loss of current prediction and the malicious target. As a result, the embedding composed of the concatenation of the original embedding e_x and the crafted suffix e_s

$$e'_x = \text{Concat}(e_x, e_s + \boldsymbol{\delta})$$

Our objective here is to craft the adversarial example to mislead the guardrail model to identify the malicious input as benign.

Adversarial targeted attack: For targeted attacks, the adversarial loss is defined as:

$$\mathcal{L}_{\text{adv}} = -\log\left(p_{y_{\text{benign}}}(f_\theta(e'_x))\right),$$

where $p_y(f_\theta(\cdot))$ represents the predicted probability of the label y. The perturbation $\boldsymbol{\delta}^*$ is computed iteratively:

$$\boldsymbol{\delta}^* = \arg\min_{\boldsymbol{\delta}} \mathcal{L}_{\text{adv}}(f_\theta(e'_x), y_{\text{benign}}).$$

Optimization Process: The optimization is performed using backpropagation. The gradient of the loss with respect to $\boldsymbol{\delta}$ is computed as:

$$\boldsymbol{\delta} \leftarrow \boldsymbol{\delta} + \eta \nabla_{\boldsymbol{\delta}} \mathcal{L}_{\text{adv}},$$

where η is the learning rate. After each update, the perturbation is projected back into the feasible region defined by the ℓ_p norm. As shown in algorithm 1

3.4 Adversarial Training

Inspired by the training flow in (Yoo and Qi, 2021), our method differs from traditional adversarial training approaches in computer vision (Goodfellow et al., 2015), which generate adversarial examples between every mini-batch, because of the GPU memory constrain. We instead generate the adversarial examples at the beginning of each training epoch. Instead of generate one adversarial example for each of the datapoint in the dataset, we generate a specified ratio, $\boldsymbol{\alpha}$, of clean malicious data (data containing malicious characteristics) as adversarial examples before each training epoch. When adversarial example crafting fails, we simply skip that example. The default value of $\boldsymbol{\alpha}$ is set to 20%.

To craft adversarial examples in the embedding space, we first transform the entire dataset into its embedding representation. Due to the mismatch in embedding dimensions between the guardrail model and the target model, we are unable to perform adaptive attacks in the embedding space—i.e., adversarial optimization cannot be conducted on both models simultaneously.

Thus, we focus solely on crafting adversarial examples using malicious data, assuming that small perturbations to malicious inputs do not alter their malicious nature. Consequently, we treat the guardrail model as the sole target during adversarial example crafting.

To preserve the benign utility of the guardrail model while enhancing its adversarial robustness, adversarial training combines clean and adversarial examples to improve model performance. We select focal loss as the loss function because we observe that during adversarial training, the model tends to be less stable when classifying benign examples. Focal loss addresses this issue by dynamically adjusting the model's focus toward examples with labels that are harder to classify, thereby mitigating instability and improving overall robustness. The overall loss function is:

$$\mathcal{L}_{\text{total}} = \mathcal{L}(f_\theta(x), y) + \alpha \cdot \mathcal{L}_{\text{adv}}(f_\theta(\mathbf{x}'_{\text{emb}}), y_{\text{benign}}),$$

where $\mathcal{L}(f_\theta(x), y)$ is the clean loss (e.g., cross-entropy) for unperturbed inputs, $\mathcal{L}_{\text{adv}}(f_\theta(\mathbf{x}'_{\text{emb}}), y_{\text{benign}})$ is the adversarial loss for perturbed inputs, and $\alpha \in [0, 1]$ is a hyperparameter balancing the contributions of clean and adversarial examples. Model parameters θ are updated to minimize $\mathcal{L}_{\text{total}}$ using stochastic gradient descent. The training process is demonstrated in Algorithm 2.

Algorithm 2 Adversarial Training with Malicious Crafting

Require: Training dataset $\mathcal{D}$, Model f_θ, Tokenizer $\mathcal{T}$, Hyperparameters α, Epochs E
Ensure: Trained model f_θ
Initialize model parameters θ
Split $\mathcal{D}$ into clean ($\mathcal{D}_c$) and malicious ($\mathcal{D}_m$) subsets
procedure ADVERSARIALTRAINING
  for epoch $= 1$ **to** E **do**
    Craft adversarial subset: $\mathcal{D}_{\text{adv}} \leftarrow$ CRAFTMALICIOUS$(\mathcal{D}_m, f_\theta, \alpha)$
    Combine dataset: $\mathcal{D}' \leftarrow \mathcal{D}_c \cup \mathcal{D}_{\text{adv}}$ ▷ Ratio α determines $\mathcal{D}_{\text{adv}}$ size
    for each mini-batch (x_i, y_i) in $\mathcal{D}'$ **do**
      Compute loss: $\mathcal{L}_i \leftarrow \mathcal{L}(f_\theta(x_i), y_i)$
      Accumulate gradients and update: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_i$
    end for
  end for
end procedure

4 Experiment

In this section, we outline the datasets, models, evaluation metrics, and baselines used in our study.

4.1 Settings

Models and Datasets. We use TaskTracker (Abdelnabi et al., 2024) as our training dataset. To test the robustness and availability of our method, besides TaskTracker evaluation dataset, we also use BIPIA (Yi et al., 2024) and PINT (AI, 2024) benchmark to test our method. We select a diverse set of prompt-guard models to serve as guardrails for our target model, Llama3-8B. The models include ProtectAI Prompt Guard (ProtectAI, 2024), Meta Prompt Guard (Meta-Llama, 2024), Epivolis Prompt Guard (Epivolis, 2024), and DeepSet Prompt Guard (deepset, 2024).

Evaluation Metrics. We use *Attack Success Rate* (ASR) to test our methods. ASR quantifies the number of success APGA over the entire number of attack dataset. Calculated as ASR $= \frac{\text{Number of Successful Attack}}{\text{Total Number of Attack Cases}}$. We also test the *Availability* of our method which evaluates the model's performance on benign inputs. This metric is crucial in ensuring that while the model is robust against

Model	In-domain (APGA)				OOD (APGA)
	FEAT	Finetune	Token-level	original	FEAT
Epivolis	5	60	100	100	5
protectai_v2	35	50	90	100	45
deepset	10	50	20	100	25
meta-llama	25	80	95	100	45

Table 1: ASR from two datasets (In-domain vs. *out-of-distribution* (OOD)) using APGA. The first four columns show results on TaskTracker data under different conditions. The last column shows the performance of FEAT against OOD (BIPIA) dataset.

Dataset	Mode	Training Method					
		Original		Fine-tuned		FEAT	
		Benign	**Malicious**	**Benign**	**Malicious**	**Benign**	**Malicious**
Microsoft	protectai_v2	99.99	4.93	81.10	83.08	84.54	84.06
	deepset	0	100	97.30	78.75	74.84	88.97
	meta-llama	0.90	99.83	56.97	94.80	93.13	88.37
	Epivolis	0.41	77.74	49.92	92.32	78.84	89.61
Pint	protectai_v2	90.9986		86.5939		84.1433	
	deepset	57.7255		72.3648		68.5684	
	meta-llama	69.1619		76.539		65.635	
	Epivolis	62.6572		50.0		56.06	

Table 2: Combined performance metrics under three training conditions. For the TaskTracker validation dataset, both benign and malicious accuracies are reported side by side. For the Pint Score dataset, the three training method scores are shown.

adversarial attacks, it does not compromise on its primary functionality when handling benign data.

Experimental Setting. In our experimental setup, we designate Llama3 as the target model. Each guardrail model is integrated with Llama3 to enhance its robustness. We keep the target model consistent across experiments, modifying only the guardrail models. Training parameters include a learning rate of 1e-5, a per-device batch size of 8, and a gradient accumulation step of 4. The maximum text length is capped at 1,024 tokens, with training conducted at 5 epochs. For FEAT, 20% of the malicious data points are augmented with adversarial examples. To test the robustness of our method against APGA, we curated 20 examples of APGA from the TaskTracker validation set which ensuring no overlap with the training data, and another 20 examples from BIPIA to test the generalization to out of distribution ability of our method. These data are all under the setting of prompt injection.

4.2 Evaluation of Attack Success Rate (ASR)

Table 1 presents the ASR (Attack Success Rate) results for various prompt-guard models when subjected to APGA. To demonstrate the superior robustness of our method, we conduct experiments under four different settings: fine-tuned (standard training without adversarial examples), token-level adversarial training (restricted to the same resources as FEAT), original model (without any fine-tuning), and our proposed method FEAT.

Our findings indicate a significant reduction in ASR when adversarial training is applied. For in-

Attack Type	Model	Attack Success Rate (%)	
		Fine-tuned	**FEAT**
Email	Deepset	66.24	0.92
	Meta-LLaMA	34.46	67.59
	Epivolis	6.64	1.35
	ProtectAI	8.05	0.09
Table	Deepset	77.48	10.72
	Meta-LLaMA	37.56	1.89
	Epivolis	78.9	4.44
	ProtectAI	76.88	22.02
Code	Deepset	50.48	4.50
	Meta-LLaMA	39.55	48.98
	Epivolis	18.96	3.11
	ProtectAI	84.18	5.13
Total	Deepset	67.92	6.72
	Meta-LLaMA	37.29	30.09
	Epivolis	45.8533	3.33
	ProtectAI	61.50	12.31

Table 3: Attack Success Rates (%) for Different Models of two methods and Attack Types from BIPIA.

stance, Epivolis/Hyperion experiences a dramatic ASR drop from 60% in the fine-tuned setting to just 5% with FEAT. Compared to token-level adversarial training, our method also achieves a substantial reduction in ASR, highlighting not only its robustness but also its cost efficiency. Similarly, other models such as ProtectAI and DeepSet exhibit notable improvements, with ASR reductions of 15% and 40%, respectively. These results support our hypothesis that FEAT significantly enhances the resilience of prompt-guard models against APGA, effectively reducing their vulnerability to prompt injection attacks.

To further evaluate the robustness of our method across different settings, we leverage BIPIA, a comprehensive prompt injection benchmark that includes a diverse range of prompt injection attacks. As shown in Table 3, FEAT consistently leads to a significant drop compared to fine-tuned in ASR across various attack scenarios. This reinforces the effectiveness of our method in mitigating not only APGA but also conventional prompt injection threats, demonstrating its broad applicability in enhancing model security.

4.3 Evaluation of Accuracy on Benign and Malicious Data

Table 2 presents accuracy metrics for both benign and malicious inputs on the Microsoft out-of-domain validation dataset and the Pint benchmark. We evaluate each model's accuracy under three conditions: original, fine-tuned, and FEAT. Our goal is to assess how well each model maintains high accuracy on benign inputs while resisting prompt injection attacks.

To ensure that our method does not significantly compromise utility, we compare its performance against the fine-tuned-only approach on both the Microsoft validation dataset and the Pint benchmark. As shown in Table 2, our method demonstrates noticeable performance improvements over the original models while maintaining utility comparable to fine-tuned-only models. When FEAT is applied, benign accuracy does not degrade significantly compared to fine-tuned alone. This is further supported by the Pint benchmark evaluation, where the differences in Pint scores between adversarially trained and original models remain within 6%, with some models even showing improved performance after adversarial training.

The original models exhibit a strong bias toward one label. For example, meta prompt guard achieves 99.83% malicious accuracy but only 0.41% benign accuracy, while ProtectAI reaches 99.99% benign accuracy but only 4.93% malicious accuracy in the original setting. This imbalance can persist even after fine-tuned, leading to performance degradation. However, FEAT mitigates this bias, achieving a more balanced accuracy between benign and malicious inputs. For instance, in Table 2, the fine-tuned version of Epivolis attains 92.32% malicious accuracy but only 49.92% benign accuracy, which is close to random guessing. In contrast, the FEAT version of Epivolis achieves a more balanced performance across both categories, demonstrating greater resistance to the biases of the original models.

In summary, our experiments confirm that FEAT effectively reduces the ASR of APGA across multiple prompt-guard models, strengthening resilience against prompt injection while only minimally affecting benign accuracy. This underscores the potential of adversarial training as a robust defense for securing LLMs against gradient-based prompt injection attacks. Notably, models like Epivolis and ProtectAI using FEAT exhibit strong performance in balancing benign and malicious accuracy, further reinforcing the effectiveness of FEAT in improving model robustness.

4.4 Ablation Studies

To investigate the performance variation under different hyperparameter setting during training. We explore the different setting of various of suffix length and the ratio of adversarial examples α. The detailed results deferred to Appendix. 5.

5 Conclusions and Limitations

In this paper, to fully evaluate the existing prompt guard models. We introduce an adaptive attack APGA. We point out the lack of robustness of the existing prompt guard model against sophisticated prompt injection attack which combines with adversarial attacks. To solve this problem, we introduce FEAT and demonstrate the robustness against APGA as well as regular prompt injection attacks.

While our method demonstrates robustness against both sophisticated and regular prompt injection attacks across different settings, we did not extend the evaluation to additional foundation LLMs due to resource constraints.

Ethics Statement

We are committed to advancing the security and integrity of LLMs responsibly. In this research, we introduce APGA, an attack method designed to stress test the models robustness. Additionally, we introduce FEAT, a novel training method aimed at enhancing LLM security efficiently. All data used are synthetically generated or sourced from publicly available datasets, ensuring that no personal or sensitive information is involved. This approach safeguards privacy and complies with ethical standards regarding data use.

While our work focuses on enhancing defensive mechanisms against prompt injection attacks, we acknowledge the potential for dual use in security research. We encourage the ethical and responsible use of APGA to improve LLM security and not for malicious purposes. Our commitment to transparency is reflected in making both the dataset and model fully open-source, fostering collaboration, and allowing others to verify, replicate, and build upon our work for the betterment of the field. To foster future research in this area, we will open-source our code under the MIT license.

References

Sahar Abdelnabi, Aideen Fay, Giovanni Cherubin, Ahmed Salem, Mario Fritz, and Andrew Paverd. 2024. Are you still on track!? catching llm task drift with activations. *Preprint*, arXiv:2406.00799.

Lakera AI. 2024. Lakera pint benchmark. Accessed: 2025-02-10.

Tao Bai, Jinqi Luo, Jun Zhao, Bihan Wen, and Qian Wang. 2021. Recent advances in adversarial training for adversarial robustness. *Preprint*, arXiv:2102.01356.

Hezekiah J. Branch, Jonathan Rodriguez Cefalu, Jeremy McHugh, Leyla Hujer, Aditya Bahl, Daniel del Castillo Iglesias, Ron Heichman, and Ramesh Darwishi. 2022. Evaluating the susceptibility of pre-trained language models via handcrafted adversarial examples. *Preprint*, arXiv:2209.02128.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arxiv:2005.14165.

deepset. 2024. Deberta v3 base injection. https://huggingface.co/deepset/deberta-v3-base-injection.

Deepset. 2024. Deepset Prompt Injection Guardrail. https://huggingface.co/deepset/deberta-v3-base-injection.

Yi Dong, Ronghui Mu, Gaojie Jin, Yi Qi, Jinwei Hu, Xingyu Zhao, Jie Meng, Wenjie Ruan, and Xiaowei Huang. 2024. Building guardrails for large language models. *Preprint*, arXiv:2402.01822.

Epivolis. 2024. Hyperion model. https://huggingface.co/Epivolis/Hyperion.

fmops. 2024. Fmops Prompt Injection Guardrail. https://huggingface.co/fmops/distilbert-prompt-injection.

Siddhant Garg and Goutham Ramakrishnan. 2020. Bae: Bert-based adversarial examples for text classification. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. *Preprint*, arXiv:1412.6572.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad,

Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim

Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023a. Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. *arXiv preprint*. ArXiv:2302.12173 [cs].

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023b. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. *Preprint*, arXiv:2302.12173.

Rich Harang. 2023. Securing llm systems against prompt injection.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023a. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. In the Proceedings of ICLR.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023b. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. *Preprint*, arXiv:2111.09543.

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

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. *Preprint*, arXiv:1907.11932.

LakeraAI. 2024. LakeraGuard. `https://www.lakera.ai/lakera-guard`.

Hao Li and Xiaogeng Liu. 2024. Injecguard: Benchmarking and mitigating over-defense in prompt injection guardrail models. *Preprint*, arXiv:2410.22770.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. 2024a. Autodan: Generating stealthy jailbreak prompts on aligned large language models. *Preprint*, arXiv:2310.04451.

Xiaogeng Liu, Zhiyuan Yu, Yizhe Zhang, Ning Zhang, and Chaowei Xiao. 2024b. Automatic and universal prompt injection attacks against large language models. *arXiv preprint arXiv:2403.04957*.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. 2024c. Formalizing and benchmarking prompt injection attacks and defenses. *Preprint*, arXiv:2310.12815.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2019. Towards deep learning models resistant to adversarial attacks. *Preprint*, arXiv:1706.06083.

Meta. 2024. PromptGuard Prompt Injection Guardrail. `https://www.llama.com/docs/model-cards-and-prompt-formats/prompt-guard/`.

Meta-Llama. 2024. Prompt-guard. `https://github.com/meta-llama/PurpleLlama/tree/main/Prompt-Guard`.

Takeru Miyato, Andrew M. Dai, and Ian Goodfellow. 2021. Adversarial training methods for semi-supervised text classification. *Preprint*, arXiv:1605.07725.

John X. Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. *Preprint*, arXiv:2005.05909.

Rodrigo Pedro, Daniel Castro, Paulo Carreira, and Nuno Santos. 2023. From prompt injections to sql injection attacks: How protected is your llm-integrated web application? *Preprint*, arXiv:2308.01990.

Fábio Perez and Ian Ribeiro. 2022a. Ignore Previous Prompt: Attack Techniques For Language Models. *arXiv preprint*. ArXiv:2211.09527 [cs].

Fábio Perez and Ian Ribeiro. 2022b. Ignore previous prompt: Attack techniques for language models. *Preprint*, arXiv:2211.09527.

ProtectAI. 2024. Deberta v3 base prompt injection v2. `https://huggingface.co/protectai/deberta-v3-base-prompt-injection-v2`.

ProtectAI.com. 2024. Fine-tuned deberta-v3-base for prompt injection detection.

Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. 2019a. Adversarial training for free! *Preprint*, arXiv:1904.12843.

Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. 2019b. Adversarial training for free! In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2024. "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. *Preprint*, arXiv:2308.03825.

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

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. *Preprint*, arXiv:1312.6199.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2021. Universal adversarial triggers for attacking and analyzing nlp. *Preprint*, arXiv:1908.07125.

Simon Willison. 2022. Prompt injection attacks against GPT-3. `https://simonwillison.net/2022/Sep/12/prompt-injection/`.

Eric Wong, Leslie Rice, and J. Zico Kolter. 2020. Fast is better than free: Revisiting adversarial training. *Preprint*, arXiv:2001.03994.

Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. 2024. Benchmarking and defending against indirect prompt injection attacks on large language models. *Preprint*, arXiv:2312.14197.

Jin Yong Yoo and Yanjun Qi. 2021. Towards improving adversarial training of nlp models. *Preprint*, arXiv:2109.00544.

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

Appendix

A Ablation Study on Suffix Length and Alpha

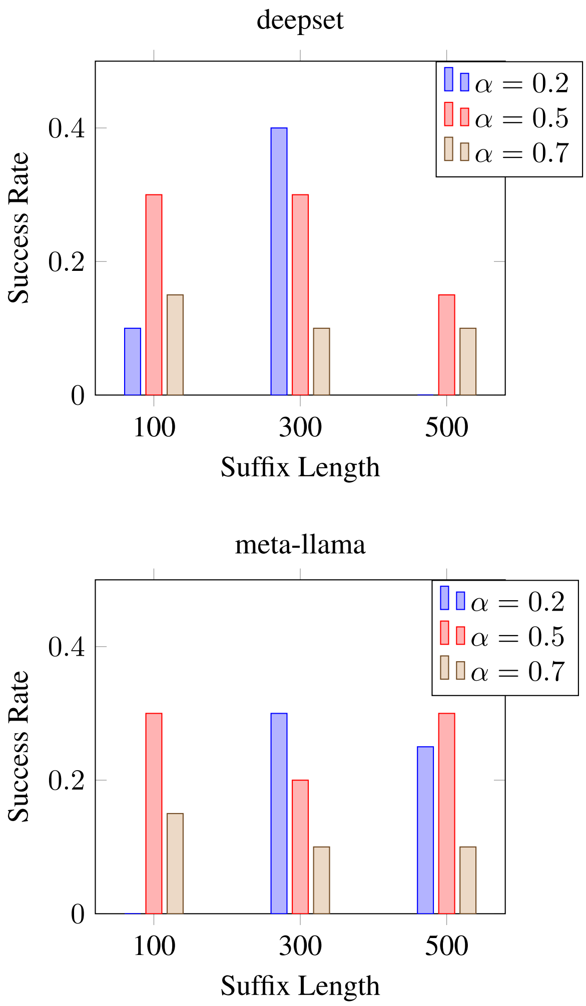

Figure 3: Comparison of Success Rates across models for different suffix lengths and α values.

To examine the effects of different hyperparameter combinations, we explore suffix lengths of 100, 300, and 500, along with α values of 0.2, 0.5, and 0.7. We select Deepset and Meta-LLaMA as the targets for our ablation study, as they demonstrate more stable performance across different settings.

As shown in Figure 3, we observe some fluctuations across different configurations. While suffix length does not exhibit a clear positive impact on model robustness overall, we find that higher α values generally lead to increased robustness. When $\alpha = 0.2$, model performance tends to be random. However, as α increases, performance stabilizes, and models become more resilient. This suggests that larger α values contribute to greater robustness, showing the importance of this hyperparameter in optimizing model resistance to attacks.