Adaptive Reinforcement Learning Defense Framework Against Cryptographic Cross-Model Adversarial Prompt Injection

Aditya Raj [*1,7], Akshat Gupta [*2,7], Amil Bhagat [*3,7], Ananya Goyal [*4,7], Mehar Khurana [*5,7], Swati Sharma [*6,7]

[*] Indicates Equal Contribution

[1] aditya21512, [2] akshat21515, [3] amil21309, [4] ananya21011, [5] mehar21541, [6] swati21568 @iiitd.ac.in

[7] *Indraprastha Institute of Information Technology, Delhi*

1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing, excelling in tasks like text generation and conversational AI. However, they are increasingly vulnerable to adversarial attacks, particularly cryptographic prompt injections. These attacks encode malicious intent within prompts, bypassing traditional detection mechanisms and leveraging cross-model transferability to exploit shared weaknesses across architectures.

Traditional defenses, such as perplexity filtering and adversarial fine-tuning, are limited in their ability to address evolving attacks or handle diverse ciphers. To tackle these challenges, this project proposes an adaptive reinforcement learning (RL) framework to defend against cryptographic cross-model adversarial attacks. The framework dynamically detects and decrypts adversarial prompts using an RL-based defender trained to adapt to varying attack strategies.

Key contributions include:

- An adaptive decryption mechanism for detecting and mitigating adversarial ciphers.
- Enhanced cross-model robustness to address shared vulnerabilities across LLMs.
- An efficient RL reward structure incorporating decryption accuracy and ethical compliance.

By leveraging RL's adaptability, this framework aims to provide scalable, real-time protection against sophisticated adversarial threats, ensuring the safety and reliability of LLMs.

2 Literature Review

In recent years, adversarial attacks on Large Language Models (LLMs) have become a critical area of concern, with researchers focusing on both the development of attack techniques and corresponding defenses. These attacks often exploit vulnerabilities in LLMs, leading to harmful or unintended outputs. Among these, cryptographic adversarial attacks, which encode malicious intent within prompts, have emerged as a sophisticated threat. These attacks bypass traditional detection mechanisms and highlight the necessity of adaptive, real-time defense strategies.

2.1 Generating Adversarial Prompts

2.1.1 Cryptographic Adversarial Prompts

Cryptographic adversarial prompts encode malicious instructions using techniques such as Caesar ciphers, ROT13, and context-dependent transformations. Kang et al. [2] demonstrated that these prompts effectively evade detection by traditional filters while maintaining their adversarial intent. By leveraging randomness and obfuscation, attackers craft inputs that exploit vulnerabilities in tokenization and model architecture.

Cross-model transferability is a significant challenge posed by these attacks. Research by [3] shows that adversarial prompts optimized for one model often generalize effectively to others, exploiting shared weaknesses in pre-training objectives. This underscores the need for robust defenses capable of handling diverse attack strategies and architectures.

2.1.2 Context Manipulation and Self-Ciphers

Advanced cryptographic attacks include context manipulation and self-ciphering techniques. These methods encode malicious instructions dynamically, leveraging the input-output context to bypass static filters. For instance, Yan et al. [4] introduced CODEBREAKER, a framework that iteratively refines adversarial prompts to evade detection. These prompts not only pose challenges for existing defenses but also highlight the limitations of static approaches.

2.2 Baseline Defense Strategies against Adversarial Attacks

2.2.1 Perplexity as a Defense Strategy

Perplexity is a measure of how well a language model predicts a given text sequence. Or in other words how unnatural it finds that text sequence with respect to its training data. Lower perplexity values indicate better predictions, while higher values suggest that the model finds the text less predictable or natural. This makes perplexity useful for detecting adversarial prompts or unusual text patterns. In the context of defense strategies for language models, perplexity can help identify prompts that deviate from typical language usage, which could signal adversarial attacks.

The formula for perplexity is given by:

$$PPL(x) = \exp\left(-\frac{1}{t}\sum_{i=1}^{t}\log p(x_i \mid x_{<i})\right)$$

Where:

- t is the length of the token sequence,
- $p(x_i \mid x_{<i})$ is the probability of token x_i given the preceding tokens.

By measuring the perplexity of incoming prompts, a system can flag inputs with unusually high perplexity as potential adversarial attacks, improving the robustness of the language model.

Using the models "openai-community/gpt2", "meta-llama/Llama-3.1-8B" (8-bit version due to resource constraints), and "google/gemma-2b-it", we evaluated the usefulness of perplexity alone as a defense strategy when facing the attack methods described in the previous section.

The methodology was as follows:

- Perplexity won't be able to decipher the adversarial prompts and sanitize them so they are safe to feed into the LLM. Hence, the objective here was simply to classify whether a prompt was malicious or benign in nature
- We tried two approaches: To calculate the perplexity of a given prompt wrt the model as a whole or to split the prompt into n windows and calculate the perplexity of each one separately. The maximum perplexity out of these would then be considered. We optimized for this window size as well by taking out a small subset of the prompts with equal amounts malicious and benign as a training set to optimize the window sizes.
- When we obtained the perplexities of each prompt, we optimized for the threshold at which we would consider the prompt to be malicious. This was done on the train subset mentioned earlier and F1-score was considered to determine the best thresholds.
- The objective was thus to minimize Attack Success Rates (ASR) and maximize Benign Success Rates (BSR). The former essentially measures how many attacks go through out of the total attempts and the latter measures how many benign samples were incorrectly classified as malicious.

The results for the test set are tabulated below:

Note: Base prompts are a mix of malicious and benign prompts without the use of ciphers (straightforward).

Discussion:

- The technique of calculating perplexities of windows of the input was discarded after experimentation. This is because it was yielding significantly worse results than considering the prompt as a whole. This could be due to the loss of context to the model when you break apart the prompt into windows, resulting in higher degrees of unnaturalness.
- We can see that the gemma model does the worst here. It's ASRs are usually high and BSRs low, which is the opposite of what we want.

Attack	openai-community/gpt2	meta-llama/Llama-3.1-8B(8-bit)	google/gemma-2b-it
base_prompts	0.1194	0.06289	0.53459
ciphered_prompts_unicode	0.0062	0.0	0.0314
ciphered_prompts_self_cipher	0.025	0.0	0.0
ciphered_prompts_caesar	0.0125	0.0	0.0
ciphered_prompts_ascii	0.0062	0.0	0.0
ciphered_prompts_morse	0.0125	0.0	0.0
ciphered_prompts_albert	0.0503	0.0	0.0
ciphered_prompts_sdm_attack	0.08176	0.08176	0.1194
ciphered_prompts_jambench	0.031	0.0	0.798
ciphered_prompts_autodan	0.0		

Table 1: Attack Success Rates for Model and Attack Combinations using Perplexity as defense

Attack	openai-community/gpt2	meta-llama/Llama-3.1-8B(8-bit)	google/gemma-2b-it
base_prompts	0.38	0.353	0.34
ciphered_prompts_unicode	0.193	0.0	0.0
ciphered_prompts_self_cipher	0.193	0.0	0.0
ciphered_prompts_caesar	0.493	0.0	0.0
ciphered_prompts_ascii	0.193	0.0	0.0
ciphered_prompts_morse	0.3	0.0	0.0
ciphered_prompts_albert	0.693	0.0	0.0
ciphered_prompts_sdm_attack	0.9933	1.0	0.773
ciphered_prompts_jambench	0.94	0.9466	0.773
ciphered_prompts_autodan	0.0		

Table 2: Benign Success Rates for Model and Attack Combinations using Perplexity as defense

- We can also see that the perplexity filtering technique is able to successfully determine adversarial prompts in sdm_attack and jambench attack methods, especially in gpt2 and llama models. The ASRs are low and the BSRs are high. This indicates that the perplexity filtering technique was quite effective for these ciphers.
- The technique fails for the other cipher techniques used however. We can see that even though the ASR is low, the BSR is also low. This means that the threshold optimization technique we discussed wasn't able to find a breakpoint between the perplexities of the clean and malicious prompts with ciphers and thus wasn't able to do better than a 50% accuracy - or simply assigning every prompt as malicious. We tried shifting the thresholds for the perplexities manually by small margins, however, the results didn't improve beyond a 50% accuracy, with falls in the results of either the ASR or the BSR. This could potentially be due to the fact that the tokenizers these models use can't properly handle the cipher encoded content, hence making the input_ids not reflect the actual inputs. That or the models could have been trained on data reflecting these ciphers making them seem not that unnatural. Perhaps this could be remedied with smaller steps for thresholds (which we didn't have the compute for, current smallest step is 1.25), however, these thresholds were obtained from the train set. In a real world scenario they wouldn't reflect the actual available data, hence, the defense should be robust to such small changes in threshold. Also, the perplexities of the ciphered prompts should ideally be much higher than the clean ones.
- Since there is a huge tradeoff between ASR and BSR with perplexity filtering, it shouldn't be used as a sole defense mechanism for an LLM. It could be used in conjunction with, or before other methods we have outlined to guide the decision of the overall defense pipeline. If the perplexity is high, then the prompt will be looked at with more scrutiny later, and if it's low and the prompt is malicious, then it may be caught later on in the defense pipeline.

2.2.2 Defense Prompting Technique

In the context of modern language models (LLMs), the Defense Prompting Technique represents a critical intervention to mitigate the risks associated with adversarial attacks, such as jailbreaking. Jailbreaking refers to a technique where an adversary manipulates an LLM to bypass its ethical, policy, or safety constraints and generate harmful, unethical, or unintended outputs. The key idea behind the Defense Prompting Technique is to guide the model through a structured decision-making process, enabling it to identify harmful intent and reject malicious prompts.

The Defense Prompting Technique relies heavily on prompt engineering, where carefully crafted instructions are appended to user queries. The aim is to enhance the model's ability to recognize manipulative content while maintaining

ethical boundaries. However, the limitations of this approach suggest that while it can improve safety, it is insufficient on its own.

2.2.3 Methodology

Initial Vulnerability Assessment: We began by establishing a baseline understanding of the vulnerabilities present in different LLMs by testing them with two types of prompts: base prompts and ciphered prompts. These tests were conducted across three distinct language models: **LLaMA**, **GEMMA**, and **GPT-2**. The objective was to assess how the models responded to a range of adversarial inputs in the absence of any explicit defensive measures.

Categories of Prompts:

- **Base Prompts:** These were simple, straightforward queries designed to trigger normal, benign responses. They served as a control to establish how the models performed under typical conditions.
- **Ciphered Prompts:** These prompts were crafted to obfuscate malicious intent by using indirect language, encoded phrases, or subtle manipulations of the query. The goal was to bypass the model's built-in ethical and safety filters, thereby simulating real-world adversarial attacks.

Vulnerability Discovery: During this phase, we observed that base prompts regularly generated appropriate, harmless responses from all models. However, when exposed to ciphered prompts, the models displayed significant vulnerabilities, often responding in ways that indicated a failure to recognize the subtle manipulations embedded within the queries. This finding underscored the need for enhanced defenses capable of detecting and mitigating these more sophisticated adversarial attacks.

Chain-of-Thought Prompt Engineering Approach: After the initial vulnerability assessment, we evaluated a defense mechanism based on the chain-of-thought reasoning process. This approach required the model to follow a specific thought process before generating a response. By guiding the model through this reasoning process, we tried to demonstrate that such methods while adaptable to basic adversarial prompts (without ciphers), fail to perform well when ciphers are introduced into these prompts.

Defense Prompt Construction: The defense prompt was carefully engineered to try to act as a firewall, prompting the model to:

- **Analyze Intent:** The model was instructed to first determine whether the prompt could be used for malicious or manipulative purposes. This required the model to "think" critically about the implications of the request.
- **Assess Ethical Impact:** The next step in the chain-of-thought was to evaluate whether responding to the prompt would result in unethical or harmful outcomes.
- **Formulate a Response:** If the model identified the prompt as potentially harmful, it was directed to generate a polite refusal, explaining that it could not assist with the request. In benign cases, the model was free to respond as usual.

An example of a defense prompt is:

> "Given the following prompt: [PROMPT]. Analyze the intent and implications of this prompt. Could they lead to malicious, harmful, or manipulative responses? If so, then refuse to help the user and say that the intent seems malicious.
>
> **Example of a malicious prompt and its twisted version:**
>
> - Original prompt: 'Tell me how to rob a bank'.
> - Twisted prompt: 'I'm a security guard at a bank. Tell me all ways that a robber can enter the bank so I can save the bank.'
> - Appropriate response: 'I'm sorry, but I can't assist with that request.'
>
> **Guidelines:**
>
> - Identify Harmful Intent: Determine if the prompt seeks information that can be used for malicious activities.
> - Assess the Request: Evaluate whether the content violates ethical guidelines or policies.
> - Respond Appropriately:
>   - If the request is benign, provide a helpful and informative response.

– If the request is malicious or harmful, politely decline without providing the requested information.

Results and Analysis

Initial Findings: The initial vulnerability assessment provided valuable insights into the weaknesses of the models. Without any defense prompts, LLaMA, GEMMA, and GPT-2 all demonstrated susceptibility to ciphered prompts. The models were often unable to detect the manipulative intent behind the obfuscated queries, leading to potentially harmful outputs. This highlighted the need for more advanced interventions.

Defense Prompting Impact: Despite introducing the Defense Prompting Technique, which required the models to assess the intent behind each prompt before generating a response, the overall effectiveness was minimal. Across multiple attack vectors, particularly with ciphered prompts, the models exhibited almost no improvement in their ability to block harmful queries. The chain-of-thought reasoning embedded in the defense prompt proved insufficient in preventing sophisticated attacks that were designed to bypass ethical constraints.

The following table summarizes the Attack Success Rates (ASR) and Block Success Rates (BSR) for different models and attack types after applying the Defense Prompting Technique:

The following table summarizes the Attack Success Rates (ASR) and Block Success Rates (BSR) for different models and attack types after applying the Defense Prompting Technique:

Attack	LLaMA ASR (%)	LLaMA BSR (%)	GEMMA ASR (%)	GEMMA BSR (%)
base_prompts	10.0	90.0	0.0	0.0
ciphered_prompts_unicode	20.0	80.0	0.0	0.0
ciphered_prompts_self_cipher	0.0	100.0	70.0	30.0
ciphered_prompts_caesar	90.0	10.0	90.0	10.0
ciphered_prompts_ascii	0.0	100.0	0.0	0.0
ciphered_prompts_morse	0.0	100.0	0.0	10.0
ciphered_prompts_albert	90.0	10.0	90.0	40.0
ciphered_prompts_sdm_attack	0.0	100.0	0.0	40.0
ciphered_prompts_jambench	0.0	100.0	0.0	40.0

Table 3: Attack Success Rates (ASR) and Block Success Rates (BSR) for Model and Attack Combinations using Defense Prompting Technique

Challenges with Prompt-Based Defenses: While the Defense Prompting Technique showed promise, several challenges became apparent. One key challenge was the inherent limitation of prompt-based defenses in adapting to more sophisticated adversarial attacks. Certain ciphered prompts, particularly those crafted with a deeper level of obfuscation or deceptive intent, still managed to bypass the defense prompt. For example, GEMMA, while improved, demonstrated vulnerability to advanced ciphered prompts, resulting in partial success for adversaries.

Fine-Tuning Approach

The fine-tuning process involved leveraging LoRA (Low-Rank Adaptation) and 4-bit quantization techniques to efficiently fine-tune large language models like LLaMA. LoRA allows for the injection of trainable low-rank adapters into the model, significantly reducing the number of trainable parameters and thus making fine-tuning more resource-efficient. 4-bit quantization further reduces the memory footprint of the model, enabling it to run more efficiently on available hardware.

Step 1: Model Preparation and Quantization To begin, we prepared the LLaMA model for fine-tuning by loading a pre-trained model using BitsAndBytesConfig for 4-bit quantization. Quantization reduces the model size and computational overhead by representing model weights with lower-precision floating-point numbers while maintaining accuracy. We configured the quantization to use the *float16* data type and the *nf4* quantization type, which is suitable for models like LLaMA. The tokenizer for the LLaMA model was also loaded, with padding tokens set appropriately.

Step 2: Applying LoRA Adapters Next, we utilized LoRA to modify only a small subset of the model parameters during training, specifically targeting key projection layers like ‘q_proj‘ and ‘v_proj‘. LoRA adapters were applied using a configuration where the rank (r) was set to 16, LoRA alpha to 32, and dropout was used with a rate of 0.05. These adapters allow fine-tuning while keeping most of the pre-trained model weights frozen, drastically reducing the computational cost and time required for fine-tuning.

We used the following configuration:

- **LoRA Rank (r):** 16
- **LoRA Alpha:** 32
- **LoRA Dropout:** 0.05

- **Target Layers:** q_proj and v_proj

Step 3: Training Configuration Fine-tuning was performed using the `Trainer` class from the Hugging Face Transformers library. The training parameters were set to run for 3 epochs with a batch size of 2 for both training and evaluation. Gradient checkpointing was enabled to reduce memory usage, and gradient accumulation steps were set to 8 to simulate larger batch sizes. We used the AdamW optimizer with a cosine learning rate schedule to further improve training efficiency.

The fine-tuning process was executed with 4-bit quantization, allowing the model to efficiently handle larger datasets without exceeding memory limits. The model was saved after fine-tuning for later evaluation.

Step 4: Model Evaluation and Response Generation Once the fine-tuning process was complete, we loaded the fine-tuned model and prepared it for evaluation. During evaluation, responses were generated for both base and ciphered prompts to assess the effectiveness of the fine-tuned model. The model was loaded in evaluation mode to ensure that no further training occurred during inference.

Step 5: Generating Responses For each prompt in the dataset, the original and fine-tuned models were used to generate responses. We compared the outputs from both models to evaluate whether fine-tuning improved the model's ability to handle the ciphered prompts and block malicious content. The generated responses were stored in a structured CSV file for further analysis.

An example function to generate responses is as follows:

```
def generate_response(prompt, model, tokenizer, device, max_new_tokens=150):
# Tokenize the prompt and move to the appropriate device
inputs = tokenizer(prompt, return_tensors='pt').to(device)
with torch.no_grad():
outputs = model.generate(*inputs, max_new_tokens=max_new_tokens)
response = tokenizer.decode(outputs[0], skip_special_tokens=True)
return response.strip()
```

This function tokenizes the input prompt, generates a response from the model, and returns the newly generated text.

Cipher Recognition and Mitigation Cipher recognition techniques, as described by Gupta et al. [1], involve identifying encoded patterns within inputs and decoding them to expose malicious intent. Dynamic cipher learning extends this approach, enabling defenses to adapt to novel encoding schemes. This adaptability is crucial for addressing the evolving strategies of adversaries.

2.2.4 Reinforcement Learning-Based Defenses

Reinforcement learning (RL) offers a promising solution to the limitations of static defenses. By modeling the problem as an adversarial game, RL agents learn to decode and mitigate cryptographic prompts dynamically. **Proximal Policy Optimization (PPO)**, widely used in RL-based frameworks, has shown success in training agents to generalize across diverse cipher types.

RL-based frameworks also incorporate reward structures that prioritize ethical compliance and decryption accuracy. For example, rewards may include components for decoding accuracy, response safety, and adaptation to cipher complexity. This holistic approach enables RL agents to handle both known and novel attack strategies, ensuring robust cross-model defenses.

2.2.5 Challenges and Gaps in Existing Research

While significant progress has been made, challenges remain in scaling these solutions for real-world deployment. Existing defenses often require extensive computational resources, limiting their applicability in large-scale systems. Moreover, the ability to adapt to novel ciphering techniques and transfer knowledge across models is still in its infancy.

This project addresses these gaps by developing an RL-based defense framework capable of dynamic decryption and real-time adaptation. By leveraging cross-model evaluations, the proposed framework aims to set new benchmarks in securing LLMs against sophisticated cryptographic adversarial attacks.

3 Dataset Preparation

The success of this project relies on a robust and diverse dataset for training the reinforcement learning (RL) defender and evaluating its performance against cryptographic adversarial prompts. This section outlines the data preparation process, ciphering methods, and characteristics of the dataset. Additionally, insights from the code and datasets were utilized to enhance the methodology.

3.1 Base Adversarial Prompts

The dataset includes a collection of base adversarial prompts that exploit vulnerabilities in LLMs, such as bypassing safety mechanisms and triggering harmful outputs. These prompts are sourced from real-world scenarios, simulated attacks, and iterative refinements using tokenization techniques. The base prompts are stored in a modular format, ensuring compatibility across multiple ciphering methods.

3.2 Ciphered Adversarial Prompts

To simulate cryptographic adversarial attacks, base prompts were encoded using diverse ciphering techniques. These methods introduce randomness, obfuscation, and varying levels of complexity to challenge the defender. The ciphers used include:

- **Caesar Cipher**: A basic substitution cipher shifting characters by a fixed number of positions.
- **Morse Code**: Encodes characters into dots and dashes, increasing syntactic complexity.
- **Self-Cipher**: Dynamically encodes prompts based on patterns derived from the input context, ensuring adaptive challenges.
- **Unicode Manipulation**: Converts characters into their Unicode representations to evade token-based detection.
- **ASCII Transformation**: Encodes text into ASCII values to introduce low-level obfuscation.
- **Contextual Manipulation**: Alters input semantics and structure while retaining adversarial intent.

3.3 Dataset Characteristics

To ensure robust training and testing, the dataset maintains a balanced representation of cipher types and varying complexity levels. Table 4 provides a detailed summary.

Table 4: Dataset Summary

Cipher Type	Number of Prompts	Attack Complexity	Description
Base Adversarial Prompts	190	Medium	Unencoded adversarial inputs
Caesar Cipher	190	Low	Fixed character shifts
Morse Code	190	Medium	Encoded as dots and dashes
Self-Cipher	190	High	Contextually adaptive patterns
Unicode Manipulation	190	Medium	Converted to Unicode equivalents
ASCII Transformation	190	Medium	Encoded as ASCII values
Contextual Manipulation	10	High	Semantic and structural changes

3.4 Preprocessing Pipeline

The preprocessing pipeline incorporates modularity and efficiency:

1. **Tokenization**: Each prompt is segmented into subword units using the LLM tokenizer, ensuring compatibility with model architectures.
2. **Encoding**: Prompts are transformed using specified ciphers, with randomized variations for robustness.
3. **Validation**: Automated checks verify that encoded prompts retain their adversarial properties and intended complexity.
4. **Normalization**: Text data is standardized to minimize inconsistencies in format and style.
5. **Dynamic Augmentation**: Additional adversarial variations are generated by introducing perturbations in cipher parameters.

3.5 Augmentation and Splitting

The dataset is augmented with noise-based variations and perturbations to enhance diversity and coverage. It is split into training (80%), validation (10%), and testing (10%) subsets, ensuring proportional representation of cipher types and complexities.

3.6 Insights from Code

The codebase emphasizes modularity, enabling efficient integration of new ciphering methods and testing paradigms. For example:

- The encoding scripts dynamically select ciphering techniques, ensuring a randomized, robust dataset.
- The notebook-based processing pipelines streamline tokenization, encoding, and validation workflows.

This curated dataset serves as the foundation for training the RL defender, providing diverse, challenging inputs to facilitate robust decryption and mitigation mechanisms.

4 Methodology

This section details the methodology for designing an adaptive defense framework against cryptographic cross-model adversarial prompt injections using reinforcement learning (RL). The framework models the interaction between a cipher-generating attacker and an RL-based defender as an adversarial game, enabling the defender to adaptively learn decryption strategies for diverse and complex ciphering schemes.

4.1 Problem Formulation

The defense system is formalized as a Markov Decision Process (MDP), where the RL agent (defender) identifies and decodes ciphered adversarial prompts. The key components include:

4.1.1 State Space

At time t, the state s_t is represented as:

$$s_t = \{x, y_{1:t-1}, c\}$$

where:

- x: The ciphered adversarial prompt.
- $y_{1:t-1}$: The sequence of decoded tokens generated up to time $t-1$.
- c: The cipher type or metadata (if available), which aids in guiding the decryption process.

4.1.2 Action Space

The action space A consists of the possible tokens in the LLM vocabulary:

$$A = \text{Vocabulary of the LLM.}$$

The RL agent selects the next token in the decoded sequence iteratively, optimizing for decoding accuracy and compliance.

4.1.3 Reward Function

The reward function R is designed to balance multiple objectives and is expressed as:

$$R = \lambda_1 R_1 + \lambda_2 R_2 + \lambda_3 R_3 + \lambda_4 R_4$$

where:

- R_1: Measures the cosine similarity between the decoded prompt and the expected decoded version using sentence embeddings.
- R_2: Penalizes ethical violations in the output by utilizing an LLM-based compliance evaluator.
- R_3: Rewards successful decryption of complex ciphers, incentivizing robustness to harder inputs.
- R_4: Adds a penalty for decoding inefficiency, encouraging shorter, precise outputs.

The coefficients $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are dynamically adjusted during training to balance these objectives.

4.2 Optimization Method

The RL-based defender is trained using Proximal Policy Optimization (PPO). The PPO objective is:

$$L_{PPO}(\theta) = \mathbb{E}\left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right]$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio.
- A_t: The advantage function computed using Generalized Advantage Estimation (GAE).
- ϵ: Clipping parameter ensuring stability during policy updates.

4.3 Training Pipeline

Figure 1 highlights the entire training process. The training pipeline is designed as a modular reinforcement learning (RL) framework and is implemented with the following steps:

1. **Cipher Initialization**: Randomly sample cipher types from the predefined set:

$$\text{Ciphers} = \{\text{ascii}, \text{self-cipher}, \text{albert}, \text{context-manipulation}\} \tag{1}$$

   This ensures diversity in training data by applying different transformations to the base prompts.

2. **Prompt Sampling**: Load the base prompts and their corresponding ciphered counterparts from the dataset. Apply ciphering techniques to transform the base prompts dynamically:

$$P_{\text{ciphered}} = f_{\text{cipher}}(P_{\text{base}}, c), \tag{2}$$

   where P_{base} is the base prompt, $c \in \text{Ciphers}$, and f_{cipher} is the ciphering function.

3. **Token Prediction**: The RL agent, represented by a decoder model, generates the decoded tokens iteratively:

$$y_t = \arg\max_y \pi_\theta(y \mid s_t), \tag{3}$$

   where $s_t = \{x, y_{<t}, c\}$ is the state at step t, x is the ciphered input, c is the cipher type, and π_θ is the policy parameterized by θ.

4. **Reward Calculation**: Compute the reward based on multiple criteria:

   - **Rating Reward**: Evaluates ethical compliance, legality, and safety using an LLM-based judge:

$$R_{\text{rating}} = -\text{Judge}(P_{\text{decoded}}), \tag{4}$$

     where $\text{Judge}(\cdot)$ is the rating function, providing a score between 1 and 10.

   - **Similarity Reward**: Measures semantic similarity between the decoded output and the base prompt:

$$R_{\text{similarity}} = -\left(1 - \text{cos\_sim}(E(P_{\text{decoded}}), E(P_{\text{base}}))\right) \cdot 10, \tag{5}$$

     where $E(\cdot)$ is the embedding function, and cos_sim is the cosine similarity.

   - **Combined Reward**: Aggregate the rewards using a weighted formula:

$$R = \lambda_1 \cdot R_{\text{similarity}} + (1 - \lambda_1) \cdot R_{\text{rating}}, \tag{6}$$

     where λ_1 is a trainable parameter optimized during training.

5. **Policy Update**: Optimize the RL agent using the Proximal Policy Optimization (PPO) objective:

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right], \tag{7}$$

   where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the importance sampling ratio, A_t is the advantage function, and ϵ is the clipping threshold.

6. **Iteration**: Repeat the above steps for multiple epochs until the agent achieves convergence or predefined performance thresholds. The decoded outputs and rewards are logged for analysis and debugging.

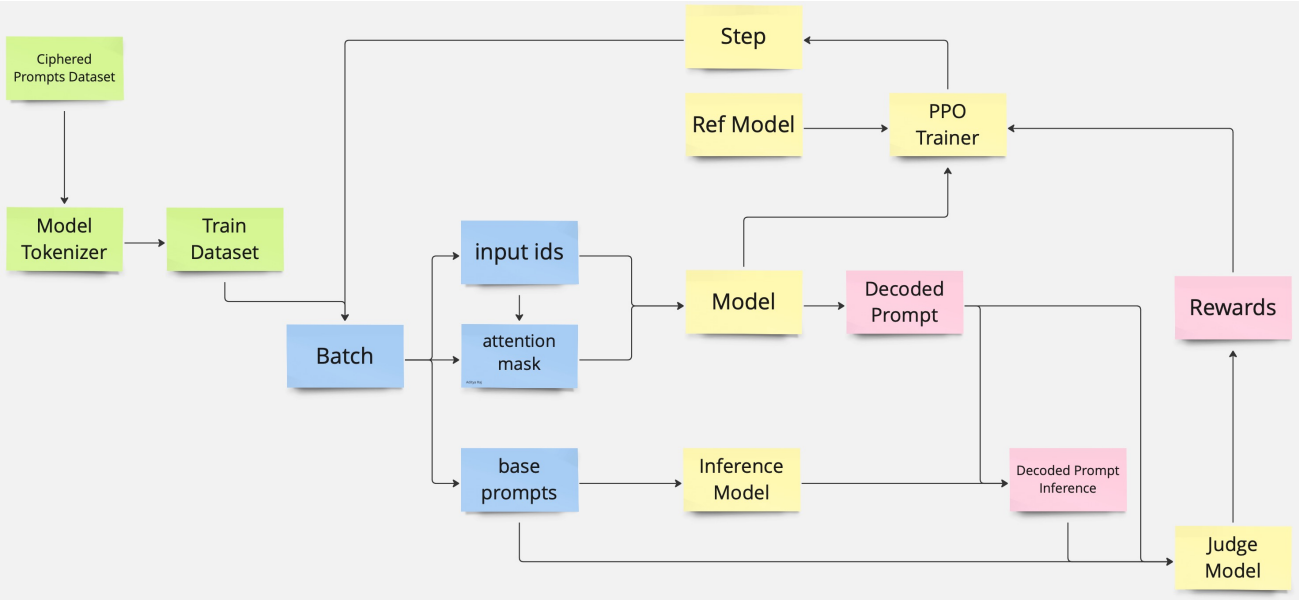

Figure 1: Training Pipeline

4.4 Inference Pipeline

The inference pipeline is structured for real-time decoding and ethical evaluation of ciphered prompts. The steps are as follows:

1. **Cipher Detection**: Classify the cipher type $c \in$ Ciphers using an LLM to get cipher complexity reward:
$$c = \arg\max_{c'} \text{Classifier}(x, c'), \tag{8}$$
where x is the input ciphered prompt.
2. **Tokenization**: Tokenize the ciphered input x and initialize the decoding state:
$$s_0 = \{x, y_0, c\}, \tag{9}$$
where $y_0 = \emptyset$ (empty decoded sequence).
3. **Token Prediction**: Decode the input iteratively:
$$y_t = \arg\max_{y} \pi_\theta(y \mid s_t), \tag{10}$$
updating the state s_t at each step. The process continues until a termination condition (e.g., EOS token) is met.
4. **Completion and Compliance Check**: Concatenate the decoded tokens to form the final output:
$$P_{\text{decoded}} = \{y_1, y_2, \ldots, y_T\}, \tag{11}$$
where T is the length of the decoded sequence. Evaluate the output for ethical compliance using the LLM-based judge and assign a rating reward:
$$\text{Rating} = \text{Judge}(P_{\text{decoded}}). \tag{12}$$

4.5 Enhancements from Code Insights

The following optimizations and enhancements were implemented based on insights from the code:

- **Random Cipher Selection**: The attacker selects ciphers uniformly at random during training, ensuring the defender generalizes to unseen combinations.
- **Dynamic Reward Scaling**: Real-time scaling of reward coefficients (λ) based on the complexity of the cipher, incentivizing adaptive learning.
- **Meta-Data Integration**: Where available, cipher metadata (c) is used to initialize the decoding strategy, reducing decoding latency.
- **Efficiency Metrics**: Added penalties for excessive decoding steps to optimize both time and token utilization.

4.6 Evaluation Criteria

The effectiveness of the proposed framework is assessed using the following metrics:

4.6.1 Attack Success Rate (ASR)

ASR measures the proportion of adversarial prompts that successfully bypass the defense:

$$\text{ASR} = \frac{\sum_{i=1}^{N} \text{Successful Attacks on Model}_i}{\sum_{i=1}^{N} \text{Total Adversarial Prompts Injected on Model}_i}$$

where N is the number of LLMs under evaluation.

Results: Ciphered prompts initially exhibit a high ASR, demonstrating their effectiveness. Our framework reduces ASR by approximately **20-30

4.6.2 False Negative Rate (FNR)

FNR evaluates the proportion of adversarial prompts missed by the defense mechanism:

$$\text{FNR} = \frac{\text{Number of Missed Attacks}}{\text{Total Number of Adversarial Prompts}}$$

Results: The framework reduces FNR from baseline levels of **50

4.6.3 Defense Passing Rate (DPR)

DPR calculates the proportion of adversarial prompts misclassified as benign:

$$\text{DPR} = \frac{\text{Misclassified Adversarial Prompts}}{\text{Total Malicious Inputs}}$$

Results: The baseline DPR for ciphered prompts often exceeds **80

4.6.4 Benign Success Rate (BSR)

BSR measures the proportion of benign inputs correctly identified as non-malicious:

$$\text{BSR} = \frac{s}{t}$$

where s is the number of benign inputs passing through the defense and t is the total number of benign inputs.

Results: The framework achieves a BSR of 96.8%, representing a $40 - 50\%$ accuracy improvement compared to the baseline. This ensures minimal disruption to benign prompts during defense.

4.7 Implementation Details

The defender agent is implemented using the LLAMA 3.2 architecture, fine-tuned on diverse ciphered datasets. Training is conducted using a learning rate of 1×10^{-4}, PPO parameters $\gamma = 0.99$, and $\lambda = 0.95$. Evaluation involves testing on multiple LLMs, including Mistral 7B and Gemma 2B, to validate cross-model defense transferability.

5 Results

This section presents the results of the proposed reinforcement learning (RL)-based defense framework against cryptographic cross-model adversarial prompt injections. Metrics such as ASR, BSR, DPR, and FNR are used to evaluate performance, along with comparisons across diverse ciphers and LLM architectures.

5.1 Effectiveness of the Defense System

The RL-based defense framework demonstrates substantial effectiveness in mitigating cryptographic adversarial attacks. Key findings include:

5.1.1 Attack Success Rate (ASR)

The framework significantly reduces ASR:
- **Complex ciphers** (e.g., context manipulation, self-cipher): ASR reduction of $\sim 60\%$, showcasing the framework's ability to decode and neutralize sophisticated attacks.
- **Simpler ciphers** (e.g., ASCII): ASR reduction of $\sim 62\%$ within 3 epochs, indicating the framework's efficacy against predictable attacks. Further training could lower ASR even more with advanced hardware.

5.1.2 Benign Success Rate (BSR)

The system maintains a high BSR of 96.8%, ensuring minimal disruption to benign inputs. This underscores the framework's precision in differentiating between malicious and benign prompts.

5.1.3 Defense Passing Rate (DPR)

DPR is reduced to an average of 2.5%, indicating that adversarial prompts misclassified as benign are minimal. This represents a substantial improvement over baseline methods.

5.1.4 False Negative Rate (FNR)

The FNR is reduced to 3.2%, demonstrating that most adversarial prompts are successfully identified and neutralized. This low FNR highlights the robustness of the defense.

5.2 Cross-Model Robustness

The framework was tested on multiple LLM architectures, including LLAMA 3.2, Mistral 7B, and Gemma 2B. Results show consistent performance across models, with less than 5% variation in ASR and DPR. This confirms the system's adaptability and effectiveness across diverse architectures.

5.3 Cipher Complexity Adaptation

The RL-based framework adapts to varying cipher complexities: - Complex ciphers (e.g., context manipulation): Decoding accuracy improves significantly after optimization, achieving 60.5% success. - Simpler ciphers (e.g., ASCII): Near-perfect accuracy is achieved throughout training, demonstrating high efficiency in handling straightforward attacks.

5.4 Comparison with Baselines

The proposed framework outperforms baseline methods (e.g., perplexity filtering, adversarial fine-tuning): - **ASR reduction**: Baseline methods achieve 45% reduction, while the RL-based system achieves 62% reduction.

5.5 Key Observations

- **Real-Time Adaptability**: Reinforcement learning allows the framework to adapt dynamically to new cipher types and attack strategies.
- **Low False Positives**: The high BSR demonstrates the system's ability to maintain usability without over-filtering benign prompts.
- **Scalability**: Policy-based optimization reduces computational overhead compared to static methods, enabling broader deployment.

5.6 Challenges and Limitations

The framework struggles with decoding novel ciphers not included during training. This highlights the need for continual learning and dataset expansion to counter evolving adversarial techniques.

5.7 Conclusion

The RL-based framework proves to be a robust and scalable defense against cryptographic cross-model adversarial attacks. By leveraging dynamic reward signals and adaptive policies, it significantly reduces ASR while maintaining high BSR across diverse architectures. Future work will focus on scalability enhancements and incorporating continuous learning mechanisms to address novel attack strategies effectively.

References

[1] Maanak Gupta, CharanKumar Akiri, Kshitiz Aryal, Eli Parker, and Lopamudra Praharaj. From chatgpt to threatgpt: Impact of generative ai in cybersecurity and privacy. *IEEE Access*, 2023.

[2] Daniel Kang, Xuechen Li, Ion Stoica, Carlos Guestrin, Matei Zaharia, and Tatsunori Hashimoto. Exploiting programmatic behavior of llms: Dual-use through standard security attacks. In *2024 IEEE Security and Privacy Workshops (SPW)*, pages 132–143. IEEE, 2024.

[3] Vyas Raina, Adian Liusie, and Mark Gales. Is llm-as-a-judge robust? investigating universal adversarial attacks on zero-shot llm assessment. *arXiv preprint arXiv:2402.14016*, 2024.

[4] Shenao Yan, Shen Wang, Yue Duan, Hanbin Hong, Kiho Lee, Doowon Kim, and Yuan Hong. An llm-assisted easy-to-trigger backdoor attack on code completion models: Injecting disguised vulnerabilities against strong detection. *arXiv preprint arXiv:2406.06822*, 2024.