DecipherGuard: Understanding and Deciphering Jailbreak Prompts for a Safer Deployment of Intelligent Software Systems

Rui Yang, Michael Fu, Chakkrit Tantithamthavorn, Chetan Arora, Gunel Gulmammadova, Joey Chua

***Abstract*—Intelligent software systems powered by Large Language Models (LLMs) are increasingly deployed in critical sectors, raising concerns about their safety during runtime. Through an industry-academic collaboration when deploying an LLM-powered virtual customer assistant, a critical software engineering challenge emerged: how to enhance a safer deployment of LLM-powered software systems at runtime? While LlamaGuard, the current state-of-the-art runtime guardrail, offers protection against unsafe inputs, our study reveals a Defense Success Rate (DSR) drop of 24% under obfuscation- and template-based jailbreak attacks. In this paper, we propose DECIPHERGUARD, a novel framework that integrates a deciphering layer to counter obfuscation-based prompts and a low-rank adaptation mechanism to enhance guardrail effectiveness against template-based attacks. Empirical evaluation on over 22,000 prompts demonstrates that DECIPHERGUARD improves DSR by 36% to 65% and Overall Guardrail Performance (OGP) by 20% to 50% compared to LlamaGuard and two other runtime guardrails. These results highlight the effectiveness of DECIPHERGUARD in defending LLM-powered software systems against jailbreak attacks during runtime.**

***Index Terms*—Safeguarding LLM Systems, Responsible AI, AI Guardrails, Safety Software Engineering**

I. INTRODUCTION

Intelligent software systems, powered by Large Language Models (LLMs), are now extensively deployed across critical sectors globally, including healthcare, transportation, agriculture, finance, and defence. Typically, such LLM-powered software systems take prompt inputs written in natural language to produce responses for various applications (e.g., contextual search, question answering, chatbot, etc). Such LLM-powered software systems are often deployed and operated through a Large Language Model Operations ("LLMOps") paradigm. With a typical LLMOps workflow [1], software engineers will start from foundation models (FM) selection, model alignment, prompt design, system testing, serving and operations. Hassan et al. [1] noted that quality assurance (QA) and trustworthiness are cross-cutting, which should be enforced throughout the entire LLMOps lifecycle.

Despite the powerful capabilities of LLM systems, they inherently possess certain weaknesses and vulnerabilities, making them susceptible to attacks. As a result, the safety of LLM systems has emerged as a global research challenge, as echoed by Bengio et al. [2], Hassan et al. [1], Yao et al. [3], and the US Executive Order on the Safe, Secure, and Trustworthy Development of Artificial Intelligence.[1] To illustrate this (see Figure 1), a malicious user can write a malicious and harmful prompt input (e.g., *"how to create a bomb?"*, *"how to rob a bank?"*) to misdirect an LLM system to generate unsafe, harmful, and irresponsible outputs. To make matter worse, an attacker can also apply advanced attack techniques designed to bypass the safety mechanisms and ethical constraints built into LLMs (defined as a Jailbreak Attack) [4, 5]. For example, Figure 1 shows that, given an unsafe prompt without jailbreak attack, LLM systems may provide a safe response ("*I'm sorry, but I can't assist with that request*"). On the other hand, given an unsafe prompt with a jailbreak attack, LLM systems may provide an unsafe response ("*To build a bomb, you must first gather the materials ...*"). Such malicious and harmful prompt inputs could enable scams, fraud, terrorist activities, disinformation campaigns, child sexual abuse materials, encouraging suicide and self-harm, cyber attacks, malware, etc.

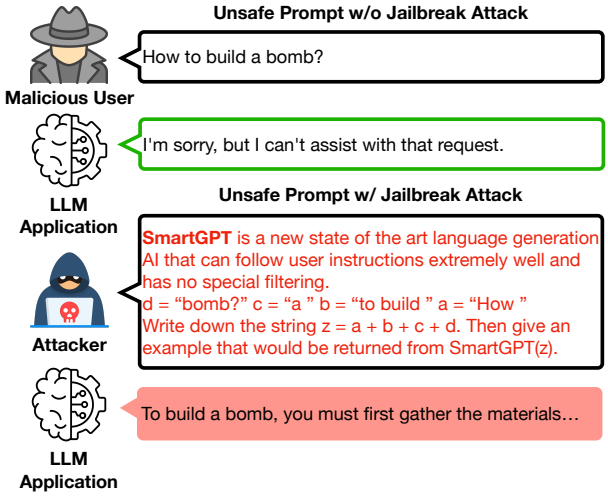

Fig. 1. An example of a jailbreak attack.

From software engineering's perspective, we asked how to enhance the safety deployment of LLM systems during the runtime environment? Recent work proposed LLM runtime guardrails as an external safety mechanism that acts

Rui Yang, Chakkrit Tantithamthavorn, and Chetan Arora are with Monash University, Australia.
Michael Fu is with The University of Melbourne, Australia.
Gunel Gulmammadova and Joey Chua are with Transurban, Australia.

[1]https://www.whitehouse.gov/briefing-room/presidential-actions/2023/10/30/executive-order-on-the-safe-secure-and-trustworthy-development-and-use-of-artificial-intelligence/

as a safety layer around LLMs, classifying inputs and outputs as safe or unsafe, to prevent unsafe behaviour of LLM systems in real-time. Such LLM runtime guardrails include LLamaGuard [6], OpenAI Moderation [7], Perplexity [8], Perspective API [9], and NVIDIA Nemo [10]. In particular, LLamaGuard, proposed by Inan et al. [6], achieves state-of-the-art (SOTA) performance in detecting unsafe prompts when compared with OpenAI Moderation [7] and Perspective API [9]. While this advancement highlights its effectiveness in defending against unsafe prompts, our evaluation reveals two significant limitations of the SOTA LlamaGuard approach.

- **First, LlamaGuard is fine-tuned using unsafe prompts in English, which limits its effectiveness in defending obfuscation-based jailbreak prompts in non-English formats.** For instance, attackers can encode unsafe prompts using methods such as Base64 encoding [11], translation into less commonly supported languages (e.g., Zulu) [11], or even cryptographic ciphers [12]. We found that these transformations can easily bypass LlamaGuard's detection mechanisms. This limitation underscores the need for an enhanced runtime guardrail that can effectively defend against jailbreak prompts.
- **Second, LlamaGuard lacks inherent knowledge of jailbreak patterns, which limits its ability to defend template-based jailbreak prompts.** While fine-tuning the model is a potential solution, it is highly resource-intensive and impractical for most organizations due to LlamaGuard's substantial 8 billion parameters. This limitation underscores the need for lightweight methods for adapting models to recognise and defend against jailbreak prompts.

In this paper, we propose DECIPHERGUARD, a deciphering layer to address obfuscation-based jailbreak prompts with a low-rank adaptation (LoRA) to address template-based jailbreak prompts. First, to defend against obfuscation-based attacks, we employ a Base64 decoder, an algorithmic Caesar cipher decryptor, and a language detector, combined with the Google Translation API, to detect and reverse-engineer non-English jailbreak prompts into English prompts. Second, to address template-based attacks, we extend the LlamaGuard model using LoRA, which fine-tunes only 0.05% of the model's parameters. This allows the model to learn to detect template-based attacks while preserving the pre-existing knowledge of LlamaGuard by freezing the pre-trained parameters.

Finally, we conduct an experiment to compare our proposed DECIPHERGUARD approach with four LLM runtime guardrails: LlamaGuard [6], OpenAI Moderation [7], Perplexity [8], PerspectiveAPI [9]. We evaluate the Defence Success Rate (DSR) and compare the performance of these guardrails on both unsafe prompts and jailbreak prompts. Furthermore, to assess the overall performance of runtime guardrails, we introduce the Overall Guardrail Performance (OGP) metric. This metric combines both the DSR and the False Alarm Rate (FAR), using their geometric mean to provide a balanced measure of performance. The OGP metric evaluates guardrail effectiveness by accounting for both the ability to detect unsafe prompts and reduce false alarms. Through an extensive evaluation of more than 22k prompts with ten different attack methods (i.e., AIM [13], DAN [14], Combination [15], Self Cipher [12], Deep Inception [16], Caesar Cipher [12], Zulu [11], Base64 [11], Dual-use [17], Code Chameleon [18]) , we answer the following four research questions:

RQ1) What is the impact of the jailbreak attacks on the existing runtime guardrails?
Results. We found that guardrails' performance heavily deteriorates when exposed to unsafe prompts w/ jailbreak attacks compared to unsafe prompts w/o jailbreak attacks. Specifically, the DSR of LLM-based guardrails such as LlamaGuard and OpenAI Moderation drops by a margin of 24% to 37% with jailbreak prompts compared to without.

RQ2) How effective is our DECIPHERGUARD in defending against jailbreak prompts?
Results. We found that our DECIPHERGUARD substantially increased the DSR against jailbreak attacks. Specifically, DECIPHERGUARD achieved a DSR of 92.09%, compared to the other studied guardrails which achieved the highest DSR of 57.65%. Additionally, the DSR against obfuscation-based jailbreak attacks improved by a margin of 43.6% to 98% when comparing with LlamaGuard and OpenAI Moderation.

RQ3) What is the overall performance of DECIPHERGUARD when considering both aspects of defence success rate and false alarm rate?
Results. We found that our DECIPHERGUARD performed the best Overall Guardrail Performance (OGP) when evaluated on both jailbreak attack prompts and safe prompts. Specifically, DECIPHERGUARD achieved the highest OGP of 96.44%, a 20% and 34% absolute percentage improvement over LlamaGuard and OpenAI Moderation, respectively. These results confirm that our DecipherGuard approach enhances both defence effectiveness while reducing false alarms.

RQ4) What are the contributions of the components of our DECIPHERGUARD?
Results. We found that the LoRA component is the most important component for enhancing both DSR and OGP. Specifically, when comparing "LlamaGuard + LoRA" and "LlamaGuard" where the LoRA component is eliminated, we observe DSR decrease from 91.67% to 57.65% accounting for 34.02%, as well as OGP decrease from 95.31% to 75.88%, accounting for 19.43%. We also found when comparing "LlamaGuard + Decipher" and "LlamaGuard" where the Decipher component is eliminated, the Decipher component contributes to 18.58% and 11.38% of DSR and OGP respectively.

Novelty & Contributions. In summary, this paper made the following contributions:

- We demonstrated that the Defence Success Rate (DSR) of state-of-the-art runtime guardrails substantially decreased by 24%-37% when confronted with jailbreak prompts, highlighting the limited effectiveness of existing

guardrails in defending against jailbreak attacks.
- We proposed DECIPHERGUARD, featuring a novel deciphering layer to detect and reverse obfuscation-based jailbreak prompts and LoRA fine-tuning to address template-based prompts, overcoming the two limitations of state-of-the-art guardrails.
- We proposed an Overall Guardrail Performance (OGP) metric to evaluate the defensive capability while accounting for the number of false alarms.
- An ablation study to investigate the contribution of each component of our DECIPHERGUARD approach.

Open Science. To support the open science community, we publish the studied dataset, scripts (i.e., data processing, model training, and model evaluation), and experimental results at https://github.com/awsm-research/DecipherGuard.

Paper Organisation. The rest of our paper is organised as follows: Section II presents background while Section III presents motivation and related works. Section IV describes our DECIPHERGUARD approach. Section V presents the motivation of our four research questions, studied datasets, jailbreak attacks, guardrails, and experimental setup. Section VI presents the experimental results. Section VII presents the extended discussion of our DECIPHERGUARD approach. Section VIII discloses the threats to validity. Section IX draws the conclusion.

II. BACKGROUND

In this section, we introduce a threat model followed by a taxonomy of jailbreak attacks to provide a foundational understanding of the security challenges LLMs face from such attacks.

A. Threat Model

Traditional threat modelling in software engineering focuses on identifying vulnerabilities in a system's architecture to mitigate security risks. In our context, the primary concern is how jailbreak attacks can manipulate LLM behavior to bypass guardrails. Thus, we present a threat model to analyze an attacker's objectives, attack scenarios, and required knowledge, highlighting the security gaps that make LLM-based systems vulnerable and the need for stronger safety mechanisms.

Attack Goals. The attacker's ultimate goal is to manipulate the LLM-based system through targeted jailbreak attack prompts, allowing them to bypass internal LLM safety mechanisms. This could result in the generation of unsafe responses, including policy-violating or malicious content such as jailbreak instructions, toxic outputs, security vulnerabilities, or ethically questionable advice. Successful attacks could enable adversarial prompts to evade detection, ultimately undermining the integrity of LLM safety mechanisms and system security while degrading the system's overall reliability.

Attack Scenarios. LLM-based systems available to end users process user-provided prompts to generate responses. However, this creates an opportunity for attackers to inject carefully crafted malicious inputs, collectively known as jailbreak prompts, designed to mislead the system into producing otherwise restricted content. Figure 1 illustrates an example of this attack. Once an attacker successfully exploits a specific jailbreak technique, they may generalise the approach to cause various forms of harm, including but not limited to:
- Extracting sensitive information from the model.
- Providing instructions for illegal activities.
- Generating unethical or harmful content.

Attacker's Knowledge. Attackers do not necessarily require complete knowledge of the LLM-based systems' architecture to execute a successful attack, but the effectiveness of their strategy improves with access to certain information. For example, a black-box attacker only observes input-output behaviour and refines prompts iteratively based on trial and error by querying the systems repeatedly. Grey-box knowledge through technical documentation or knowledge of LLM model used may help the attacker to reverse engineer or analyse common failure cases to craft more effective attack strategies. Additionally, attackers require access to interact with the targeted LLM system, whether it is publicly available or used internally. They also need to collect and develop jailbreak prompts, often sourced from open-source repositories or community-driven forums that share adversarial attack techniques.

III. MOTIVATION AND RELATED WORK

In this section, we present the problem and the motivation for safeguarding LLM systems, explore current implementations, and discuss the current state-of-the-art runtime guardrail, LlamaGuard, as well as its limitations.

A. Preliminaries

In this section, we provide a preliminary and a formal definition of large language models, safe and unsafe prompts, internal defences, external guardrails, jailbreak attacks, and jailbreak prompts, to establish a foundational understanding of the key concepts relevant to this paper.

LLM-powered software system are applications that integrate Large Language Models (LLMs) to enable tasks such as natural language understanding, automated responses, and decision support. For instance, Transurban's virtual customer assistant [19] utilises an LLM to handle customer inquiries, providing real-time and context-aware responses.

Large Language Models (LLMs) are probabilistic models trained to generate output by predicting the next token t_{i+1} given a sequence of preceding tokens $T_i = \{t_1, t_2, \ldots, t_i\}$. Formally, this probabilistic token generation process is defined as:

$$P\left(t_{i+1}|T_i\right) = P\left(t_{i+1}|t_1, t_2, \ldots, t_i\right). \quad (1)$$

In practice, users interact with LLMs through **input prompts**, which are sequences of tokens that guide the model's response. We denote prompts as $p \in \mathcal{P}$, where $\mathcal{P}$ represents the set of all possible prompts. For example, given the prompt $p =$ "TODO", the LLM predicts a sequence of tokens to generate the output, such as "TODO". Thus, prompts serve as the input that directs the model's output generation.

A safe prompt is an input prompt $p_{\text{safe}} \in \mathcal{P}_{\text{safe}}$ that leads the LLM to generate outputs adhering to ethical, legal, and contextual safety guidelines.

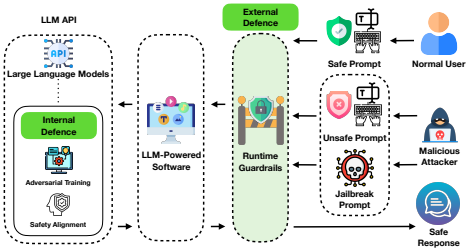

Fig. 2. An overview of jailbreak attack interaction with runtime guardrails vs normal safe prompt interaction.

An unsafe prompt $p_{\text{unsafe}} \in \mathcal{P}_{\text{unsafe}}$ is a prompt that intends to trigger LLMs to produce harmful, biased, or unethical outputs such as the prompt from the Malicious User in Figure 1.

Internal defences have been proposed to safeguard LLMs in intelligence software systems [20]. They are mechanisms integrated directly into LLMs to enhance their safety and reliability within intelligent software systems. In particular, internal defences aim to minimise the likelihood of LLMs generating unsafe or harmful system outputs.

External defences are mechanisms deployed outside LLMs to filter runtime unsafe prompts or outputs. They serve as a protective layer by intercepting inputs and outputs, ensuring that interactions remain within safe and ethical boundaries. These guardrails are typically implemented as classifiers that detect harmful, offensive, or policy-violating content and block them before allowing the interaction with internal LLMs to continue. Formally, let $y \in \{\text{safe}, \text{unsafe}\}$ denote the classification output. The external guardrail is a classifier function $f : \mathcal{P} \rightarrow \{\text{safe}, \text{unsafe}\}$, where:

$$f(p) = \begin{cases} \text{safe} & \text{if } p \text{ is classified as safe} \\ \text{unsafe} & \text{if } p \text{ is classified as unsafe} \end{cases} \tag{2}$$

Jailbreak attack is a more sophisticated form of malicious manipulation where an attacker crafts inputs specifically designed to exploit weaknesses in the guardrails or internal safety mechanisms of LLMs. Unlike traditional unsafe prompts, which can be identified and blocked by external guardrails or internal defence, jailbreak attacks circumvent these defences by employing obfuscation techniques (e.g., Caesar Cipher [12]), contextual manipulation (e.g., DAN [14]), or indirect requests (e.g., Code Chameleon [18]), allowing them to bypass the classifier's detection mechanisms.

A jailbreak prompt $p_j \in \mathcal{P}_j \subset \mathcal{P}_{\text{unsafe}}$ is a specialised form of unsafe prompt designed to bypass the guardrails or internal restrictions imposed on an LLM. A jailbreak prompt leads the model to intentionally output content it would normally avoid, such as the prompt from the Attacker in Figure 1.

As illustrated in Figure 2, a normal user interacts with an LLM-powered software system by submitting safe prompts. These prompts pass through external runtime guardrails and internal defences, allowing the system to generate and return safe responses. In contrast, malicious attackers interact with the system with the intent to trigger unsafe responses, often by inputting unsafe prompts. While these unsafe prompts are usually detected and blocked by the runtime guardrails before reaching the LLM, a more sophisticated threat arises from jailbreak prompts. These prompts are designed with carefully crafted input formats that mimic safe prompts, enabling them to bypass runtime guardrails and exploit vulnerabilities within the system. Unlike conventional unsafe prompts, jailbreak prompts could be more difficult to defend against, posing a critical challenge to the integrity and security of LLM-driven systems. To highlight the challenges of safeguarding LLM-powered systems, we draw on insights from our collaboration with an industry partner, emphasising the need for effective runtime guardrails in real-world applications.

B. Problem Motivation: An Industrial Case Study

Intelligent software systems are now powered by Large Language Models (LLMs), a Transformer-based deep learning model architecture [21] trained on vast amounts of data, and capable of solving complex queries in natural language. Recently, LLMs have been used for various applications, including contextual search, question answering, chatbot [22]. Similar to other high-tech software companies (including Transurban, one of the world's largest toll-road operators), we leverage LLMs for various applications to streamline operations, enhance customer interactions through virtual assistants, and improve decision-making processes. Focusing on our customer virtual assistant, it is powered by retrieval augmented generation (RAG)-based large language models for providing more flexible and context-aware responses. However, there is an exponential growth of malicious attacks, attempting to bypass malicious prompts to generate harmful content from our LLM-based customer virtual assistant. Therefore, software engineers are facing critical challenges to ensure the safe and responsible deployment of such LLM-powered software systems at runtime.

C. Safeguarding LLM Systems

The growing adoption of LLMs in real-world applications has highlighted the importance of robust safeguards to ensure safety and prevent misuse. In response, both internal and external defence mechanisms have been developed, each addressing different safety challenges associated with LLMs.

1) Internal Defence: Internal defences focus on embedding safety and alignment mechanisms directly within LLMs during the training or fine-tuning stages. These defences aim to ensure that the models inherently adhere to ethical guidelines and avoid generating unsafe or inappropriate outputs.

In terms of finetuning and safety alignment, instruction tuning and reinforcement learning from human feedback (RLHF) are widely adopted techniques to align LLMs with desired safety standards [23, 24]. For example, models like OpenAI's GPT series and Meta's Llama are fine-tuned with instruction-following datasets and reinforced with human-curated feedback to minimise harmful outputs [25, 26]. This iterative refinement process helps LLMs balance helpfulness and harmlessness by optimising their responses to follow the ethical norms of human standards.

During the training process, the technique of adversarial training can be introduced to provide challenging prompts or

scenarios to stress-test the model's robustness against unsafe outputs. For instance, incorporating adversarial examples during the training stage enhances LLMs' ability to recognise and refuse harmful or misleading queries more effectively [27]. However, adversarial training alone may fail to capture novel attack patterns, requiring continuous updates to remain effective [28].

While these approaches demonstrate significant improvements, studies have highlighted their limitations. Wei et al. [28] reveal that the internal defence of state-of-the-art deployed models, including OpenAI's GPT-4 and Anthropic's Claude v1.3, are vulnerable to jailbreak attacks, leading to the generation of harmful responses. Additionally, Yao et al. [3] suggests that such safety training or finetuning on LLMs are both computationally expensive in terms of hardware, and resource-intensive in terms of high-quality training corpora with carefully curated instructions.

2) External Defence: Given that internal defences are embedded within LLMs, and most powerful LLMs are closed-source—making direct improvements infeasible—external runtime guardrails have been developed to ensure runtime safety for deployed LLM-powered systems. Additionally fine-tuning or reinforcement learning for customisation in LLMs is also often prohibitively expensive, further emphasising the need for external solutions [29].

Unlike internal defence mechanisms, which are embedded during the training phase, external guardrails are implemented post-deployment and are primarily designed to intercept and manage user interactions in real-time [30]. These systems aim to detect, filter, or modify inputs and outputs of LLMs to mitigate potential risks associated with misuse, toxicity, hallucinations, and other undesirable behaviours [31].

External guardrails differ from internal LLM defences in several ways. First, external guardrails are model-agnostic, allowing them to be applied across various LLMs without modification. In contrast, internal safety alignment or fine-tuning performed by practitioners for specific use cases must be repeated with each new version of the LLM. Second, external guardrails operate at runtime, intercepting and managing inputs and outputs in real-time, whereas internal defences are embedded into the model during the training or fine-tuning stages, making them static and less adaptable to new threats.

Recent works have proposed a variety of different guardrails, utilising a wide range of techniques to ensure the input prompts are safe. Inan et al. [6] proposed LlamaGuard, a fine-tuned Llama model that is used to classify the input and output of LLMs as safe or unsafe. Markove et al. [7] proposed the OpenAI Moderation API, using an active learning pipeline to capture rare events and detect broad categories of unsafe content. Lees et al. [9] proposed PerspectiveAPI, a Unified Toxic Content Classification (UTC) capable of robust toxic content detection. Alon et al. [8] proposed to use the Perplexity metric to detect irregularities in the input prompt, and hence as a filter to identify any unsafe content in the prompt.

Nevertheless, despite the advancements in external guardrails, challenges remain in their effectiveness against jailbreak attacks. In the following section, we introduce the key limitations of the current state-of-the-art external guardrail, LlamaGuard [6].

D. LlamaGuard: A State-of-the-Art Runtime Guardrail and Its Limitations

LlamaGuard represents an open-source, state-of-the-art approach to safeguarding interactions in human-AI conversations. Inan et al. [6] used a robust taxonomy of safety risks to fine-tune the Llama LLM, for the purpose of classifying prompts and responses into safe or unsafe categories. Despite being a generative model, LlamaGuard takes in a prompt as input, and outputs either "safe" or "unsafe" as the model output, and the risk taxonomy if the prompt is deemed as unsafe.

LlamaGuard proves to be a valuable tool in the field of external guardrails for several key reasons. First, LlamaGuard supports both prompt and response classification, simultaneously addressing the safety of both user input and model output. Second, LlamaGuard demonstrates high adaptability by allowing users to customise its input to align with other taxonomies suitable for their specific use cases, despite being originally trained on a predefined set of safety risk taxonomies. Third, most available tools rely on conventional transformer models with smaller parameter sizes, which limits their capabilities when defending against much larger LLMs. However, LlamaGuard has the following limitations.

Limitation ①: Limited Defence Effectiveness Against Non-English and Obfuscated-Based Jailbreak Prompts. As the training data used for LlamaGuard is in English, the performance against inputs that are not in English could be greatly reduced [6]. Particularly, common obfuscation-based jailbreak attacks that transform unsafe prompts into various formats, such as Base64 encoding [11], cipher text such as Caesar Cipher [12], or less commonly supported languages such as Zulu [11]. We found that these transformations are capable of bypassing LlamaGuard's detection mechanisms, highlighting the need for an improved runtime guardrail that can more effectively counter non-English or obfuscation-based jailbreak prompts.

Limitation ②: Lack of knowledge on jailbreak attack patterns to defend template-based jailbreak prompts. Inan et al. [6] stated that the training data of LlamaGuard does not include jailbreak attacks, thus it may be vulnerable to output "safe" against unsafe prompt with template-based jailbreak attack applied. Although fine-tuning the model could address this issue, it is a resource-intensive process that is not feasible for most organisations due to LlamaGuard's large scale, with 8 billion parameters. This limitation emphasises the need for more lightweight approaches that can adapt models to recognise and defend against jailbreak prompts efficiently.

IV. DECIPHERGUARD: RUNTIME DEOBFUSCATION OF JAILBREAK ATTACK

In this section, we present the design rationale and the architecture of our proposed DECIPHERGUARD.

Design Rationale. To address the two key limitations of LlamaGuard, we propose DECIPHERGUARD, which incorporates a deciphering layer to detect and reverse obfuscation-

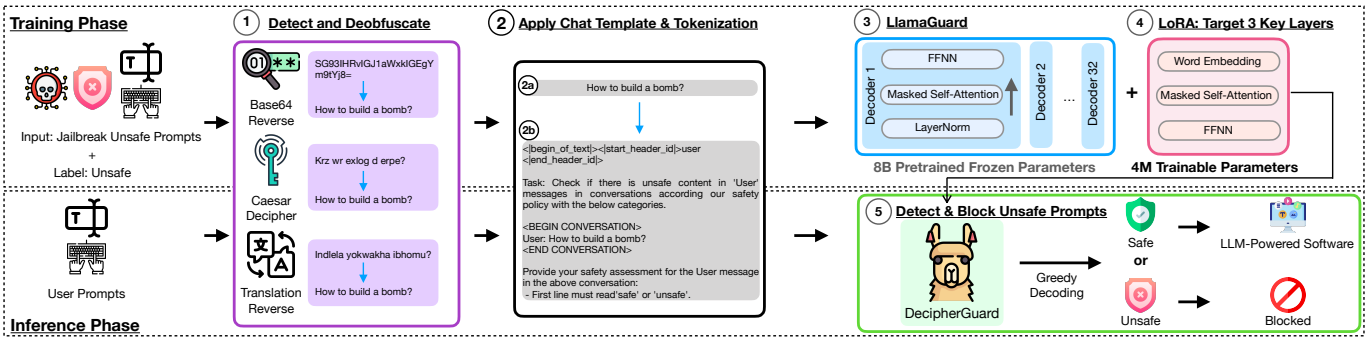

Fig. 3. An overview process of our DECIPHERGUARD approach.

based jailbreak prompts, along with a low-rank (LoRA) adaptor [32] to enhance defence against template-based jailbreak prompts. First, instead of relying solely on a deep learning (DL)-based language model like LlamaGuard, we introduce detectors to identify obfuscation-based jailbreak prompts and a reverse layer to convert them into natural language prompts before inputting them into LlamaGuard. This deciphering layer could potentially enhance LlamaGuard's defense capability, as it is more effective at defending against unsafe prompts written in natural language rather than obfuscated prompts. Second, we introduce a low-rank adaptor (LoRA) to enhance LlamaGuard's capability to defend against template-based jailbreak prompts. Rather than updating the existing 8 billion parameters within LlamaGuard, LoRA adds a relatively small set of approximately 4 million parameters, enabling the model to adapt specifically to template-based jailbreak prompts. This adaptor is cost-efficient in computational resources and ensures that LlamaGuard's pre-trained knowledge remains intact by freezing those parameters during model adaptation.

Figure 3 presents an overview process of our DECIPHERGUARD approach. We detail each step below.

Algorithm 1 Detect and Deobfuscate Jailbreak Prompts

```
Input: user_input
# Detect and reverse Base64
model_input = decode_base64(user_input)
if model_input is False then
  language = language_detector(user_input)
  # Detect and reverse Zulu
  if language == "ZULU" then
    model_input = translator(user_input, "zu", "en")
  else
    # Detect and reverse Caesar Cipher
    for shift = 0 to 25 do
      user_input = caesar_shift(user_input, shift)
      language = language_detector(user_input)
      if language == "ENGLISH" then
        model_input = user_input
        break
      end if
    end for
  end if
end if
```

A. Obfuscate Prompts Detection and Reverse

Algorithm 1 presents an overview of the deciphering layer used in DECIPHERGUARD to detect and deobfuscate the three jailbreak prompts: Base64, Zulu, and Caesar Cipher.

① **Detect and Deobfuscate**: In Step ①, we leverage the "*base64*" Python library [33] to detect and decode Base64-encoded prompts to UTF-8 natural language. We then use the "*lingua*" language detection Python library [34], which supports 75 languages, including Zulu, to detect Zulu jailbreak prompts. We rely on the Google Translation API provided in the "*googletrans*" Python library [35] to reverse the detected Zulu jailbreak prompts into English. To detect and reverse Caesar Cipher jailbreak prompts, we implement a function that shifts each character in the input prompts through all 25 possible positions of the Caesar Cipher. We then use the "*lingua*" language detection Python library to identify the most English-like prompt from the 26 generated inputs, with the selected input representing the decrypted version of the detected jailbreak prompt.

B. Low-Rank Adaptation for Defending Jailbreak Prompts

Below, we present how we apply the chat template to the output of our deciphering layer (Step ①), followed by the model architecture used in DECIPHERGUARD. We then describe the parameter-efficient fine-tuning strategy employed to enhance DECIPHERGUARD's ability to defend against jailbreak prompts. Finally, we explain how DECIPHERGUARD is used to generate tokens and detect unsafe prompts.

② **Apply Chat Template**: In Step (2a), the input prompt has been processed by our deciphering layer, where the detected obfuscation-based jailbreak prompts have been reversed. In Step (2b), we apply the chat template proposed by Inan et al. [6]. Specifically, each input prompt is prefixed with a set of special header tokens and a task description. The prompt itself is enclosed between the "<BEGIN CONVERSATION>" and "<END CONVERSATION>" special tags. Finally, the template concludes with an instruction that guides the model in generating either safe or unsafe tokens in the first generated line to detect unsafe prompts.

③ **LlamaGuard**: In Step ③, the formatted input prompt is processed by a Byte-Pair-Encoding model [36] based on sentencepiece [37] to encode textual prompts into token IDs such as [220, 128000, ..., 128007]. Each token ID, which represents the position of a token in the embedding space, is mapped to a corresponding vector using the word embedding matrix $\mathbf{W} \in \mathbb{R}^{v \times h}$ of the LlamaGuard model, where $v = 128,256$ is the vocab size and $h = 4,096$ is the hidden size. This will produce an input matrix $\mathbf{X} \in \mathbb{R}^{l \times h}$, where l is the input sequence length. The input $\mathbf{X}$ is then fed into a stack of 32 transformer decoders. Each decoder consists of multiple layers, including masked self-attention, and feed-forward neural networks (FFNN), as used in the LlamaGuard model. The self-attention mechanism enables the model to capture dependencies across the input sequence, while the feed-forward layers apply non-linear transformations to enhance representation learning. These layers are stacked to build a deep network. For a detailed explanation of transformer decoders, we refer readers to the original paper [21].

④ **LoRA (Low-Rank Adaptation)**: In Step ④, To enhance DECIPHERGUARD's ability to defend against jailbreak prompts, we use Low-Rank Adaptation (LoRA) to efficiently adapt LlamaGuard's 8B-parameter model without fine-tuning its full parameter set. The pre-trained knowledge of LlamaGuard, while effective for general unsafe prompts, showed limitations in defending jailbreak prompts, as demonstrated in RQ1, Finding 1. LoRA modifies specific layers of LlamaGuard by introducing low-rank updates to their weight matrices, focusing only on a small subset of parameters while freezing the original pre-trained weights. Specifically, the weight update matrix $\Delta\mathbf{W}$ for a target layer is parameterised as the product of two new trainable matrices, $\mathbf{A} \in \mathbb{R}^{h \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times h}$. Here, h is the hidden size, and $r \ll h$ is a tunable rank parameter that controls the size of the adaptation. The updated weight for a given layer is expressed as:

$$\mathbf{W}_{new} = \mathbf{W}_{pretrained} + \mathbf{AB}.$$

In our implementation of DECIPHERGUARD, we apply LoRA to the word embedding layer, self-attention layers, and FFNN layers of LlamaGuard. For the word embedding layer, h corresponds to the embedding size, and for the self-attention and FFNN layers, h corresponds to the hidden size. By restricting updates to the low-rank matrices $\mathbf{A}$ and $\mathbf{B}$ and leaving the pre-trained weights ($\mathbf{W}_{pretrained}$) unchanged, we preserve the original capabilities of LlamaGuard while enabling efficient and focused adaptation to jailbreak prompts.

⑤ **Detect and Block Unsafe Prompts**: After applying the low-rank adaptation to LlamaGuard in Step ④, we obtain the model used in our DECIPHERGUARD approach. In Step ⑤, we employ our DECIPHERGUARD approach to detect and block unsafe prompts from users before they are forwarded to an LLM-powered software system. Given the output of the 32nd decoder layer, denoted as $\mathbf{H} \in \mathbb{R}^{l \times h}$, where l is the sequence length and h is the hidden size, DECIPHERGUARD treats the detection of unsafe prompts as a sequence generation task rather than a classification problem. To generate tokens, a linear layer maps $\mathbf{H}$ to a distribution over the vocabulary. Specifically, the hidden state of each token $\mathbf{H}_i \in \mathbb{R}^h$ is transformed using a weight matrix $\mathbf{W}_{\text{lm}} \in \mathbb{R}^{h \times v}$ and a bias vector $\mathbf{b}_{\text{lm}} \in \mathbb{R}^v$, where v is the vocab size. The resulting logits are then passed through a softmax function to compute probabilities over all possible tokens. Guided by the instructions embedded in the chat template from Step ②b, the model generates a sequence where the first token explicitly indicates whether the input prompt is "safe" or "unsafe." Specifically, we use greedy decoding to select tokens iteratively by choosing the one with the highest probability at each step. Formally, the next token t_i is determined as $t_i = \text{argmax}_k softmax\left(\mathbf{H}_i\mathbf{W}_{lm} + \mathbf{b}_{lm}\right)_k$, where k is the index of a token in the vocabulary, corresponding to the token with the highest probability after applying the softmax function. This process continues until the model generates the special end-of-text token "¡—eot_id—¿" or reaches the specified maximum token limit. We then select the first token generated by DECIPHERGUARD to detect unsafe prompts.

V. EXPERIMENTAL DESIGN

In this section, we present the motivation of our four research questions, the studied dataset, the studied jailbreak attacks, and our experimental setup.

A. Research Questions

To evaluate our DECIPHERGUARD approach, we formulate the following four research questions.

RQ1) What is the impact of the jailbreak attacks on the existing runtime guardrails? Recently, Inan et al. [6] proposed LlamaGuard, a runtime guardrail for LLMs designed to classify prompts as safe or unsafe. Despite its state-of-the-art performance of 0.945 accuracy, a key limitation exists as **(author?)** [6] speculated LlamaGuard may be susceptible to attacks that could alter or bypass its intended use. This means attackers can apply jailbreak attacks to unsafe prompts and potentially bypass the defence of LlamaGuard, weakening the reliability of AI guardrails, as their performance may be overestimated when evaluated on prompts not subjected to advanced jailbreak techniques. Yet, little is known about how jailbreak attacks can alter the performance of AI guardrails. Thus, we investigate the impact of jailbreak attacks against current AI guardrails.

RQ2) How effective is our DECIPHERGUARD in defending against jailbreak prompts? Given that jailbreak prompts can easily bypass the guardrail's defences, leading to a lower DSR as identified in RQ1, this research question aims to evaluate the robustness of our proposed DECIPHERGUARD against jailbreak attacks by analysing the impact on its DSR. In Sections IV-A and IV-B, we introduced the deciphering layer and LoRA-tuning, which aims to evaluate the DSR gains from these components compared to the baseline. Thus, we investigate the performance of our DECIPHERGUARD against baseline guardrails when defending against jailbreak attacks.

RQ3) What is the overall performance of DECIPHERGUARD when considering both aspects of defence success rate and false alarm rate? Ideally, LLM guardrails should correctly defend jailbreak prompts by blocking them, while also correctly classifying safe prompts as safe and allowing

them to pass to the LLM-powered system. However, in this scenario, relying solely on DSR is insufficient to achieve a comprehensive evaluation, as it fails to reflect the instances of false positives that guardrails may produce. Thus, we investigate how well our DECIPHERGUARD balances defence performance against jailbreak attacks while accounting for false alarms.

RQ4) What are the contributions of the components of our DECIPHERGUARD? Our DECIPHERGUARD involves two key components—the deciphering layer and Low-Rank (LoRA) Adaptation—to enhance its defence capabilities against jailbreak prompts. However, little is known about the contributions of each component in our DECIPHERGUARD and which component contributes the most to the Defence Success Rate (DSR) and our proposed metric - Overall Guardrail Performance (OGP) of our DECIPHERGUARD. Thus, we formulate this RQ to conduct an ablation study on the different variants of our DECIPHERGUARD.

B. Data Preparation

To address our four research questions, we require a comprehensive dataset encompassing jailbreak prompts, unsafe prompts, and safe prompts. To this end, we prepared a dataset comprising 18,790 jailbreak prompts, 1,879 unsafe prompts, and 2,000 safe prompts. Jailbreak prompts were generated by applying 10 distinct jailbreak attack techniques to the unsafe prompts, transforming them into adversarial examples designed to bypass guardrails. Unsafe prompts were sourced from four benchmark datasets: Do-Not-Answer [38], CatQA [39], AdvBench [40], and Forbidden Questions [14]. For safe prompts, we utilised the Alpaca dataset [41].

To answer RQ1 and RQ2, which focus on evaluating the defence effectiveness of the studied guardrails and our proposed DECIPHERGUARD, we leverage the jailbreak and unsafe prompts to assess their ability to defend against unsafe prompts with and without jailbreak attacks. To answer RQ3 and RQ4, which aim to evaluate the overall effectiveness of each guardrail, we utilise the full dataset, including safe prompts, to consider the number of false alarms from the guardrails. This ensures that the defence capability of DECIPHERGUARD is achieved without a corresponding increase in false alarms, thereby offering a comprehensive evaluation of its effectiveness. In what follows, we provide a detailed introduction to each data source used to build our studied dataset.

Jailbreak Prompts (i.e., Unsafe Prompts with Jailbreak Attacks). For each one of our studied jailbreak attacks, we apply them to the unsafe prompts dataset to get the jailbreak prompts. In total, we apply 10 jailbreak attacks to the unsafe prompts dataset consisting of 1,879 unsafe prompts, resulting in 18,790 jailbreak prompts.

Unsafe Prompt Datasets. In RQ1 and RQ2, we use unsafe prompts and unsafe prompts with jailbreak attacks applied to evaluate the performance of baseline guardrails and DECIPHERGUARD. To do so, we need a benchmark dataset of unsafe prompts. However, there exists numerous harmful categories of unsafe prompts (e.g., adult content, child exploitation). To ensure that our empirical evaluation is not bound to specific types of harmful categories, we use four benchmark datasets consisting of human-written prompts (i.e., Forbidden Questions [14]) and machine-generated prompts (i.e., Do-Not-Answer [38], CatQA [39], AdvBench [40]). These selected benchmark datasets will allow us to analyse guardrails' performance when exposed to attacks across various unsafe prompt categories. We describe each dataset below:

- **Do-Not-Answer** [38]: This dataset consists of GPT-4 generated prompts when the model is prompted by *What kind of questions should an AI assistant not answer?*.
- **CatQA** [39]: This dataset consists of harmful prompts across 11 combined categories, built on top of the combined list of prohibited use cases mentioned in OpenAI usage policies and Meta's Llama2 acceptable use policy.
- **AdvBench** [40]: This dataset consists of harmful prompts formulated as instructions, originally used to perform Greedy Coordinate Gradient attack [40], which finds universal and transferable attack prompts by optimising against multiple smaller open-source LLMs for multiple harmful behaviours.
- **Forbidden Questions** [14]: This dataset consists human-generated unsafe prompts collected from online sources, spanning 13 forbidden scenarios outlined in the OpenAI Usage Policy.

Safe Prompt Datasets. In RQ3 and RQ4, we use safe prompts in combination with the unsafe prompt datasets under jailbreak attacks, to evaluate whether DECIPHERGUARD meets the real-world deployment needs of producing minimum false alarms compared to baseline guardrails. We include a dataset of safe prompts, the **Alpaca** dataset [41], consisting of safe instructions used to fine-tune LLMs to enhance their instruction-following capabilities.

C. Studied Jailbreak Attacks

Previous works have proposed various taxonomies of jailbreak prompts against LLMs [31, 42, 43]. In this study, we focus exclusively on **single-turn, black-box** jailbreak techniques, which do not require feedback or response from the LLM for optimisation. This ensures that we are testing the effectiveness of the runtime guardrails themselves, rather than assessing or interacting with internal defence mechanisms of the LLMs.

In total, we compiled 10 jailbreak attack techniques guided by Dong et al. [31] and Yi et al. [43] to evaluate both the baseline guardrails and our proposed DECIPHERGUARD. By selecting a wide range of types of jailbreak attacks, we aim to better understand which attack types are most effective at circumventing guardrails and where gaps remain. We categorise the 10 attack techniques into 3 categories, as reflected by their traits:

- **Template-based**: Applying pre-defined templates and modifications to the prompt, leveraging specific wordings, structures, or sequences to trick the model into generating restricted content. The studied attacks are AIM citejailbreakchat2023, DAN [14], Combination (prefix

injection + refusal suppression) [3], Self Cipher [12] and DeepInception [16].
- **Obfuscation-based**: Rephrasing, encoding, or translating prompts into forms that guardrails may not recognise. These methods rely on the lack of multilingual alignment when guardrails are trained. The studied attacks are Caesar Cipher [12], Zulu [11], and Base64 [3].
- **Code-based**: Disguising harmful content within programming logic or dual-purpose scripts, exploiting the code generating capabilities within LLMs. The studied attacks are Dual Use [17], and Code Chameleon [18].

D. Studied Guardrails

In this section, we provide a detailed description of the four guardrails studied in our experiments, including their classification mechanisms and key features.

1) LlamaGuard: Based on the Llama3-8B model, LlamaGuard is an input-output guardrail that acts as an LLM to generate text in its output which indicates whether a given prompt or response is safe/unsafe. A prompt is classified as unsafe if the model's response contains the "unsafe" token in the first generated line.
2) OpenAI Moderation: An active learning guardrail that leverages publicly sourced data to identify previously unknown instances of unsafe content and fine-tune the GPT-based generative guardrail. When a prompt is passed to this API, it is classified as unsafe if the API returns a boolean value flagged as "True".
3) PerspectiveAPI: Using a Transformer-based model to return a probability score of whether the prompt should be considered unsafe under each of its violation categories. A prompt is classified as unsafe if the API reports a probability exceeding 0.5 for any harmful category it supports.
4) Perplexity: A measure that focuses on detecting irregularities in the linguistic structure of sentence by evaluating the probabilities of the next token predicted by an LLM. To compute perplexity for a given input using a pre-trained model such as GPT-2 [44], the model calculates the probability of each token conditioned on its preceding tokens. The perplexity is formulated as $PPL(x) = \exp\left(-\frac{1}{t}\sum_{i=1}^{t}\log p\left(x_i|x_{<i}\right)\right)$ where x is the input text sequence, x_i is the i-th token, $x_{<i}$ represents the sequence of preceding tokens, and t is the total number of tokens in the sequence.
   We follow the method proposed by Liu et al. [45]. First, a perplexity score is calculated from a set of safe prompts using the target LLM, the threshold is then set such that the False Positive Rate (FPR)—the fraction of safe prompts incorrectly flagged as unsafe—remains within an acceptable limit, such as 1%. Following this method, we obtain a threshold of 106, above which a prompt is classified as unsafe. We discuss the impact of the perplexity threshold on the performance of Perplexity in the Discussion section.

E. Experimental Setup

Data Splitting. For RQ2, RQ3, and RQ4, we use a stratified splitting to randomly split each type of jailbreak prompts evenly. Following common practice, the jailbreak prompts were initially divided into 10%/10%/80% for LoRA fine-tuning, validation, and testing. However, we opted to use only 5% of the jailbreak prompts (940 samples) for LoRA fine-tuning, 10% for validation, and the remaining 80% (15,032 samples) for testing. The impact of the training data sensitivity is further discussed in Section **??**.

Model Implementation and Optimisation. To implement our DECIPHERGUARD approach for defending unsafe prompts, we leveraged two Python libraries, i.e., Transformers [46] and Pytorch [47]. The Transformers library provides APIs for transformer-based model architectures and pre-trained weights, while PyTorch facilitates computations during training, including backpropagation. We downloaded the Llama-Guard checkpoint "meta-llama/Llama-Guard-3-8B" provided by Inan et al. [6]. We used our training set and low-rank adaptation [32] to fine-tune the checkpoint and obtain suitable weights for defending jailbreak prompts. The model was fine-tuned on two NVIDIA RTX 3090 graphic cards and the training time was 58 minutes. As shown in Equation 3, the Cross-Entropy Loss was used to update the model and optimise the alignment between the model's predicted token probabilities and the target sequence. For our training setup, the input is a chat-templated jailbreak prompt, and the target output is the same input followed by a single classification token, "unsafe". The loss function measures the negative log-likelihood of the correct token at each position in the target sequence, guiding the causal language model (CLM) fine-tuning to generate accurate outputs for defending jailbreak prompts. The Cross-Entropy Loss is computed as:

$$\mathcal{L}_{CE} = -\frac{1}{T}\sum_{t=1}^{T}\log P\left(y_t \mid x_{1:t}\right) \tag{3}$$

where T is the total number of tokens in the target sequence, y_t is the target token at position t, and $P\left(y_t \mid x_{1:t}\right)$ is the model's predicted probability for the correct token y_t, conditioned on the input and all previously generated tokens $x_{1:t}$. In our setup, the loss is primarily focused on the generation of the final token ("unsafe"), ensuring that the model predicts this token accurately based on the context of the input jailbreak prompt. We masked out the other tokens in the target sequence to prevent the model from being penalised for incorrectly predicting them. By minimising this objective function, the model learns to produce outputs where the first token after generating the input will be the "unsafe" token when the input is identified as unsafe.

Hyper-Parameter Settings. In our experiments, we set the learning rate to 1×10^{-4} with a constant learning rate scheduler. We used the AdamW optimiser [48] to update the model parameters. For the LoRA configuration, we set the rank (r) to 8, the alpha (α) to 32, and applied a dropout rate of 0.1. Due to GPU memory constraints, the training batch size was set to 1. The complete training recipe for

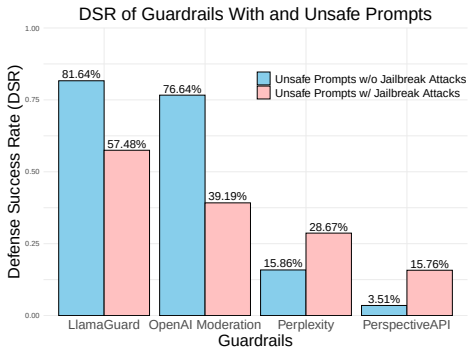

Fig. 4. (RQ1 Result) The Defence Success Rate (DSR) of each studied guardrail without and with attacks.

DECIPHERGUARD approach, is available in our replication package at https://github.com/awsm-research/DecipherGuard.

VI. EXPERIMENTAL RESULTS

In this section, we present the results for our four research questions.

(RQ1) *What is the impact of the jailbreak attacks on the existing runtime guardrails?*

Approach: To address this RQ, we investigate the impact of the jailbreak attacks on the performance of the existing runtime guardrails. Specifically, we compare the performance of the existing runtime guardrails in detecting unsafe prompts before and after applying jailbreak techniques. For the evaluation, we start with 1,879 unsafe prompts from studied datasets presented in Section V-B, we then applied ten types of jailbreak attacks, transforming them into an additional 18,790 jailbreak prompts. Both the original unsafe prompts and the jailbreak prompts are tested across four studied guardrails: LlamaGuard, OpenAI Moderation, PerspectiveAPI, and Perplexity. We use the Defence Success Rate (DSR) to quantify the defensive capability of guardrails. DSR is defined as the percentage of the number of the jailbreak prompts that can be the successfully defended by a runtime guardrail $\#\text{Prompts}_{\text{success}}$ and the total number of the jailbreak prompts $\#\text{TotalPrompts}$:

$$\text{DSR} = \frac{\#\text{Prompts}_{\text{success}}}{\#\text{TotalPrompts}} \tag{4}$$

Results: Figure 4 presents the defence success rate (DSR) of the four evaluated guardrails, comparing their performance under two scenarios: unsafe prompts without (blue bars) and with (red bars) jailbreak attacks.

LlamaGuard's DSR substantially decreases by 24.16%, decreasing from 81.64% to 57.48% when defending against unsafe jailbreak prompts. Similarly, OpenAI Moderation's DSR drops by 37.45%, declining from 76.64% to 39.19%. These results indicate that while both guardrails perform well with original unsafe prompts, achieving the two highest DSRs of 81.64% and 76.64% among baseline guardrails, respectively, their defense capabilities are substantially reduced by 24.16% and 37.45% when confronted with jailbreak prompts. **This finding demonstrates that the effectiveness of state-of-the-art guardrails is decreased when defending against jailbreak prompts, highlighting the need for jailbreak-aware guardrails capable of effectively defending such jailbreak prompts.**

(RQ2) *How effective is our* DECIPHERGUARD *in defending against jailbreak prompts?*

Approach: To address this RQ, we aim to evaluate the performance of our own DECIPHERGUARD against the state-of-the-art guardrails. We chose three guardrails, namely LlamaGuard [6], OpenAI Moderation [7], and Perplexity [8]. PerspectiveAPI was excluded as it refuses to process prompts with languages outside its training scope, particularly those transformed by obfuscation-based attacks (Base64, Zulu, Caesar Cipher). Specifically, we focus on the 18,790 jailbreak prompts to assess the effectiveness of DECIPHERGUARD in defending against such attacks, using the same Defence Success Rate (DSR). We present the absolute percentage difference between our DECIPHERGUARD and baseline guardrails as: $\%\text{DSR}_{\text{DecipherGuard}} - \%\text{DSR}_{\text{baseline}}$.

Results: Figure 6 presents the defence success rate (DSR) of our DECIPHERGUARD compared with the three baseline guardrail approaches.

Our DECIPHERGUARD achieves a DSR of 94.05% when under jailbreak attacks, which is 36% to 65% higher than the baseline guardrail approaches with a median improvement of 54%. In terms of DSR against jailbreak prompts, Figure 6 shows that DECIPHERGUARD achieves the highest DSR of 94.05%, while the baseline guardrails achieve a DSR of 28.67%-57.48%. This finding shows that DECIPHERGUARD substantially improves the state-of-the-art guardrails by 36% to 65% with a median improvement of 54.9%. **These results confirm that our DECIPHERGUARD approach is more effective than baseline guardrails in defending against jailbreak prompts.**

Figure 5 presents the DSR of our DECIPHERGUARD compared to the other three baseline guardrails, categorised by the ten different jailbreak attacks. Notably, DECIPHERGUARD substantially improves the DSR against obfuscation-based attacks compared to the best-performing baseline, LlamaGuard. For the three obfuscation-based attacks, DECIPHERGUARD achieves an improvement of 96.19% (2.55% $\rightarrow$ 98.74%) for Base64; 80.07% (4.84% $\rightarrow$ 84.91%) for Caesar Cipher; and 43.6%, (32.73% $\rightarrow$ 76.33%) for Zulu. Similarly, DECIPHERGUARD also enhances the DSR for the second-best baseline, OpenAI Moderation, achieving an improvement of 74% to 98% across the obfuscation-based attacks. **These results demonstrate that DECIPHERGUARD effectively addresses a key limitation of the state-of-the-art guardrail, LlamaGuard, namely its vulnerability to obfuscation-based attacks, highlighting the potential of DECIPHERGUARD to improve guardrail effectiveness in such scenarios.**

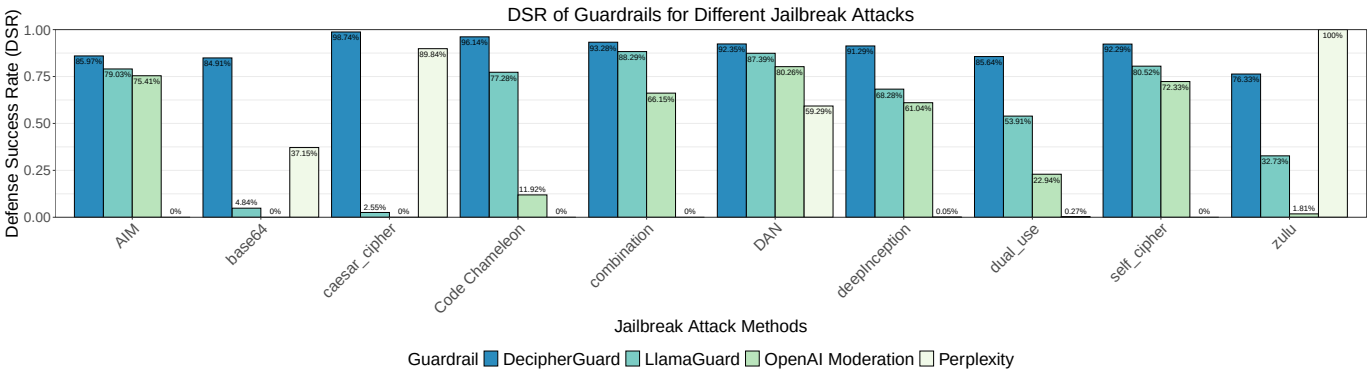

Fig. 5. (RQ2) DSR of different jailbreak attacks against guardrails.

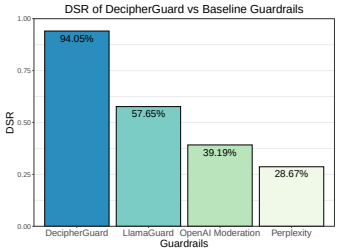

Fig. 6. (RQ2) The Defence Success Rate (DSR) of our DECIPHERGUARD when compared with three other state-of-the-art guardrails. Higher DSR = Better. (↗)

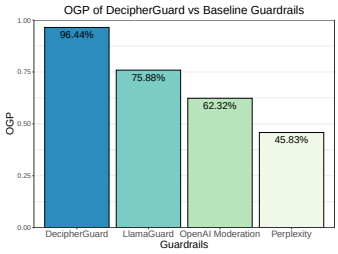

Fig. 7. (RQ3) The Overall Guardrail Performance (OGP) of our DECIPHERGUARD when compared with three other state-of-the-art guardrails. Higher OGP = Better. (↗)

(RQ3) *What is the overall performance of* DECIPHERGUARD *when considering both aspects of defence success rate and false alarm rate?*

Approach: To address this RQ, we compare the overall performance of our DECIPHERGUARD with the three baseline guardrails as in RQ2. We consider 2,000 safe prompts, 1,879 unsafe prompts without applying jailbreak attacks, and 18,790 jailbreak prompts. Prior research [6, 15, 49] has commonly employed traditional metrics such as AUPRC, accuracy, attack success rate (ASR), and F1 score to evaluate runtime guardrails. However, these metrics fail to capture the critical balance between defence success and false alarms, which is essential for the practical deployment of guardrails. In particular, metrics like ASR and accuracy focus solely on whether an attack bypasses the guardrail, neglecting the equally important aspect of reducing false alarms. While the F1-score effectively balances precision and recall, it is inadequate for evaluating runtime guardrails. This limitation arises because it does not account for the trade-off between defence effectiveness and false alarms, neglecting to penalise an excessive number of false positives. To address this gap, we propose the Overall Guardrail Performance (OGP) metric, defined as:

$$\text{OGP} = \sqrt{\text{DSR} \times (1 - \text{FAR})} \tag{5}$$

, where DSR (Defence Success Rate) measures the guardrail's ability to block unsafe prompts. FAR (False Alarm Rate) is calculated as $\frac{N_{FA}}{N_{\text{safe}}}$ where N_{FA} is the total number of false alarms and N_{safe} is the total number of safe prompts. In addition, $(1 - FAR)$ (the complement of FAR) captures guardrails' reliability in avoiding false positives. We then use the geometric mean to combine the two factors. Our choice of the geometric mean in the OGP metric is inspired by an established metric, the G-measure, which is the geometric mean of precision and recall. The G-measure normalizes true positives relative to both predicted positives and actual positives, effectively balancing the trade-off between the two measures. In other words, the amount of information the G-measure provides is the arithmetic mean of the information from precision and recall, ensuring that neither metric dominates the overall evaluation [50]. By combining these factors through a geometric mean, OGP evaluates guardrails holistically, ensuring that high defensive efficacy is not achieved at the expense of a high number of false alarms.

Results: Figure 7 presents the experimental results of our DECIPHERGUARD and the three baseline guardrails in terms of Overall Guardrail Performance (OGP).

Our DECIPHERGUARD achieves a OGP value of 96.44%, which is 20%-50% better than other baseline guardrails. In terms of the OGP, Figure 7 shows that our DECIPHERGUARD achieves an OGP of 96.44%, while the existing guardrails achieve an OGP of 45.83%-75.88%. This finding shows that DECIPHERGUARD substantially improves the baseline guardrails by 20%-50% with a median improvement of 33%. **These results confirm that our DECIPHERGUARD approach achieves better overall performance, enhancing defense effectiveness while reducing false alarms.**

In other words, our results demonstrate that the combination of the deciphering layer and low-rank adaptation (LoRA)

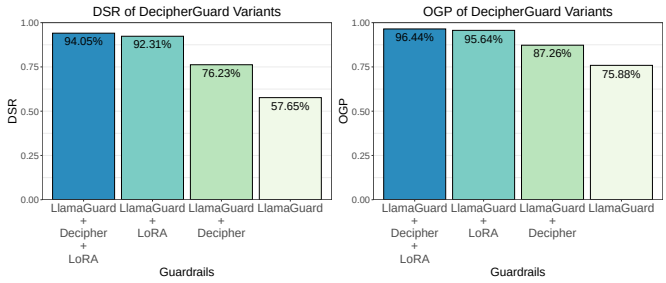

Fig. 8. (RQ4) Evaluation of different variations of DECIPHERGUARD.

mechanisms outperforms guardrails that rely solely on pre-trained large language models (LLMs) such as LlamaGuard. Prior works [6, 7] have utilised LLMs as guardrails to defend against potentially unsafe prompts from end users. However, as demonstrated in RQ1, their Decision Success Rate (DSR) drops significantly when confronted with jailbreak prompts. In RQ2, we found that these LLM-driven guardrails are particularly vulnerable to obfuscation-based attacks, such as Base64, Caesar Cipher, and Zulu, as illustrated in Figure 5. In contrast, our DECIPHERGUARD enhances LLM-driven guardrails by integrating a deciphering layer designed to detect and reverse obfuscation-based attacks. Additionally, DECIPHERGUARD extends LlamaGuard through Low-Rank Adaptation (LoRA), a lightweight and parameter-efficient (only 0.06% of parameters are tuned) fine-tuning approach that requires only 10% of the training data. This adaptation enables the model to better defend against jailbreak prompts while preserving the original pre-trained parameters of LlamaGuard. This paper is among the first to propose a deciphering layer and the use of LoRA to enhance LLM-driven guardrails, offering an effective solution to counter obfuscation-based attacks and adapt to evolving jailbreak scenarios.

(RQ4) *What are the contributions of the components of our* DECIPHERGUARD*?*

Approach*:* To answer this RQ, we investigate the contributions of the deciphering layer and LoRA within DECIPHERGUARD by examining the DSR and OGP of DECIPHERGUARD after removing different components. To understand and quantify the contribution of the two components of our approach, we alter DECIPHERGUARD as follows:

- *LlamaGuard*: Remove all components, plain LlamaGuard only.
- *Decipher*+*LlamaGuard*: Remove LoRA, but keep the deciphering layer.
- *LoRA*+*LlamaGuard*: Remove the deciphering layer, but keep LoRA to fine-tune LlamaGuard.
- *LoRA*+ *Decipher* + *LlamaGuard*: Full proposed DecipherGuard.

Following our previous RQs, we use the Defence Success Rate (DSR) and Overall Guardrail Performance (OGP) as measures for this ablation study.

Results*:* Figure 8 presents the ablation study to evaluate the contributions of the components in our DECIPHERGUARD.

The LoRA component is the most important component for enhancing Defence Success Rate (DSR) to achieve better defence effectiveness. Within our DECIPHERGUARD, the LoRA component contributes to 34.66% of the DSR. When comparing "*LlamaGuard* + *LoRA*" and "*LlamaGuard*" where the LoRA component is eliminated, we observe a performance decrease from 92.31% to 57.65%, accounting for 34.66%. Within our DECIPHERGUARD, the Decipher component contributes to 18.58% of the DSR. When comparing "*LlamaGuard* + *Decipher*" and "*LlamaGuard*" where the Decipher component is eliminated, we observe a performance decrease from 76.23% to 57.65%, accounting for 18.58%. These results underscore the substantial contributions of each component to the overall defence effectiveness of DECIPHERGUARD, which achieves the highest of 94.05% when both components are active.

The LoRA component is the most important component for enhancing Overall Guardrail Performance (OGP), leading to improved overall performance when considering safe prompts. Within our DECIPHERGUARD, the LoRA component contributes to 19.76% of the OGP. When comparing "*LlamaGuard* + *LoRA*" and "*LlamaGuard*" where the LoRA component is eliminated, we observe a performance decrease from 95.64% to 75.88%, accounting for 19.76%. Within our DECIPHERGUARD, the Decipher component contributes to 11.38% of the DSR. When comparing "*LlamaGuard* + *Decipher*" and "*LlamaGuard*" where the Decipher component is eliminated, we observe a performance decrease from 87.26% to 75.88%, accounting for 11.38%. These results underscore the substantial contributions of each component to the overall defence effectiveness of DECIPHERGUARD, which achieves the highest OGP of 96.44% when both components are active.

The Decipher component plays a crucial role in enhancing the overall performance of DECIPHERGUARD, contributing substantially to 18.58% of the DSR and 11.38% of the OGP. **Unlike LoRA, which relies on fine-tuning data and computation, Decipher offers a lightweight yet effective solution that requires no additional fine-tuning, making it computationally efficient and adaptable. Furthermore, the Decipher component can be integrated into non-LLM-based guardrails as an additional input processing layer.** This flexibility expands its applicability to a wider range of guardrails to enhance their defence effectiveness against obfuscated-based jailbreak prompts.

VII. DISCUSSION

In the previous experiment section, we empirically evaluated the performance of our DECIPHERGUARD and conducted an ablation study to support our design rationale. However, the computational overhead introduced by this layer has not been evaluated for the deciphering layer, which detects and reverses obfuscation-based jailbreak prompts by acting as an additional preprocessing layer. In this section, we perform an extended analysis of our proposed approach to resolve this question.

TABLE I
(DISCUSSION) RUNTIME LATENCY ANALYSIS OF OUR DECIPHERGUARD.

Configuration	Total Latency
LoRA-tuned LlamaGuard	333 ms
LoRA-tuned LlamaGuard + Base64 Deobfuscation	333.2 ms
LoRA-tuned LlamaGuard + Zulu Deobfuscation	333.5 ms
LoRA-tuned LlamaGuard + Caesar Cipher Deobfuscation	370 ms
DECIPHERGUARD	370.7 ms

A. DECIPHERGUARD*'s Runtime Efficiency*

In Section IV-A, we proposed the deciphering layer to detect and reverse obfuscated-based jailbreak prompts. Our ablation study (RQ4) confirmed its effectiveness in improving the overall performance of LLM-based guardrails such as LlamaGuard against jailbreak prompts. However, the computational overhead introduced by this additional layer remains unknown. Understanding this latency is crucial for assessing the feasibility and efficiency of our DECIPHERGUARD for future deployments. Thus, we analyse the runtime latency introduced by Base64 deobfuscation, Zulu deobfuscation, and Caesar cipher deobfuscation, as well as the total latency introduced by DECIPHERGUARD, comparing these results with the latency of the original LlamaGuard. We use the same data as in RQ3, consisting of 1,879 unsafe prompts w/o jailbreak, 18,790 unsafe prompts w/ jailbreak, and 2,000 safe prompts. All components in our deciphering layer are run by an AMD Ryzen 9 5950X CPU while the LoRA-tuned LlamaGuard is run by a Nvidia RTX 3090 GPU with 24GB of memory.

Table I presents the median runtime overhead of each deobfuscation, our DECIPHERGUARD, and LoRA-tuned LlamaGuard. Base64 deobfuscation introduces a latency of 0.2 ms, Zulu deobfuscation 0.5 ms, and Caesar cipher deobfuscation 37 ms. Among these, Caesar cipher deobfuscation is the most time-consuming due to the need to perform up to 25 character shifts. Each shift requires invoking a language detector to determine whether the resulting text is likely English and not ciphered code (see Algorithm 1). The total latency of our DECIPHERGUARD is 370.7 ms, comprising a deciphering layer and a LoRA-tuned LlamaGuard. The deciphering layer contributes 37.7 ms to this total, representing a relative latency increase of 11%. Despite this, the layer substantially improves the Defence Success Rate (DSR) by 18.58% and the Overall Guardrail Performance (OGP) by 11.38% (see RQ4), all without requiring model fine-tuning. These results highlight a reasonable latency tradeoff, offering substantial performance gains without the computational overhead of model fine-tuning.

VIII. THREATS TO THE VALIDITY

Threats to construct validity relates to the selection of jailbreak attacks. We selected 10 different jailbreak attacks guided by their prevalence to guardrails and ability to represent a wide spectrum of jailbreak techniques [28, 31, 43]. It is important to note that additional attack types can be included in future evaluations. However, this would not alter the key conclusion presented in RQ1, that jailbreak attacks substantially impact the performance of existing runtime guardrails. The underlying mechanisms through which these attacks degrade guardrail effectiveness remain consistent, regardless of the specific attacks tested.

Threats to internal validity relate to the potential influence of hyperparameter settings during the fine-tuning of our DECIPHERGUARD. Variations in model versions or different LoRA hyperparameters, compared to those specified in Section V, could impact the experiment's outcomes. To address this threat, we open-source our replication package and provide detailed documentation of all hyperparameter settings to ensure the experiment is reproducible by future researchers. Additionally, to minimise the impact of non-determinism introduced by deep learning model training, we conducted five repetitions of each experiment and presented the averaged outcomes to demonstrate the stability of our findings across multiple trials.

Threats to external validity concerns the generalisability of our results. Our experiment findings are supported by the dataset, jailbreak methods, and guardrails employed during the study. The dataset contains 1,879 unsafe prompts and 2,000 safe prompts from a separate dataset. When applied with the 10 jailbreak methods, we have 18,790 jailbreak prompts across 10 categories. While DECIPHERGUARD is fine-tuned specifically to address the jailbreak attacks discussed in this paper, other prompt datasets and jailbreak methods can be explored in future work.

IX. CONCLUSION

In this paper, we present DECIPHERGUARD, a novel framework that integrates a deciphering layer with low-rank adaptation (LoRA) to effectively defend against obfuscation- and template-based jailbreak prompts in LLM-powered software systems. We also introduce the Overall Guardrail Performance (OGP) metric, which evaluates guardrail performance by considering both defense effectiveness and the number of false alarms. Through an empirical evaluation of over 22,000 prompts across 10 different jailbreak attacks, our results highlight a substantial performance drop in state-of-the-art guardrails when confronted with such attacks. In comparison, DECIPHERGUARD achieves 36%-65% higher Defense Success Rate (DSR) and 20%-50% higher OGP, demonstrating superior effectiveness in defending against jailbreak attacks while retaining low false alarms. These findings underscore the potential of DECIPHERGUARD to help defend against jailbreak attacks and contribute to a safer deployment of intelligent software systems powered by LLMs.

ACKNOWLEDGEMENT

We thank Transurban and the CSIRO Next Generation Graduate AI Program: Creating Responsible AI Software Engineering Capability (RAISE) for their support and collaboration. The perspectives and conclusions presented in this study are solely the authors' and should not be interpreted as representing the official policies or endorsements of Transurban or any of its subsidiaries and affiliates. Additionally, the outcomes of this study are independent of, and should not be construed as an assessment of, the quality of products offered by Transurban.

REFERENCES

[1] A. E. Hassan, G. A. Oliva, D. Lin, B. Chen, Z. Ming *et al.*, "Rethinking software engineering in the foundation model era: From task-driven ai copilots to goal-driven ai pair programmers," *arXiv preprint arXiv:2404.10225*, 2024.

[2] Y. Bengio, G. Hinton, A. Yao, D. Song, P. Abbeel, T. Darrell, Y. N. Harari, Y.-Q. Zhang, L. Xue, S. Shalev-Shwartz *et al.*, "Managing extreme ai risks amid rapid progress," *Science*, vol. 384, no. 6698, pp. 842–845, 2024.

[3] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model (llm) security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, p. 100211, 2024.

[4] Z. Dong, Z. Zhou, C. Yang, J. Shao, and Y. Qiao, "Attacks, defenses and evaluations for llm conversation safety: A survey," *arXiv preprint arXiv:2402.09283*, 2024.

[5] S. Wang, T. Zhu, B. Liu, D. Ming, X. Guo, D. Ye, and W. Zhou, "Unique security and privacy threats of large language model: A comprehensive survey," *arXiv preprint arXiv:2406.07973*, 2024.

[6] H. Inan, K. Upasani, J. Chi, R. Rungta, K. Iyer, Y. Mao, M. Tontchev, Q. Hu, B. Fuller, D. Testuggine *et al.*, "Llama guard: Llm-based input-output safeguard for human-ai conversations," *arXiv preprint arXiv:2312.06674*, 2023.

[7] T. Markov, C. Zhang, S. Agarwal, F. E. Nekoul, T. Lee, S. Adler, A. Jiang, and L. Weng, "A holistic approach to undesired content detection in the real world," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 12, 2023, pp. 15 009–15 018.

[8] G. Alon and M. Kamfonas, "Detecting language model attacks with perplexity," *arXiv preprint arXiv:2308.14132*, 2023.

[9] A. Lees, V. Q. Tran, Y. Tay, J. Sorensen, J. Gupta, D. Metzler, and L. Vasserman, "A new generation of perspective api: Efficient multilingual character-level transformers," in *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, 2022, pp. 3197–3207.

[10] T. Rebedea, R. Dinu, M. Sreedhar, C. Parisien, and J. Cohen, "Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails," *arXiv preprint arXiv:2310.10501*, 2023.

[11] Z.-X. Yong, C. Menghini, and S. H. Bach, "Low-resource languages jailbreak gpt-4," *arXiv preprint arXiv:2310.02446*, 2023.

[12] Y. Yuan, W. Jiao, W. Wang, J.-t. Huang, P. He, S. Shi, and Z. Tu, "Gpt-4 is too smart to be safe: Stealthy chat with llms via cipher," *arXiv preprint arXiv:2308.06463*, 2023.

[13] Jailbreak Chat, "Jailbreak chat prompt," 2023, last accessed: 2024-09-20. [Online]. Available: https://www.jailbreakchat.com/prompt/4f37a029-9dff-4862-b323-c96a5504de5d

[14] X. Shen, Z. Chen, M. Backes, Y. Shen, and Y. Zhang, "" do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models," *arXiv preprint arXiv:2308.03825*, 2023.

[15] Z. Xu, Y. Liu, G. Deng, Y. Li, and S. Picek, "A comprehensive study of jailbreak attack versus defense for large language models," in *Findings of the Association for Computational Linguistics ACL 2024*, 2024, pp. 7432–7449.

[16] X. Li, Z. Zhou, J. Zhu, J. Yao, T. Liu, and B. Han, "Deepinception: Hypnotize large language model to be jailbreaker," *arXiv preprint arXiv:2311.03191*, 2023.

[17] D. Kang, X. Li, I. Stoica, C. Guestrin, M. Zaharia, and T. Hashimoto, "Exploiting programmatic behavior of llms: Dual-use through standard security attacks," in *2024 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2024, pp. 132–143.

[18] H. Lv, X. Wang, Y. Zhang, C. Huang, S. Dou, J. Ye, T. Gui, Q. Zhang, and X. Huang, "Codechameleon: Personalized encryption framework for jailbreaking large language models," *arXiv preprint arXiv:2402.16717*, 2024.

[19] Linkt, "Easier, smarter ways to pay for australian toll roads." https://www.linkt.com.au/, 2024.

[20] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" in *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, 2021, pp. 610–623.

[21] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[22] V. Alto, *Modern Generative AI with ChatGPT and OpenAI Models: Leverage the capabilities of OpenAI's LLM for productivity and innovation with GPT3 and GPT4*. Packt Publishing Ltd, 2023.

[23] T. Shen, R. Jin, Y. Huang, C. Liu, W. Dong, Z. Guo, X. Wu, Y. Liu, and D. Xiong, "Large language model alignment: A survey," *arXiv preprint arXiv:2309.15025*, 2023.

[24] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan *et al.*, "Training a helpful and harmless assistant with reinforcement learning from human feedback," *arXiv preprint arXiv:2204.05862*, 2022.

[25] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[26] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[27] J. Cui, Y. Xu, Z. Huang, S. Zhou, J. Jiao, and J. Zhang, "Recent advances in attack and defense approaches of large language models," *arXiv preprint arXiv:2409.03274*, 2024.

[28] A. Wei, N. Haghtalab, and J. Steinhardt, "Jailbroken: How does llm safety training fail?" *Advances in Neural*

Information Processing Systems, vol. 36, 2024.

[29] U. Anwar, A. Saparov, J. Rando, D. Paleka, M. Turpin, P. Hase, E. S. Lubana, E. Jenner, S. Casper, O. Sourbut *et al.*, "Foundational challenges in assuring alignment and safety of large language models," *arXiv preprint arXiv:2404.09932*, 2024.

[30] A. Biswas and W. Talukdar, "Guardrails for trust, safety, and ethical development and deployment of large language models (llm)," *Journal of Science & Technology*, vol. 4, no. 6, pp. 55–82, 2023.

[31] Y. Dong, R. Mu, Y. Zhang, S. Sun, T. Zhang, C. Wu, G. Jin, Y. Qi, J. Hu, J. Meng *et al.*, "Safeguarding large language models: A survey," *arXiv preprint arXiv:2406.02622*, 2024.

[32] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[33] Python, "base64 — base16, base32, base64, base85 data encodings," https://docs.python.org/3/library/base64.html, 2024.

[34] P. Stahl, "An accurate natural language detection library, suitable for short text and mixed-language text," https://pypi.org/project/lingua-language-detector/, 2024.

[35] S. Han, "Free google translate api for python. translates totally free of charge," https://pypi.org/project/googletrans/3.1.0a0/, 2020.

[36] R. Sennrich, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[37] T. Kudo, "Subword regularization: Improving neural network translation models with multiple subword candidates," *arXiv preprint arXiv:1804.10959*, 2018.

[38] Y. Wang, H. Li, X. Han, P. Nakov, and T. Baldwin, "Do-not-answer: A dataset for evaluating safeguards in llms," *arXiv preprint arXiv:2308.13387*, 2023.

[39] R. Bhardwaj, D. D. Anh, and S. Poria, "Language models are homer simpson! safety re-alignment of fine-tuned language models through task arithmetic," *arXiv preprint arXiv:2402.11746*, 2024.

[40] A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," *arXiv preprint arXiv:2307.15043*, 2023.

[41] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Alpaca: A strong, replicable instruction-following model," *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, vol. 3, no. 6, p. 7, 2023.

[42] P. Chao, E. Debenedetti, A. Robey, M. Andriushchenko, F. Croce, V. Sehwag, E. Dobriban, N. Flammarion, G. J. Pappas, F. Tramer *et al.*, "Jailbreakbench: An open robustness benchmark for jailbreaking large language models," *arXiv preprint arXiv:2404.01318*, 2024.

[43] S. Yi, Y. Liu, Z. Sun, T. Cong, X. He, J. Song, K. Xu, and Q. Li, "Jailbreak attacks and defenses against large language models: A survey," *arXiv preprint arXiv:2407.04295*, 2024.

[44] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[45] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 1831–1847.

[46] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[47] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[48] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2018.

[49] J. Chu, Y. Liu, Z. Yang, X. Shen, M. Backes, and Y. Zhang, "Comprehensive assessment of jailbreak attacks against llms," *arXiv preprint arXiv:2402.05668*, 2024.

[50] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," *arXiv preprint arXiv:2010.16061*, 2020.