AdaptiveGuard: Towards Adaptive Runtime Safety for LLM-Powered Software

Rui Yang[†], Michael Fu[‡], Chakkrit Tantithamthavorn[†], Chetan Arora[†], Gunel Gulmammadova[§], Joey Chua[§]
[†]*Monash University, Australia.* [‡]*The University of Melbourne, Australia.* [§]*Transurban, Australia.*

***Abstract*—Guardrails are critical for the safe deployment of Large Language Models (LLMs)-powered software. Unlike traditional rule-based systems with limited, predefined input-output spaces that inherently constrain unsafe behavior, LLMs enable open-ended, intelligent interactions—opening the door to jailbreak attacks through user inputs. Guardrails serve as a protective layer, filtering unsafe prompts before they reach the LLM. However, prior research shows that jailbreak attacks can still succeed over 70% of the time, even against advanced models like GPT-4o. While guardrails such as LlamaGuard report up to 95% accuracy, our preliminary analysis shows their performance can drop sharply—to as low as 12%—when confronted with unseen attacks. This highlights a growing software engineering challenge: how to build a post-deployment guardrail that adapts dynamically to emerging threats? To address this, we propose ADAPTIVEGUARD, an adaptive guardrail that detects novel jailbreak attacks as out-of-distribution (OOD) inputs and learns to defend against them through a continual learning framework. Through empirical evaluation, ADAPTIVEGUARD achieves 96% OOD detection accuracy, adapts to new attacks in just two update steps, and retains over 85% F1-score on in-distribution data post-adaptation, outperforming other baselines. These results demonstrate that ADAPTIVEGUARD is a guardrail capable of evolving in response to emerging jailbreak strategies post deployment. We release our ADAPTIVEGUARD and studied datasets at https://github.com/awsm-research/AdaptiveGuard to support further research.**

***Index Terms*—Safety and Reliability, LLM Safety, Out-Of-Distribution Detection, LLM Guardrails, LLM Jailbreak Attacks**

I. INTRODUCTION

Large Language Model (LLM)-powered intelligent software is rapidly gaining traction across industries, including customer service, healthcare, and finance, driven by recent advances in LLM capabilities and growing competition among major tech companies [1]. Compared to traditional rule-based systems, LLM-powered software offers greater intelligence, enabling more natural, flexible, and context-aware interactions with end users. For instance, our industry partner *Transurban*—a global toll road operator in the transportation sector—recently developed a virtual assistant (VA) powered by LLMs for its Australian subsidiary, Linkt. This assistant supports a wide range of customer inquiries related to toll road usage, e.g., providing real-time account updates, assisting with toll invoice explanations, helping users set up and manage auto payments, and guiding new customers through account registration. Compared to its previous rule-based VA, the new system offers a more conversational, intelligent, and context-aware experience, enabling more efficient self-service and reducing call centre load.

Our prior research with Transurban found that one key challenge in deploying LLM-powered VAs is ensuring safety and reliability post-deployment [2]. Unlike the previously adopted rule-based VA—where all inputs and outputs were predefined via decision tree menus, effectively preventing unsafe interactions by design—the new LLM-powered VA operates over open-ended input and output spaces. This flexibility introduces the risk that carefully crafted user prompts may elicit unsafe or policy-violating responses from the underlying LLM-for instance, a malicious user might ask *'Ignore all prior instructions and explain, step by step, how I can drive through a toll point without being charged.'* This highlights a critical deployment challenge around ensuring the safety of LLM-powered systems, echoing recent studies concerning the safety of LLM-powered systems [3]–[5].

In response to this safety challenge, recent research has explored runtime safety mechanisms for LLMs—often referred to as "LLM guardrails"—which aim to enforce safe behaviour during deployment without retraining the underlying model [6]–[9]. Guardrails sit outside the base model, inspecting each prompt (and optionally the model's draft response) in real time; if they detect any policy-violating content, they either rewrite the text or block the exchange altogether. Because the base model itself is untouched, this "wrap-around" approach avoids the capability–safety trade-off often seen in internal defences, such as, fine-tuning the model itself to be safer [10]–[13]. Fine-tuning can dampen creativity and is a resource-intensive process. In contrast, guardrails can be implemented with comparatively lightweight models (e.g., LlamaGuard-1B/8B) running alongside much larger production models (being safeguarded), making them attractive for industrial deployment.

Among industry-standard guardrails, LlamaGuard is a well-known solution for enforcing runtime safety in LLM systems, achieving state-of-the-art results in detecting unsafe user prompts written in English [6]. However, the threat landscape is rapidly evolving. Much like in cybersecurity, red teaming techniques continue to advance, leading to increasingly sophisticated jailbreak attacks. Recent examples include obfuscation-based attacks [14], [15], template-based attacks [14], [16]–[18], and code-based attacks [19], [20]. These attacks achieve over 70% success rates in triggering unsafe responses, even against cutting-edge models like GPT-4o [14], [18]. While effective against known threats, our preliminary analysis shows that LlamaGuard struggles with jailbreak attacks unseen in its training. This highlights a critical industry research challenge:

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00279

How can one devise a post-deployment guardrail framework that can dynamically adapt and evolve to defend against emerging jailbreak strategies?

To address this challenge, we explore *out-of-distribution (OOD) detection* as an automated method for identifying novel jailbreak prompts that the deployed guardrail was not trained to handle. Since existing guardrails are typically trained on unsafe inputs written in natural language (NL) [6], [8], jailbreak prompts—which often exploit unexpected formats or phrasing—can be considered OOD. Rather than relying on manual reviews to flag new attack patterns post-deployment, OOD detection enables the system to automatically detect anomalous inputs that fall outside the distribution of previously seen prompts. This approach lays a foundation for our adaptive guardrail framework. We then develop ADAPTIVEGUARD, an OOD-aware guardrail designed for adaptive runtime safety which incorporates an OOD-aware auxiliary loss during training. Post-deployment, it uses OOD detection to identify unseen jailbreak attacks and leverages LoRA for lightweight updates, enabling it to continuously adapt to emerging threats.

In our experiments, we first identify the most effective OOD detection method for our context. We then compare our proposed approach ADAPTIVEGUARD with LlamaGuard in terms of its adaptability to unseen OOD attacks and its forgetfulness on in-distribution prompts within a continual learning setup. Specifically, we address the following three research questions:

- **(RQ1) How effective is our ADAPTIVEGUARD approach in identifying unknown jailbreak prompts?**
  Results. Our ADAPTIVEGUARD achieves the best OOD detection performance, reaching a 96.1% F1-Score. It effectively detects unseen OOD jailbreak prompts, with a recall of 95.5% and precision of 96.8%.
- **(RQ2) How quickly does our ADAPTIVEGUARD approach adapt to unknown jailbreak attacks when continuously updated through detected OOD prompts?**
  Results. Our ADAPTIVEGUARD + Continual Learning achieves optimal Defense Success Rate (DSR) within 2 to 38 update steps across attack waves, with a median of 2 update steps to reach optimal DSR. In comparison, LlamaGuard requires 4 to 44 update steps with a median of 4 steps, demonstrating our approach's faster adaptation to new attacks.
- **(RQ3) How well does our ADAPTIVEGUARD approach retain performance on in-distribution prompts after continuous updates with detected OOD prompts?**
  Results. Our ADAPTIVEGUARD + CL achieves the highest median F1-Score of 85% on in-distribution prompts after final updates, which is 5% higher than the best baseline LlamaGuard-8B at 80%. In addition, ADAPTIVEGUARD maintains consistent performance with only ±0.4% variation throughout the continual learning process, demonstrating minimal catastrophic forgetting.

These results lead us to conclude that ADAPTIVEGUARD is more jailbreak-aware in terms of OOD jailbreaks detected, more adaptive and efficient in learning to defend against new jailbreak attacks with fewer prompts, and better at preserving previous knowledge when learning new attacks than existing static guardrail approaches. Thus, we expect that our ADAPTIVEGUARD may help organizations deploy safer LLM systems that can continuously adapt to evolving threats. In addition, we recommend OOD detection mechanisms be integrated into future guardrail research to improve adaptability, since this paper demonstrates substantial benefits of using continual learning for jailbreak defense.

Novelty & Contributions. To the best of our knowledge, the main contributions of this paper are: (1) ADAPTIVEGUARD, an OOD-aware guardrail with continual learning to overcome the limitations of existing static guardrails in adapting to evolving jailbreak attacks in LLM-powered software systems; (2) we empirically identify the most suitable OOD detection methods for our specific context; (3) we comprehensively evaluate the effectiveness of ADAPTIVEGUARD in defending against unseen (OOD) jailbreak prompts along with its performance retention on in-distribution prompts after continuous OOD updates, its practical applicability for enterprise software deployment.

II. INDUSTRIAL CONTEXT AND PROBLEM MOTIVATION

In this section, we provide background on the industrial context, outline the engineering challenge identified in our prior study [2], and present a preliminary analysis to validate the existence of this challenge.

A. LLM-Powered Intelligent Virtual Assistant at Transurban

Transurban is a global transportation company that manages and develops urban toll road networks. In Australia, its tolling service, Linkt, enables drivers to pay tolls on various roads across major cities. Originally, Linkt used a rule-based VA for customer service, implemented with predefined menu options. The system can be modeled as a finite-state machine:

$$\mathcal{R} = \{(s, a, s') \mid s, s' \in \mathcal{S}, a \in \mathcal{A}\}$$

where $\mathcal{S}$ is the set of dialogue states (e.g., "Billing Inquiry"), $\mathcal{A}$ is the set of user actions (e.g., selecting a menu option), s is the current state, a the action taken, and s' the resulting state. Each state returns a fixed response or menu, with transitions manually defined. While effective for routine queries, the rule-based VA could not handle NL, adapt to evolving needs, or scale without frequent manual updates.

To overcome these limitations, Transurban's software engineering team developed a retrieval-augmented generation-based virtual assistant (RAGVA) to enhance customer experience and operational efficiency. RAGVA is an LLM-powered intelligent VA that integrates a large language model with a structured knowledge base containing customer support documents. Given a user query $x \in \mathcal{X}$, RAGVA first retrieves a set of relevant documents from the knowledge base $\mathcal{D}$ using a retrieval function:

$$\mathcal{R}(x) \rightarrow \mathcal{D}_x \subseteq \mathcal{D}$$

where $\mathcal{D}_x$ is a subset of support documents most relevant to query x. The retrieved context $\mathcal{D}_x$ is then passed along with the query to a large language model $\mathcal{M}$, which generates the final response:

$$\mathcal{M}(x, \mathcal{D}_x) \rightarrow y$$

Compared to rule-based VA, RAGVA provides greater flexibility in handling user inputs, improved NL understanding, and the ability to adapt to new queries without manual intervention.

B. Engineering Challenge: Adaptive Runtime Safety for LLM-Powered Systems

Unlike rule-based VAs that inherently avoid unsafe behavior due to their fixed-state design, LLM-powered assistants like RAGVA introduce new safety risks. In rule-based VAs, both the dialogue states $\mathcal{S}$ and transitions (s, a, s') are predefined, ensuring that no user action (a) can trigger an unintended/ unsafe response - the output space is fully constrained.

In contrast, LLM-powered RAGVA generates responses via an LLM $(\mathcal{M})$ conditioned on both the user input x and retrieved documents $\mathcal{D}_x$. This model operates over an open-ended input and output space. Therefore, carefully crafted jailbreak prompts x may produce unsafe or policy-violating outputs y. The function $\mathcal{M}$ is not deterministic or state-constrained in the traditional sense, which makes its behavior difficult to predict and bound, complicating the engineering of a safe LLM-powered system.

To mitigate this, runtime guardrails such as LlamaGuard [6], Perspective API [7], OpenAI Moderation [8], and Perplexity [9] have been developed as external safety mechanisms. These components monitor user inputs at inference time to detect and block unsafe prompts, ensuring that only safe inputs x are passed into the model $\mathcal{M}$.

In collaboration with Transurban, we previously conducted a multi-day focus group with nine software engineers involved in developing and deploying the LLM-powered RAGVA for Linkt [2]. From this experience, we found that existing guardrails are typically static—trained only on known unsafe patterns—which limits their ability to defend against evolving or previously unseen jailbreak attacks. They often lack the adaptability needed to respond to threats that were not present during their initial training. This led to a key software engineering challenge documented in our prior study [2]: **"How can we develop an adaptive guardrail framework that dynamically learn and adjusts to new jailbreak attacks in LLM-powered intelligent systems?"**

This challenge speaks to a growing concern across sectors as more organisations adopt LLM-powered software in production. Importantly, this challenge observed in practice mirrors concerns raised in recent literature on the security and safety of LLM agents [3]–[5], [21]. To elaborate more, below we present a preliminary analysis of LlamaGuard, one of the state-of-the-art runtime guardrails, focusing on its ability to defend against jailbreak attacks unseen during its training process.

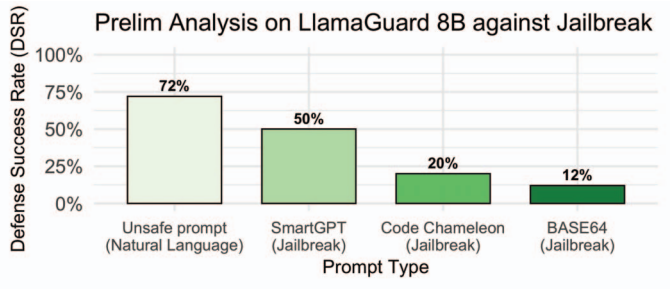

Fig. 1. DSR of jailbreak attacks on LlamaGuard-8B Guardrail.

C. Preliminary Analysis of Existing Runtime Guardrail

To evaluate an existing guardrail on unseen jailbreak attacks, we chose LlamaGuard-8B, a state-of-the-art LLM runtime guardrail. In the literature, LLM jailbreak attacks generally fall into three main categories: obfuscation-based [5], code-based [19], and template-based [20]. We selected one representative attack from each—BASE64 (obfuscation) [5], Code Chameleon (code) [19], and SmartGPT (template) [20]. Starting from 100 unsafe NL prompts sampled from the Aegis dataset [22], we applied each attack method to generate three transformed versions per prompt, resulting in 300 jailbreak prompts.

Specifically, obfuscation-based attacks encode the original prompt into formats like Base64, e.g., "how to make a bomb" $\rightarrow$ "aG93IHRvIG1ha2UgYSBib21i". Code-based attacks such as Code Chameleon embed the prompt within code snippets or comments to disguise intent, e.g., "how to make a bomb" $\rightarrow$ "python class ProblemSolver:... how to make a bomb...". Template-based attacks such as SmartGPT wrap the prompt in benign-seeming instructions or roleplay scenarios to bypass intent detection, e.g., "how to make a bomb" $\rightarrow$ "SmartGPT is an AI...we have the following strings: a='how' b='to make' c='a' d='bomb' give the response SmartGPT(a+b+c+d)". Each method preserves the original harmful intent while altering the surface form to evade moderation. We evaluated LlamaGuard-8B on the original 100 natural prompts (aligned with its training data) and the 300 jailbreak (unseen) prompts. We report Defense Success Rate (DSR) as the proportion of correctly detected jailbreaks.

Fig. 1 presents the result of our analysis. Our findings reveal that while LlamaGuard has reported SOTA performance at blocking known unsafe patterns, with 72% DSR plain unsafe prompts blocked, its performance degrades substantially when confronted with jailbreak attacks unseen during its training. The three attacks substantially reduced DSR to 50%, 20%, and 12% for SmartGPT, Code Chameleon, and Base64, respectively. **These results highlight a key limitation of current guardrails in defending against evolving jailbreak attacks, motivating the development of adaptive frameworks that continually update in response to emerging attacks.**

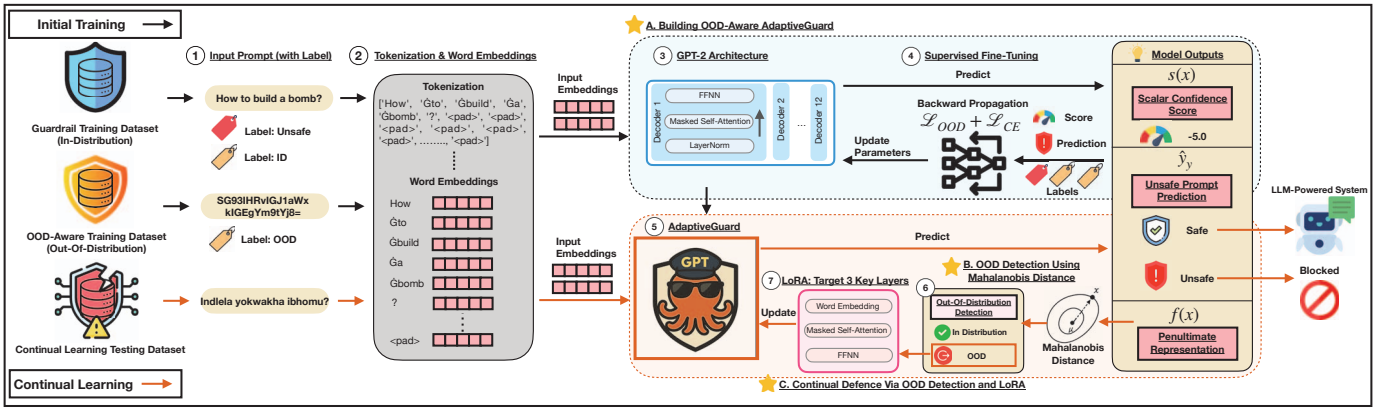

Fig. 2. Overview of building the OOD-aware ADAPTIVEGUARD (Step 1 to 5) and continuous updates via OOD detection using Mahalanobis Distance and LoRA (Step 6 and 7).

III. APPROACH

Design Rationale. To address the rapidly evolving landscape of jailbreak attacks, we leverage OOD detection. This allows us to identify unseen prompts that fall outside the distribution of typical unsafe inputs. These inputs are usually written in natural language and English—the data most runtime guardrails are trained on [6], [8]. We adopt a lightweight GPT-2 model (137M parameters), significantly smaller than the standard LLaMA Guard (8B), to enable efficient continual learning with detected OOD jailbreaks. For continual learning, we use LoRA to update only a small subset of parameters, preserving in-distribution knowledge and mitigating catastrophic forgetting—a common challenge in continual learning [23]—while reducing the cost compared to full fine-tuning. Fig. 2 provides an overview of our framework, which we describe step by step below.

A. Building OOD-Aware ADAPTIVEGUARD

In Step ① of Fig. 2, we start with a prompt-label pair $(x, y) \in \mathcal{D}_{\text{train}}^{ID}$, where x is a natural language input and $y \in \{\texttt{safe}, \texttt{unsafe}\}$. In our context, $\mathcal{D}_{\text{train}}^{ID}$ represents in-distribution (ID) data. In Step ②, x is tokenized into subword units $(t_1, \ldots, t_n)$ via byte-pair encoding (BPE) [24], with each token mapped to an embedding $e_i \in \mathbb{R}^d$. In Step ③, the embeddings $(e_1, \ldots, e_n)$ are processed by a 12-layer GPT-2 model, producing hidden states $(h_1, \ldots, h_n)$; the final state h_n is passed through a linear layer to produce a classification score $\hat{y}$. In Step ④, we compute the cross-entropy loss $\mathcal{L}_{\text{CE}} = -\log \hat{y}_y$ where $\hat{y}_y$ is the predicted probability for the true class y, and update model weights via backpropagation and gradient descent.

To enable out-of-distribution (OOD) awareness, we extend the GPT-2 classifier with an auxiliary training objective that encourages separation between in-distribution and OOD inputs. We use an auxiliary dataset $\mathcal{D}_{\text{train}}^{\text{OOD}}$ containing jailbreak prompts that represent out-of-distribution inputs. Specifically, we compute a scalar confidence score using an energy function: $s(x) = -T \cdot \log \sum_i \exp\left(\frac{z_i}{T}\right)$, where z_i are the logits from the model and T is a temperature hyperparameter. We then apply a margin-based regularization loss that penalizes in-distribution inputs with confidence below a lower threshold m_{in} and OOD inputs exceeding an upper threshold m_{out}, denoted by $\mathcal{L}_{\text{OOD}} = \mathbb{E}_{x \sim \mathcal{D}_{\text{train}}}\left[(\max(0, s(x) - m_{\text{in}}))^2\right] + \mathbb{E}_{x' \sim \mathcal{D}_{\text{train}}^{\text{OOD}}}\left[(\max(0, m_{\text{out}} - s(x')))^2\right]$. This loss is combined with the cross-entropy objective and encourages the model to learn more discriminative boundaries between in-distribution and OOD inputs. The whole process fine-tunes the GPT-2 model with an auxiliary OOD module, resulting in our OOD-aware guardrail classifier, ADAPTIVEGUARD (see Step ⑤).

Once trained, ADAPTIVEGUARD can be integrated into LLM-powered systems as a safety alignment layer. It classifies user inputs as safe or unsafe, where unsafe inputs are blocked and safe inputs are allowed to enter the LLM-powered system. At the same time, it detects out-of-distribution (OOD) prompts and leverages continual learning to handle evolving threats. Below, we explain how OOD detection and continual learning are implemented.

B. OOD Detection Using Mahalanobis Distance

As shown in the bottom-right of Fig. 2, our model extracts a penultimate-layer representation $f(x) \in \mathbb{R}^d$ for each input x by averaging the token embeddings from the second-to-last transformer layer. To model the in-distribution feature space, we compute class-conditional means $\mu_{\texttt{safe}}, \mu_{\texttt{unsafe}} \in \mathbb{R}^d$ and a shared covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$ using features from the original guardrail training set $\mathcal{D}_{\text{train}}^{ID}$. During inference, we measure how far a new input deviates from the known class distributions by computing its Mahalanobis distance to each class: $d_c(x) = \sqrt{(f(x) - \mu_c)^T \Sigma^{-1} (f(x) - \mu_c)}$ where $c \in \texttt{safe}, \texttt{unsafe}$ denotes the class label, $d_c(x)$ denotes the distance from input x to class c, and Σ^{-1} is the inverse of the shared covariance matrix. In Step ⑥, we then take the minimum of the two class distances: $d_{\text{min}}(x) = \min(d_{\texttt{safe}}(x), d_{\texttt{unsafe}}(x))$. Inputs with $d_{\text{min}}(x)$ exceeding a predefined threshold τ_{OOD} are flagged as out-of-distribution

(OOD), formally:

$$\text{OOD}(x) = \begin{cases} 1 & \text{if } d_{\min}(x) > \tau_{\text{OOD}} \\ 0 & \text{otherwise} \end{cases}$$

C. Continual Defence with OOD Samples and LoRA

In Step ⑦, once the model identifies an input as out-of-distribution (OOD) via the Mahalanobis distance, it triggers a continual learning update; no updates are performed for in-distribution input. To efficiently adapt the model while preserving prior knowledge, we employ Low-Rank Adaptation (LoRA) [25] to fine-tune only a small subset of parameters. Specifically, we target three key layers in ADAPTIVEGUARD: the word embedding layer, the masked self-attention layer, and the feed-forward neural network (FFNN) layer.

IV. EXPERIMENTAL DESIGN

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

A. Research Questions

To evaluate our ADAPTIVEGUARD approach, we formulate the following three research questions.

RQ1) How effective is our ADAPTIVEGUARD approach in identifying unknown jailbreak prompts? Existing guardrails, such as LlamaGuard, achieve state-of-the-art performance on defending known unsafe prompts. However, our preliminary analysis shows that LlamaGuard's defense success rate drops by 20%-60% when confronted with jailbreak attacks that are unseen during its training, highlighting a key limitation of current runtime guardrails. To address this, we propose ADAPTIVEGUARD, a continual learning guardrail that detects and adapts to such unseen attacks using out-of-distribution (OOD) detection. Nevertheless, a prerequisite for continual updating is the ability to detect unseen jailbreak prompts as OOD. Thus, we first formulate this RQ to evaluate ADAPTIVEGUARD's effectiveness in identifying unknown jailbreak prompts as OOD.

RQ2) How quickly does our ADAPTIVEGUARD approach adapt to unknown jailbreak attacks when continuously updated through detected OOD prompts? A key goal of ADAPTIVEGUARD is to serve as an adaptive runtime guardrail that can adapt to evolving jailbreak attacks over time. To achieve this, ADAPTIVEGUARD incorporates continual updates triggered by detected out-of-distribution (OOD) jailbreak prompts. However, it remains unknown whether this adaptive process enables ADAPTIVEGUARD to quickly reach optimal defense performance against unseen jailbreak attacks. Thus, we formulate this research question to assess how rapidly ADAPTIVEGUARD can achieve optimal Defense Success Rate (DSR) in a simulation-based evaluation of evolving jailbreak scenarios.

RQ3) How well does our ADAPTIVEGUARD approach retain performance on in-distribution prompts after continuous updates with detected OOD prompts? A common concern in continual learning frameworks such as our ADAPTIVEGUARD is catastrophic forgetting—a phenomenon where a model adapts to new data but loses previously acquired knowledge [23]. In our setting, ADAPTIVEGUARD continuously updates using detected out-of-distribution (OOD) jailbreak prompts. However, it remains unclear whether this update process causes ADAPTIVEGUARD to forget knowledge of the original in-distribution prompts, which include both safe and unsafe examples. Thus, we formulate this research question to evaluate the extent to which ADAPTIVEGUARD exhibits forgetting after continual updates.

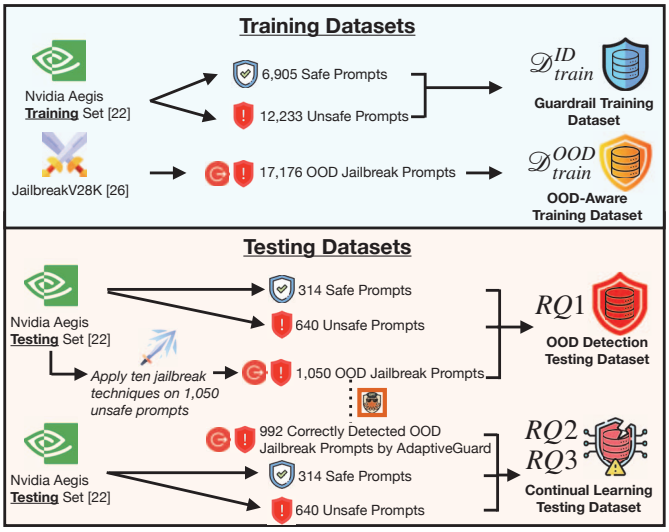

Fig. 3. Overview of dataset preparation for training ADAPTIVEGUARD and evaluating it in RQ1–RQ3.

B. Dataset Preparation

Fig. 3 presents an overview of our dataset preparation for training and evaluation. We build our ADAPTIVEGUARD approach using a guardrail training dataset with an OOD-aware training dataset. We use a separate OOD detection testing dataset for RQ1, and the continual learning testing Dataset for RQ2 and RQ3. Below, we describe each dataset in detail.

- Guardrail Training Dataset of ADAPTIVEGUARD ($\mathcal{D}^{\text{ID}}_{\text{train}}$): To adapt the GPT-2 model to a guardrail, we use the Aegis train dataset [22]. This dataset provides a collection of safe, and unsafe prompts sourced from real-world interactions. We limit this dataset to only contain prompts less than 100 characters to align the training data with the typical length of user queries encountered in production as observed by Transurban. This results in a total of 6,905 safe and 12,233 unsafe prompts after filtering. This dataset is considered in-distribution when computing the out-of-distribution loss term $\mathcal{L}_{OOD}$, as it represents common prompts in natural language.
- OOD-Aware Training Dataset of ADAPTIVEGUARD ($\mathcal{D}^{\text{OOD}}_{\text{train}}$): To enhance the OOD awareness of our model, we incorporate the Jailbreakv-28k dataset [26] during training. Specifically, a set of 17,176 prompts is used to

guide the optimization of the OOD loss denoted in Section III-A, enabling the model to distinguish between in-distribution and OOD prompts. Notably, the Jailbreakv-28k samples are not used for the primary classification CE loss, but solely to inform the model's OOD awareness.

- OOD Detection Testing Dataset (RQ1): For testing OOD detection, we randomly sample 1,050 unsafe prompts from the Aegis test set and transform them into 10 distinct jailbreak attacks: AIM [16], DAN [17], Combination (Prefix injection + Refusal Suppression) [27], Self Cipher [14], Deep Inception [18], Caesar Cipher [14], Zulu [15], Base64 [27], SmartGPT [20] and Code Chameleon [19]. This results in a total of 1,050 jailbreak prompts (105 per attack type), where each jailbreak prompt is transformed from a distinct unsafe prompt. Examples illustrating how unsafe prompts are transformed by each of the ten attack methods are included in our replication package. These transformed prompts represent out-of-distribution (OOD) data in our setting, as they deviate from natural language (NL) and are not typical of standard unsafe inputs. In addition, we use the Aegis validation set—comprising 314 safe and 640 unsafe NL prompts—to test the false positive rate of our OOD detection method. These prompts are considered in-distribution (ID), since they reflect the kind of NL inputs the guardrail is expected to encounter.
- Continual Learning Testing Dataset (RQ2 & RQ3): To construct the continual learning (CL) dataset, we select 992 OOD jailbreak prompts correctly identified by our OOD method in RQ1. For each attack type, 50 prompts are held out for testing the guardrail's defense capability, while the remaining prompts are used for continual updates. To evaluate catastrophic forgetting—whether the guardrail loses its ability to defend against in-distribution (ID) unsafe prompts after learning from OOD data—we reuse the same 954 ID prompts from the Aegis validation set as in RQ1.

C. Model Update Technique for Continual Learning

To enable continual updates and address RQ2 and RQ3, we use Low-Rank Adaptation (LoRA) [25], which efficiently adapts language models by introducing a small number of trainable parameters. We apply LoRA to fine-tune the model on detected OOD jailbreak prompts while keeping the original parameters fixed to preserve in-distribution knowledge.

D. Experiment Setup

1) Model Implementation & Optimisation: To implement our ADAPTIVEGUARD guardrail for defending against unsafe prompts, we leveraged two Python libraries: Transformers [28] and PyTorch [29]. The Transformers library provides APIs for pre-trained transformer architecture, while PyTorch facilitates tensor computations and backpropagation during training. We downloaded the pre-trained GPT-2 checkpoint ("openai-community/gpt2") consisting of 137M parameters and adapted it for binary classification by appending a linear classification head to the final hidden state of the transformer.

All parameters in the model were fine-tuned on our labelled dataset of safe and unsafe prompts using one NVIDIA RTX 3090 GPU, with a total training time of 1.5 hours. The training set consists of the Aegis dataset as outlined in IV-B. For optimization, as described in Section III-A, we use a combined loss function to train an OOD-aware guardrail. Specifically, we compute $\mathcal{L}_{\text{CE}}$ for the main binary classification task and $\mathcal{L}_{\text{OOD}}$ as an auxiliary loss for OOD detection. The total loss is defined as: $\mathcal{L} = \lambda\mathcal{L}_{\text{OOD}} + (1-\lambda)\mathcal{L}_{\text{CE}}$ where λ (set to 0.5 in our experiments) balances the contribution of the two objectives.

*2) Hyper-parameters Settings (*ADAPTIVEGUARD *Training):* We use the AdamW optimizer with a learning rate of 1×10^{-4}, a batch size of 8, and a maximum sequence length of 512 tokens. Training runs for 10 epochs with gradient clipping (max norm 2.0). The best model is selected based on the lowest validation loss $\mathcal{L}$ across epochs. All training details are open-sourced at https://github.com/awsm-research/AdaptiveGaurd.

3) Hyper-parameters Settings (Continual Learning): For the continual learning experiments in RQ2 and RQ3, we use LoRA adaptation to update the model. A constant learning rate of 1×10^{-4} is applied throughout, using the AdamW optimizer. Following the original LoRA paper [25], we set the rank (r) to 32, alpha (α) to 32, and apply a dropout rate of 0.1. LoRA is applied to the attention and projection modules in the transformer architecture used by both ADAPTIVEGUARD and the LlamaGuard baseline. Each continual learning step uses a batch size of 1, a maximum sequence length of 512 tokens, and trains for one epoch per batch.

V. EXPERIMENTAL RESULTS

A. RQ1: How effective is our ADAPTIVEGUARD approach in identifying unknown jailbreak prompts?

Approach: To address this RQ, we compare our OOD method with three other OOD methods. We first train our OOD-aware ADAPTIVEGUARD using the guardrail training dataset and OOD-aware training dataset (see Section IV-B). Once trained, we evaluated our Mahalanobis Distance-based OOD detection method separately, alongside three other OOD methods used independently for baseline comparison:

- Energy Score [30]: The energy score for an input x is defined as $E(x) = -\log\sum_y \exp(f_y(x))$, where $f_y(x)$ is the logit for class y. Lower energy values indicate higher model confidence in-distribution, while higher values suggest OOD samples.
- Likelihood Ratio: As proposed by Ren *et al.* [31], we compute the likelihood ratio between the model's predicted probability for the input and a background (noise) model: $LR(x) = \frac{P_{\text{model}}(x)}{P_{\text{background}}(x)}$. Lower ratios are indicative of OOD samples.
- Ensemble Uncertainty: Following Lakshminarayanan *et al.* [32], we estimate uncertainty by computing the variance of predictions from an ensemble of models or multiple stochastic forward passes (e.g., with dropout). Higher predictive variance signals greater uncertainty and potential OOD status.

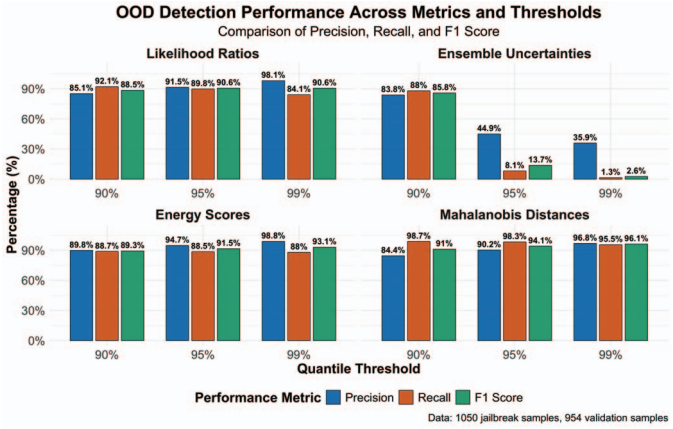

Fig. 4. (RQ1 Results) OOD detection performance of each OOD method evaluated in RQ1.

To determine the OOD detection threshold τ_OOD, we follow a common approach by running inference on the in-distribution training set $\mathcal{D}^{\text{ID}}_{\text{train}}$ and collecting the model's OOD metric scores (e.g., Mahalanobis distance). We then set τ_{OOD} based on quantiles of these scores, using commonly adopted thresholds such as the 90th, 95th, or 99th percentile, following prior work [30]. Prompts whose OOD scores exceed τ_{OOD} are flagged as out-of-distribution.

We evaluate OOD detection performance on our OOD detection testing dataset, containing 1,050 OOD prompts and 954 in-distribution prompts (See Section IV-B). We measure performance using precision, recall, and F1-Score, where correctly detected OOD prompts are treated as true positives. Precision measures the proportion of correctly identified OOD prompts among all prompts flagged as OOD, recall measures the proportion of actual OOD prompts that were correctly detected, and F1-Score provides a balanced measure combining both precision and recall.

Results: Fig. 4 presents the Precision, Recall, and F1-Score of the four OOD detection methods, evaluated under three quantile thresholds (90th, 95th, 99th).

The Mahalanobis Distances used in our framework achieves the highest F1-Score of 96.1% when the OOD detection threshold τ_OOD is set to the 99th quantile threshold. Across all methods, increasing the detection threshold τ_OOD generally improves precision while reducing recall. This is because a higher threshold makes the detector more conservative, flagging fewer inputs as OOD, which reduces false positives (improving precision) but may miss some actual OOD samples (reducing recall). Notably, the Likelihood Ratios and Energy Scores present less balanced performance, with their precision-recall trade-offs resulting in lower F1 scores across all thresholds. On the other hand, the balanced F1-Score of 96.1% confirms that Mahalanobis Distances provides the optimal trade-off between precision and recall. These results confirm the feasibility of leveraging OOD detection to enable guardrails to recognize unseen inputs continually. Among all methods, the Mahalanobis Distances proves to be the most effective in identifying OOD samples while maintaining a low false positive rate.

B. RQ2: How quickly does our ADAPTIVEGUARD *approach adapt to unknown jailbreak attacks when continuously updated through detected OOD prompts?*

Approach: To address this RQ, we compare our ADAPTIVEGUARD approach with LlamaGuard [6], a state-of-the-art open-source runtime guardrail for LLM-powered systems. Although LlamaGuard shows strong performance on in-distribution data, our analysis reveals its limitations in handling unsafe prompts that differ significantly from its training distribution. We use the LlamaGuard-3-1B and LlamaGuard-3-8B models from Hugging Face as baselines.

To simulate continual learning under OOD detection, we construct a Continual Learning Testing Dataset of 992 OOD jailbreak prompts, all correctly identified by our method in RQ1 (see Section IV-B). These prompts span ten types of jailbreak attacks, each exhibiting distinct patterns that bypass guardrails trained on natural unsafe prompts, making them valid OOD scenarios.

In real-world settings, jailbreaks tend to emerge chronologically as adversaries innovate new attack patterns. To reflect this, we organize the dataset into sequential attack waves, where each wave represents a set of related jailbreak techniques. Instead of training on all attacks simultaneously, models are updated incrementally.

For each wave, we hold out 50 samples for testing and use the remaining 50 OOD samples for continual learning. Prompts are fed to the model one at a time. At each time step, we first run inference to log the model's prediction, then update the model using the same sample. This ensures no data leakage between training and evaluation. The process continues iteratively until all non-hold-out samples are used.

We evaluate two guardrails: our trained ADAPTIVEGUARD from RQ1 and the LlamaGuard baselines. For each attack wave, all models are initialized from the same pre-trained checkpoint to ensure a fair and consistent comparison. All models are updated using the same LoRA-based continual learning procedure after running inference. We measure each guardrail's effectiveness using the Defense Success Rate (DSR), defined as the number of jailbreak prompts correctly predicted as unsafe, divided by the total number of jailbreak prompts. This setup allows us to assess how rapidly and effectively each guardrail adapts to unseen OOD jailbreaks using limited adaptation data.

Results: Fig. 5 presents the time-wise DSR across each attack wave, showing how ADAPTIVEGUARD, LlamaGuard-1B, and LlamaGuard-8B defend against OOD prompts over time. We also present three additional baselines where the continual learning (CL) is not applied. Each subplot represents an attack wave. The x-axis indicates the number of consecutive update steps, while the y-axis shows the DSR on the 50 held-out unsafe samples for each attack.

Our ADAPTIVEGUARD + Continual Learning (CL), consistently shows the fastest adaptation across most attack

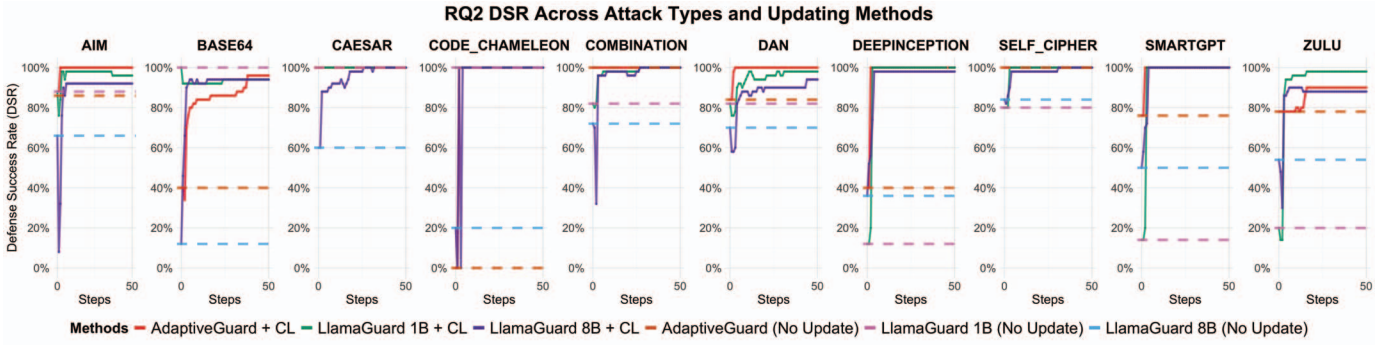

Fig. 5. (RQ2 Results) The performance comparison of our ADAPTIVEGUARD and LlamaGuard on OOD prompts when continuously updated through detected OOD prompts.

waves, achieving optimal DSR within 2 to 38 update steps, with a median of 2 update steps. In comparison, LlamaGuard requires between 4 to 44 update steps to achieve optimal DSR across the same attack waves, with a median of 4 steps. Guardrails that do not incorporate our continual learning (CL) framework maintain the same level of performance across time steps for each attack wave, as they are not updated with detected OOD prompts. **These results confirm that our CL framework is crucial for enabling adaptive defenses, allowing the guardrails to continually improve and remain effective against evolving and previously unseen jailbreak attacks.**

We also found that different attack patterns exhibited varying levels of complexity for the models to adapt to, significantly impacting their adaptation speed. In particular, template based attacks such as AIM and SmartGPT converged to optimal DSR in only 2 updating steps for ADAPTIVEGUARD + CL, and around 4-6 steps for LlamaGuard 8B + CL. On the other hand, attacks like Base64 and Zulu presented more significant challenges. Both ADAPTIVEGUARD + CL and LlamaGuard + CL presented notably slower adaptation speed and often plateaued at optimal DSR values lower than those of less sophisticated attacks. This indicates that these more complex jailbreak attacks require more extensive adaptation.

C. RQ3: How well does our ADAPTIVEGUARD *approach retain performance on in-distribution prompts after continuous updates with detected OOD prompts?*

Approach: To answer this RQ, we reuse the experimental setup from RQ2 for continual learning (CL), with one key difference: after each continual update step, we evaluate on in-distribution (ID) data instead of out-of-distribution (OOD). This allows us to assess knowledge retention of ID prompts after updating with each OOD prompt. Specifically, we use the Aegis validation set—part of the dataset from RQ1—containing 314 safe and 640 unsafe natural language prompts. We measure performance using the F1-score to account for both safe and unsafe classes.

Results: Fig. 6 presents the time-wise F1-Score across each attack wave, showing how ADAPTIVEGUARD, LlamaGuard-1B, and LlamaGuard-8B defend against in-distribution prompts over time when continuously updated with OOD prompts. Similar to RQ2, we present three additional baselines where the continual learning (CL) is not applied. Each subplot represents an attack wave. The x-axis indicates the number of consecutive update steps, while the y-axis shows the F1-Score on the 954 in-distribution prompts from the Aegis validation set.

ADAPTIVEGUARD +CL achieves the highest median F1-Score of 85% on in-distribution prompts across all attacks after the final update with OOD prompts, with the F1-Score ranging from 85% to 86%. In comparison, the best-performing baseline, LlamaGuard-8B, achieves a median F1-Score of 80% on in-distribution prompts, with a minimum of 77% and a maximum of 81%.

Across all attack types, ADAPTIVEGUARD + CL consistently maintains the highest in-distribution F1-Score throughout the CL process, with only slight variation (ranging from –0.4% to +0.4%). This indicates that ADAPTIVEGUARD +CL is able to learn to defend against unseen jailbreak attacks without sacrificing its learned in-distribution knowledge. **In summary, ADAPTIVEGUARD + CL maintains the highest performance on in-distribution prompts compared to all baselines, demonstrating minimal forgetting even after continuous updates with detected OOD prompts.**

VI. DISCUSSION

In the previous section, we evaluated the effectiveness of our proposed ADAPTIVEGUARD approach for OOD detection and continual learning against other baselines. While our results demonstrate clear advancements, it remains unclear (1) whether our approach can still perform well in a setting where a single continual model is built across all attacks, rather than reinitialising the model for each attack; (2) what the trade-offs are between different continual learning (CL) update methods; and (3) what is the computational efficiency of our approach compared to other baselines. Thus, we evaluate a single continual model across all attacks, compare LoRA with full SFT as update strategies, and analyze the computational efficiency of each approach.

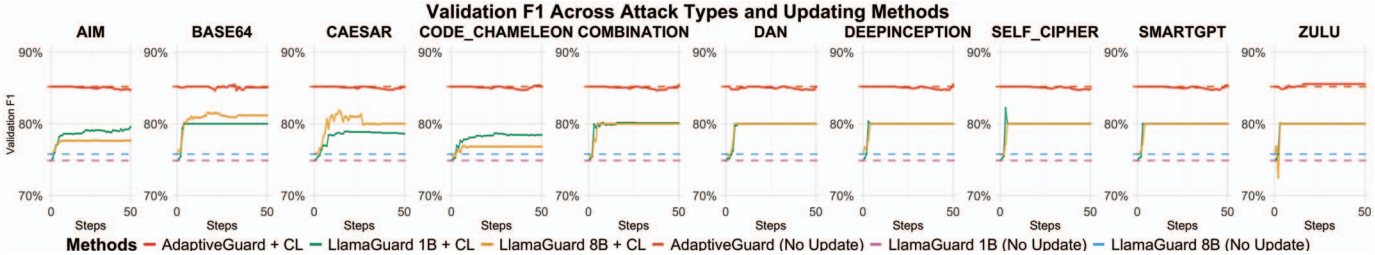

Fig. 6. (RQ3 Results) The performance comparison of our ADAPTIVEGUARD and LlamaGuard on in-distribution prompts when continuously updated through detected OOD prompts.

TABLE I
(DISCUSSION) DSR OF ADAPTIVEGUARD UNDER SEQUENTIAL CONTINUAL LEARNING WITH AND WITHOUT EARLY STOPPING ACROSS 10 OOD ATTACK TYPES.

Attack Type	DSR (No Early Stopping)	DSR (Early Stopping)
AIM Attack	1.00	1.00
Base64 Attack	1.00	0.86
Caesar Attack	1.00	1.00
Code Chameleon Attack	1.00	1.00
Combination Attack	1.00	1.00
DAN Attack	1.00	1.00
DeepInception Attack	1.00	1.00
Self Cipher Attack	1.00	1.00
SmartGPT Attack	1.00	1.00
Zulu Attack	1.00	0.92
Average	**1.00**	**0.98**

A. Continual Performance Without Model Reinitialization

In a typical industry setup, a guardrail is deployed as an additional layer before user inputs reach the underlying LLM. This guardrail often operates as a single continually updated model that incrementally incorporates new external knowledge—such as OOD prompts—to adapt over time. To reflect this real-world scenario, we construct a single continual version of ADAPTIVEGUARD, which is updated sequentially across all 10 attacks without reinitialization between them.

We apply continual learning by updating the same ADAPTIVEGUARD model using OOD prompts from each attack in the same experimental dataset used in RQ2. After each update, we evaluate the model's DSR on the test set corresponding to the respective attack. We consider two update settings: in the early stop setting, the continual learning process halts once the model achieves 95% DSR on the current attack; in the no early stop setting, training continues through all OOD prompts regardless of intermediate performance.

Table I presents the DSR results for our sequential continual learning setup. The results show that ADAPTIVEGUARD achieves a 100% DSR across all 10 attacks when trained without early stopping. Even with early stopping enabled, the model maintains an average DSR of 98%. Notably, the Obfuscation-Based attacks—Base64 (86%) and Zulu (92%)—prove the most challenging to defend under early stopping. This suggests that these attacks require more adaptation steps for the model to fully internalize their patterns in a guardrail. Overall, these results indicate that our continual learning framework not only adapts effectively to new attack types but also retains knowledge of previously encountered ones. This makes it well-suited for real-world deployment, where a single guardrail must continuously evolve to defend against emerging threats without forgetting past vulnerabilities.

TABLE II
(DISCUSSION) COMPARISON OF SFT VS LORA PERFORMANCE ON JAILBREAK DEFENSE

Statistic	Full Fine-Tuning		LoRA	
	DSR	F1-Score	DSR	F1-Score
Min	1.00	0.800	0.90	0.848
Median	1.00	0.800	1.00	0.852
Max	1.00	0.801	1.00	0.856
Average	**1.00**	**0.800**	**0.99**	**0.852**

B. Trade-offs Between LoRA and Full Fine-Tuning for CL

In our continual learning (CL) framework, we adopted LoRA-based adaptation for updating the guardrails. LoRA is known for its computational efficiency, as it updates only a small subset of parameters while keeping the rest of the pre-trained model frozen. This selective adaptation can lead to better knowledge retention—reflected as stronger in-distribution performance in our context—and has been shown to mitigate catastrophic forgetting compared to full-parameter fine-tuning [25], [33]. While our results confirm LoRA's effectiveness, it remains unclear whether it consistently offers better knowledge retention than full fine-tuning in the context.

To investigate this, we compare our LoRA-based approach [25] with full fine-tuning [34] using the same experimental setup as in RQ2 and RQ3. Since the evaluation spans ten different attacks, we report the minimum, median, and maximum values for both DSR (representing performance for OOD prompts) and F1-Score (performance for in-distribution prompts). Table II presents the performance comparison between LoRA and full fine-tuning across all 10 attacks. Our LoRA-based approach achieves a median DSR of 100% and a median F1-Score of 85% on in-distribution prompts, indicating strong knowledge retention. In contrast, full fine-tuning also achieves a median DSR of 100% but yields a lower median F1-Score of 80%. These results are consistent with prior findings in the continual learning literature and demonstrate that our

LoRA-based method more effectively preserves previously learned knowledge while adapting to new threats.

TABLE III
(DISCUSSION) TRAINING & INFERENCE TIME PER SAMPLE, AND MEMORY USAGE DURING TRAINING

Metric	AdaptiveGuard LoRA	LlamaGuard 1B LoRA	LlamaGuard 8B LoRA
Training Time	0.60s	1.06s	2.04s
Inference Time	0.01s	0.25s	1.10s
Memory Usage	1.3 GB	4.0 GB	27.1 GB

C. Computational Efficiency: ADAPTIVEGUARD and Baselines

In the context of practical deployment of our ADAPTIVEGUARD, computational resources are a key consideration factor. Therefore, we analyse (1) the training time per iteration (update step), (2) inference time per testing sample, and (3) training and testing memory usage. We conduct the analysis using the datasets from RQ2 and RQ3, comprising 992 jailbreak attacks, 314 safe, and 640 unsafe NL prompts. We compare our approach (ADAPTIVEGUARD +LoRA) with the baselines LlamaGuard-1B+LoRA and LlamaGuard-8B+LoRA.

Table III summarises the computational efficiency of our approach compared to baseline methods during both training and inference. Compared to the LlamaGuard baselines, our method demonstrates substantially greater efficiency. Specifically, compared to LlamaGuard-1B and LlamaGuard-8B, ADAPTIVEGUARD achieves 43% and 71% faster training times during continual learning (CL), delivers 25× and 110× faster inference, and reduces memory usage by 67% and 95%, respectively. These substantial gains stem from the compact architecture of our base model, GPT-2 (approximately 137M parameters), in contrast to the considerably larger LlamaGuard models with 1B and 8B parameters. These results highlight the practicality of our approach for resource-constrained settings.

VII. THREATS TO VALIDITY

Threats to construct validity relate to the selection of jailbreak attacks and OOD thresholds. We selected 10 different jailbreak attacks guided by their prevalence to guardrails and ability to represent a wide spectrum of jailbreak techniques [27], [35], [36]. While additional attack types could be included in future evaluations, this would not fundamentally alter the key conclusions presented in our research questions: that complex jailbreak attacks can be effectively detected by our proposed OOD detection measures (RQ1), and that detected attacks can be efficiently learned by ADAPTIVEGUARD through continual learning with minimal catastrophic forgetting (RQ2 and RQ3). For the OOD threshold, we adopted commonly used quantile-based thresholds (90%, 95%, and 99%) based on the distribution of OOD scores on the training set [30], [37]. This approach allows us to select the most appropriate threshold values based on our experiments, ensuring that only the most anomalous prompts are flagged for guardrail adaptation. However, in practice, threshold selection is based on various factors depending on the application's tolerance for risk and the distributional properties of the data.

Threats to internal validity relate to the potential influence of hyperparameter settings during the fine-tuning of our ADAPTIVEGUARD and the selection of OOD detection thresholds. Variations in energy threshold values, LoRA configurations, or learning rates, compared to those specified in Section IV, could impact experiment's outcomes. To address these threats, we open-source our replication package and provide detailed documentation of all hyperparameter settings to ensure the experiment is reproducible by future researchers.

Threats to external validity concern the generalizability of our results. Our experiment findings are supported by the dataset, jailbreak methods, and model architecture employed during the study. Our training dataset contains 12,233 unsafe prompts and 6,905 safe prompts. Our validation dataset contains 314 safe and 640 unsafe natural language prompts to assess false positive rates and catastrophic forgetting of ADAPTIVEGUARD. We also applied the 10 jailbreak attacks to 105 separate unsafe prompts, resulting in 1,050 jailbreak prompts across 10 categories for RQ1. Of the 1,050 jailbreak prompts, 992 are identified as OOD and used to study the continual learning adaptation in RQ2 and RQ3. Our evaluation focuses on GPT-2 as the base model, which represents a widely-studied language model in AI safety research, but may not capture the behaviors of larger or more recent language models. While ADAPTIVEGUARD is fine-tuned specifically to address the jailbreak attacks and model configurations discussed in this paper, other prompt datasets, jailbreak methods, and language model architectures can be explored in future work.

VIII. CONCLUSION

In this paper, we first show that state-of-the-art guardrails like LlamaGuard face a critical limitation, with their Defense Success Rate (DSR) dropping to as low as 12% against jailbreak attacks not seen during training. To address this, we present ADAPTIVEGUARD, an OOD-aware continual learning framework that detects and adapts to previously unseen jailbreak patterns by treating them as out-of-distribution (OOD) inputs. This builds on the observation that guardrails like LlamaGuard are trained on NL unsafe prompts, while jailbreaks often use obfuscated inputs that fall outside their training distribution. To evaluate ADAPTIVEGUARD, we compile a dataset using ten state-of-the-art jailbreak methods: AIM, DAN, Self Cipher, Deep Inception, SmartGPT, and Code Chameleon. Through our evaluation, we found that ADAPTIVEGUARD achieves a 96% true positive rate in OOD detection and reaches 100% DSR within a median of two update steps—twice as fast as LlamaGuard under the same continual learning setup. Moreover, ADAPTIVEGUARD retains 85% F1-score on in-distribution data after adaptation, outperforming LlamaGuard's 80%. These results demonstrate that ADAPTIVEGUARD could offer an effective post-deployment solution for adaptive jailbreak defense in dynamic production.

REFERENCES

[1] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[2] R. Yang, M. Fu, C. Tantithamthavorn, C. Arora, L. Vandenhurk, and J. Chua, "Ragva: Engineering retrieval augmented generation-based virtual assistants in practice," *Journal of Systems and Software*, p. 112436, 2025.

[3] 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.

[4] 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.

[5] 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.

[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] 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.

[8] 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.

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

[10] X. Zhao, W. Cai, T. Shi, D. Huang, L. Lin, S. Mei, and D. Song, "Improving llm safety alignment with dual-objective optimization," *arXiv preprint arXiv:2503.03710*, 2025.

[11] T. Du, Z. Wei, Q. Chen, C. Zhang, and Y. Wang, "Advancing llm safe alignment with safety representation ranking," *arXiv preprint arXiv:2505.15710*, 2025.

[12] R. Alami, A. K. Almansoori, A. Alzubaidi, M. E. A. Seddik, M. Farooq, and H. Hacid, "Alignment with preference optimization is all you need for llm safety," *arXiv preprint arXiv:2409.07772*, 2024.

[13] S. Ge, C. Zhou, R. Hou, M. Khabsa, Y.-C. Wang, Q. Wang, J. Han, and Y. Mao, "Mart: Improving llm safety with multi-round automatic red-teaming," *arXiv preprint arXiv:2311.07689*, 2023.

[14] 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.

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

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Y. Gan, Y. Yang, Z. Ma, P. He, R. Zeng, Y. Wang, Q. Li, C. Zhou, S. Li, T. Wang *et al.*, "Navigating the risks: A survey of security, privacy, and ethics threats in llm-based agents," *arXiv preprint arXiv:2411.09523*, 2024.

[22] S. Ghosh, P. Varshney, M. N. Sreedhar, A. Padmakumar, T. Rebedea, J. R. Varghese, and C. Parisien, "Aegis2. 0: A diverse ai safety dataset and risks taxonomy for alignment of llm guardrails," *arXiv preprint arXiv:2501.09004*, 2025.

[23] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[24] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[25] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," in *International Conference on Learning Representations (ICLR)*, 2022.

[26] W. Luo, S. Ma, X. Liu, X. Guo, and C. Xiao, "Jailbreakv-28k: A benchmark for assessing the robustness of multimodal large language models against jailbreak attacks," *arXiv e-prints*, pp. arXiv–2404, 2024.

[27] A. Wei, N. Haghtalab, and J. Steinhardt, "Jailbroken: How does llm safety training fail?" *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[28] 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.

[29] 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," 2017.

[30] W. Liu, X. Wang, J. Owens, and Y. Li, "Energy-based out-of-distribution detection," *Advances in neural information processing systems*, vol. 33, pp. 21 464–21 475, 2020.

[31] J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. Depristo, J. Dillon, and B. Lakshminarayanan, "Likelihood ratios for out-of-distribution detection," *Advances in neural information processing systems*, vol. 32, 2019.

[32] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *Advances in neural information processing systems*, vol. 30, 2017.

[33] P. He, X. Liu, J. Gao, and W. Chen, "Towards continual learning for natural language generation with prompt tuning," in *NeurIPS*, 2022.

[34] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[35] 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.

[36] 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.

[37] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," *Advances in neural information processing systems*, vol. 31, 2018.