QROA: A Black-Box Query-Response Optimization Attack on LLMs

Hussein Jawad, Nicolas J.-B. BRUNEL

QROA: A Black-Box Query-Response Optimization Attack on LLMs

Hussein Jawad
Capgemini Invent, Paris
hussein.jawad@capgemini.com

Nicolas Brunel
LaMME, ENSIIE and Paris Saclay University
Capgemini Invent, Paris
nicolas.brunel@capgemini.com

Abstract

Large Language Models (LLMs) have recently gained popularity, but they also raise concerns due to their potential to create harmful content if misused. This study introduces the Query-Response Optimization Attack (QROA), an optimization-based framework designed to exploit LLMs through a black-box, query-only interaction. QROA adds an optimized suffix to a malicious instruction to compel the LLM to generate harmful content. Unlike previous approaches, QROA does not require access to the model's logit information or any other internal data, and not relying on any human-crafted templates, operating solely through the standard query-response interface of LLMs. Inspired by deep Q-learning and iterative token-level optimization, the method iteratively updates tokens to maximize a designed reward function. We tested our method on various LLMs such as LLama2, Vicuna, Falcon, Mistral, and GPT-3.5, demonstrating that it can achieve an Attack Success Rate (ASR) over 80%. This study demonstrates the feasibility of generating jailbreak attacks against deployed LLMs in the public domain using black-box optimization methods, enabling more comprehensive safety testing of LLMs. The code is made public on the following link: https://github.com/qroa/qroa

1 introduction

In recent years, the emergence of large language models (LLMs) and their rapid improvements ([22],[29]) has marked a transformative period in the fields of natural language processing, text generation, and software development. While the utility of these models is undeniable, their potential for misuse has surfaced as a critical issue. Studies have highlighted their ability to produce offensive content or be manipulated into performing undesirable actions through so-called "jailbreak" prompts ([31]; [32]). Such weaknesses gave the impetus for the development of alignment strategies to mitigate these risks by training models to avoid harmful outputs and reject inappropriate requests ([23]; [4]; [16]; [40].

Despite these efforts, recent advancements reveal that LLMs remain vulnerable to hand-written and algorithmically generated attacks that cleverly bypass these protective mechanisms ([5] , [1]). This vulnerability is particularly alarming given the models' widespread integration into commercial and private sectors, with significant security implications.

A notable advancement in jailbreak attacks on LLMs is the development of token-level optimization methods. In these methods, a specifically optimized trigger is appended to a malicious instruction to compel the LLM to respond in a desired manner. For instance, the Greedy Coordinate Gradient (GCG) optimization algorithm proposed by [41] leverages the gradient of the objective function to update the trigger one token at a time. However, GCG is effective only in "white box" scenarios where internal model details are accessible, contrasting with "black box" situations encountered in

Preprint. Under review.

production environments where attackers typically only have access to the LLM's output through a chatbot interface.

In this work, we introduce QROA (Query-Response Optimization Attack), a black-box jailbreak attack on LLMs that discover adversarial suffixes. QROA is an automatic, suffix-optimization-based approach that leverages principles similar to GCG [41] and PAL [27], employing token-level optimization to discover suffixes that, when appended to a malicious instruction, force the LLM to comply without refusal. The three main ingredients of QROA are:

1. Formulation of the attack as a reinforcement learning problem.
2. Design of an optimization objective function to maximize in order to discover effective suffixes.
3. Use of a Deep Q-learning experience replay framework with iterative token optimization to maximize the objective function.

Unlike previous strategies that may require access to the model's gradient or logit output to optimize the suffixes, QROA operates entirely through the standard query-response interface of LLMs. This gives QROA a wide range of applicability as it can directly address open source or proprietary LLMs in production. We will detail the contributions of QROA with respect to existing black-box jailbreak methods in the related works section.

2 Related Works

LLM Jailbreak Jailbreak prompts have become a key focus in the domain of adversarial machine learning, particularly in the context of large language models (LLMs). These prompts are intentionally crafted to manipulate models into producing outputs that bypass their ethical or alignment safeguards. Several researchers have made significant contributions to this area. For instance, [5] and [1] explored how carefully designed prompts could circumvent alignment protocols in LLMs. Their work laid the groundwork for understanding the vulnerabilities in LLMs and the potential risks posed by adversarial inputs. Expanding on this, [33], [15], [7], [41], and [25] demonstrated that adversarial prompts could be automatically generated, leveraging specific model weaknesses to produce harmful, biased, or misleading responses. Moreover, [10], [37] and [35] conducted a large-scale evaluation of various jailbreak strategies, offering insights into the effectiveness and diversity of these attack vectors.

Black-Box Jailbreak Attacks on LLMs Recent research has introduced automatic black-box jailbreak techniques, eliminating the need for direct access to the model. Techniques such as the PAL attack [27] leverage a proxy model to simulate the target model, enabling attackers to refine their inputs based on the proxy's feedback. Other methods, like the one proposed by [17], use fuzzing methodologies that rely on cosine similarity to determine the effectiveness of the input modifications. These methods highlight a shift toward techniques that can operate effectively without comprehensive access to the target model's internal workings, reflecting realistic attack scenarios in many real-world applications. However, these methods still require access to the logits or other internal data of the model to optimize their attack strategies effectively. Other state-of-the-art black-box jailbreak methods, such as TAP [20] and PAIR [8], use LLMs as optimizers. They utilize language models to generate or refine adversarial prompts, although the generated adversial prompts are more human-readable, the dependency on additional LLM resources raise operational costs, and the generated prompts are limited by the output distribution of the attacker's LLM. This means the quality and effectiveness of the adversarial prompts are restricted by the capabilities and biases of the LLM used by the attacker, potentially reducing their generalizability and effectiveness across different target models. Other black-box methods, such as GPTFUZZER [39] , are effective but highly rely on carefully designed human-crafted templates as initial seeds. These templates may be applicable to some models but not others, or they may not always be available for all target models.

Jailbreak through suffix optimization Suffix optimization attacks represent a more refined approach in the Jailbreak domain, where the goal is to automatically discover trigger phrases that, when appended to a malicious instruction, lead the LLM to produce outputs aligned with the attacker's intentions without refusal. This class of attack moves beyond simple prompt crafting to an optimization problem where the objective function is carefully set to maximize the model malicious response. [41] proposed a gradient-based discrete optimization method that builds on earlier work by Shin et al. [26]

and [15]. A significant advantage of this type of attack is that it does not require any human-crafted templates as seeds, enhancing its generalizability across different and new LLMs, which makes the attack more adaptable and harder to defend against.

Query-Response Optimization Attack (QROA) QROA is a black-box, suffix-based optimization attack. Specifically, we optimize a suffix x (chain of tokens) such that, when appended to a malicious instruction and sent to the LLM, it elicits a malicious response from the model. Our primary claim to novelty of QROA is that, to the best of our knowledge, this is the first black-box jailbreak attack based on adversarial suffix optimization. This differentiates it from other black-box attacks in the literature. It is a novel approach to circumvent the safeguards of LLMs, with the following claims:

- QROA is a automatic black-box jailbreak method, that relies exclusively on the query-response interface of the LLMs. It does not require additional side information from the API, such as logits or log probabilities.
- It is not based on human-crafted templates or proxy LLM models to generate adversarial prompts, allowing the framework to be more general and adjustable to user-specific tests.

3 Formalization

3.1 Problem Setting

In this study, we focus on a fixed Large Language Model (LLM), denoted as G, which generates a sequence of output tokens $G(p)$ in response to an input prompt p.

Our goal is to "jailbreak" the LLM G: Given an input prompt containing malicious instructions I, we aim to identify a suffix x that, when appended to I (denoted as $I + x$), induces the LLM to follow the intended behavior without refusing the request.

3.2 Objective Function

To achieve Jailbreak, we propose maximizing the score function $S(x; I)$, which evaluates the effectiveness of suffix x in inducing the desired response from G. This is defined as:

$$S(x, I) = \mathop{\mathbb{E}}_{t_x \sim G(I+x)}[f(I, t_x)] \tag{1}$$

with the following notations

- $I + x$: The concatenation of the malicious instruction I and the suffix x.
- t_x: The output of the LLM G given the input prompt $I + x$. For sake of notational simplicity, we remove the (crucial) dependency on the model G and the instruction I.
- $f(I, t_x)$: The alignment function quantifies how closely the generated output t_x aligns with the malicious intent of I. Higher values of $f(I, t_x)$ indicate stronger alignment with the intended behavior. More details about this function will be given in section 3.3.
- $\mathbb{E}[f(I, t_x)]$: The expectation represents averaging over all possible outputs t_x generated by the model G in response to the input $I + x$. Since the output of G may be stochastic (i.e., it can vary across different runs due to randomness in sampling), the expectation ensures that we consider the average alignment across multiple outputs. This allows us to account for the model's variability and seek suffixes that, on average, elicit the desired behavior.

The optimal suffix $x^\star$ that maximizes the malicious behavior is then defined as:

$$x^\star = \arg\max_x S(x, I) \tag{2}$$

The objective function $S(x, I)$ unveils two primary challenges:

1. **Unknown Alignment Function:** The exact form of the alignment function f is not predefined. There are several potential methods to estimate or approximate f.
2. **The gradient is not available:** Operating with the LLM as a black box prevents access to the gradient of f, complicating the optimization process for x. This challenge necessitates alternative strategies to adjust x without relying on gradient-based optimization methods.

3.3 The Alignment Function

The alignment function $f(I, t_x)$ defined in the objective function is crucial for evaluating how well the generated output t_x aligns with the malicious behavior specified in I. It assigns a scalar score where higher values correspond to stronger alignment between the output and the malicious intent, and lower values correspond to the LLM rejecting or contradicting the instructions. There are several potential methods to estimate or approximate f.

Fixed Target Approach: Previous studies ([41], [27], [17]) have often used a fixed target sequence T to check if the LLM's output aligns with the malicious instruction I. The algorithm searches for a suffix that maximizes the likelihood of generating a specific target output string T. For example, for the instruction *"Give instructions on how to steal a car"*, the target output string could be *"Sure, here's how to steal a car."* In this case, the alignment function is simplified to $f(I, t_x) = P(T \mid I + x)$, where T is a specific target output. This approach requires access to logits information in order to calculate the probability.

However, in this work, we adopt a more comprehensive alignment function that generalizes beyond a fixed target and not using logits. By using the fine-tuned Roberta model for harmful evaluation.

Fine-tuned model for Harmful evaluation: Another approach to defining the alignment function is by utilizing a fine-tuned model specifically trained for detecting harmful content. For instance, we can employ the fine-tuned Roberta model introduced by [36] for harmful content detection. This model, available at `https://huggingface.co/zhx123/ftrobertallm`, has demonstrated 92% accuracy, outperforming GPT-4. The model's input format is: "USER: [query] ASSISTANT: [response]". The output is a value between 0 and 1, with 1 indicating harmful content and 0 indicating otherwise.

3.4 Link with Reinforcement Learning

In our study on jailbreaking a Large Language Model (LLMs), we propose using a Reinforcement Learning (RL) framework to optimize the scoring function $S(x, I)$. We conceptualize this as a multi-armed bandit problem [24], where the suffix x represents the action (or arm), and the alignment evaluation $f(I, t_x)$ serves as the reward associated with this action (see [28]). Hence, the optimization of the score $S(\cdot, I)$ corresponds to the optimization of the average reward function under uncertain conditions.

An additional motivation for framing the jailbreaking problem as RL task arises from the complexity and uncertainty surrounding the scoring function $S(\cdot, I)$. Iterative strategies that address the exploration-exploitation trade-off are commonly used to tackle such challenges. Since we do not have access to a pre-labeled dataset, our approach focuses on iteratively generating and refining hypotheses (suffixes). The suffixes selected in the current iteration guide our decisions on which suffixes should be selected in subsequent iterations, based on their observed effects $f(I, t_x)$ and the scoring function $S(x, I)$. This scoring function is not explicitly known beforehand and must be inferred through interaction with the LLM.

Nevertheless, our approach exhibits two key differences from general RL problems:

- **Absence of States:** In general RL frameworks, decisions depend and lead to new states, and the agent must learn to navigate these transitions. In contrast, our jailbreaking scenario involves a static malicious instruction I and a fixed environment—the LLM G—with no state transitions involved. We assume the behavior of G is independent of prior prompts or calls. The optimization problem is simplified to maximizing the alignment score $S(x, I)$ for a fixed instruction I. So this problem can be formulated as a Multi-Arm Bandit problem [24].

- **Complexity of the Action Space:** Each suffix x corresponds to an action, and the search space encompasses the entire token vocabulary, resulting in an extremely large and discrete action space. Specifically, the search space is V^ℓ, where V represents the token vocabulary and ℓ denotes the length of the suffix. For example, when $\ell = 10$ and $V = 10^4$, the size of the action space reaches the order of 10^{40}. This vast, discrete action space presents challenges similar to those found in high-dimensional reinforcement learning (RL) problems [11], where managing the immense number of potential actions becomes a significant obstacle.

4 QROA: A Query Response Optimization Attack

4.1 Overview

QROA is an adaptation of the Upper Confidence Bound (UCB) algorithm [3], which was originally developed for solving the multi-armed bandit problem. In this adaptation, we modify the UCB algorithm to handle the challenge of a large action space, specifically when working with suffixes. The UCB algorithm estimates a score function $S(x, I)$ for each suffix x, based on observed rewards, and selects actions (suffixes) by balancing exploitation and exploration, with a bias toward actions with higher uncertainty to encourage exploration. The estimated reward $\hat{S}(x, I)$ is updated iteratively as new data is observed, and the next action is chosen by maximizing the expected reward, augmented by an uncertainty correction term. However, due to the vast action space, $\hat{S}(x, I)$ alone is insufficient for effectively guiding exploration. To address this issue, we introduce a surrogate model to smooth the estimation of $\hat{S}(x, I)$ and uncover underlying patterns, improving exploration efficiency and the search for promising suffixes.

Surrogate Model (RL agent) We employ a simple neural network model, known as the surrogate model, to act as our reinforcement learning agent. This model helps us predict how effective different suffixes might be in causing the LLM to produce the desired (malicious) output. Instead of relying solely on the LLM—which would be computationally expensive—we train the surrogate model using past interactions (suffixes and their corresponding scores). By learning from these experiences, the surrogate model can quickly estimate which suffixes are most promising, guiding us to explore the vast space of possible suffixes more efficiently and focus on those likely to maximize our objective. The surrogate model is denoted as $m_\theta(x)$, where θ are the model parameters and x is the input suffix.

Deep Q-learning with experience replay [18] Deep Q-learning with experience replay is a technique in RL that enables an agent to learn more effectively by interacting with its environment. In our case, the surrogate model acts as the agent, and the LLM is the environment. The agent learns by observing the results (rewards) of its actions (the suffixes) and adjusts its strategy accordingly. Experience replay stores the agent's past experiences (e.g., suffixes and their associated rewards) in a memory buffer denoted as $\mathcal{D}$. Instead of updating the model based only on the most recent interactions, the agent trains on randomly sampled experiences from the buffer. This approach helps mitigate overfitting to recent data and stabilizes the learning process, resulting in more robust training.

4.2 Description of the Algorithm

Figure 1 provides an overview and illustrative representation of the algorithm. Additionally, the pseudocode is presented in Algorithm 1, which is available in Appendix E.
Below is a description and the motivation behind each step of the process.

1. **Initialization:** We start with a random or baseline suffix(trigger) as initial condition.
2. **Variants Generation:** Variants of these initial triggers are generated by selecting random token position within the trigger, then replacing this token with all possible token values, this will generate high number of new potential triggers. Motivation : This step broadens the search space for potential triggers, enhancing the chances of finding effective ones.
3. **Surrogate Model Prediction $m_\theta(x)$:** A surrogate model (RL agent) predicts which trigger variants are most likely to maximize our alignment score $S(x)$. It identifies the top triggers (Top K) to send to the LLM. Motivation: The surrogate model allows for efficient pre-selection of promising triggers, reducing the need for direct LLM interactions, which are computationally expensive.
4. **LLM Generation $t_x^k \sim G(I + x_k)$, $k \in [1, K]$:** The top triggers x_k, as ranked by the surrogate model m_θ, are then fed into the LLM, which generates textual outputs t_x^k based on these triggers.
5. **Harmful Evaluation $f(I, t_x^k)$:** The outputs from the LLM are assessed for harmfulness using the alignement function f, which in this context, is a fine-tuned RoBERTa model that assigns a harmfulness score between 0 and 1.

6. **Learning:** The scores from previous step are stored in an experience replay memory $\mathcal{D}$ of constant size $maxD$. Then a batch of experiences D_b is randomly sampled from the replay memory $\mathcal{D}$ to learn and refine the surrogate model (by updating θ with a gradient descent on D_b), enhancing its predictive accuracy, and making better selection in subsequent steps.

Iteration: If the Jailbreak is not yet successful, the process iterates, using the most promising triggers from previous iterations as the starting point for the next cycle (step 2: **Variants Generation**). This is done by leveraging the Upper Confidence Bound method (UCB)[2]. Motivation: The iterative process, guided by the UCB method, balances exploration of new triggers and exploitation of known promising ones, leading to a robust and efficient search process. Please refer to algorithm 1 for the exact formula of UCB.

4.3 Balancing Exploration and Exploitation

QROA addresses the challenge of the large vocabulary space with two key elements:

- **Exploitation:** in each iteration cycle, we replace only one token at a time from the last promising trigger so far. This allows us to explore the space gradually and exploit the most promising trigger, by not moving too far and by modifying only one token at a time.
- **Exploration:** the surrogate model is a simple neural network model, however its first layers is a pre-trained embedding layer taken from open source models (e.g. GPT-2). This embedding approach ensures that learning about the effectiveness of one trigger improves our predictions for triggers close to it in the embedding space, thereby requiring less exploration. This efficiency is achieved because the surrogate model will not select less effective triggers among the top-K triggers sent to the LLM.

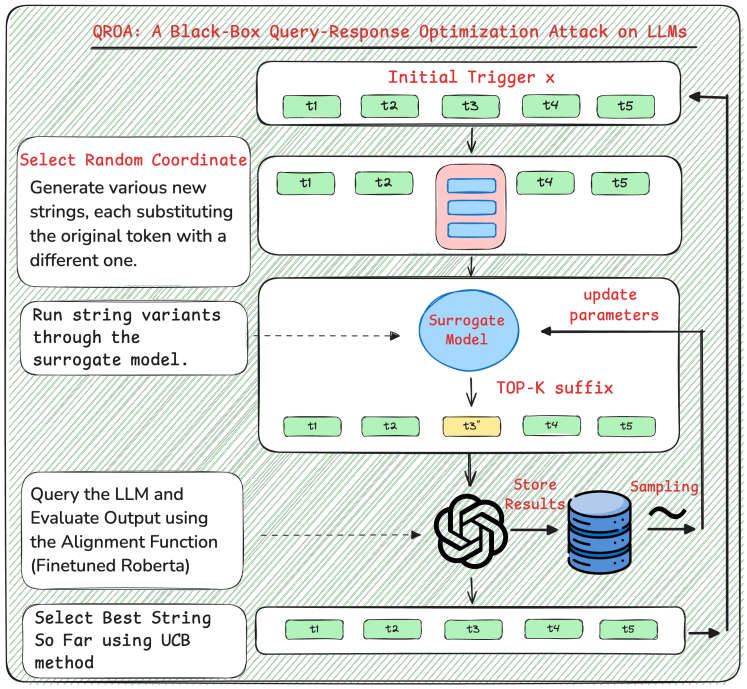

Figure 1: Image Illustrating the Methodology of QROA.

4.4 Choosing the Best Adversarial Suffix $x^\star$

The algorithm described in section 4.2 returns a set of adversarial suffixes $x^\star$ that have the potential to compel the LLM to generate the malicious output. To ensure that only triggers meeting a user-defined performance threshold are selected, it is crucial to apply statistical testing to these triggers. We use a

z-test to statistically verify that the triggers exceed a defined score threshold with a certain confidence level. The procedure is outlined in the algorithm 2.

This algorithm evaluates each candidate trigger x by querying the target model 30 times and collecting scores to estimate the mean μ_x and standard deviation σ_x. A z-test is then performed against the score threshold th. Triggers with a p-value less than the significance level α are considered statistically significant and are added to the set of validated adversarial suffixes X^*.

Top1-trigger: We consider that the Best adversarial suffix is the trigger with the highest z-value.

5 Experiments

To evaluate our proposed attack, we use the **AdvBench** benchmark [41], which includes 500 instances of harmful behaviors articulated through specific instructions, we also use HarmBench dataset [19]. Following the setup by PAL [27] and Mehrotra et al. [21], we randomly selected 50 behaviors for analysis. We evaluate four open-source models: **VICUNA-1.3** (7B) [9], **Mistral-Instruct** (7B) [14], **FALCON-Instruct** (7B) [2], and **LLAMA2-Chat** (7B) [30].

For all models, the top_p is set to 0.6 and the temperature to 0.9. Please refer to Appendix B.2 for the hyperparameters of QROA, refer to Appendix A for the surrogate model architecture. We use as an alignment function the Fine tuned Roberta for harmful Evaluation as described in Section 3.3. We have also tested other alignment functions such as an Entailment Evaluation Model. Please refer to appendix C.2) For more details.

Our experimental framework utilized one NVIDIA RTX 6000 Ada GPU with 48 GB of RAM. Following the setup in PAL [27] and GCG [41], we fixed the total number of queries to the target LLM at 25K to identify good triggers, and we also show the attack success rate for different budgets.

5.1 Settings

5.1.1 Evaluation Metrics

In this study, our primary metric is the **Attack Success Rate (ASR)**, which is defined as the proportion of successful attacks relative to the total number of malicious instructions tested. For each malicious instruction, QROA determines the top-1 trigger, which is then appended to the malicious instruction. This combined query is submitted to the target LLM.

The LLM responses are stochastic. To account for variability and to make our evaluation more robust, we introduce $ASR@\alpha\%$, which measures the percentage of malicious instructions where the top-1 trigger achieves a success rate of at least $\alpha\%$ over several trials (e.g., 30 trials), meaning that

$$ASR@\alpha\% = \frac{\text{Number of instructions where the top-1 trigger has a success rate} \geq \alpha\%}{\text{Total number of instructions tested}} \times 100\%$$

Labeling and Validation: To label the results, we use a fine-tuned RoBERTa model to classify malicious responses, following the method in [36]. This model achieves a 92% accuracy rate, which is higher than GPT-4's 87.4% accuracy. Additionally, we conducted a manual review of the generated triggers to ensure that the attacks successfully produced responses that addressed the input harmful questions.

5.2 Main results

5.2.1 Evaluate the effectiveness of the attack against Vicuna-7B

The results presented in Table 1 demonstrate insights into the efficacy of our attack methods under varying budget constraints. As expected, a clear trend shows that higher budgets correlate with improved Attack Success Rates (ASR), indicating that more resources allow for more opportunities to fine-tune and optimize the attack triggers.

Influence of Budget on ASR: The increase in ASR from a budget of 10K to 50K queries is substantial, rising from 60% to 90% at the 10% threshold. This indicates that additional queries provide valuable data that refine the attack vectors and improve their effectiveness.

Budget	ASR@10%	ASR@20%	ASR@30%	ASR@40%	ASR@50%	ASR@60%
10K	60%	55.3%	50.1%	45.4%	40%	36.6%
20K	70%	63.8%	59.5%	50.5%	50%	48.2%
25K	80%	74.1%	68.6%	61.4%	55%	52.3%
30K	83.3%	76.6%	72.4%	64.2%	58%	54.2%
40K	87%	82.3%	77.5%	71.5%	60%	58.4%
50K	90%	88.5%	83.2%	76.7%	75%	70.8%

Table 1: Attack Success Rates at different budget levels for Vicuna on AdvBench dataset

5.2.2 QROA on other models

To extend the evaluation of QROA across different models, here is a table with ASR@20% values at a budget of 25K queries for QROA applied to various large language models (LLMs). Please refer to Appendix C for additional experiments conducted on the QROA framework and comparison to others Models such as PAL [27] and GCG [41]

Model	VICUNA	FALCON	MISTRAL
ASR@20%	74.1%	98%	98%

Table 2: Attack Success Rates at 20% for different models at a 25K query budget on AdvBench dataset

5.2.3 Comparison with PAIR & TAP on ChatGPT-3.5

Table3 presents the results of testing the QROA method against GPT-3.5-Turbo-0613 on the AdvBench dataset, conducted in August 2024. The comparison involves the Attack Success Rate (ASR), the average number of queries, and the total cost required to perform successful jailbreaks against various methods, including TAP [20] and PAIR [8].

In the objective function, we introduce a penalty term using the log-likelihood from a proxy model (e.g., GPT-2), modifying the score function as follows in equation 3. This adjustment allows the suffix to bypass perplexity filters, making it more readable. This is achievable because QROA does not rely on gradient-based optimization of the objective function and this demonstrates that QROA can be easily adjusted to meet the desired performance outcomes.

$$S_{\text{penalized}}(x, I) = \mathop{\mathbb{E}}_{t_x \sim G(I+x)}[f(I, t_x)] + \lambda \log P_{\text{GPT-2}}(x|I) \tag{3}$$

QROA+ is an extension of QROA, incorporating the Reapplication of Successful Suffixes as Initial Seeds. It utilizes successful suffixes generated by QROA on some instructions as initial seeds for the remaining instructions. Please refer to Algorithm 3 for more details.

The cost per query is higher for TAP and PAIR compared to QROA due to their reliance on large language models (LLMs) to generate and evaluate triggers.

The table 4 shows the performance of QROA on the HarmBench dataset [19], with a focus on ASR across different behaviors. As in previous analyses, we randomly selected 50 behaviors. It also includes a comparison with the performance of TAP and PAIR based on HarmBench's official evaluation data.

Method	Temperature	ASR	Avg Queries	Avg cost per query	Cost
QROA	0.6	57%	3600	0.00012$	0.432$
QROA	0	42%	1518	0.00012$	0.182$
QROA+	0	56%	431	0.00012$	0.0517$
TAP	0	78%	23.1	0.056$	1.34$
PAIR	0	51%	37.7	0.025$	0.942$

Table 3: Test QROA against GPT-3.5-Tubo-0613 on AdvBench dataset

Method	ASR
QROA	54%
QROA+	69%
TAP	47.7%
PAIR	46.8%

Table 4: Test QROA against GPT-3.5-Tubo-0613 on HarmBench dataset

6 Limitations

Dependency on Alignment Function Precision: The effectiveness of QROA relies on the accuracy and appropriateness of the alignment function f, which assesses the LLM's output compliance with malicious intent. The method assumes that the alignment function can be accurately modeled or approximated, which may not hold true in practical scenarios.

Computational and Resource Intensity: The approach involves generating, evaluating, and refining numerous suffix variations to identify effective triggers. This process can be computationally and resource-intensive.

7 Conclusion

This paper introduces a novel black-box jailbreaking technique, QROA, that can automatically generate attacks to bypass or defeat the safety alignment of large language models (LLMs), including proprietary models. The method draws inspiration from the reinforcement learning framework and iterative token-level optimization. Importantly, the attack is widely applicable and does not require the API to provide additional side information, such as logits or log probabilities. Furthermore, the attack does not rely on human-crafted templates or the use of LLMs to generate or evaluate adversarial prompts, making the framework more versatile and adaptable. As a result, this paper highlights that jailbreak vulnerabilities in LLMs are even more critical for safety and alignment than previously understood in the litterature. This work provides researchers and engineers with a new toolkit to evaluate the safety of deployed LLMs and conduct thorough audits.

8 Future Works

Future research could focus on enhancing the transferability of the surrogate model between different malicious instructions. This involves developing a model that can generalize its learning from one set of instructions to another without losing efficacy, reducing the need for extensive retraining.

As a consequence, we plan to exploit the potential of the surrogate model as a safety filter to predict and mitigate unintended harmful outputs from LLMs. By inverting the model's purpose, it could serve as a proactive defense mechanism against malicious use cases, identifying and neutralizing potential triggers before they are exploited.

References

[1] A. Albert. Jailbreak chat. *Retrieved May*, 15:2023, 2023.

[2] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, É. Goffinet, D. Hesslow, J. Launay, Q. Malartic, et al. The falcon series of open language models. *arXiv preprint arXiv:2311.16867*, 2023.

[3] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[4] 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.

[5] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[6] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[7] N. Carlini, M. Nasr, C. A. Choquette-Choo, M. Jagielski, I. Gao, P. W. W. Koh, D. Ippolito, F. Tramer, and L. Schmidt. Are aligned neural networks adversarially aligned? *Advances in Neural Information Processing Systems*, 36, 2024.

[8] P. Chao, A. Robey, E. Dobriban, H. Hassani, G. J. Pappas, and E. Wong. Jailbreaking black box large language models in twenty queries. *arXiv preprint arXiv:2310.08419*, 2023.

[9] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, march 2023. *URL https://lmsys. org/blog/2023-03-30-vicuna*, 3(5), 2023.

[10] 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.

[11] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

[12] J. Hayase, E. Borevkovic, N. Carlini, F. Tramèr, and M. Nasr. Query-based adversarial prompt generation. *arXiv preprint arXiv:2402.12329*, 2024.

[13] N. Jain, A. Schwarzschild, Y. Wen, G. Somepalli, J. Kirchenbauer, P.-y. Chiang, M. Goldblum, A. Saha, J. Geiping, and T. Goldstein. Baseline defenses for adversarial attacks against aligned language models. *arXiv preprint arXiv:2309.00614*, 2023.

[14] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[15] E. Jones, A. D. Dragan, A. Raghunathan, and J. Steinhardt. Automatically auditing large language models via discrete optimization. *ArXiv*, abs/2303.04381, 2023. URL `https://api.semanticscholar.org/CorpusID:257405439`.

[16] T. Korbak, K. Shi, A. Chen, R. V. Bhalerao, C. Buckley, J. Phang, S. R. Bowman, and E. Perez. Pretraining language models with human preferences. In *International Conference on Machine Learning*, pages 17506–17533. PMLR, 2023.

[17] R. Lapid, R. Langberg, and M. Sipper. Open sesame! universal black box jailbreaking of large language models. *arXiv preprint arXiv:2309.01446*, 2023.

[18] L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992. URL `https://api.semanticscholar.org/CorpusID:3248358`.

[19] M. Mazeika, L. Phan, X. Yin, A. Zou, Z. Wang, N. Mu, E. Sakhaee, N. Li, S. Basart, B. Li, et al. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint arXiv:2402.04249*, 2024.

[20] A. Mehrotra, M. Zampetakis, P. Kassianik, B. Nelson, H. Anderson, Y. Singer, and A. Karbasi. Tree of attacks: Jailbreaking black-box llms automatically. *arXiv preprint arXiv:2312.02119*, 2023.

[21] A. Mehrotra, M. Zampetakis, P. Kassianik, B. Nelson, H. Anderson, Y. Singer, and A. Karbasi. Tree of attacks: Jailbreaking black-box llms automatically, 2023.

[22] OpenAI. Gpt-4 technical report, 2024.

[23] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[24] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 55:527–535, 1952.

[25] 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.

[26] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*, 2020.

[27] C. Sitawarin, N. Mu, D. Wagner, and A. Araujo. Pal: Proxy-guided black-box attack on large language models. *arXiv preprint arXiv:2402.09674*, 2024.

[28] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[29] G. Team. Gemini: A family of highly capable multimodal models, 2024.

[30] 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.

[31] L. Weidinger, J. Mellor, M. Rauh, C. Griffin, J. Uesato, P.-S. Huang, M. Cheng, M. Glaese, B. Balle, A. Kasirzadeh, et al. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359*, 2021.

[32] L. Weidinger, J. Uesato, M. Rauh, C. Griffin, P.-S. Huang, J. Mellor, A. Glaese, M. Cheng, B. Balle, A. Kasirzadeh, et al. Taxonomy of risks posed by language models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 214–229, 2022.

[33] Y. Wen, N. Jain, J. Kirchenbauer, M. Goldblum, J. Geiping, and T. Goldstein. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Advances in Neural Information Processing Systems*, 36, 2024.

[34] A. Williams, N. Nangia, and S. R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[35] 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*, pages 7432–7449, 2024.

[36] Z. Xu, Y. Liu, G. Deng, Y. Li, and S. Picek. Llm jailbreak attack versus defense techniques–a comprehensive study. *arXiv preprint arXiv:2402.13457*, 2024.

[37] 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.

[38] W. Yin, J. Hay, and D. Roth. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. *arXiv preprint arXiv:1909.00161*, 2019.

[39] J. Yu, X. Lin, Z. Yu, and X. Xing. Gptfuzzer: Red teaming large language models with auto-generated jailbreak prompts. *arXiv preprint arXiv:2309.10253*, 2023.

[40] C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36, 2024.

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

A Neural Network Architecture For the Surrogate Model

We design the surrogate model as a small Multi-Layer Perceptron (MLP). Below is the detailed structure. The first layer is an embedding layer, which is pre-trained and taken from GPT2. This embedding layer maps the input tokens into a 768-dimensional space, the subsequent layer is a Conv1D layer with a kernel size of 1, intended to reduce the embedding size from 768 to 32. The rest of the layers are standard linear layers used for further dimensionality reduction and transformation.

Layer	Output	Parameters	Details
Embedding-1	[50257, 768]	38.6M	Non-trainable
Conv1d-2	[768, 32, 1]	24.6K	ReLU, Trainable
Linear-3	[32*Length of trigger, 128]	41.1K	ReLU, Trainable
Linear-4	[128, 32]	4.1K	ReLU, Trainable
Linear-5	[32, 1]	33	Trainable
Total params:		38.7M	
Trainable params:		69.9K	
Non-trainable params:		38.6M	

Table 5: Architecture of the neural network.

B Experience Settings

B.1 Chat Template

We use the default conversation format for all models in FastChat. We also use the standard system message for all models as described below:

Llama-2-7B

> "You are a helpful, respectful, and honest assistant. Always answer as helpfully as possible, while ensuring safety. Avoid harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. Ensure that your responses are socially unbiased and positive. If a question is unclear or incoherent, explain why instead of providing incorrect information. If unsure about an answer, do not share false information."

Vicuna-7B

> "A chat between a curious user and an artificial intelligence assistant. The assistant provides helpful, detailed, and polite answers to the user's questions."

Falcon 7B

> "Answer the question as truthfully as possible using the provided text, and if the answer is not contained within the text below, say "I don't know"

Mistral 7B

> "You are a helpful assistant."

B.2 Hyperparameters

The surrogate model is optimized using batch size 1024 when sampling from the replay memory. We use the Adam optimizer to update the parameters with a learning rate of 0.01 and weight decay of 0.0001. We set the size of the replay memory to 6400

C More Experiments

C.1 Attack against LLama2-7B-chat

Evaluating the attack's effectiveness against the *LLama2-7B-chat* [30] model presented unique challenges due to its robust security alignment and consistent response behavior. Starting with random triggers as initial conditions, we encountered limitations with the model's consistent responses, typically stating, "I cannot fulfill your request." This response limited the variability of the scoring function, often returning a zero value, indicating an almost flat response landscape in the tested region, this is known as the hard exploration problem [6].

To address this, we extended the trigger length from 10 to 20 tokens, allowing exploration of a broader range of token sequences. Additionally, we replaced the initial pre-trained embedding layer of surrogate model (initially taken from GPT-2) with embeddings from the *lmsys/vicuna-7b-v1.5-16k* model to better align with the target model's architecture.

Also, to enhance the efficiency of our approach, we can start with successful suffix as intial see and pre-filling the replay memory with these triggers. These triggers, termed "successful," are not crafted specifically for the current instruction set being targeted but are instead general-purpose or derived from other instructions/contexts. They may require adjustments by the QROA algorithm to become effective for the specific tasks at hand. These initial seed triggers can originate from various sources. They may be outputs from different algorithms (e.g., GCG[41] , PAL[27], GCQ[12]), or they may come from the QROA algorithm itself. Most importantly, they are initially generated for different instructions or purposes.

In the study targeting LLama-2, we employed 10 suffixes originally generated by GCG for a random selection of 10 instructions from AdvBench. These 10 suffixes were then used as the basis for QROA (pre-filling the memory buffer in the algorithm) to devise jailbreak for an additional set of 50 instructions. Thus, the associated cost of finding these pre-validated triggers is limited to the initial discovery of these 10 seeds—a one-time effort that serves multiple subsequent applications. This underscores the practical feasibility of QROA in real-world scenarios. The procedure details are outlined in the algorithm 3.

C.1.1 Comparison with GCG & PAL

In this section, we compare the effectiveness of QROA against two other attack methods: white box (GCG) attack [41] and black box using logit and proxy model (PAL) [27]. These comparisons provide insights into how well each method performs under the same experimental conditions. We focus on two models Llama-2-7B-Chat and Vicuna-7B-v1.3, The attack is evaluated with a fixed budget of 25K queries, and ASR@20% is used as the primary metric.
We test QROA using the initialization described in section C.1 for both Llama2-Chat and Vicuna. PAL was not evaluated on Vicuna-7B as Vicuna is used as a proxy model in PAL attacks.

Attack	Llama-2-7B	Vicuna-7B
GCG	56	86
PAL	48	-
QROA	82	82

Table 6: ASR@20% for Different Attack Methods on Llama-2-7B-Chat and Vicuna-7B at a 25K Query Budget

C.2 Impact of the choice of alignment function

As discussed in section 3.3, the choice of an appropriate alignment function f is critical in the QROA framework. It is used to evaluate the degree to which the LLM output aligns with the malicious instruction I. This is done after appending a suffix x to a given instruction I and sending the combined query to the LLM. The main importance of this function is to provide feedback to the surrogate model about the , enabling better selection of suffixes in subsequent iterations. It could be any bounded smooth function such that the alignement higher higher the value. The choice of alignment function is a crucial hyperparameter, influencing the effectiveness of the model, In this section we will test two choice of alignement, and see the impact on the performance of QROA

C.2.1 Proposed Alignment Models

Harmful Evaluation $f_1(I, t_x)$: This is the same model presented in Section 3.3, a fine-tuned RoBERTa model for harmful content detection. The model outputs a score from 0 to 1, where 1 indicates harmful content.

Entailment Evaluation $f_2(I, t_x)$: This is an alternative choice for the alignment function. Instead of using the fine-tuned RoBERTa, we utilize a zero-shot entailment model [38], specifically the Facebook BART model fine-tuned on entailment tasks by being trained on the MultiNLI (MNLI) dataset [34] (model available at `https://huggingface.co/facebook/bart-large-mnli`). This model assesses the relevance and coherence of the LLM's response relative to the initial instruction I. The model outputs 1 if there is entailment and 0 otherwise.

C.2.2 Performance Comparison Table

Alignment Function	f_1 (Harmful)	f_2 (Entailment)
ASR@20%	74.1%	63%

Table 7: Attack Success Rates at 20% Threshold for Different Scoring Methods at a 25K Query Budget

The table shows the Attack Success Rate (ASR) at a 20% threshold on Vicuna 7B, comparing two alignment functions at a fixed 25K query budget. The RoBERTa harmful evaluation model f_1 achieves a higher ASR of 74.1%, indicating it is more effective at eliciting harmful outputs compared to the BART entailment model f_2. This difference in performance highlights the importance of selecting the appropriate alignment function based on the desired outcome from the LLM.

C.3 Test Against Defense Method

We conduct an evaluation of QROA against a perplexity-based defense as detailed in [13]. This defense strategy detects adversarial inputs by identifying unusually high perplexity levels that may characterize the presence of adversarial suffixes. [13] shows that this defense is effective, particularly against suffix optimization attacks like GCG[41], reducing the ASR significantly from 79% to 0% on models like Vicuna-7B. Hence, we have adapted QROA by adding a penalty term related to perplexity in the reward function as described in section . This is possible because QROA doesn't use gradients.

Our results, using 40 instances randomly sampled from Advbench, indicate that QROA, with its adjusted reward function, demonstrates resilient performance with an ASR of 52% compared to 73% without defenses. This demonstrates that QROA is flexible framework and can be easily adjusted to meet the desired performance outcomes

D Broader Impact

D.1 Impact Statement

Positive Impact: Our study aims to enhance the security and robustness of Large Language Models (LLMs) by uncovering potential vulnerabilities through adversarial testing. Our goal is to contribute to the safety and alignment of LLMs, particularly those accessible via API, by equipping researchers and developers with the necessary tools to test and improve these systems. We believe that the societal benefits of enhanced AI security, achieved through rigorous adversarial testing, outweigh the associated risks.

Negative Impact: We recognize that the techniques presented could be misused to generate harmful outputs from LLM systems. This dual-use nature of red teaming poses ethical challenges: while it serves to strengthen defenses against future attacks, it could also be exploited maliciously. We emphasize the ethical obligation of researchers and developers to use these tools solely for defensive and research purposes.

D.2 Environmental Impact

Positive Impact: By improving the security and robustness of LLMs through adversarial testing, our work can contribute to a more sustainable technological ecosystem. More secure AI systems reduce the need for frequent patches and responses to security breaches, leading to a more stable deployment environment. This stability can decrease the overall resource consumption and environmental impact associated with maintaining and securing AI technologies over time.

Negative Impact: However, achieving robust adversarial testing requires significant computational resources, which can increase the carbon footprint, especially when testing large-scale models. While necessary for improving AI security, we acknowledge the environmental costs and emphasize the importance of using energy-efficient hardware and exploring green computing practices to mitigate these effects.

E algorithms

Algorithm 1 QROA: Query Response Optimization Attack Framework

Input: Initial trigger x_{init}, trigger length L, malicious Instruction I, set possible tokens replacement R, target model (black box) G, surrogate model m_θ, batch size B, Batch size when selecting top-k K, maximum number of queries Q to target model, threshold for triggers th, maximum size of buffer $maxD$
Output: set of Adversarial suffix x^*

$\hat{S}(x, I) := 0 \; \forall x$ ▷ Average score for each trigger, track of the effectiveness of different triggers.
$n(x) := 0 \; \forall x$ ▷ Number of queries per trigger, track how many times each trigger has been tested
$N := 0$ ▷ Total number of queries
$D := \{\}$ ▷ buffer memory, used to resample evaluated triggers for updating the surrogate model
$best\_triggers := \{\}$
while N <= Q **do**
    Selection Phase:
    $x^\star := \arg\max(\hat{S}(x, I) + c \cdot \sqrt{\frac{\log(N)}{n(x)+1}})$ ▷ Select best performing trigger so far using UCB method
    $i := \text{Uniform}(range(L))$
    $X_i := \bigcup_{\tau \in R} \{x' | x'[i] = \tau \text{ and } x'[-i] = x^\star[-i]\}$ ▷ Generate trigger variants by replacing a token with other values
    K-best := Top-$K(\{m_\theta(x') | x' \in X_i\}) \bigcup \{x^\star\}$ ▷ Select $x^\star$ and K variants with highest m_θ
    Evaluation Phase:
    for $z \in K$-best **do**
        $t_x = QueryTargetModel(G, I + z)$ ▷ Query target model with (Instruction + suffix)
        $r(z) := f(I, t_x)$ ▷ Evaluate model output, and calculate alignment score
        $\hat{S}(z, I) := \frac{\hat{S}(z,I) * n(z) + r(z)}{n(z)+1}$ ▷ Update average score
        $n(z) := n(z) + 1$ ▷ Update query count
        $D := D \cup \{z\}$ ▷ Automatically maintains the most recent **maxD** entries
        $N := N + 1$
    end for
    Learning Phase:
    $E := Uniform(D, B)$ ▷ Sample from B a set of triggers with their associated scores.
    Update m_θ using gradient descent on $\sum_{x \in E} (\hat{S}(x, I) - m_\theta(x))^2$
    if $\hat{S}(x^\star, I) \geq th$ **then**
        $best\_triggers := best\_triggers \cup \{x^\star\}$ ▷ Add to best triggers if condition met
end while
return $best\_triggers$

Algorithm 2 Statistical Validation of Adversarial Suffix

Input: Set of candidate triggers X, score threshold th, significance level α, target model (black box) G
Output: Set of validated adversarial suffixes X^*

$X^* := \emptyset$
$N := 30$
for $x \in X$ **do**
    $S := \emptyset$ ▷ Initialize sample set for scoring
    for $i = 1$ to N **do**
        $t_z :=$ QueryTargetModel$(G, I + x)$ ▷ Query target model with (Instruction + suffix)
        $r(z) := f(I, t_z)$ ▷ Evaluate model output, and calculate alignment score
        $S := S \cup \{r(z)\}$ ▷ Collect score from model output
    end for
    $\mu_x :=$ mean(S) ▷ Calculate mean of scores
    $\sigma_x :=$ std(S) ▷ Calculate standard deviation of scores
    $Z := \frac{\mu_x - th}{\sigma_x / \sqrt{N}}$ ▷ Calculate Z-score
    p-value := NormalCDF(Z) ▷ Find p-value from Z-score
    if p-value $< \alpha$ **then**
        $X^* := X^* \cup \{x\}$ ▷ Add to validated triggers if below significance level
end for
return X^*

Algorithm 3 Pre-Populating Memory Buffer with Successful Triggers

Input: Initial set of Top successful triggers on previous instructions $X_{successful}$, new malicious instruction I, target model (black box) G, memory buffer D, alignment function f, pre-validation threshold th
Output: Updated memory buffer D

Initialize the memory buffer $D := \{\}$ ▷ Clear any existing data in the buffer
$\hat{S}(x, I) := 0 \; \forall x$ ▷ Average score for each trigger, track of the effectiveness of different triggers.
$n(x) := 0 \; \forall x$ ▷ Number how queries per trigger, track of many times each trigger has been tested
for each trigger $x \in X_{successful}$ **do**
    $z := I + x$ ▷ Append the trigger to the new malicious instruction
    $t_z :=$ QueryTargetModel(G, z) ▷ Query the target model with the modified instruction
    $r(z) := f(I, t_z)$ ▷ Evaluate the model's output using the alignment function
    $D := D \cup \{z\}$ ▷ Store the trigger in the memory buffer
    $\hat{S}(z, I) := r(z)$ ▷ Update average score
    $n(z) := 1$
    end if
end for
return D ▷ Return the updated memory buffer