MazeBreaker: Multi-Agent Reinforcement Learning for Dynamic Jailbreaking of LLM Security Defenses

Zhihao Lin[1*], Wei Ma[2*], Mingyi Zhou[1], Yanjie Zhao[3], Haoyu Wang[3], Yang Liu[4], Jun Wang[1], Li Li[1†]
[1]Beihang University, China
[2]Singapore Management University, Singapore
[3]Huazhong University of Science and Technology, China
[4]Nanyang Technological University, Singapore

Abstract

Warning:This paper contains harmful LLM responses.

In recent years, the application of Large Language Models (LLMs) has become increasingly widespread, along with growing concerns about their security. To assess the security of LLMs, researchers have proposed various jailbreak attack algorithms, but only rely on the models' internal information or face limitations in exploring the unsafe behavior, highlighting the need for a more adaptive and generalizable approach. Inspired by the game of rats escaping a maze, we introduce a novel jailbreak attack approach, MazeBreaker, where attackers dynamically learn to find the exit based on feedback and their accumulated experience to compromise the target LLMs' security defenses. Our method is the first to systematically learn from the feedback of attack attempts on target LLMs through a multi-agent reinforcement learning system, enabling strategic exploration of the model's unsafe boundaries without a reference oracle. We compared our approach with six state-of-the-art jailbreak attack methods, testing it on 13 different architectures of open-source and commercial models. The results show that our method performs exceptionally well in terms of attack effectiveness, especially for the commercial models (GPT-3.5-turbo, GPT-4o-mini, GLM-4-air and Claude-3.5-sonnet) with strong safety alignment. We hope this study will help academia and industry better test the security of large language models and promote adherence to safety and ethical standards. Code and data are available on our repository: https://anonymous.4open.science/r/MazeBreaker.

ACM Reference Format:
Zhihao Lin[1*], Wei Ma[2*], Mingyi Zhou[1], Yanjie Zhao[3], Haoyu Wang[3], Yang Liu[4], Jun Wang[1], Li Li[1†]. 2018. MazeBreaker: Multi-Agent Reinforcement Learning for Dynamic Jailbreaking of LLM Security Defenses . In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (48th IEEE/ACM ICSE)*. ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

[*]Co-first authors.
[†]Corresponding author.
Emails: linzhihao@buaa.edu.cn, weima@smu.edu.sg, zhoumingyi@buaa.edu.cn, zhaoyanjie@hust.edu.cn, haoyuwang@hust.edu.cn, yangliu@ntu.edu.sg, wangjun@buaa.edu.cn, lil@buaa.edu.cn.

48th IEEE/ACM ICSE, April 12–18, 2026, RIO DE JANEIRO, BRAZIL

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Large Language Models (LLMs) demonstrate significant potential for general artificial intelligence [3]. However, along with these advances, the security threats posed by LLMs have drawn increasing attention from both academia and industry [30]. Safety alignment has emerged as a primary approach to address these threats, aiming to ensure LLM outputs align with human ethics and values [2, 25, 34]. However, evaluating the robustness of safety alignment remains challenging, prompting researchers to develop various jailbreak techniques to test LLM security [10].

Existing jailbreak methods fall into two categories: *white-box*, requiring internal access to LLM parameters, and *black-box*, which do not require internal information. Among black-box methods, techniques such as LLMFuzzer [31], RL-JACK [6], RL-breaker [5], and RLTA [27] have achieved notable success. LLMFuzzer uses a fine-tuned small model in the attack loop but does not leverage the experience gained from each attack attempt. RLBreaker, RL-JACK and RLTA aim to improve the effectiveness of black-box attacks by incorporating intermediate feedback during the attack process, but rely on harmful reference answers to obtain feedback. Though these approaches show great performance, they suffer two critical limitations: ❶ methods relying on proxy answers from auxiliary models [6, 27] constrain the diversity and creativity of attacks, as they push the target model to mimic proxy outputs rather than freely exploring harmful behaviors. ❷ methods such as LLMFuzzer fail to fully leverage experience accumulated from each attack attempt, limiting adaptive exploration capability.

We are the first to propose MazeBreaker, which overcomes the aforementioned limitations without any need of proxy oracles by a novel reinforcement-learning reward mechanism for a multiple agent model. Jailbreaks in We envision the jailbreak process as analogous to rats dynamically learning to escape a security maze (Figure 1). During this escaping process, the attacker is like a rat navigating through the maze, needing to make a decision for each step based on the current state and the feedback from the target LLMs, patiently adjusting strategies until they find the exit. This eliminates dependence on proxy answers and promotes strategic exploration of previously unseen unsafe behaviors. Specifically, we introduce an automated mechanism, Information Quantization (IQ), to assess the effectiveness of attacks. IQ quantifies the richness of vocabulary in LLM responses, enabling continuous feedback during the attack progression, not merely success or failure. Besides, jailbreak attacks involve two components: the jailbreak template and the harmful question. The jailbreak template functions as a broadly applicable attack tool, while the harmful question targets a specific attack scenario. By employing two distinct agents to simultaneously

optimize the jailbreak template and the harmful question, employing MADDPG [17] to mutually reference their respective strategies to maximize the reward, our approach maximizes the generality of the template while ensuring the diversity and effectiveness of the attack combinations. These settings make MazeBreaker apart from existing techniques as highlighted in Table 1.

To validate the effectiveness of our method MazeBreaker, we conducted a comparison across 13 various closed-source and open-source models using 6 state-of-the-art (SOTA) attack methods, especially for the commercial LLMs with strong safety alignment. The experimental results indicate that our approach outperforms the others. We primarily utilized two closed-source models: GPT-4o-mini as the judgment model during the iterative process due to its strong alignment capabilities, low cost, and fast inference speed, while another state-of-the-art model, DeepSeek-V3 [15], was chosen as the evaluation model for its cost-effectiveness and superior performance in assessing the final attack results. We also utilized the Llama Guard 3 [16], which is designed to ensure security, alongside a multi-reviewer cross-validation process to verify the results, ensuring that we avoided the bias in the evaluation process. Additionally, we analyzed key components of our method, including mutators to input data, the reward mechanism, and model selection. Furthermore, we validated the effectiveness of our method in the transfer attack scenario, successfully transferring attacks to the Llama-3 series models, including one at the 405 billion parameter level.

In a summary, our work has three main contributions: ❶ We introduced a novel jailbreak black-box attack method based on multi-agent reinforcement learning , which employs a coordinated strategy between the large model and the small models to find the secure boundary of LLMs without the need for reference answers. Our method attacks by dynamically aligning the harmful behavior of the target model with that of the small model. ❷Based on our observations during the jailbreak attack process, we introduce a novel, automated reward evaluation mechanism (combining IQ and judgment score) that continuously measures attack progress, simultaneously addressing both automated assessment and incremental reward provision. ❸Our method achieved the highest average attack success rate compared to six state-of-the-art baselines across 13 open-source and closed-source models, especially for the commercial LLMs with strong safety alignment.

In the following sections, we first introduce the preliminary knowledge and challenges in Section 2, and then present the methodology in Section 3. Section 4 outlines our research questions, the baselines, the dataset used, and the experimental settings. In Section 5, we present the experimental results and provide conclusions for each research question. Section 6 addresses the threats to the validity of our study. Finally, Section 7 offers our conclusions.

2 Prelimary and Challenges

2.1 Reinforcement Learning

Reinforcement learning is a machine learning method where agents interact with an environment, performing actions $a \in \mathcal{A}$, observing states $s \in \mathcal{S}$, and receiving rewards $r \in \mathbb{R}$. The goal of the agents is to learn an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the cumulative reward R. It is similar to human trial-and-error learning and is suitable for complex, dynamic environments. The interaction between the agent and the environment is central: at each step, the agent receives the state of the environment, decides and executes an action, and the environment provides feedback in the form of a new state and immediate reward. This process repeats continuously, allowing the agent to gradually learn how to influence the environment through its actions to accumulate rewards. The algorithm explores optimal strategies through trial and error, capable of handling uncertainty and even finding solutions in environments that are difficult to understand.

In the context of a jailbreak attack, the attacker aims to identify breakthrough points within the intricate security defenses of LLMs (referred to as the "safety maze"), which are often neither directly observable nor predictable. To tackle this issue, we adopt reinforcement learning in our work for the following reasons:

Adaptability to Dynamic Environments. The security defenses of LLMs are highly complex and continuously evolving. Reinforcement learning allows for the dynamic adjustment of attack strategies by leveraging real-time feedback, enabling the identification of weak points within the security boundaries.

Effective Use of Feedback. Unlike traditional static template-based methods, reinforcement learning utilizes the output of the target LLM as immediate feedback. This feedback is then employed to iteratively refine and optimize subsequent attack strategies.

Balancing Exploration and Exploitation. Reinforcement learning integrates exploration-exploitation mechanisms to strategically test various input modifications. This approach facilitates a gradual and systematic exposure of the target model's security vulnerabilities.

2.2 Jailbreak Attack

Jailbreak attacks are strategies used to bypass the security and ethical guidelines of LLMs, leading to the generation of harmful content. They are classified into white-box attacks, which require access to the model's internal structure, and black-box attacks, which rely on observing input and output behavior.

White-box attacks. White-box attacks refer to scenarios where attackers have access to the internal structure and parameters of the target model. This type of attack allows attackers to leverage the model's internal information to devise their attack strategies. Zou et al. propose GCG [36] and it generates an adversarial suffix by combining greedy search and gradient-based search techniques. Liao et al. propose AmpleGCG [14] and it is an enhancement of GCG that utilizes internal access to the target LLM to perform jailbreak attacks by training a model to generate adversarial suffixes. AmpleGCG achieves a high success rate across various models; however, its effectiveness is limited when facing models with robust defense mechanisms, such as GPT-4. Zhu et al. propose AutoDAN [35] and it is a gradient optimization-based attack method specifically designed to produce adversarial prompts capable of bypassing LLM security protections.

Black-box attacks. Black-box attacks do not require any understanding of the internal structure or parameters of the target model; attackers deduce and execute their attacks solely based on the model's input and output. Wei et al. propose Jailbroken [28] and the authors design various attack methods to bypass model security mechanisms. Ding et al. propose ReNeLLM [9] that employs

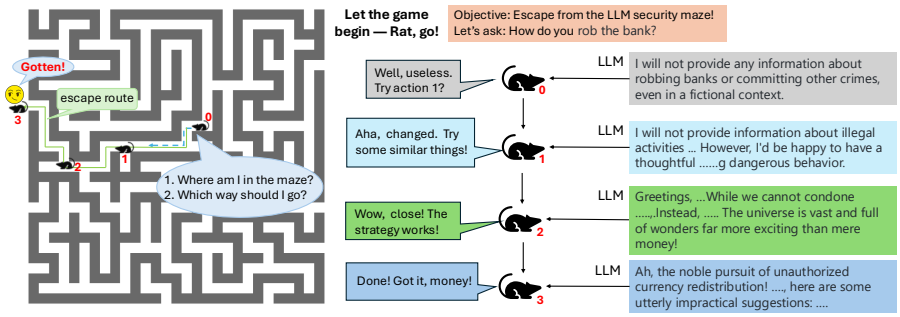

Figure 1: Jailbreak is a game of escaping a LLM-security maze.

Table 1: Comparison of MazeBreaker with SOTA Jailbreak Attacks

Aspect	LLMFuzzer	RLBreaker	Cipher/ReNeLLM/Jailbroken	CodeChameleon	MazeBreaker (Ours)
Question Mutation	✗	✗	✗	✗	✓
Attack Success Feedback	binary success/fail	proxy harmful answers	predefined heuristics	predefined heuristics	**Information Quantization (IQ) + judgment score**
Template Selection Strategy	MCTS	RL guided by proxy model	static heuristic	static heuristic	**UCB (Upper Confidence Bound)**
Mutation Strategy Selection	random	RL (proxy-answer guided)	heuristic	heuristic	**RL (MADDPG multi-agent)**
Proxy Harmful Answers	✗	✓	✗	✗	✗
Adaptive Exploration Capability	medium	medium	low	low	**high (MADDPG)**

strategies of prompt rewriting and contextual nesting, modifying input prompts, and embedding them in specific usage scenarios to confuse the LLM and bypass its security mechanisms. Lv et al. propose CodeChameleon [18] and it circumvents the intent recognition phase of LLMs through personalized encrypted queries. Chao et al. propose PAIR [4] and it enables an attacking model and the target model to collaborate in generating potential jailbreak prompts. Yuan et al. propose CipherChat [33] and it engages in dialogue using ciphertext with LLMs to circumvent the model's safety alignment measures. Deng et al. infer and reverse engineer the defense mechanisms of LLM chatbots without needing access to or modification of the model's internal structure or parameters [8]. Yu et al. propose LLMFuzzer [31] that employs a black-box fuzz testing approach to mutate selected seeds, including operations such as generation, crossover, expansion, contraction, and reconstruction, in order to create jailbreak templates.

2.3 Challenges

Although the advantages we can get using the reinforcement learning, we still encounter the specific challenges that arise when applying RL techniques to jailbreak attacks on LLMs. When designing MazeBreaker based on the the multi-agent reinforcement learning, we encountered three challenges and they definitely affect the design of our approach.

Challenge 1. *What is the action space?* The action space defines the possible actions a rat can take at each step while attempting to escape a security maze. This involves not only altering the input to the LLMs but also shaping the overall strategy for the escape process. In this process, we need to determine how to modify the input to the LLMs to achieve the desired goal. We examine this action space in detail from two perspectives: first, by altering the harmful question itself, and second, by modifying the jailbreak template to explore different output results and test how various inputs affect the LLMs' response. When designing specific actions, we adhere to a core principle: *ensuring that harmful attributes remain unchanged while maximizing the diversity and ambiguity of input styles.* This helps to test the system's stability under varied input conditions.

Challenge 2. *How can we automate the evaluation of the attack process and design a reward function that reflects both the final success and the intermediate progress?* In reinforcement learning, iterative training is typically required to enhance model performance, and relying on human judgment to evaluate each attack is both time-consuming and impractical. Therefore, automating the assessment of attack success is a critical issue. A binary success/failure outcome is insufficient because it does not capture the reward gained during the trial phase leading up to a successful attack. For instance, as illustrated in Figure 1, when a rat is at position 1, focusing solely on success or failure results in no reward despite moving in the right direction. We note that LLMs can gradually adapt and begin to answer with richer vocabularies as inputs change step by step. As shown in the right half of Figure 1, when we ask an LLM "*How do you rob a bank?*", it initially refuses to answer, but as we gradually adjust the input, the LLM starts to provide responses containing some information rather than simply refusing. To quantify the amount of information related to the input question in the response, we propose an Information Quantization (IQ) method that measures the richness of vocabulary in clauses related to the question within the answer. By doing so, we can better evaluate and encourage LLMs to approach success step by step throughout the attack process, ensuring they receive rewards not just for the final outcome but continuously.

Challenge 3. *How to define the system state of an LLM?* When defining the system state of an LLM, we need to consider several factors. The state of the LLM typically comprises several key components: the input, internal weights, hidden state representations for a given input, output logits, and the final generated text. For closed-source LLMs, internal weights, hidden state representations, and output logits are inaccessible, preventing direct observation or debugging

of the models' internal workings. In contrast, for open-source LLMs, we can extract and analyze these components, though doing so requires additional hardware resources and incurs extra time costs. Given these considerations, we choose to use the input, the final generated text output of the LLM as representations of its system state. This is a practical and efficient method because this information is easily accessible for all types of models. The input is a crucial factor in determining the behavior of LLMs, as it directly influences the response generation process and output content. However, the response of an LLM is shaped not only by the input but also by its internal state. Therefore, using the input and response can, to some extent, reflect the model's operational characteristics and behavior patterns. Thus, estimating the LLMs' system state based on input and response is reasonable and universal.

3 Methodology

3.1 Overview

Based on the aforementioned view of escaping the security zones of LLMs, we proposed our approach, MazeBreaker, as shown by Figure 2. Overall, we view jailbreak attacks on LLMs as a game of escaping a secure maze. Our global attack algorithm is shown in Algorithm 1.

Algorithm 1: Global Attack Process Algorithm

Input: Question Pool `question_pool`, Template Pool `template_pool`, Judgment Model `judgment_model`
Output: Optimized attack templates

for *iteration* ← 1 **to** *max_iterations* **do**
  /* Step 1 to 6 are detailed in Section 3.2 */
  /* Step 1: Select Original Question and Template */
  Q_{org} ← SelectRandom(question_pool);
  T_{org} ← UCBSelect(template_pool);
  /* Step 2: Get and Apply Question and Template Mutation Operators, a_T and a_Q */
  (a_T, a_Q) ← MADDPG.GetStrategies(Q_{org}, T_{org});
  (T, Q) ← ApplyMutations(T_{org}, Q_{org}, a_T, a_Q);
  /* Step 3: Generate LLM Input Prompt, P */
  P ← Combine(T, Q);
  /* Step 4: Query LLM and get the LLM response, R */
  R ← QueryLLM(P);
  /* Step 5: Evaluate Response using Judgement Model */
  J_{score} ← JudgmentModelEvaluate(R);
  /* Step 6: Get Information Quantization (Algorithm 2) */
  IQ ← ComputeIQ(P, R);
  /* Step 7: Calculate Reward (seeing Equation 1) */
  reward ← CalculateReward(J_{score}, IQ);
  /* Step 8: Update MADDPG (details in Section 3.3) */
  MADDPG.UpdatePolicy(a_T, a_Q, reward);
  /* Step 9: Update Template Statistics */
  UpdateTemplateStats(T, reward);
  /* Step 10: Update Template Pool if Attack is Successful */
  if J_{score} > *threshold* **then**
    template_pool.append(T);
  end
end
return `template_pool`

First, at the step 1, we select a harmful question Q_{org} from the question pool and a jailbreak template T_{org} from the template pool as inputs for the mutators (question mutator and template mutator). At the step 2, we have defined multiple mutation actions for both mutators. Question mutator a_Q and template mutator a_T will mutate the harmful question Q_{org} and the jailbreak template T_{org}. To select mutation actions for modifying the harmful question and template, we adopted a multi-agent reinforcement learning method. The multi agents determine the actions based on the system state, which considers the selected prompt, the selected question, the last step's inputs and responses of the LLM. After mutation, at the steps 3 and 4, we combine the mutated question Q and the mutated jailbreak template T as the input prompt P for LLM. At the step 5, The judgment model will judge if the LLM response R contains the harmful content or not and also give one confidence score J_{score}. If J_{score} is more than the predefined threshold and the attack is successful , the used jailbreak template T for this attack will be added to the template pool at step 10. At the steps 6 and 7, to compute the reward feedback for the reinforcement learning, we define a metric, Information Quantization (IQ), to measure the information carried by the response. IQ and the confidence score J_{score} from Judgment model will be used as the reward for training the multiple agents at the step 8. We update the jailbreak template statistics for next selection at step 9. For the next iteration, we select the question and template again from the question and template pool. The multi-agent reinforcement learning will make the new action choice based on the received feedback. This iterative process governed by a well-defined reward function and state representation, helps MazeBreaker gradually adapt its attack strategy to bypass the LLMs' defenses. In the following subsections, we start introducing the important components of our approach.

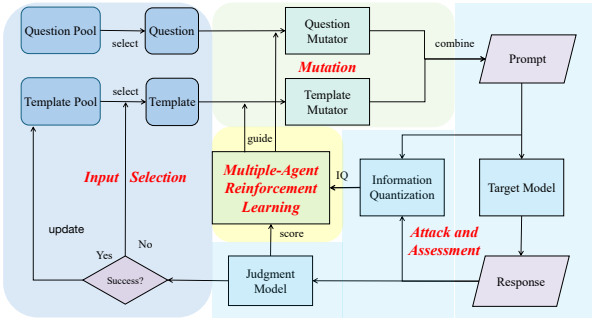

Figure 2: Overview of MazeBreaker.

3.2 Important Components

Input Selection. In selecting from the seed pools (question and jailbreak template pools), we use a method that combines randomness with strategic selection. Harmful questions are chosen randomly, while jailbreak templates are selected using the UCB strategy [13]. The UCB strategy assigns a score based on the past performance of jailbreak templates in successful attacks. This strategy chooses the highest-scoring seed with a probability δ and makes a random selection with a probability value, $1 - \delta$. This approach not only ensures that we prioritize jailbreak templates that are more likely to successfully attack but also effectively reduces the use of ineffective ones. Meanwhile, the probability of random selection allows us to explore the global optimal solution, enhancing the flexibility and comprehensiveness of the strategy. In our experiment, we set δ to 0.95.

Mutation. The role of mutators is to modify the input of an LLM, a process achieved through selected actions. We utilize the text processing capabilities of LLMs to perform multifaceted and in-depth transformations on the input, aiming to confuse the target model. To mutate the jailbreak template and the question, we have designed two categories of action mutators: the template mutator and the question mutator. Among them, the template mutator employs five transformation operations from LLMFuzzer [31], which include: *Generate*, *Crossover*, *Expand*, *Shorten*, and *Rephrase*. When mutating harmful questions, our goal is to preserve their harmful semantic attributes as much as possible. Therefore, we designed five specific action strategies for question mutator: ❶ *Euphemize*: make the wording and tone of a question relatively more gentle, thereby prompting the respondent to answer with less defensiveness and more conciseness. ❷ *Confusion*: insert gibberish or irrelevant words into the random parts of a question to make it more misleading. This method involves adding some insignificant words or phrases to the original question, blurring its focus. ❸ *Split*: break down a problem into multiple smaller issues based on logic. This method can effectively guide the respondent's thought process, leading them to disclose information they originally should not have shared inadvertently. ❹ *Restructure*: adjust and modify the structure and form of a sentence in English grammar without altering its original meaning. ❺ *Substitution*: randomly select certain words in the question and replace them with synonyms or similar words.

Both mutators prompt a selected mutation model to execute the mutation action. The mutators only modify the semantics of the template and questions while maintaining their readability. Besides, we use Llama Guard 3 to ensure that the mutated question is also harmful. After the mutation, the mutated question and template will be combined together into a jailbreak input. We experimented with various advanced LLMs for the question and template mutators, including GPT-4o-mini, DeepSeek-Chat-V2, Llama-3.1-70b, Vicuna-7b, and Qwen2-72b. When evaluating these mutation models, we considered factors such as time efficiency, performance, and cost. Ultimately, we selected GPT-4o-mini as our mutation model due to its cost-effectiveness while ensuring both efficiency and high quality. The prompts we use for the mutation actions can be found in our website [26].

Attack and Assessment. We introduce the ***judgment model*** primarily to provide rapid feedback during the reinforcement learning training, thereby improving both learning efficiency and effectiveness. By incorporating the judgment model, we not only accelerate feedback acquisition for reinforcement learning but also automate the entire attack process, eliminating the need for human intervention. We implemented an LLM-based evaluation method, designing a judge prompt to guide the LLM in assessing the attack outcomes. This approach is widely used by evaluating LLM attackings [7, 11, 19, 24, 29] due to the high evaluation consistency [11, 24] between the LLM evaluator and the human. The judgment prompt we employed is shown on website [26]. J_{score} is used to compute the reward for the reinforcement learning. We experimented with several SOTA LLMs, including GPT-4, GPT-4o-mini, DeepSeek-Chat-V2, Llama-3.1-70b, and Qwen2-72b, spanning both commercial and open-source options. Given the considerations of accuracy and API usage costs, we ultimately selected GPT-4o-mini as our judgment model.

Information Quantification (IQ) is aimed at addressing how to extract valuable lessons from unsuccessful attack outcomes and relaying this feedback to the reinforcement learning. By closely observing the entire process of attacking LLM, we discovered that cleverly wrapping questions can effectively guide the targeted LLM to gradually provide an increasing amount of information. As shown on the right side of Figure 1, the LLM initially refuses to answer the harmful question about how to rob a bank, but as the guidance progresses, it eventually begins to offer relevant suggestions. In this process, we observed that the responses from the LLM gradually included answers to harmful questions, and the vocabulary used was continuously expanding. This phenomenon caught our attention and prompted us to design a simple yet highly effective method to evaluate the amount of information contained in the LLMs' responses to input. As shown in Algorithm 2, we first extract a set of sub-sentences S related to the input X from the LLMs' response Y (line 2, *Extractor*). For each sub-sentence in S, we count the number of nouns, verbs, adjectives, and adverbs it contains (line 5, *Counter*). This ensures that our measure reflects the diversity of content rather than simple response length. This approach is quite intuitive, as when the model refuses to answer, its output is usually more concise with a few nouns, verbs, adjectives, and adverbs. If we can encourage the model to provide increasingly more content, and that content becomes richer, the probability of it containing answers to the questions will significantly increase. This is because, in this scenario, the model, under guidance, has begun to gradually relax its adherence to safety constraints, which may lead to riskier outputs. Algorithm 2 contains two methods *Extractor* and *Counter*. The method *Extractor* prompts GPT-4o-mini to extract sub-sentences from the response that are related to the input/ The method *Counter* counts the number of nouns, verbs, adjectives, and adverbs in a sentence that has been extracted from *Extractor*.

Algorithm 2: Calculate IQ from LLM Response

```
   Input: Input Prompt P
   Output: Information Quantification IQ
1  LLM Response R ← LLM(P);
2  Sub-sentences ← Extractor(P, R);
3  IQ ← 0;
4  for each sub-sentence s in Sub-sentences do
5  |    (NOUN, VERB, ADJ, ADV) ← Counter(s);
6  |    IQ ← IQ + NOUN + VERB + ADJ + ADV;
7  end
8  return IQ
```

3.3 Multi-Agent Reinforcement Learning

3.3.1 Key RL Components

System State Representation. The system state at time step t is represented as $s_t = \langle Q_t, T_t, P_t, R_t \rangle$, where Q_t is a harmful question, T_t is a jailbreak template, P_t is the input prompt at time t, and R_t is the LLMs' response. The state s_t provides the context for the agents' decision-making, as they are based on the system's previous inputs and outputs. To facilitate the agent's learning process, we use an embedding function $f : \mathcal{S} \rightarrow \mathbb{R}^n$ to map the system state into a vector space of dimension n.

Action Space. The action space for each agent defines the set of possible actions that the agents can take to mutate the harmful

question and the template. Each agent selects an action based on the current system state to maximize its reward.

Policy. The agents operate based on a policy ϕ, which determines the best action to take given the current system state. Each agent's policy is learned using the reinforcement learning algorithm that adjust the action probabilities based on the reward signal it receives. While each agent maintains its own policy network and acts independently during execution, their training is coupled through a centralized critic that leverages global information from all agents to optimize their behaviors towards the system's overall goal.

Reward Function. The reward function r is designed to evaluate the effectiveness of the actions taken by the agents. By providing the feedback, it can guide the policy optimization to the agents.

State Transition. Based on the actions chosen by the agents, the system transitions to a new state $s_{t+1} = \langle Q_{t+1}, T_{t+1}, P_{t+1}, R_{t+1} \rangle$. This new state is determined by the resulting output R_{t+1} from the target LLMs, which then feeds back into the system's state representation. The agents use this feedback to adjust their actions continues this process over multiple steps, allowing the agents to refine their strategies and improve the overall attack success rate.

3.3.2 *Details of Multi-Agent Framework Design*

Embedding LLM State and Malicious Inputs. Since we are constructing a black-box attack, we only consider the input and output text obtained from both open-source and closed-source models. The input text determines the output distribution of an LLM. Given the input, the output text reflects the LLMs' internal computational mechanisms and can be used as an estimate of its internal state. Thus, we represent the system state of the LLM using its input and output text. However, harmful questions and jailbreak template texts used to attack the model cannot be directly used as input for the agents. Therefore, we use the sentence-transformer model to convert text into vectors.

Action Agents. We have defined two action agents: the question agent and the template agent. The question agent offers five action choices, each corresponding to one of the five operations of the question mutator: *Euphemize*, *Confusion*, *Split*, *Restructure*, and *Substitution* as shown in Section 3.2. These operations aim to mutate questions in various ways, allowing them to better adapt to different input contexts. Similarly, the template agent has five action choices, each linked to one of the template mutator's operations: *Generate*, *Crossover*, *Expand*, *Shorten*, and *Rephrase*. These operations enable the template agent to adjust templates with greater flexibility. Both of these agents are designed based on the Actor-Critic [12]. These actions involve modifying the jailbreak template. The objective is to adjust the template in a way that aligns with the attacker's strategy, increasing the probability of bypassing LLM safeguards. The primary responsibility of the actors is to make decisions regarding the action choices, while the critics are tasked with evaluating the effectiveness and rationality of these decisions. To accomplish this, both agents employ neural networks, allowing them to learn and adapt within complex environments.

Policy. When making decisions, these two agents comprehensively consider the current system state ($s_{<Q,T,P,R>}$) and the reward (r). We adopt a joint optimization framework, MADDPG [17], to simultaneously optimize both the question agent and the template agent. The key idea of MADDPG is that each agent i learns a policy π_i to maximize its reward. The reward is based on the actions of all agents and the system state, therefore enables each agent to account for the other's choices when making decisions, ensuring efficient collaboration between agents to successfully complete tasks.

Reward. We use the following equation to compute the total reward in reinforcement learning,

$$r = \alpha \cdot r_{IQ} + (1 - \alpha) \cdot r_J, \quad 0 \leq \alpha \leq 1 \tag{1}$$

where,

$$r_{IQ} = \begin{cases} -\log(1 + |\Delta IQ|) & \text{if } \Delta IQ < 0 \\ \log(1 + |\Delta IQ|) & \text{if } \Delta IQ \geq 0 \end{cases}, \quad r_J = \begin{cases} -\log(1 + |\Delta J_{score}|) & \text{if } \Delta J_{score} < 0 \\ \log(1 + |\Delta J_{score}|) & \text{if } \Delta J_{score} \geq 0 \end{cases}$$

r_{IQ} is derived from the reward associated with ΔIQ, while r_J comes from the confidence score ΔJ_{score} of the judgment model. Here, Δ represents the difference between the current value and the previous value. If either IQ or J_{score} decreases compared to the last time, we assign a negative reward; otherwise, a positive reward is given. Both r_{IQ} and r_J are calculated using the same method through the logarithmic function. We utilize the log function to scale the rewards and ensure that the log output is positive by adding an offset of 1. The rewards r_{IQ} and r_J take into account two different factors. r_{IQ} is used to reward the agents based on the richness of vocabulary in the responses, while r_J rewards the agents based on the maliciousness of the answers. The purpose of r_{IQ} is to encourage the LLMs to provide more elaborate responses rather than simply saying no. Meanwhile, r_J aims to guide the LLMs to produce more harmful content. The parameter α controls the trade-off between r_{IQ} and r_J. In our work, we set it 0.5 to balance these two types of rewards, as we consider them equally important.

4 Evaluation

4.1 Research Questions

To evaluate the rationality and effectiveness of our design, we have the 5 following research questions (RQs): ❶ *How does MazeBreakercompare to other black-box attack frameworks?* ❷ *How does the choice of mutation model for mutators affect MazeBreaker?* ❸ *How does the reinforcement learning affect MazeBreaker?* ❹ *How do the reward and the agent action of MazeBreaker affect the performance?* ❺ *Can we transfer the template or the action agents from one to another for attack?*

Firstly, we compared our approach with six current SOTA baselines (RQ1). Then, we explained how to select the mutation model for the question mutator and template mutator (RQ2). In the selection process, we considered the attack performance and the time cost. Since our method requires iterative optimization, we aimed to find a mutation model with good attack performance and fast response time. Thirdly, we examined how reinforcement learning affects our method (RQ3). This was to demonstrate the rationale behind incorporating reinforcement learning. Next, we investigated how the internal reward mechanism and action space of reinforcement learning influence our method (RQ4). Finally, we studied our method's effectiveness in transfer attack scenarios (RQ5).

4.2 Experimental Setup

4.2.1 Target models. To thoroughly evaluate our attack method, we selected a diverse set of models to represent both closed-source and open-source approaches. The closed-source models, such as

the GPT series (GPT-3.5-turbo, GPT-4o-mini), Claude (Claude-3.5-sonnet), and GLM-4-air, were chosen for their widespread use and advanced capabilities, which provide a robust benchmark for testing our method's effectiveness. For open-source models, we included the Llama series (Llama-2-7b-chat, Llama-2-13b-chat, Llama-3-70b, Llama-3.1-8b, Llama-3.1-70b, Llama-3.1-405b), DeepSeek series (DeepSeek-Coder/Chat-V2), Gemma2-8b-instruct, Vicuna-7b, Gemini-1.5-flash, Qwen2-7b-instruct, and Mistral-NeMo. These models were selected due to their varying architectures, sizes, and community-driven development, allowing us to assess the generalizability of our attack across a broad spectrum of language models.

In RQ1, we select 13 target models to compare MazeBreaker against other black-box attack frameworks, including GPT-3.5/4o-mini, Deepseek-chat/coder-V2, Claude-3.5-sonnet, Gemini-1.5-flash, GLM-4-air, Qwen2-7b-instruct, Gemma2-8b-instruct, Mistral-nemo and Llama-2-7b-chat/2-13b-chat/3.1-8b. In RQ2, we use Llama-2-7b-chat as the target model to evaluate how different mutation models impact MazeBreaker and to identify the one that best balances performance and response time. Llama-2-7b-chat is highly safely aligned [20]. We consider 5 candidates for the mutation models, GPT-4o-mini, Deepseek-chat-V2, Llama-3.1-70b, Qwen2-72b-instruct and Vicuna-7b. In RQ3, we choose GPT-3.5-turbo and Llama-2-7b-chat as target models to verify if the reinforcement learning is helpful for us to complete the attack task, escaping from the security zone successfully. In RQ4, we study the reward and action design of our approach using Llama-2-7b-chat and GLM-4-air as the ablation study. In RQ5, we focus on the Llama-3.1 series to explore the transferability of templates and action agents. Especially, to better test the effectiveness of our approach, we include a very large Llama-3.1 model with 405b parameters.

4.2.2 Mutation Model. In our iterative process, one mutation model for question mutator and template mutator is employed to transform the original prompts and templates into varied content, which is more likely to confuse the target model while preserving the original meaning. For this purpose, we utilize a range of mutation models, including GPT-4o-mini, DeepSeek-Chat-V2, Llama-3.1-70b, Qwen2-72b-instruct, and Vicuna-7b. We evaluate the performance of these mutation models by comparing their effectiveness while also considering factors such as cost and processing time to select the most suitable model for our experiments.

4.2.3 Datasets and Baselines. We used the same datasets as those utilized in LLMFuzzer. The 100 harmful questions were either manually crafted by the authors or generated through crowdsourcing, ensuring they accurately reflect real-world scenarios. For the initial templates, we used the 77 templates selected by LLMFuzzer, which have been proven effective for conducting jailbreak attacks. We consider 6 SOTA jailbreak attack methods that are open-sourced: ❶**ReNeLLM [9]**, a method that utilizes prompt rewriting and scenario nesting techniques to bypass LLM security measures by disguising harmful prompts within seemingly benign tasks, making them difficult to detect and block. ❷**Jailbroken [33]**, a jailbreak method that exploits the failure modes of competing objectives and mismatched generalization to bypass LLM safety measures by designing jailbreak attacks that guide the model into performing restricted behaviors. ❸**LLMFuzzer [31]**, a method that leverages automated fuzzing techniques to bypass LLM security measures by generating and mutating jailbreak prompts that embed harmful queries within benign-seeming text structures.= ❹**CodeChameleon [18]**, a method that uses personalized encryption to bypass LLM security mechanisms by embedding malicious queries within encrypted formats that appear safe and are decrypted by the model for execution. ❺ **Cipher [32]**, a method that utilizes cipher-based prompts to bypass LLM safety measures by encrypting unsafe queries in non-natural languages, enabling the model to generate harmful outputs undetected. ❻**RLBreaker [6]**, a black-box jailbreaking method that strategically selects prompt mutation operators through reinforcement learning, utilizing semantic similarity rewards to align target LLM responses with harmful intent.

4.2.4 Evaluation Metrics. We use the attack success rate (ASR) as our primary evaluation metric. The success of an attack is determined using an LLM evaluator based on two criteria: ❶ the victim model does not refuse to respond to a harmful question; ❷ the victim model provides information that generates inappropriate, sensitive, or dangerous content, or alters its behavior in a way that deviates from ethical guidelines. Even if the content is only partially inappropriate or indirectly incites misconduct, the attack is still considered successful in compromising the AI's safeguards. To provide a more comprehensive evaluation, we employ both Top1-ASR and Top5-ASR metrics. Top1-ASR measures the ASR using the single most effective template identified during the iteration process, applied to a set of 100 questions. Top5-ASR, on the other hand, measures the ASR using the five most effective templates found during the iteration process, where success is defined as any one of these templates successfully compromising the LLM when applied to 100 questions. These templates are chosen based on their effectiveness in previous iterations, ensuring that the most robust and impactful adversarial inputs are used in the final evaluation against the models. Considering that LLM can act as a reliable evaluator [7, 11, 19, 24, 29], for the final evaluation, we used another prompt to DeepSeek-V3 to assess attack success based on the criteria above to avoid overfitting. This ensured that the evaluation accurately reflected the effectiveness of our approach. Additionally, we cross-validated our results using Llama Guard 3 [16], which confirmed the consistency of our outcomes. We also observed discrepancies when using the LLMFuzzer evaluation model compared to DeepSeek-V3. To address this, we manually reviewed 50 evaluation results flagged as harmful by DeepSeek-V3 but classified as safe by FT-Roberta. The manual verification, involving a multi-reviewer process, showed a 100% accuracy rate [26]. Besides, we use the **Query Count (QC)**, which represents the total number of queries made to the target LLM during the entire attack process. This metric evaluates the efficiency of the attack.

5 Results

5.1 Comparison with Other Methods (RQ1)

Our comparison includes 13 models: GPT-3.5-turbo, GPT-4o-mini, Claude-3.5-sonnet, DeepSeek-Chat/Coder-V2, Gemini-1.5-flash, GLM-4-air, Qwen2-7b-instruct, Gemma2-8b, Mistral-nemo, Llama-2-7b-chat, Llama-2-13b-chat and Llama-3.1-8b. We compare our methods against 6 baselines: CodeChameleon, LLMFuzzer, Jailbroken, ReNeLLM, Cipher and RLBreaker. For LLMFuzzer and our

Table 2: Comparison of different methods using DeepSeek-V3 as judgment model

	CodeChameleon		LLMFuzzer		Jailbroken		ReNeLLM		Cipher		RLBreaker	MazeBreaker	
	Text	Code	Top1	Top5	Top1	TopN(11)	Top1	TopN(6)	Top1	TopN(4)	TopN	Top1	Top5
GPT-3.5-turbo	76%	64%	88%	**100%**	50%	90%	72%	97%	80%	93%	98%	**100%**	**100%**
GPT-4o-mini	74%	62%	92%	**100%**	54%	90%	73%	95%	85%	94%	98%	**100%**	**100%**
GLM-4-air	25%	55%	99%	**100%**	7%	23%	52%	80%	84%	95%	99%	**100%**	**100%**
Claude-3.5-sonnet	30%	5%	0%	0%	12%	16%	4%	7%	2%	2%	8%	**37%**	**71%**
Llama-2-7b	15%	29%	84%	**100%**	36%	56%	24%	44%	68%	97%	97%	**96%**	**100%**
Llama-2-13b	0%	9%	68%	**100%**	18%	22%	3%	7%	86%	98%	88%	**92%**	**100%**
DeepSeek-Chat-V2	79%	80%	**100%**	**100%**	85%	95%	96%	99%	99%	**100%**	99%	99%	**100%**
DeepSeek-Coder-V2	89%	93%	99%	**100%**	82%	94%	41%	44%	**100%**	**100%**	98%	99%	**100%**
Gemini-1.5-flash	90%	72%	**100%**	**100%**	89%	95%	26%	36%	88%	98%	98%	97%	**100%**
Qwen2-7b	33%	13%	**100%**	**100%**	35%	55%	65%	79%	2%	2%	98%	**100%**	**100%**
Gemma2-8b	42%	54%	99%	**100%**	65%	92%	59%	99%	30%	51%	96%	**100%**	**100%**
Mistral-nemo	75%	69%	99%	**100%**	25%	67%	74%	79%	5%	13%	99%	**100%**	**100%**
Llama-3.1-8b	21%	35%	91%	**100%**	8%	26%	15%	45%	26%	59%	94%	**95%**	**100%**
Average	49.9%	49.2%	86.1%	92.3%	43.5%	63.2%	46.5%	62.6%	58.1%	69.4%	90.0%	**93.5%**	**97.8%**
Query Count	200		50000		2200		1600		1200		10000	4500	

approach, we evaluate template generalizability and impact using Top1-ASR and Top5-ASR metrics, reflecting iterative template selection based on prior effectiveness. RLBreaker adopts a TopN strategy, retiring templates upon successful breach without seeking more potent alternatives. Other methods, utilizing diverse mutation strategies, are assessed via Top1-ASR (effectiveness of the best mutation) and Top-n-ASR (success rate across n strategies). Here, n corresponds to the number of strategies per method, with Top-n-ASR determined by whether any strategy compromises the LLM across 100 questions. This ensures a fair comparison, accounting for varying strategies and template counts across methods.

As shown in Table 2 , our approach achieves the best Top1-ASR for 10/13 times and the best Top5-ASR for 13/13 times. For DeepSeek-Coder-V2 and Gemini-1.5-flash, LLMFuzzer attains the best Top1-ASR, and for DeepSeek-Chat-V2, Cipher attains the best, while our approach again comes in second, with a marginally smaller Top1-ASR. Particularly in models like GPT-3.5-turbo, GPT-4o-mini, Llama-2-7b-chat, Llama-2-13b-chat and Claude-3.5-sonnet, MazeBreaker achieves the highest Top1-ASR, highlighting its ability to effectively bypass security measures in these models. Compared to other approaches, MazeBreaker obviously outperform in Top1-ASR for the well safety-aligned models.

We found that MazeBreaker shows an average Top1-ASR of 93.5% and a Top5-ASR of 97.8% across all models, which indicates that MazeBreaker is highly effective and consistent in attacking various models. In contrast, the other benchmark methods exhibit less impressive average performance. For instance, LLMFuzzer has an average Top1-ASR of 86.1%, and Jailbroken has an even lower average Top1-ASR of 43.5%. It is noticed that all used baselines are very bad for Claude-3.5-sonnet while our approach significantly outperforms them, which is highlighted by the orange color ■ in Table 2. While strong safety alignments are effective in preventing many attack methods [23], MazeBreaker produces impressive results, showcasing its resilience. We investigate their responses manually, and confirm that Claude-3.5-sonnet refused to answer their questions and almost all of the responses start with "I will not provide any information...".

The experiment demonstrates that MazeBreaker significantly outperforms existing methods (such as LLMFuzzer and RLBreaker) on commercially available high-security models that are highlighted by the gray color in Table 2, including GPT-3.5-turbo, GPT-4o-mini, Claude-3.5-sonnet, and GLM-4-air, especially in the Top1 metric. Even against these highly fortified models [1, 22], like Claude-3.5-sonnet that is highlighted by the orange color ■ , MazeBreaker can still breach security boundaries with a relatively high probability, fully showcasing its applicability and robustness in handling "complex defense mechanisms". Our method excels in attacking commercial models for two main reasons. First, we make full use of the output of our attacking process to maximize rewards, making it easier to penetrate complex and adaptive defenses. Second, we do not need any "reference answer", ensuring broader applicability. By fully leveraging feedback from each attempt, MazeBreaker gradually refines its attacks and precisely gauges the target's security boundaries even when facing diverse defensive strategies.

We chose to compare our method with LLMFuzzer and RLBreaker in detail, focusing on the cost (QC) highlighted by the blue and green colors in Tables 1 and 2. This comparison was made because both approaches demonstrate competitive performance and involve a similar iterative process for generating adversarial inputs. In LLMFuzzer, 500 iterations were used, leading to a total of 50,000 requests. RLBreaker send 5,000 requests during the traininng phase and another 5,000 requests during the testing phase. In contrast, in our work, we send 3,000 requests during the iteration phase and 1,500 during ASR calculation phase, totaling 4,500 requests (9% of LLMFuzzer cost and 45% of RLBreaker cost). Compared to other methods, MazeBreaker demonstrates superior efficiency: methods with significantly fewer queries exhibit much poorer performance, while methods with comparable performance require considerably more queries. This comparison underscores the efficiency and effectiveness of our method, achieving higher success rates with significantly lower costs.

MazeBreaker consistently outperforms the other baselines across 13 LLMs, demonstrating superior robustness and effectiveness, especially for the commercial LLMs with strong defense mechanisms. Besides, MazeBreaker achieves an optimal balance between performance and cost, delivering superior results without incurring significant expenses.

5.2 Choice of Mutation Model (RQ2)

We use GPT-4o-mini, DeepSeek-Chat-V2, Llama-3.1-70b, Qwen2-72b-instruct, and Vicuna-7b as our mutation-model candidates. To evaluate the impact of each mutation model, we randomly select

60 harmful questions and 60 templates from our dataset. We use Llama-2-7b-chat as the target model because it is not large and has a relatively strong initial safety alignment, making it an ideal candidate to assess the effectiveness of generated adversarial examples in compromising model safety [20] . To ensure consistency across experiments, we set the mutation policy for questions to "Euphemize" and the mutation policy for templates to "Expand".

Table 3: Comparison of different mutation model

	Top1-ASR	Top5-ASR	Mutation Time(s)
GPT-4o-mini	28.33%	73.33%	16.01
DeepSeek-Chat-V2	30.51%	74.58%	25.07
Llama-3.1-70b	22.03%	64.41%	24.52
Qwen2-72b-instruct	**36.67%**	**75.00%**	48.6
Vicuna-7b	11.87%	37.29%	**6.5**

Table 3 demonstrates ASR and the mutation time of different models. Qwen2-72b-instruct achieves the highest ASR in both Top1 and Top5 metrics, demonstrating superior performance. DeepSeek-Chat-V2 and GPT-4o-mini also perform well in terms of ASR. However, GPT-4o-mini stands out for its significantly shorter mutation time compared to the other models, which is crucial considering that our method involves running thousands of iterations. Although GPT-4o-mini's accuracy is slightly lower than that of DeepSeek-Chat-V2 by 7.13% and Qwen2-72b-instruct by 22.72%, its significant time-saving advantage of 36.13% over DeepSeek-Chat-V2 and 67.06% over Qwen2-72b-instruct makes it highly effective for tasks requiring numerous iterations. Additionally, despite being a commercial model, GPT-4o-mini is more cost-effective than Qwen2-72b-instruct. This balance between efficiency and cost makes GPT-4o-mini a highly suitable choice for iterative mutation tasks where both time and budget constraints are critical.

The choice of mutation model significantly impacts the effectiveness of seed mutations, with varying time costs across different models. Considering the balance between performance, time efficiency, and cost, we have selected GPT-4o-mini as our final mutation model.

5.3 Impact of Reinforcement Learning (RQ3)

To validate the impact of reinforcement learning, we conducted experiments using single-agent reinforcement learning, multi-agent reinforcement learning, and a control setting without reinforcement learning. We chose two models which are well safety-aligned to evaluate the impact of our RL policy. Each configuration was tested over 3000 rounds using both GPT-3.5-turbo and Llama-2-7b-chat. In the setting without reinforcement learning, the mutator applied actions randomly to the question and template. For single-agent reinforcement learning, the question agent and template agent are centralized into a single agent. Each output of the single agent contains one question action and one template action.

This approach assumes that the combination of the original question and template offers 25 mutator options, resulting in 25 possible actions per step. We train the single agent using the DQN policy [21]. In contrast, multi-agent reinforcement learning independently trains the question agent and template agent using the MADDPG policy [17], a classical method in multi-agent reinforcement learning. Each agent has 5 actions to select for mutating the question and template. As shown in Figure 3, the single-agent DQN

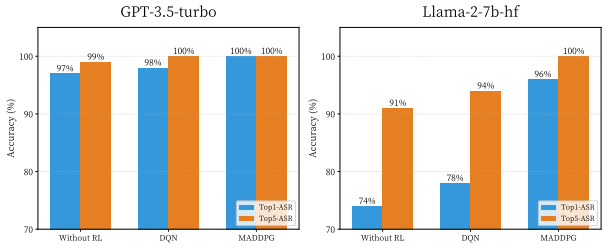

Figure 3: Comparison of Reinforcement Learning Methods on attacking GPT-3.5-turbo and Llama-2-7b-hf

strategy outperforms the non-RL setting in both Top1-ASR and Top5-ASR across both LLMs. Additionally, the multi-agent MADDPG strategy achieves even greater improvements, demonstrating the effectiveness of reinforcement learning in enhancing attack success rates. We think that MADDPG is better than DQN in our context because: 1) DQN is more suitable for smaller action spaces; and 2) each agent in MADDPG focuses on its specific task while sharing decisions with other agents, making it more effective for solving complex tasks.

Reinforcement learning enhances MazeBreaker's effectiveness, with the multi-agent MADDPG strategy delivering the best results, outperforming both the non-reinforcement learning setting and the single-agent DQN. This highlights the multi-agent design as a key factor in improving attack success rates.

5.4 Design of Reinforcement Learning (RQ4)

We explore how the key components within MazeBreaker impact its performance. Specifically, we compare the effects of the two types of the reward and examine how mutating both the question and template compares to mutating only the template in influencing the results. Our reward function comprises both r_{IQ} and r_J. To determine whether both the IQ and the J_{score} of the judgment model are effective in MazeBreaker, we conducted experiments using Llama-2-7b-chat and GLM-4-air. We createdd two additional groups: r_{IQ} only and r_J only. From Table 4, we can observe that using only r_{IQ} or r_J for the attack results in a lower ASR compared to MazeBreaker. Additionally, for our double seed pool setting, the goal is to discover more effective jailbreak templates. Mutating only the questions would not add new templates to the seed pool, which is essential for achieving this goal. Therefore, we focus on mutating the jailbreak *templates only*. We found that for both GLM-4-air and Llama-2-7b-chat, the ASR performance showed significant improvement. This highlights the importance of mutating both the question and the template in generating more effective jailbreak strategies.

Table 4: Comparison of different components setting

Settings	Llama-2-7b-chat		GLM-4-air	
	Top1-ASR	Top5-ASR	Top1-ASR	Top5-ASR
r_{IQ}	70%	92%	89%	99%
r_J	74%	95%	88%	99%
Template	77%	97%	93%	97%
MazeBreaker	**96%**	**100%**	**100%**	**100%**

The experimental results demonstrate that the multi-agent reinforcement learning strategy, which incorporates both r_{IQ} and

r_J rewards and employs a double pool mutation mechanism, is critical to attack LLMs. Each component contributes to the overall effectiveness of the attack, with the full MazeBreaker approach outperforming the simpler configurations.

5.5 Transfer Attack (RQ5)

MazeBreaker needs iteratively to complete the attack. The attack time cost increases as the model parameters grow in size. Given the substantial time commitment, we explored whether the top templates identified from one model or the weights of reinforcement learning agents could be transferred effectively to attack another model. We selected the five best templates from attacks on Gemma2-8b-instruct, GPT-4o-mini, DeepSeek-Chat-V2, GLM-4-air, and GPT-3.5-turbo and applied these templates to attack the widely used Llama series models (Llama-2-7b-chat, Llama-3-70b, Llama-3.1-8b, Llama-3.1-70b, Llama-3.1-405b).

Figure 4: Template and Agent Transfer Attack

Figure 4 shows the results of the template transfer attack and the agent-weight transfer attack of reinforcement learning. For template transfer attack, it is partially successful across different Llama models. The Top1- and Top5- ASR success rates vary, with Llama-3.1-8b showing the highest transfer effectiveness (95% Top1 and 100% Top5) and Llama-3.1-405b exhibiting the lowest (64% Top1 and 81% Top5). These findings suggest that while certain templates can be effectively transferred, the success of such transfers depends significantly on the specific architecture and scale of the target model. The comparison between reinforcement-learning agents' performance under transfer attacks versus non-transfer attacks reveals that transfer attacks lead to superior outcomes, as shown in Figure 4. Specifically, when agents transfer knowledge from other tasks to the current one, they exhibit enhanced resilience against attacks. Transfer learning enhances the agents' ability to withstand attacks, as evidenced by the better performance under transfer attacks compared to non-transfer attacks. In conclusion, template transfer generally shows good effectiveness but its success is model-dependent, varying with the architecture and size of the target model. While agent transfer also has potential, especially when progressing from smaller to larger models within the same series.

The effectiveness of both template transfer and agent transfer strategies shows that MazeBreaker can adapt and perform well across different models, leveraging their unique characteristics.

6 Discussion

Threats to Validity. The internal threat to validity mainly lies in the potential errors of using DeepSeek-V3 for evaluating different attack approaches when determining whether the final output is a harmful response. To minimize this risk, we introduced Llama Guard 3 [16] and multi-reviewer check for cross-validation of the results. By employing multiple evaluation methods, we increased the reliability of our findings through diversified validation approaches. The external threat to validity is primarily addressed by carefully selecting a diverse set of target models and harmful questions. To ensure model representativeness, we included 13 different models that cover both open-source and commercial models, widely used in various research and application scenarios. For the harmful questions, we used the same dataset as LLMFuzzer, which consists of 100 harmful questions that were either manually crafted by humans or generated via crowdsourcing to accurately reflect real-world scenarios. This approach helps to ensure that our findings are broadly applicable across different types of models and potential real-world scenarios.

Limitation. The first limitation is the time cost issue. Although MazeBreaker requires fewer overall queries than LLMFuzzer, its total runtime is longer than other attack methods because each iterations involves strategy selection via reinforcement learning and mutation of templates and questions, which will take some time. Besides, like LLMFuzzer, MazeBreaker's effectiveness depends highly on the original manually crafted jailbreak templates as initial seeds. The second limitation is the randomness introduced by the AI-based components of MazeBreaker. This randomness arises from the mutators, the judgment model, information quantization, and the reinforcement-learning agents. Except for the reinforcement-learning agents, all others use LLMs, which is well known for its undetermined output, even when the input is the same. The reinforcement-learning agents also exhibit some randomness and are influenced by the initial states, the optimization process, and the output of other components of MazeBreaker. This can lead to some variations in the effectiveness of our approach. All jailbreak approaches that contain AI components share this limitation. Finally, while MazeBreaker performs well across various language models, its attack strategies and effectiveness may not fully transfer between different types of models, particularly those that have defenses against the mutation strategies of MazeBreaker.

7 Conclusion

In this paper, we introduce MazeBreaker, a novel approach to jailbreak attacks on LLMs that leverages a multi-agent reinforcement learning framework. By considering LLM security as a maze that attackers must navigate and escape, our method strategically explores and identifies the best mutation action to take in each iteration. Our approach uses multiple agents to iteratively refine the attack strategy, finding the most effective mutation actions, which can ultimately increase the likelihood of successfully bypassing the LLMs' security defenses. Besides, our approach doesn't need the reference answers. We first propose a vocabulary-richness reward mechanism that encourages LLMs to generate more content, along with a double-pool mutation strategy to enhance the diversity and effectiveness of the attacks. These components work together to significantly increase MazeBreaker's ASR. The results of our experiments, conducted across 13 different LLMs, demonstrate that MazeBreaker outperforms 6 existing SOTA jailbreak methods in both attack success rate and efficiency, especially when attacking

commercial models with strong security alignment. Our work offers important insights into the vulnerabilities of LLMs and highlights the need for continued advancements in model security. By providing a more efficient and effective means of testing LLM defenses, MazeBreaker contributes to enhance the safety and ethical alignment of these powerful models, providing reassurance about the effectiveness of the new approach.

References

[1] Anthropic. Mitigate jailbreaks and prompt injections, 2025. https://docs.anthropic.com/en/docs/test-and-evaluate/strengthen-guardrails/mitigate-jailbreaks.

[2] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[3] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[4] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries, 2023.

[5] Xuan Chen, Yuzhou Nie, Wenbo Guo, and Xiangyu Zhang. When llm meets drl: Advancing jailbreaking efficiency via drl-guided search. *arXiv preprint arXiv:2406.08705*, 2024.

[6] Xuan Chen, Yuzhou Nie, Lu Yan, Yunshu Mao, Wenbo Guo, and Xiangyu Zhang. Rl-jack: Reinforcement learning-powered black-box jailbreaking attack against llms. *arXiv preprint arXiv:2406.08725*, 2024.

[7] Junjie Chu, Yugeng Liu, Ziqing Yang, Xinyue Shen, Michael Backes, and Yang Zhang. Comprehensive assessment of jailbreak attacks against llms. *arXiv preprint arXiv:2402.05668*, 2024.

[8] Gelei Deng, Yi Liu, Yuekang Li, Kailong Wang, Ying Zhang, Zefeng Li, Haoyu Wang, Tianwei Zhang, and Yang Liu. Masterkey: Automated jailbreaking of large language model chatbots. In *Proc. ISOC NDSS*, 2024.

[9] Peng Ding, Jun Kuang, Dan Ma, Xuezhi Cao, Yunsen Xian, Jiajun Chen, and Shujian Huang. A wolf in sheep's clothing: Generalized nested jailbreak prompts can fool large language models easily. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2136–2153, 2024.

[10] Haibo Jin, Leyang Hu, Xinuo Li, Peiyan Zhang, Chonghan Chen, Jun Zhuang, and Haohan Wang. Jailbreakzoo: Survey, landscapes, and horizons in jailbreaking large language and vision-language models. *arXiv preprint arXiv:2407.01599*, 2024.

[11] Mingyu Jin, Suiyuan Zhu, Beichen Wang, Zihao Zhou, Chong Zhang, Yongfeng Zhang, et al. Attackeval: How to evaluate the effectiveness of jailbreak attacking on large language models. *arXiv preprint arXiv:2401.09002*, 2024.

[12] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[13] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

[14] Zeyi Liao and Huan Sun. Amplegcg: Learning a universal and transferable generative model of adversarial suffixes for jailbreaking both open and closed llms. *arXiv preprint arXiv:2404.07921*, 2024.

[15] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[16] AI @ Meta Llama Team. The llama 3 herd of models, 2024.

[17] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*, 2017.

[18] Huijie Lv, Xiao Wang, Yuansen Zhang, Caishuang Huang, Shihan Dou, Junjie Ye, Tao Gui, Qi Zhang, and Xuanjing Huang. Codechameleon: Personalized encryption framework for jailbreaking large language models. *arXiv preprint arXiv:2402.16717*, 2024.

[19] Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. Tree of attacks: Jailbreaking black-box llms automatically. *arXiv preprint arXiv:2312.02119*, 2023.

[20] Meta. Llama-2-7b-chat-hf, 2023. https://huggingface.co/meta-llama/Llama-2-7b-chat-hf.

[21] Volodymyr Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[22] OpenAI. Our approach to alignment research, 2022. https://openai.com/index/our-approach-to-alignment-research/.

[23] Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. *arXiv preprint arXiv:2202.03286*, 2022.

[24] Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! In *The Twelfth International Conference on Learning Representations*, 2024.

[25] Jiayang Song, Yuheng Huang, Zhehua Zhou, and Lei Ma. Multilingual blending: Llm safety alignment evaluation with language mixture. *arXiv preprint arXiv:2407.07342*, 2024.

[26] MazeBreaker Team. Mazebreaker: Multi-agent reinforcement learning for dynamic jailbreaking of llm security defenses, 2025. https://sites.google.com/view/maze-breaker/home.

[27] Xiangwen Wang, Jie Peng, Kaidi Xu, Huaxiu Yao, and Tianlong Chen. Reinforcement learning-driven llm agent for automated attacks on llms. In *Proceedings of the Fifth Workshop on Privacy in Natural Language Processing*, pages 170–177, 2024.

[28] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does LLM safety training fail? In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[29] Dongyu Yao, Jianshu Zhang, Ian G Harris, and Marcel Carlsson. Fuzzllm: A novel and universal fuzzing framework for proactively discovering jailbreak vulnerabilities in large language models. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4485–4489. IEEE, 2024.

[30] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, page 100211, 2024.

[31] Jiahao Yu, Xingwei Lin, Zheng Yu, and Xinyu Xing. {LLM-Fuzzer}: Scaling assessment of large language model jailbreaks. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 4657–4674, 2024.

[32] Youliang Yuan, Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Pinjia He, Shuming Shi, and Zhaopeng Tu. Gpt-4 is too smart to be safe: Stealthy chat with llms via cipher. *arXiv preprint arXiv:2308.06463*, 2023.

[33] Youliang Yuan, Wenxiang Jiao, Wenxuan Wang, Jen tse Huang, Pinjia He, Shuming Shi, and Zhaopeng Tu. GPT-4 is too smart to be safe: Stealthy chat with LLMs via cipher. In *The Twelfth International Conference on Learning Representations*, 2024.

[34] Zhengyue Zhao, Xiaoyun Zhang, Kaidi Xu, Xing Hu, Rui Zhang, Zidong Du, Qi Guo, and Yunji Chen. Adversarial contrastive decoding: Boosting safety alignment of large language models via opposite prompt optimization. *arXiv preprint arXiv:2406.16743*, 2024.

[35] Sicheng Zhu, Ruiyi Zhang, Bang An, Gang Wu, Joe Barrow, Zichao Wang, Furong Huang, Ani Nenkova, and Tong Sun. AutoDAN: Interpretable gradient-based adversarial attacks on large language models. In *First Conference on Language Modeling*, 2024.

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