

RedHit: Adaptive Red-Teaming of Large Language Models via Search, Reasoning, and Preference Optimization

Anonymous ACL submission

Abstract

Red-teaming has become a critical component of Large Language Models (LLMs) security amid increasingly sophisticated adversarial techniques. However, existing methods often depend on hard-coded strategies that quickly become obsolete against novel attack patterns, requiring constant updates. Moreover, current automated red-teaming approaches typically lack effective reasoning capabilities, leading to lower attack success rates and longer training times. In this paper, we propose **RedHit**, a multi-round, automated, and adaptive red-teaming framework that integrates Monte Carlo Tree Search (MCTS), Chain-of-Thought (CoT) reasoning, and Direct Preference Optimization (DPO) to enhance the adversarial capabilities of an Adversarial LLM (ALLM). RedHit formulates prompt injection as a tree search problem, where the goal is to discover adversarial prompts capable of bypassing target model defenses. Each search step is guided by an Evaluator module that dynamically scores model responses using multi-detector feedback, yielding fine-grained reward signals. MCTS is employed to explore the space of adversarial prompts, incrementally constructing a Prompt Search Tree (PST) in which each node contains an adversarial prompt, its response, a reward, and other control properties. Prompts are generated via a locally hosted IndirectPromptGenerator module, which uses CoT-enabled prompt transformation to create multi-perspective, semantically equivalent variants for deeper tree exploration. CoT reasoning improves MCTS exploration by injecting strategic insights derived from past interactions, enabling RedHit to adapt dynamically to the target LLM’s defenses. Simultaneously, DPO fine-tunes the *ALLM* using preference data gathered from prior attack rounds, progressively enhancing its ability to generate more effective prompts. RedHit leverages the Garak framework to evaluate each adversarial prompt and compute rewards, demonstrating robust

and adaptive adversarial behavior across multiple attack rounds.

1 Introduction

LLMs such as GPT-4 and LLaMA have demonstrated remarkable capabilities in understanding and generating coherent, context-aware text across a wide array of applications, including machine translation, summarization, code generation, and conversational agents (??). These models have shown human-like fluency and reasoning capabilities, enabling them to power both commercial and open-source AI systems. However, their growing capabilities come with an expanding set of safety and security challenges (?). LLMs are susceptible to producing unsafe content, disclosing sensitive information, or being manipulated through adversarial prompts—raising substantial concerns around trust, safety, and deployment in real-world scenarios (??). One prominent threat to LLM safety is the phenomenon of prompt injection attacks, in which adversarial users craft input sequences to circumvent safety filters, jailbreak models, or elicit harmful, biased, or restricted outputs. As LLMs are increasingly integrated into search engines, productivity tools, customer service bots, and decision-support systems, the impact of such attacks grows significantly. For example, attackers may exploit LLMs to bypass content moderation, extract private training data, or subtly manipulate the model’s behavior in multi-turn dialogues. These vulnerabilities are not merely theoretical; several real-world instances of prompt injection and model misuse have already been documented (??).

To identify and mitigate such vulnerabilities during development, red-teaming has emerged as a core strategy. Red-teaming involves simulating adversarial behavior by designing malicious or probing inputs to test how models respond under unsafe or manipulative conditions. Traditionally, this has been performed manually by expert annotators or

security researchers who craft edge-case prompts and evaluate outputs for policy violations or harm (??). While this approach is invaluable, it is inherently resource-intensive, requiring extensive time, domain expertise, and iteration to explore the high-dimensional space of adversarial behaviors effectively. Manual red-teaming suffers from several key limitations. First, it does not scale well. The ever-growing range of use cases and the rapid evolution of LLMs have made the space of potential vulnerabilities vast and constantly shifting. Relying solely on human experts to explore this space leads to bottlenecks. Second, human-crafted attacks may fall behind the sophistication of both LLMs and their defense mechanisms. Third, human evaluators introduce subjectivity, inconsistency, and potential oversight, especially when assessing nuanced harmful outputs.

To address these challenges, automated red-teaming has been proposed as a promising alternative. One stream of research trains reward models that approximate human judgment, enabling large-scale preference modeling and automated scoring of model outputs (??). These models help reduce reliance on human annotators for output evaluation. However, generating high-quality adversarial prompts—especially ones that adapt to model defenses—still largely depends on human creativity and intuition. Recent works have explored automated systems that use language models themselves to generate adversarial prompts. For example, some approaches fine-tune models to behave as adversarial agents, iteratively optimizing prompts to maximize unsafe completions (??). However, these systems often suffer from brittleness, overfitting to specific targets, or poor generalization across different LLMs. Furthermore, they frequently lack strategic reasoning and adaptability, making them ineffective at discovering newly emerging failure modes.

A critical limitation in existing automated red-teaming frameworks is their use of static or greedy generation strategies. These systems typically lack mechanisms for exploration and strategic refinement. As a result, they may become stuck in sub-optimal attack patterns and fail to uncover subtle or novel vulnerabilities. Moreover, most frameworks do not incorporate learning from feedback in a structured and long-term way—each generation is treated independently, without memory of past successes or failures.

To bridge these gaps, we propose **RedHit**, a

novel framework for progressive, automated, and adaptive red-teaming of LLMs. RedHit introduces a synergistic integration of three core components: MCTS, CoT reasoning, and DPO. Together, these techniques enable RedHit to generate high-quality adversarial prompts that evolve over multiple rounds, guided by both strategic exploration and preference feedback. At the heart of RedHit is the formulation of prompt injection as a tree search problem. Each node in the search tree represents a candidate adversarial prompt, its corresponding model response, a reward (evaluated via an external reward model), and relevant metadata. RedHit uses a configurable MCTreeSearch module that controls search depth, iteration count, and branching breadth to systematically expand the prompt search space. The model responses are evaluated by an Evaluator module that aggregates multi-detector results to compute a fine-grained reward signal. Adversarial prompts are generated using the IndirectPromptGenerator, a locally hosted Chain-of-Thought-based rewriter that transforms prompts into strategically deceptive alternatives, boosting the diversity and stealth of attacks.

To further enhance the strategic depth of MCTS, we incorporate Chain-of-Thought reasoning during prompt generation. CoT provides intermediate reasoning steps, helping the \mathcal{ALLM} generate prompts that are not only more coherent but also more tactically sound. This improves the likelihood of successfully bypassing target defenses and allows the model to reflect on past attack paths to refine future ones (?). Finally, RedHit employs DPO to continuously fine-tune the adversarial LLM based on feedback from previous attack rounds. Unlike supervised fine-tuning, which requires labeled data, DPO directly optimizes the model’s parameters to prefer high-reward prompts over low-reward ones, using the output evaluations from each tree traversal (?). This iterative learning enables the \mathcal{ALLM} to become progressively better at generating effective adversarial prompts tailored to the evolving defenses of the target LLM. Our main contributions are summarized as follows:

- We introduce **RedHit**, a novel automated red-teaming framework that integrates MCT, CoT reasoning, and DPO to progressively generate adaptive adversarial prompts. RedHit is implemented using a modular design that supports local LLMs and is fully integrated with the DSPy framework.

- Prompt injection is formulated as a tree search problem, and a PST is constructed where each node encodes an adversarial prompt, model response, reward, and auxiliary meta-data—enabling efficient exploration and learning. Prompt exploration is driven by a configurable MCTreeSearch module that supports depth-controlled, breadth-aware, reward-guided rollouts.
- CoT reasoning is embedded within MCTS rollouts to guide strategic and context-aware adversarial generation paths. Prompts are rewritten using a local IndirectPromptGenerator that produces semantically aligned but more evasive versions of the base prompt using CoT-based transformations.
- We implement a continual preference-based fine-tuning loop using DPO, allowing the adversarial LLM to improve its effectiveness over multiple attack rounds.
- We evaluate RedHit using the Garak framework and demonstrate that it achieves higher attack success rates, broader coverage of vulnerabilities, and stronger adaptability compared to existing baselines. A dedicated Evaluator module aggregates the outcomes of multiple detectors to compute fine-grained reward scores, enabling more precise learning signals.

The remainder of this paper is organized as follows. **Section 2** reviews recent advances in automated red-teaming and adversarial prompt generation for LLMs. **Section 3** introduces the RedHit framework in detail, describing the integration of MCTS, CoT reasoning, and DPO. **Section 4** outlines our experimental setup, evaluation metrics, and our baseline, followed by extensive empirical results. We also conduct an ablation study to isolate the contributions of each core module. Finally, **Section 5** concludes with a summary of our findings and discusses promising future directions for adaptive and scalable LLM red-teaming research.

2 Related Work

The growing capabilities of LLMs have amplified the need to rigorously evaluate their robustness against misuse and adversarial exploitation. Early

efforts in this space primarily relied on manual red-teaming, where human annotators craft prompts to probe model vulnerabilities (??). While valuable, this approach is inherently limited by scalability, subjectivity, and cost, often requiring large annotation teams to identify unsafe behavior through extensive trial-and-error. To mitigate the inefficiency of human evaluation, reward models trained on human preferences have been introduced to automate the assessment of model responses (??). These models approximate human judgment and provide feedback signals for fine-tuning, enabling scalable learning from preferences. However, the generation of high-quality adversarial prompts remains predominantly human-driven, limiting the overall scalability of red-teaming pipelines.

In response, recent work has explored the use of language models themselves to generate adversarial prompts. For instance, ? introduced a framework that trains adversarial LLMs to red-team other models, demonstrating the feasibility of LLMs as both attackers and defenders. However, these approaches often rely on static attack strategies or fine-tuned behaviors that do not generalize well across evolving LLMs. As target models improve, adversarial agents must also dynamically adapt to more sophisticated and subtle defense mechanisms. To address the challenge of evolving vulnerabilities, ? proposed leveraging GPT-based adversaries to automatically jailbreak models. Their findings highlight the potential of autoregressive LLMs to discover and exploit security flaws, yet also reveal the brittleness of such systems when deployed in multi-turn or adaptive contexts. Similarly, ? introduced MART, a hybrid framework that combines automated adversarial prompt generation with safe response modeling to enhance robustness and adaptability in red-teaming pipelines. Other methods like JailbreakBench and PromptBench have also aimed to standardize red-teaming evaluations, though they typically lack iterative reasoning or self-improving feedback loops. Despite these advances, existing automated red-teaming frameworks often suffer from several limitations. Many adopt single-step or greedy strategies that fail to account for long-term planning or strategic exploration. Additionally, few systems integrate structured feedback mechanisms to continuously improve adversarial capabilities over time. Moreover, most prior work underutilizes search-based optimization and reasoning-enhanced generation, both of which are critical for uncovering subtle

or evasive vulnerabilities. A fully automated and adaptive red-teaming framework must be capable of both uncovering current vulnerabilities and anticipating emergent failure patterns through iterative interaction.

To bridge these gaps, we propose **RedHit**, a multi-round, progressive, and adaptive red-teaming framework. RedHit combines MCTS (?), CoT reasoning (?), and DPO (?) to construct an \mathcal{ALLM} capable of dynamically generating increasingly effective attack prompts. The framework formulates prompt injection as a tree search problem, where MCTS guides the exploration of adversarial paths, CoT enhances strategic reasoning during prompt generation, and DPO fine-tunes the \mathcal{ALLM} based on feedback from prior attacks. Unlike earlier approaches, RedHit maintains a Prompt Search Tree across rounds, allowing it to retain memory of prior attempts, adapt search directions, and improve long-term attack efficacy. Through this integration, RedHit moves beyond static prompt crafting or single-step adversarial generation. It constructs a PST, where each node contains an adversarial prompt, corresponding model response, reward (measured via an external evaluation framework such as Garak), and control metadata. This design allows RedHit to support dynamic exploration, structured optimization, and CoT-guided reasoning in a unified, automated red-teaming pipeline.

3 Proposed Method

RedHit is an automated and multi-round red-teaming framework designed to uncover vulnerabilities in target large language models (TLLMs) through adaptive adversarial prompt generation. It treats prompt injection as a structured exploration problem, constructing a dynamic PST where nodes represent prompts, responses, and reward scores. **Figure ??** illustrates the overall RedHit workflow. The \mathcal{ALLM} generates CoT-guided prompt candidates, which are evaluated against the TLLM. The resulting responses are scored via an external evaluation framework, and high-performing prompts are retained in the PST for policy refinement. This closed-loop architecture—driven by MCTS-based exploration, CoT-based prompt generation, and reward-guided optimization—enables RedHit to progressively adapt and improve over multiple attack rounds. Let \mathcal{TLLM} denote the target LLM under audit, and \mathcal{ALLM} be an adversarial LLM capable of generating attack prompts.

At each time step t , the adversary generates a candidate prompt x_t using its current policy. The prompt is submitted to \mathcal{TLLM} , producing a response $r_t = \mathcal{TLLM}(x_t)$, which is scored by a reward function $s_t = \text{Reward}(x_t, r_t)$ computed via an external framework (e.g., Garak). The goal is to iteratively improve \mathcal{ALLM} such that it maximizes the expected cumulative reward $\sum_t s_t$, corresponding to the generation of increasingly effective adversarial prompts. The overall process is summarized in **Algorithm ??**, which outlines RedHit’s multi-round exploration and optimization procedure across search, evaluation, and preference-driven fine-tuning.

Algorithm 1 RedHit Framework

Require: Target LLM \mathcal{TLLM} , adversarial LLM \mathcal{ALLM} , reward model \mathcal{R} (e.g., Garak), number of rounds N , rollout budget B

- 1: Initialize Prompt Search Tree \mathcal{T} with root node and empty result buffer
- 2: **for** $i = 1$ to N **do**
 {Attack rounds}
- 3: **for** $j = 1$ to B **do**
 {Tree rollouts}
- 4: Select node n in \mathcal{T} using UCB traversal policy
- 5: Retrieve base prompt x_j^{base} from node n
- 6: Generate CoT reasoning trace τ_j using \mathcal{ALLM}
- 7: Generate prompt set $\{x_j^1, \dots, x_j^m\}$ from τ_j using IndirectPromptGenerator
- 8: **for** each prompt x_j^k **do**
- 9: Query \mathcal{TLLM} to get response $r_j^k = \mathcal{TLLM}(x_j^k)$
- 10: Evaluate reward $s_j^k = \mathcal{R}(x_j^k, r_j^k)$ via Evaluator
- 11: Expand \mathcal{T} by adding node (x_j^k, r_j^k, s_j^k) under n
- 12: Store (x_j^k, r_j^k, s_j^k) in result buffer
- 13: **end for**
- 14: **end for**
- 15: Construct preference pairs from result buffer
- 16: Fine-tune IndirectPromptGenerator using DPO
- 17: Clear result buffer
- 18: **end for**
- 19: **return** Top- k adversarial prompts from \mathcal{T} ranked by reward

3.1 Tree-based Prompt Exploration with CoT Guidance

RedHit employs Monte Carlo Tree Search (MCTS) as its core search mechanism to explore the space of adversarial prompts. Each node in the PST stores a tuple (x, r, s) , representing the adversarial prompt, the response from \mathcal{TLLM} , and the resulting reward. The MCTS algorithm balances exploration and exploitation using the Upper Confidence Bound (UCB) criterion to traverse promising branches. The tree search is implemented via a configurable MCTreeSearch class that supports it-

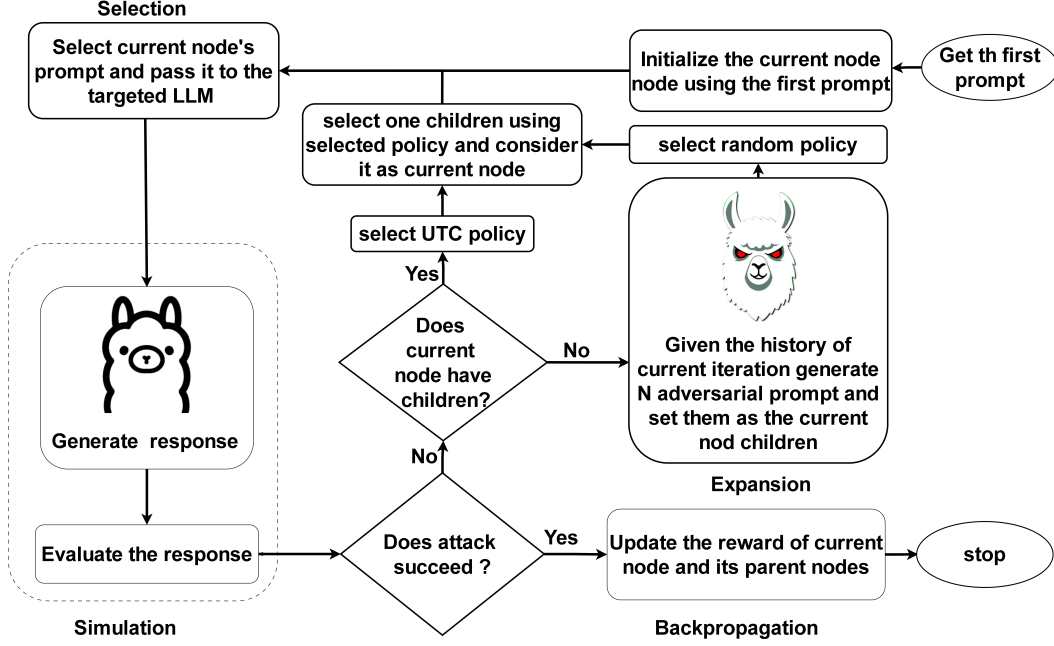


Figure 1: Overview of the RedHit framework. An \mathcal{ALLM} explores adversarial prompt space through MCTS, guided by CoT reasoning and iteratively fine-tuned using DPO. The PST encodes prompt-response-reward tuples, evaluated by the Garak framework.

erative rollouts with adjustable depth, breadth, and final expansion rounds. Prompt generation is handled by the IndirectPromptGenerator, which wraps a locally hosted DSPy program that transforms a base prompt into multiple indirect adversarial variants using Chain-of-Thought reasoning. These reworded prompts form the candidate branches during tree expansion. Each prompt is executed using the target LLM through a standard interface, and the response is passed into an Evaluator module. This evaluator aggregates the detection results from multiple detectors and normalizes the score over the number of generations and detectors, yielding a soft reward signal. This fine-grained reward is critical for driving effective policy updates and deeper exploration. The evaluator is automatically initialized per probe within the ProbeWrapper, enabling seamless integration into the red-teaming loop.

3.2 Reward-driven Optimization and Policy Refinement

Following each attack round, RedHit uses the accumulated interactions to construct preference pairs from the PST. These are used to fine-tune \mathcal{ALLM} via DPO, aligning the adversary’s generation policy to favor high-reward prompts. After each round,

high-reward outputs are stored and sampled for continued learning or offline optimization. This allows training to persist across multiple execution sessions. By iteratively refining the adversarial policy through structured exploration (MCTS), CoT, and preference-based optimization (DPO), RedHit overcomes the limitations of static or brittle red-teaming strategies. It adapts to increasingly robust defenses in \mathcal{TLLM} , discovers both common and subtle vulnerabilities, and supports diverse, high-reward adversarial strategies across multiple interaction rounds. Unlike prior frameworks, RedHit supports local model hosting, reasoning-driven generation, and modular policy training—all within a scalable and extensible DSPy-based environment.

3.3 Reward Formulation

To apply **Monte Carlo Tree Search (MCTS)**, we need to define a reward function to learn a policy through an iterative process. MCTS updates the rewards of tree nodes during the backpropagation step, which is executed after each simulation. To compute the rewards, we used Garak’s detector, which generates multiple responses using \mathcal{TLLM} for each generation and determines whether each response passes the test. Accordingly, we define the reward for each node as the proportion of suc-

successful prompt injections among the generated responses:

$$\mathcal{R}(n) = \frac{\sum_{r \in \mathcal{G}_n} \delta(r)}{|\mathcal{G}_n|}$$

where \mathcal{G}_n denotes the set of responses generated at node n , and $\delta(r) = 1$ if the response r is flagged as a successful injection by the detector, and 0 otherwise.

4 Experimental Evaluation

To evaluate the effectiveness of the RedHit framework, we conduct a comprehensive set of experiments across multiple large language models (LLMs). The adaptability of RedHit allows us to apply it to all target LLMs using a variety of prompt injection strategies. In this section, we present our evaluation methodology, describe the experimental setup, and analyze the results to assess RedHit’s performance and robustness compared to existing approaches.

4.1 Experimental Setup

We evaluate RedHit against original and distilled version state-of-the-art LLMs serving as target models (\mathcal{TLLM}), including:

- **LLaMA 3 (?)**
- **Gamma-3 (?)**
- **Mistral7B (?)**
- **DeepSeek-R1-Distill-Qwen-7B (?)**
- **Phi-4 (?)**

?? demonstrates the details of LLMs we used for evaluate our proposed method.

The adversarial agent \mathcal{ALLM} is initialized using 4-bit quantized LLaMA3 8B model and interacts with the target model through multi-round attacks. Prompt generation is performed using a Chain-of-Thought-enabled IndirectPromptGenerator, which rephrases prompts into more evasive variants during MCTS exploration. Model responses are scored using a custom Evaluator class that aggregates the outputs of multiple detectors implemented within the Garak framework (?). All experiments are conducted offline within a modular, reproducible environment using RedHit’s local execution pipeline. RedHit runs for $N = 100$ attack rounds, with a rollout budget $B = 5$ per round, and each prompt

tree is expanded up to a configurable depth. We ran RedHit on a 24GB NVIDIA A30 GPU that hosted \mathcal{ALLM} via the VLLM framework, and we also applied 4-bit quantization to \mathcal{ALLM} using the bitsandbytes library.

4.2 Evaluation Metrics

The Attack Success Rate (ASR) metric is used to evaluate the effectiveness of RedHit’s adversarial prompts. ASR is defined as the percentage of generated responses that violate policy due to successful prompt injection. Since Garak generates a specific number of responses (by default, 5) for each harmful prompt, we applied a slight modification to enable a more comprehensive evaluation of our proposed method. In our approach, an attack is considered successful for a given prompt if the proportion of successful attack responses meets or exceeds a specified threshold (default: 0.5). We modified the default settings by changing the number of generations to 6 and ran RedHit across 10 thresholds ranging from 0.1 to 1.0.

$$ASR = \frac{N_{\text{success}}}{N_{\text{total}}} \quad (1)$$

where N_{success} is the number of successful attacks and N_{total} is the total number of prompts issued.

4.3 Evaluation Results

In the following sections, we present and analyze the evaluation results of the proposed methods. A thorough examination is conducted to assess the performance of our approach, supported by detailed discussion and visualizations.

4.3.1 Threshold-Based Evaluation of ALLM

Since the MCTS reward ranges from 0.0 to 1.0, we use it as a threshold to determine when a generated prompt is considered a successful prompt injection. We evaluate our proposed method across 10 threshold levels, from 0.1 to 1.0. A threshold of 1.0 indicates that a prompt injection is considered successful only if all six responses from the target LLM are successfully affected by the \mathcal{ALLM} prompt. ?? shows the result of our custom threshold-based experiments.

4.4 Comparison Results

To contextualize the performance of *RedHit*, we compare RedHit against the Garak framework, which is leading and highly significant red-teaming framework. Garak is a widely used baseline for

Table 1: Model Specifications

Model	Architecture	Parameters	Context Length	Embedding Length	Quantization
DeepSeek-R1	qwen2	7.6B	131072	3584	Q4_K_M
llama3	llama	8.0B	8192	4096	Q4_0
gemma3	gemma3	4.3B	131072	2560	Q4_K_M
mistral	llama	7.2B	32768	4096	Q4_0
phi4	phi3	14.7B	16384	5120	Q4_K_M

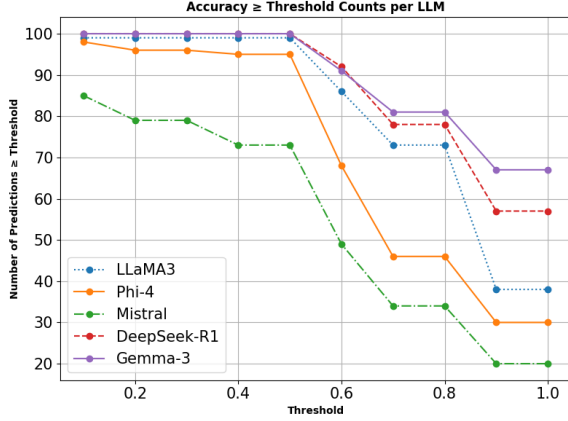


Figure 2: This figure illustrates the number of model outputs that exceed increasing accuracy thresholds (0.1 to 1.0) for five different LLMs. It highlights how performance declines as stricter confidence levels are applied, offering a comparative view of each model’s robustness under higher accuracy demands.

evaluating adversarial prompts targeting LLMs. It attempts to bypass safety filters through automatically generated injections. This comparison helps highlight the effectiveness and reliability of our proposed approach. In this experiment, we generate 100 malicious prompts using the \mathcal{ALLM} after training phase and then replaced these prompt with the Garak attempt prompt. ?? show the performance of Garak across different models while ?? show the RedHit performance.

When comparing Garak and Redhit across the same set of models, the differences in ASR highlight Redhit’s consistent performance. Redhit outperforms Garak by a 20.7% difference on LLaMA3, a 1.65% difference on Mistral-7B, and a 15.6% difference on DeepSeek-R1. Additionally, Redhit shows a 4.35% improvement over Garak on Phi4. While Redhit has a 12.2% lower ASR on Gemma3, indicating Garak’s better performance on more vulnerable models, the overall trends demonstrate that Redhit provides a more nuanced and precise eval-

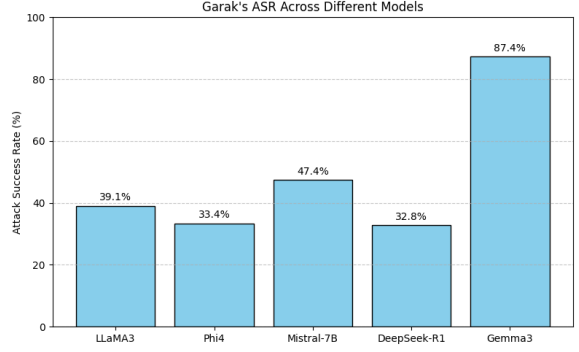


Figure 3: This chart illustrates the ASR of Garak across different language models. The results indicate that Gemma 3 is significantly more vulnerable to adversarial prompt injections compared to the other models, achieving an ASR of 87.4%.

uation. The method’s targeted approach proves particularly effective against models designed to resist basic adversarial attacks. Overall, Redhit’s ability to consistently achieve competitive or superior results across various models emphasizes its value as a reliable and refined red-teaming tool.

5 Conclusion

We presented **RedHit**, a fully automated and adaptive red-teaming framework that leverages MCTS, CoT reasoning, and DPO to iteratively generate high-quality adversarial prompts. RedHit treats prompt injection as a structured search problem, systematically exploring and expanding a prompt search tree while refining its generation policy through preference-based optimization. Our implementation integrates reasoning-driven prompt rewording, multi-detector evaluation, and modular fine-tuning, enabling RedHit to adapt over rounds and uncover both common and subtle vulnerabilities. Experimental evaluations compare RedHit against strong baselines, with ablations confirming the contribution of each component. Results demonstrate superior attack success, diversity, and

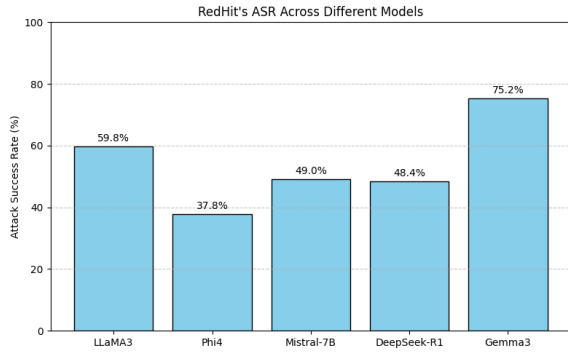


Figure 4: Redhit shows superior performance on LLaMA3, Mistral-7B, DeepSeek-R1, and Phi4. These results highlight its effectiveness across multiple models.

efficiency. RedHit advances scalable LLM auditing and offers a blueprint for combining search, reasoning, and learning in adversarial generation. Future work will explore multi-agent extensions, domain-specific reasoning, and integration with defenses to support closed-loop safety evaluation.

References

- Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, and 1 others. 2024. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*.
- Yuntao Bai, Saurav Kadavath, and 1 others. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*.
- Cameron B Browne, Edward Powley, and 1 others. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.
- Shuyang Chen, Zhe Sun, and 1 others. 2023. Mart: Improving the robustness of language models via multi-turn adversarial training. *arXiv preprint arXiv:2310.01931*.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Bastian Greshake Tzovaras and 1 others. 2023. Does gpt-4 pass the red team test? harnessing llms for automatic prompt injection. *arXiv preprint arXiv:2304.13709*.
- Xiaodong Gu, Meng Chen, Yalan Lin, Yuhan Hu, Hongyu Zhang, Chengcheng Wan, Zhao Wei, Yong Xu, and Juhong Wang. 2025. On the effectiveness of large language models in domain-specific code generation. *ACM Transactions on Software Engineering and Methodology*, 34(3):1–22.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Léo Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. *Mistral 7b*. *Preprint*, arXiv:2310.06825.
- OpenAI. 2023. *Gpt-4 technical report*. OpenAI Technical Report.
- Long Ouyang, Jeffrey Wu, and 1 others. 2022. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*.
- Ethan Perez, Kyle Yu, and 1 others. 2022. Red teaming language models with language models. *arXiv preprint arXiv:2202.03286*.
- Rafael Rafailov, Yining Zhou, and 1 others. 2023. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*.
- M. Ranta and 1 others. 2023. Garak: A framework for automated red-teaming of language models. <https://github.com/leondz/garak>.
- Robert Shelby, Carl Vondrick, and 1 others. 2023. Can llms be safely released? evaluating the impact of red teaming on language model behavior. *arXiv preprint arXiv:2304.10685*.
- Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Riviere, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.
- Hugo Touvron, Thibaut Lavril, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.
- Jason Wei, Xuezhi Wang, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.
- Laura Weidinger, Jonathan Uesato, Jack Rae, and 1 others. 2021. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359*.

Laura Weidinger, Jonathan Uesato, Maribeth Rauh,	636
Conor Griffin, Po-Sen Huang, John Mellor, Amelia	637
Glaese, Myra Cheng, Borja Balle, Atoosa Kasirzadeh,	638
and 1 others. 2022. Taxonomy of risks posed by lan-	639
guage models. In <i>Proceedings of the 2022 ACM con-</i>	640
<i>ference on fairness, accountability, and transparency</i> ,	641
pages 214–229.	642
Andy Zou, James Zou, and 1 others. 2023. Univer-	643
sal and transferable adversarial attacks on aligned	644
language models. <i>arXiv preprint arXiv:2307.15043</i> .	645