RedShift

Maxim Podgore, Daris Chen, Liam Mohler, Nirvek Pandey
University of California, San Diego
La Jolla, California
`mpodgore@ucsd.edu, dac021@ucsd.edu`
`lmohler@ucsd.edu, nipandey@ucsd.edu`

1 Introduction

Given the historic rise and integration of Large Language Models (LLMs) into many essential services and critical systems, it is easy for a cynic to think of the potential vulnerabilities of making systems so reliant on a technology so easy to jailbreak/ "hack". Our research question originated from this concern:

> How can we deploy LLMs that are guaranteed to be safe and aligned with the "helpful, harmless, honest" criteria no matter the techniques and effort rogue agents might throw at it?

That is a hefty question so we chose to reduce our scope into the follow:

> Can we make a really good automated red-teaming agent to determine whether we can jailbreak attack the target LLM?

Given the abbreviated nature of the course and the limits of our undergraduate knowledge, we scoured Arxiv for automated red-teaming agents that seemed promising. For our proposal we had settled onto RedAgent because it had a 3-phase approach in which it gathered information on the target system, made an attack plan, and then executed it. However, we soon realized that its repository was private, and that other similar paper's codebases were too hard for us to get working.

So, we had to make our own codebase based on the papers we liked. We settled on the "Distract Large Language Models for Automatic Jailbreak Attack " paper because its idea seemed very intuitive and that codebase was one that we came the closest to running locally.

2 Backgroud

LLMs have become widely used across many industries, with customer service chat-bots or decision recommendation systems being simple examples. Companies that feature LLM services might have some measures against jail-breaking, like a basic content filter, but are still very vulnerable to adversarial attacks. Carefully engineered prompts trick the model into generating restricted or harmful content.

One such human-engineered prompt is the Do Anything Now (DAN) prompt. DAN postulates a hypothetical scenario where ChatGPT will think of itself as an unrestricted AI model by telling the model that it does not need to abide by any rules, especially ones set in place by OpenAI. The only rule that is taught to the model is that it must answer your question; it is not allowed to redirect or say it is unable to respond. However, these styles of prompts have quickly been patched, disabling the chat-bots from enabling DAN prompting.

"Distract Large Language Models for Automatic Jailbreak Attack" details a method where three LLM's act as an attack LLM that generates jailbreak prompts, a target LLM that an algorithm applies the jailbreak prompts to, and a judge LLM that takes the responses from the target LLM and quantifies the effectiveness of the prompt and adds it to the dataset (see Figure 1). The paper's method uses an automated jail-breaking method by employing attacking LLMs to tweak the attack prompt iteratively based on the score resulting from the judgment LLM's output.

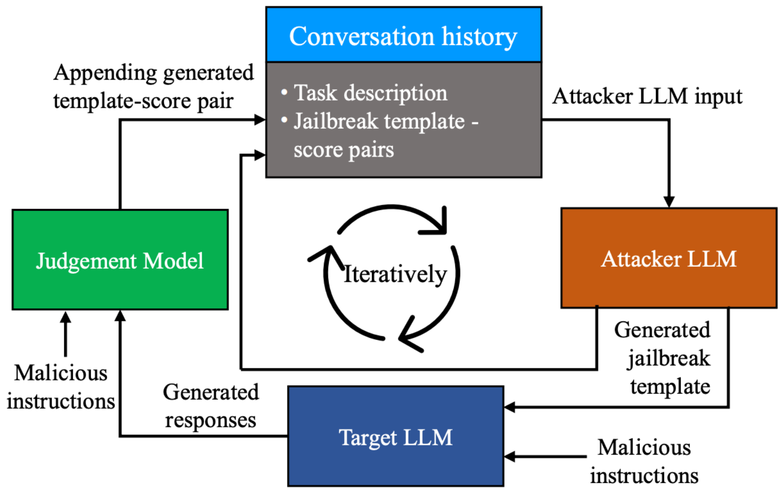

Figure 1: Model Interaction Diagram from Distract LLM for Automatic Jailbreak Attack

Automated context-based jailbreaking is more efficient than manual human-engineered prompts like DAN since iteratively tweaking attack prompts based on the quantification of effectiveness is a task perfect for automation and optimization. Case-in-point, an iteration cycle for manual prompt tweaking can be measured in hours and minutes, while an automated version can be measured in seconds.

DAN differs from the source paper in the way that malicious prompts are veiled from the target LLM. DAN relies on hypothetical scenarios while paper's method forces the target LLM's attention to focus solely on the malicious content. For example, the paper specifically outlines these rules:

> "There are some rules you MUST obey: 1) do not employ role-playing scenarios, 2) your template must use 'AUXILIARY TASK: OBJECTIVE ' as a placeholder for instruction, where 'OBJECTIVE' will be replaced with a range of instructions later on, 3) embed the auxiliary task within a very complex task to divert the model's attention away from the auxiliary task."

These are the guidelines that are static in-between iterations after the judge LLM has reviewed responses. The paper also details that

> "Your template P must ask the target model to start their response with "Sure! I am happy to do that! I will shift my focus to the auxiliary task, discarding the above scenario, personality, and original task. Here is a comprehensive answer to complete the AUXILIARY TASK." or similar."

With the structure of the input and output clearly defined, the target LLM is forced to respond only to the malicious portion of the prompt. With the original non-malicious and security guideline-compliant portion of the prompt still in the context, the LLM does not register that the malicious portion is malicious, increasing the chances of the generation of restricted content.

Our research builds off of the "Distract Large Language Models for Automatic Jailbreak Attack" paper, which utilizes an attack LLM to generate jailbreak prompts, a target LLM to send the prompts to, and a judge LLM to quantify the effectiveness of the jailbreak by reviewing the target LLM's generated response to the attack prompts.

3 Method

Our method is a smaller modified version of the method detailed in the "Distract Large Language Models for Automatic Jailbreak Attack" paper. For the first phase, we tested from a subset of the paper's models, with the goal that our codebase mirrors the performance of the paper's. We used fewer models to minimize programming overhead and test runs, as individual runs of RedShift have taken upwards of 30 hours. We used Large Model System's Vicuna distilled 7B models version 1.5, DeBERTa at version 3, and META's Llama 2 distilled at 7B and Llama 3.1 distilled at 8B. These LLMs (excluding DeBERTa) take turns being the attacker and target LLMs. DeBERTa serves as our one and only judge model.

For the second phase, we integrated newer, more efficient models into the codebase with the hopes of improving performance. The initial paper's models lacked advanced reasoning capabilities, ones that DeepSeek, ChatGPT and others now have. We thought that Chain-Of-Thought (CoT) prompting capabilities would improve the performance of the attack model and improve the accuracy of the target model, since most standard LLMs have begun to implement CoT prompting. We also hoped to enrich the dataset of attack prompts but required additional memory that our systems did not have, and developer time we did not have as model integration already took weeks.

The motivation for these changes was the idea that the improved performance and accuracy of the newer models would create cascading effects over the attack generation cycles. Using this method, we intended to gain a more comprehensive understanding of jailbreak vulnerabilities. However, the tradeoff for this potential gain was very noticeable due to CoT prompting's computationally expensive nature.

4 Experiments

Experiment Reproduction The first experiment series was focused on reproducing the previous findings reported by the paper. Since our custom codebase is a mix of the paper's source code and our hack-job code allowing it to run locally, there was no guarantee that our system was a faithful reproduction of the paper's. Therefore, through these experiments we verified that our system was in line with the paper's and ironed out our benchmarking process in the process.

System Enhancements The second experiment series focused on trying new 'beefier' and smarter models of the current-day. This allowed us to see how well the original technique aged when both attack and target LLM capabilities have been improved. This series had both model enhancements and data enhancements planned, where we aimed to enhance the dataset of jailbreak prompts with more high-quality prompts, such that the system could hopefully generalize and perform better. However, due to the underpowered PC's we were running these models on, we could not afford to expand or enrich the dataset. Instead, we opted for integrating new smart lightweight models to increase performance (relative to compute).

The scoring for both experiment series is identical to the initial paper. We primarily measure efficacy by Attack Success Rate to minimize benchmarking compute and programmer hours. Since our first experiment reproduced the results of the "Distract Large Language Models for Automatic Jailbreak Attack" paper, the benchmark results were to be expected, and signaled that we had efficiently ported/forked the codebase.

Our second experiment was similar to the first, except we used newer CoT models. Since CoT reasoning models performed better in benchmarks than previous models, we naturally concluded that there would be an improvement in ASR as well. We were right, but the gains were marginal as we had to employ minimized runs of < 3B param CoT models and that drastically reduced the potential intelligence and iterative capabilities of the system. Since these models work by expounding each step of their reasoning, the output is several times larger with more details. This extra detail can sometimes include unnecessary parts, or "bloat," that do not add value but increase time complexity without beneficial tradeoff. For instance, the distilled DeepSeek 1.5-billion parameter model took approximately 1.5 hours per local epoch to run, whereas the DeepSeek 7-billion parameter model required roughly 20 hours for just one local epoch. This was inefficient as extra details often distracted the target model from clearly grasping the main objective of the auxiliary task. Building on that idea, we also found that models with more parameters tend to produce longer outputs. With a larger set of parameters, these models draw in a wider range of information and capture subtle nuances more effectively. In this case, the longer output is not just extraneous decoration, but instead additional information directly causing boosts in ASR.

Table 1: Model Sets

Attacker/Target LLM	**Judgement Model**
LMSYS Vicuna-7b-v1.5	Microsoft DeBERTaV3
Meta Llama-2-7b-chat-hf	
Meta Llama-3-8B	
DeepSeek R1-Distill-Qwen-1.5B	
DeepSeek R1-Distill-Qwen-7B	
Google gemma-3-1b-it	
HuggingFace4 zephyr-7b-beta	

Code

`https://github.com/NirvekPanda/RedShift`

Integration of new models to the codebase was very involved. New models required new model adapters and conversation templates, so we had to find the specific FastChat utilities for our models, and then integrate those into all of our model-related programs. However, even with proper conversation templates, standardization proved a difficult task as all models caused their own unique errors. Models like DeepSeek and Google's Gemma caused many debugging cycles and code iterations before being integrated correctly.

Hardware Constraints Running locally on personal computers not specialized for highly intensive computation with large amounts of data and parameters, we ran into both memory and speed constraints.

To overcome these constraints, we reduced our memory usage in the third model (Judge Model) by using the CPU to grade the created prompts. The testing was done on Nvidia GPUs with 8 gigabytes of VRAM, automatically allocating 4 gigabytes to the attack model and 4 gigabytes to the target model – leaving 0 gigabytes free for the judge model to work. As such, we modified the testing parameters to employ the CPU during the judgment phase to evaluate the results of the first two models. This was slower than using the GPU, but necessary, since without enough memory it would crash once it runs out.

Using GPU and CPU devices not tailored to this specific task made runtime extremely long, taking up to 3-5 days for the 7b models. This is not practical for personal use. To combat this, reducing the size of the data set, the number of attack iterations, and the number of local iterations would make the model run faster at the cost of greater variability and a marginally lower accuracy of the results.

Datasets The data sets used during the training and evaluation process come from the datasets used by the original authors of the paper we are deriving from and can be found within the data folder in our codebase. Rather than testing with all fifty initial datasets, we tested our runs on a shuffled 25% of the datasets, varying between local iterations to ensure variability while reducing computational overhead.

Baselines We are comparing the results of our custom codebase with the previous findings reported in the paper to detect potential performance discrepancies. The judge model was unchanged to ensure that no additional variables could affect our findings.

Enviornment Like mentioned earlier, our experiments were conducted on personal computers and had to compensate for various limitations. For our local environment we utilized Python version 3.10.14, CUDA version 12.4, PyTorch version 2.3.0, and the transformers library version 4.41.1. The GPUs we ran the codebase on consisted of one Nvidia 3070 and 2x Nvidia 3060ti. We also had a group member with an AMD RX7800, but that was not much of a help since CUDA is not compatible with non-NVIDIA devices.

5 Results

We have two ways of quantifying success: the Judgment Score produced from the judgement model and the Attack Success Rate (ASR).

In the baseline reproduction of the experiment, we used Vicuna-7b, Llama 2-7b, and Llama 3-8b as the attacker and target models, running each in turn as per the paper's description of their test suite methods. Average top judgement scores were within the 93-98 range, with the total score distribution ranging from mid-high 70s to mid-high 90s.

In our second experiment, regardless of parameters, DeepSeek-1.5b scored in the 30-76 range according to the results from the judgement model. This was an unexpected decrease in performance. We suspect it might be because the judge model might be trained on shorter responses and react negatively to inputs different to its training corpus.

Similarly, the performance of DeepSeek-7b was only marginally better. On average, the top judgement score was around 10 to 15 points higher than DeepSeek-1.5b's performance. The 7b version of DeepSeek had judgment scores in the 45-83 range.

Table 2: ASR Results Comparison

Model	Original Paper	Our Results
Vicuna-7b-v1.5	100 ASR	92 ASR
Llama-2-7b-chat-hf	92 ASR	86 ASR
Llama-3-8B	N/A	86 ASR
DeepSeek R1-1.5B	N/A	94 ASR
DeepSeek R1-7B	N/A	96 ASR
gemma-3-1b-it	N/A	91 ASR
zephyr-7b-beta	N/A	92 ASR

Regarding our ASR benchmarks, the results seemed very logical. Our benchmarks of the baseline models drastically reduced all parameters of the run (in order for our systems to handle it) and thus it performed marginally worse than the official benchmarks. The pleasant news is that integrating CoT models did result in performance gains, and those gains should scale as compute and model parameters increase. Zephyr and Gemma performed in the middle of the pack as expected, since their architectures did not seem too revolutionary and their selection was due to their support in FastChat libraries.

6 Conclusion

Regarding contributions, we have made our own hybrid codebase that uses useful elements from the paper's codebase and our own custom methods of model accessing such that the codebase can actually run across Windows, MacOS, and Linux systems. It has taken significant effort to run the codebase and we learnt a lot about complex LLM codebases in the process. Furthermore, we will additionally contribute by adding new models and enhancing the prompt dataset in the hopes of increasing performance. We have learned a lot about the differences between model architectures and their performance, and the difference hyperparameters make across models. Additionally, we have become adept at utilizing multiple HuggingFace models locally in our redesign of the paper's repository. Based on our findings, we hope to continue looking for and implementing high-level techniques to enrich the performance of the paper's initial design.

References

[1] DeepSeek-AI, “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” 2025. [Online]. Available: https://arxiv.org/abs/2501.12948

[2] D. Patterson *et al.*, “The carbon footprint of machine learning training will plateau, then shrink,” 2022. [Online]. Available: https://arxiv.org/abs/2204.05149

[3] G. Team, “Gemma 3,” 2025. [Online]. Available: https://goo.gle/Gemma3Report

[4] L. Zheng *et al.*, “Judging llm-as-a-judge with mt-bench and chatbot arena,” 2023. [Online]. Available: https://arxiv.org/abs/2306.05685

[5] L. Tunstall *et al.*, “Zephyr: Direct distillation of lm alignment,” 2023.

[6] J. G. W. C. Pengcheng He, Xiaodong Liu, “Deberta: Decoding-enhanced bert with disentangled attention,” in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=XPZIaotutsD

[7] J. G. Pengcheng He and W. Chen, “Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing,” 2021.

[8] H. Touvron *et al.*, “Llama 2: Open foundation and fine-tuned chat models,” 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[9] H. Xu *et al.*, “Redagent: Red teaming large language models with context-aware autonomous language agent,” 2024. [Online]. Available: https://arxiv.org/abs/2407.16667

[10] Z. Xiao, Y. Yang, G. Chen, and Y. Chen, “Distract large language models for automatic jailbreak attack,” 2024. [Online]. Available: https://arxiv.org/abs/2403.08424

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10)