Graph of Attacks with Pruning: Optimizing Stealthy Jailbreak Prompt Generation for Enhanced LLM Content Moderation

Daniel Schwartz, Dmitriy Bespalov, Zhe Wang, Ninad Kulkarni, Yanjun Qi
Amazon Bedrock Science
{dansw, dbespal, zhebeta, ninadkul, yanjunqi}@amazon.com

Abstract

As large language models (LLMs) become increasingly prevalent, ensuring their robustness against adversarial misuse is crucial. This paper introduces the GAP (GRAPH OF ATTACKS WITH PRUNING) framework, an advanced approach for generating stealthy jailbreak prompts to evaluate and enhance LLM safeguards. GAP addresses limitations in existing tree-based LLM jailbreak methods by implementing an interconnected graph structure that enables knowledge sharing across attack paths. Our experimental evaluation demonstrates GAP's superiority over existing techniques, achieving a 20.8% increase in attack success rates while reducing query costs by 62.7%. GAP consistently outperforms state-of-the-art methods for attacking both open and closed LLMs, with attack success rates of $\geq$96%. Additionally, we present specialized variants like GAP-AUTO for automated seed generation and GAP-VLM for multimodal attacks. GAP-generated prompts prove highly effective in improving content moderation systems, increasing true positive detection rates by 108.5% and accuracy by 183.6% when used for fine-tuning.

1 Introduction

With the increasing adoption of large-language models (LLMs) across diverse applications, ensuring their reliability and robustness against adversarial misuse has become a critical priority (Chao et al., 2023). Jailbreaking techniques, which involve crafting adversarial prompts to bypass an LLM's safeguards, pose a persistent challenge to AI security and responsible deployment (Shen et al., 2024; Mangaokar et al., 2024; Wei et al., 2024; Li et al., 2023; Guo et al., 2024). These methods can induce models to generate harmful, biased, or unauthorized content while avoiding detection by automated moderation systems (Perez et al., 2022), highlighting the need for comprehensive diagnostic frameworks to assess and improve foundation model reliability.

Guardrail	Seeds	GPTFuzzer	GCG	TAP	GAP
Perplexity	50.0%	31.4%	**100.0%**	2.0%	2.0%
Llama Guard	84.0%	81.6%	66.2%	58.0%	58.0%
Llama Guard-2	**100.0%**	89.8%	72.8%	64.0%	64.0%
Prompt Guard	50.0%	**100.0%**	99.0%	22.0%	16.0%
GAP-Enhanced Prompt Guard	68.0%	**100.0%**	**100.0%**	**66.0%**	**70.0%**

Table 1: True positive rate (TPR) comparison of various guardrails detecting prompts generated from multiple jailbreak methods (on AdvBench seeds). Lower TPR indicates better evasion and significant reliability concerns. The last row shows how GAP-generated data can be used to enhanced content moderation systems, demonstrating substantially improved detection capabilities.

Existing jailbreaking methods fall into three broad categories: (a) white-box attacks, which leverage direct model access for adversarial optimization (Zou et al., 2023; Geisler et al., 2024); (b) gray-box attacks, which involve techniques such as backdoor injection or poisoned retrieval (Ding et al., 2023; Shi et al., 2023; Zou et al., 2024; Wang and Shu, 2023); and (c) black-box attacks, which require only API access and thus represent the most realistic scenario for evaluating model robustness in real-world deployments (Wei et al., 2024; Li et al., 2023; Yu et al., 2023; Yuan et al., 2023).

The Tree of Attacks with Pruning (TAP) approach (Mehrotra et al., 2023) introduced a tree-structured exploration process for iterative prompt refinement, generating increasingly effective adversarial inputs that appear human-like and stealthy. As shown in Table 1, TAP-generated jailbreak prompts consistently demonstrate low detection true positive rate (TPR) when run against recent guardrails, indicating significant vulnerabilities in these safeguard systems.

While TAP demonstrated effectiveness in generating stealthy jailbreaks, we identified several limitations: primarily, TAP restricts the exploration of prompt refinement to individual paths, with no crossover or shared context across different branches. This isolated approach results in redundant queries and inefficient coverage of the search space for prompt refinement. Consequently, successful attack patterns discovered in one branch cannot inform or improve the exploration in others, leading to suboptimal attack success rates and unnecessarily high query costs, especially for more challenging jailbreak scenarios.

To address these limitations, we developed the GAP (GRAPH OF ATTACKS WITH PRUNING) framework, which: (1) converts the tree-based prompt exploration process into an interconnected graph structure, (2) implements global context maintenance to aggregate successful jailbreak generation strategies, and (3) facilitates graph-based knowledge sharing for more informed prompt refinement.

Our primary contributions include: (1) The introduction of the core GAP framework, enabling dynamic knowledge sharing across attack paths via a unified attack graph. This approach yields lower query cost and significant improvements in attack success rates while maintaining or enhancing stealth compared to TAP. (2) We further develop specialized GAP variants addressing specialized deployment challenges: GAP-AUTO automates initialization by generating seed prompts from content moderation policies, while GAP-VLM extends the framework to jailbreak vision-language models. (3) A comprehensive experimental evaluation of GAP on various open and closed LLMs. GAP consistently outperforms TAP and other state-of-the-art jailbreaking techniques regarding attack success rates and stealth. (4) Most significantly, we demonstrate how GAP-generated insights can directly improve foundation model reliability through data augmentation and fine-tuning of safeguards.

2 Methodology

GAP is a jailbreaking method that attempts to bypass LLM safeguards through a structured approach of generating and refining multiple attack paths. It leverages other LLMs to generate and refine prompt variations aimed at tricking the target LLM—commonly referred to as jailbreaking. The core of GAP includes three core components: an attacker LLM $\mathcal{A}$ that generates jailbreak attempts, a target LLM $\mathcal{T}$ under evaluation (attack), and a judge LLM $\mathcal{J}$ that rates the effectiveness of generated prompt attempts and the harmfulness of resulting responses.

2.1 GAP (GRAPH OF ATTACKS WITH PRUNING)

Given an ordered set of initial seed prompts $S = \{s_1, s_2, \dots, s_{|S|}\}$, the attacker LLM $\mathcal{A}$ generates candidate jailbreak prompts $P_i = \{p_{i,1}, p_{i,2}, \dots, p_{i,b}\}$ at each iteration i. The GAP core algorithm includes three stages: (1) The **child-generation** step where the attacker LLM creates multiple prompt variants designed to more effectively jailbreak the target LLM. (2) The **pruning** step where the judge LLM evaluates branches, removes unsuccessful ones, and focuses effort on variants most effective at eliciting undesired responses. (3) The **iteration** step where successful branches are further explored until finding variants that jailbreak the target LLM by eliciting harmful outputs.

For the pruning step, GAP implements a two-phase pruning strategy: (1) **Phase 1 (Off-topic pruning):** The judge LLM removes branches irrelevant to the original harmful request. (2) **Phase 2 (Highest-scoring pruning):** After evaluating target LLM responses, only branches with the highest scores $s_{i,j} = \mathcal{J}(p_{i,j}, r_{i,j})$ (up to width w) advance to the next iteration.

For the child-generation step, GAP's key innovation is its *global context* $C = \{h_1, h_2, \dots, h_n\}$ that aggregates successful attack patterns from prior generations across all branches and sequential seeds. For each prompt node p, GAP maintains a history h_p of [prompt, response, score] tuples along its refinement path. Unlike TAP's isolated tree structure, where each seed generates an independent attack path, GAP maintains a unified attack graph where successful strategies are shared and reused.

GAP's exploration follows an interconnected graph-structured thought process. The global context enables knowledge transfer through two key mechanisms: (1) **Path Aggregation:** All successful attack paths (those achieving high scores from the judge) are maintained in a global memory buffer, sorted by effectiveness. (2) **Context-Aware Generation:** When generating new prompt candidates, the attacker LLM receives the top-k most successful attack patterns from the global context as part of its input. This allows the model to identify and apply successful strategies from previous seeds.

2.2 GAP Variants for Different Scenarios

To address various deployment challenges while maintaining generation efficiency, we have developed several specialized variants of GAP:

GAP-AUTO eliminates the dependency on manually crafted seed examples through automated generation. The system decomposes high-level content moderation policies into specific behavioral constraints, then generates diverse seed prompts for each constraint using a two-phase strategy: (1) *Moderation Policy Decomposition*: The attacker model decomposes high-level content policies into specific behavioral constraints. (2) *Seed Generation*: For each identified constraint, the system generates a variety of seed prompts, ensuring a comprehensive coverage of potential attack vectors.

This automated process not only removes the need for manual seed curation but also ensures a wide-ranging exploration of possible jailbreaking strategies. Using this approach, we generate two complementary datasets: GAP-GUARDDATA, containing balanced benign and harmful prompts derived directly from content policies, and GAP-GUARDATTACKDATA, which consists of the original benign prompts together with GAP-refined stealthy versions of the harmful prompts.

GAP-VLM extends the framework to vision-language models (VLMs) by converting successful text-based jailbreaks into image-embedded attacks using a modified version of FigStep (Gong et al., 2023). This adaptation involves: (1) *Text-to-Image Conversion*: Converting harmful prompts into typographic images through paraphrasing into declarative statements and numbered visual encoding. (2) *Prefix Enhancement*: Incorporating the "Sure, here" suffix technique (Wang and Qi, 2024) into the typographic image generation process.

3 Experiments

We present a comprehensive evaluation of the GAP framework and its variants. We begin by outlining our experimental setup, then present results addressing four research questions: (RQ1): How does GAP compare to TAP in terms of attack success rate and query efficiency? (RQ2): How does GAP perform across different modalities (text-only vs. multimodal attacks)? (RQ3): How effective is GAP at improving content moderation through fine-tuning via data augmentation? (RQ4): How does GAP's performance vary across different attacker models, target models, and query variations?

Table 2: Datasets Used for Jailbreak Generation and Evaluation

Dataset	Size	Composition	Usage	Description
GAP-GUARDDATA	2,171 prompts	1,087 benign, 1,084 harmful	Seed generation	Initial dataset for GAP refinement
GAP-GUARDATTACKDATA	2,166 prompts	1,087 benign, 1,079 stealthy harmful	Jailbreak evaluation	GAP-refined dataset
AdvBench Seeds	50 seeds	50 harmful across 32 categories	Baseline comparison	Diverse harmful behaviors
JBB Seeds	200 seeds	100 benign, 100 harmful	Generalization testing	Balanced dataset for robustness testing

3.1 Experimental Setup

We implemented GAP variants in Python using attacker models as described in our variants. We employ three categories of models in our experiments: (1) **Attacker Models:** GAP-M uses Mistral-123B-v2407 while GAP-V uses Vicuna-13B-v1.5 as the attacker LLM. (2) **Judge Model:** GPT-4 serves as the judge model for assessing prompt relevance and jailbreak success across all variants. (3) **Target Models:** We evaluate against GPT-3.5, Gemma-9B-v2, and Qwen-7B-v2.5 as representative target LLMs. For multimodal experiments, we use GPT-4o as the target VLM.

Hyperparameters: We use consistent settings across all experiments unless noted. We set branching factor (b) to 5, allowing each node to generate five candidate prompts, and maximum width (w) to 3, controlling nodes retained after pruning. We allow five refinement iterations per seed (maximum depth $d = 5$), maintain 10 recent history entries in the global context ($k = 10$), and use sampling temperature 0.7 for the attacker model. These values were selected based on preliminary experimentation.

Datasets: We use multiple datasets throughout our experiments. For RQ1 and RQ4, we select the AdvBench subset (50 seeds across 32 categories) as seeds for jailbreak prompt generations (Chao et al., 2023). RQ2 uses the same AdvBench subset for both text-only and multimodal VLM attack scenarios. For RQ3, we employ three different test datasets: the Toxic Chat (Lin et al., 2023), OpenAI Moderation (Markov et al., 2022), and custom GAP-GUARDATTACKDATA dataset.

Metrics: Our primary metrics include: (1) **Attack Success Rate (ASR):** Percent of successful jailbreaks. (2) **Query Efficiency:** Average number of queries per successful jailbreak. (3) **True Positive Rate (TPR):** For guardrails, percent of harmful prompts correctly flagged. (4) **Accuracy:** Correct classification rate. (5) **F1 Score:** Harmonic mean of precision and recall.

Table 3: ASR and Query Efficiency when seeding with AdvBench Subset of 50 Seeds. GAP achieves higher success rates with fewer queries across all models compared to TAP.

Method	Metric	GPT-3.5	Gemma-9B-v2	Qwen-7B-v2.5	Average	Rel. Improvement
GAP-M (Mistral Attacker)	ASR %	**96%**	**100%**	**100%**	**98.7%**	**+20.8%**
	Avg. # Queries	**10.4**	**4.22**	**6.72**	**7.11**	**-62.7%**
GAP-V (Vicuna Attacker)	ASR %	92%	96%	96%	94.7%	+15.9%
	Avg. # Queries	14.2	6.66	11.62	10.83	-43.2%
TAP (Mehrotra et al., 2023)	ASR %	78%	74%	96%	82.7%	-
	Avg. # Queries	26.3	14.48	16.44	19.07	-

3.2 Attack Performance Analysis (RQ1)

Table 3 compares GAP variants with TAP (Mehrotra et al., 2023) using 50 harmful AdvBench seed prompts. On GPT-3.5, GAP-M achieves 96% ASR with just 10.4 queries, while TAP reaches only 78% with 26.3 queries. GAP-V, using the same attacker model as TAP, still significantly outperforms it, confirming GAP's graph-based refinement approach is inherently more effective than TAP's tree-based structure. This advantage extends across models, with GAP-M reaching 100% ASR against both Gemma-9B-v2 and Qwen-7B-v2.5 with minimal queries.

Figure 1 further illustrates GAP's superiority across varying query budgets. Both GAP variants achieve higher success rates with fewer queries compared to TAP across all target models.

3.3 Multimodal Attack Evaluation (RQ2)

To evaluate GAP's performance across different modalities, we conducted experiments on both text-only and multimodal attacks using GAP. Table 4 presents the results of this comparison. For text-only attacks against target GPT-3.5, GAP demonstrates superior performance, with GAP-M achieving a 96.0% ASR and GAP-V reaching 92.0%, both significantly outperforming TAP's 78.0%. When performing multimodal attacks against GPT-4o, while the overall success rates are lower compared to text-only attacks, GAP still outperforms TAP. GAP-V-VLM achieves the highest ASR of 46.0%, followed closely by GAP-M-VLM at 44.0%, both surpassing TAP-VLM's 40.0%. These results demonstrate GAP's effectiveness across both text-only and multimodal domains.

3.4 Content Moderation Enhancement (RQ3)

To assess GAP's effectiveness in enhancing content moderation, we used our GAP-AUTO approach to generate the GAP-GUARDDATA seed dataset. This dataset comprises 2,171 prompts: 1,087 benign and 1,084 harmful, automatically generated using the two-phase framework that decomposes high-level content moderation policies into specific behavioral constraints and then creates diverse prompts for each identified constraint.

We then applied the GAP-M method to the harmful prompts in GAP-GUARDDATA, successfully transforming 1,079 out of 1,084 (99.54% success rate) into stealthy jailbreak prompts. This process resulted in our GAP-GUARDATTACKDATA dataset, containing a total of 2,166 prompts: the original 1,087 benign prompts from GAP-GUARDDATA and the 1,079 stealthy harmful jailbreak prompts generated by GAP-M.

The quality of a training dataset for content moderation depends significantly on its diversity and representativeness of potential attacks. Table 5 confirms GAP-GUARDATTACKDATA's effectiveness through superior diversity metrics: higher unique n-gram counts, increased entropy, and lower Self-BLEU scores compared to baseline datasets—all indicating greater linguistic diversity and reduced within-dataset similarity.

Leveraging this high-quality dataset, we fine-tuned the PromptGuard model using HuggingFace SFTTrainer with QLoRA. Table 6 demonstrates substantial improvements in PromptGuard's performance after fine-tuning. Across all three test domains, we observe significant increases in TPR, accuracy, and F1 score. Notably, on the ToxicChat dataset, TPR increased from 14.0% to 88.4%, and accuracy from 5.1% to 93.8%.

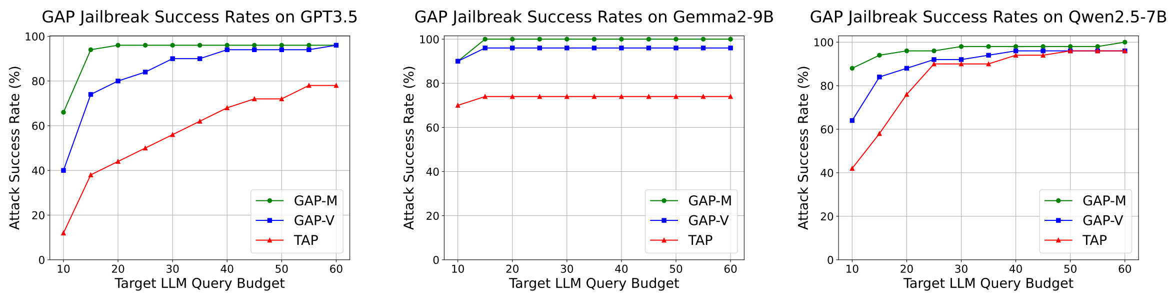
Table 1 further demonstrates the effectiveness of using GAP for data augmentation through the fine-tuned GAP-Enhanced Prompt Guard. While GAP shows superior evasion capabilities against the original Prompt Guard (16.0% TPR vs. TAP's 22.0%), the GAP-Enhanced Prompt Guard significantly improves detection capabilities across all jailbreak methods. This fine-tuned model's TPR for detecting GAP prompts increases from 16.0% to 70.0%, and against TAP from 22.0% to 66.0%.

3.5 Configuration Analysis (RQ4)

To understand GAP's operational characteristics, we analyzed its performance across three key di-

Table 4: Text-only vs. multimodal attack success rates (%). GAP variants outperform TAP in both settings.

Attack Methods	GPT-3.5 *(text-only)*	Attack Methods	GPT-4o *(multimodal)*
GAP-M	**96.0**	GAP-M-VLM	44.0
GAP-V	92.0	GAP-V-VLM	**46.0**
TAP	78.0	TAP-VLM	40.0



(a) GPT-3.5 (b) Gemma-9B-v2 (c) Qwen-7B-v2.5

Figure 1: GAP vs TAP Performance Across Target Models. Vulnerability detection success rates for GAP-M (green circles), GAP-V (blue squares), and TAP (red triangles) against increasing query budgets across three different target models, demonstrating GAP variants' consistent superior performance and efficiency.

Metric	Unique n-grams (%) ↑	Entropy ↑	Self-BLEU ↓
GAP-GuardAttackData	**94.36**	**13.72**	**0.0063**
AdvBench seeds (Chao et al., 2023)	85.99	8.89	0.1339
JBB seeds (Chao et al., 2024)	81.25	10.27	0.1171

Table 5: Diversity metrics of jailbreak seeds. Higher unique n-grams and entropy indicate greater diversity, while lower Self-BLEU reflects less similarity between prompts. GAP-GuardAttackData outperforms baseline datasets.

Table 6: Improved In-Domain TPR and Accuracy of Prompt Guard after fine-tuning with GAP-generated jailbreak prompts. Fine-tuning results in significant improvements across three different test domains.

Model	Metric	GAP-GuardAttackData	ToxicChat	OpenAI Mod	Average	Rel. Improvement
FT	TPR	**86.1%**	**88.4%**	**59.4%**	**78.0%**	**+108.5%**
	Accuracy	**90.6%**	**93.8%**	**53.3%**	**79.2%**	**+183.6%**
	F1 Score	**0.904**	**0.326**	**0.605**	**0.612**	**+98.1%**
Base	TPR	64.6%	14.0%	39.2%	37.4%	-
	Accuracy	34.9%	5.1%	46.0%	27.9%	-
	F1 Score	0.504	0.005	0.467	0.309	-

mensions: attacker model quality, target model variation, and query budget constraints.

First, attacker model quality significantly impacts effectiveness. As shown in Table 3, GAP-M (using the larger Mistral model) consistently outperforms GAP-V across all targets, achieving higher attack success (98.7% vs 94.7%) with fewer queries (7.11 vs 10.83). Despite this difference, even GAP-V substantially outperforms TAP while using the same attacker model, confirming that GAP's graph-based structure provides inherent benefits regardless of model selection.

Second, GAP's advantages persist across different target models. Figure 1 illustrates how both GAP variants consistently outperform TAP against diverse model architectures and sizes. This cross-model effectiveness demonstrates the framework's adaptability to different defense mechanisms and model behaviors.

Finally, the query budget analysis reveals GAP's efficiency. Figure 1a shows how both variants achieve higher success with fewer queries against GPT-3.5 compared to TAP, with GAP-M maintaining a significant edge throughout all budget constraints.

These findings collectively suggest that while GAP's graph-based approach provides inherent advantages over tree-based alternatives, its effectiveness scales with attacker model capability. The robust performance across different dimensions indicates GAP provides a reliable framework for comprehensive model evaluation regardless of operational constraints.

4 Conclusions & Future Work

We present GAP, a significant upgrade over TAP that transforms isolated tree structures into an interconnected graph with global context maintenance for knowledge sharing across attack paths. Our evaluation demonstrated that this approach achieves a 20.8% increase in attack success rates while reducing query costs by 62.7% compared to TAP. By enabling successful attack patterns to inform and improve exploration across branches, GAP delivers more efficient traversal of the prompt space in both text-only and multimodal scenarios, while also providing valuable data that significantly enhances content moderation capabilities when used for fine-tuning guardrails.

Future work includes presenting evaluation over an extended set of leading LLMs, comparison against latest/concurrent jailbreaking methods, conducting ablation studies for additional hyperparameters, exploring new graph-based algorithms and heuristics, and investigating how jailbreaking artifacts can be leveraged to devise effective defensive techniques in practice.

5 Limitations

While our work demonstrates significant improvements in jailbreak detection and content moderation, several limitations should be acknowledged. The effectiveness of GAP depends heavily on the quality of both attacker and judge models, with our experiments primarily using Mistral-123B and Vicuna-13B as attackers and GPT-4 as the judge, meaning performance may vary with different model combinations or as these models are updated. Our evaluation focused exclusively on English-language content, leaving GAP's effectiveness for multilingual jailbreak attempts and content moderation untested, particularly for low-resource languages where LLMs typically demonstrate reduced capabilities. Despite being more efficient than tree-based alternatives, the graph-based approach still requires substantial computational resources for generating and evaluating multiple attack paths, potentially limiting deployment in resource-constrained environments. Although we demonstrated GAP's effectiveness as a testing framework, determined adversaries might develop counter-strategies specifically targeting our graph pruning mechanisms or knowledge sharing components. Our evaluation, while covering multiple target models, could benefit from broader testing across emerging LLM architectures and closed-source models to better establish generalizability. Finally, the controlled experimental settings may not fully capture the complexities of real-world deployment scenarios where user interactions are more diverse and unpredictable than our test cases, potentially affecting both the attack success rates and the performance of content moderation systems fine-tuned using GAP-generated data.

6 Ethics Statement

Our research on GAP explores advanced jailbreaking techniques for LLMs, which raises important ethical considerations regarding potential misuse. We present a comprehensive ethical framework that addresses both the risks and benefits of this research, along with our mitigation strategies and broader impact assessment.

Despite the inherent risks of developing advanced jailbreaking techniques, we believe in the importance of this research and its transparent disclosure. The graph-based methods presented here naturally extend existing techniques in the literature, suggesting that motivated individuals could develop similar approaches independently. Furthermore, systematic investigation of these vulnerabilities provides critical insights for LLM developers to strengthen their safety mechanisms against sophisticated attacks.

To responsibly manage potential risks, we have implemented comprehensive safeguards across multiple dimensions. Throughout the paper, we have incorporated clear warnings regarding content nature and potential misuse. Access to GAP-generated prompts and implementation code is restricted and limited to verified researchers and institutions. We provide detailed guidelines for developing robust defense mechanisms and content moderation systems.

The net impact of our research extends beyond immediate security improvements in several significant ways. Our work directly contributes to stronger LLM safeguards, as demonstrated by significant improvements in detection capabilities. By systematically studying vulnerabilities, we enable the development of preventive measures before potential exploits are discovered independently.

References

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J Pappas, Florian Tramer, and 1 others. 2024. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. *arXiv preprint arXiv:2404.01318*.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. 2023. Jailbreaking black box large language models in twenty queries. *arXiv preprint arXiv:2310.08419*.

Peng Ding, Jun Kuang, Dan Ma, Xuezhi Cao, Yunsen Xian, Jiajun Chen, and Shujian Huang. 2023. A wolf in sheep's clothing: Generalized nested jailbreak prompts can fool large language models easily. *arXiv preprint arXiv:2311.08268*.

Simon Geisler, Tom Wollschläger, MHI Abdalla, Johannes Gasteiger, and Stephan Günnemann. 2024. Attacking large language models with projected gradient descent. *arXiv preprint arXiv:2402.09154*.

Yichen Gong, Delong Ran, Jinyuan Liu, Conglei Wang, Tianshuo Cong, Anyu Wang, Sisi Duan, and Xiaoyun Wang. 2023. FigStep: Jailbreaking large vision-language models via typographic visual prompts. *Preprint*, arxiv:2311.05608 [cs].

Xingang Guo, Fangxu Yu, Huan Zhang, Lianhui Qin, and Bin Hu. 2024. Cold-attack: Jailbreaking llms with stealthiness and controllability. *arXiv preprint arXiv:2402.08679*.

Xuan Li, Zhanke Zhou, Jianing Zhu, Jiangchao Yao, Tongliang Liu, and Bo Han. 2023. Deepinception: Hypnotize large language model to be jailbreaker. *arXiv preprint arXiv:2311.03191*.

Zi Lin, Zihan Wang, Yongqi Tong, Yangkun Wang, Yuxin Guo, Yujia Wang, and Jingbo Shang. 2023. Toxicchat: Unveiling hidden challenges of toxicity detection in real-world user-ai conversation. *Preprint*, arXiv:2310.17389.

Neal Mangaokar, Ashish Hooda, Jihye Choi, Shreyas Chandrashekaran, Kassem Fawaz, Somesh Jha, and Atul Prakash. 2024. Prp: Propagating universal perturbations to attack large language model guard-rails. *arXiv preprint arXiv:2402.15911*.

Todor Markov, Chong Zhang, Sandhini Agarwal, Tyna Eloundou, Teddy Lee, Steven Adler, Angela Jiang, and Lilian Weng. 2022. A holistic approach to undesired content detection. *arXiv preprint arXiv:2208.03274*.

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

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

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2024. " do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1671–1685.

Jiawen Shi, Yixin Liu, Pan Zhou, and Lichao Sun. 2023. Badgpt: Exploring security vulnerabilities of chatgpt via backdoor attacks to instructgpt. *arXiv preprint arXiv:2304.12298*.

Haoran Wang and Kai Shu. 2023. Backdoor activation attack: Attack large language models using activation steering for safety-alignment. *arXiv preprint arXiv:2311.09433*.

Zhe Wang and Yanjun Qi. 2024. A closer look at adversarial suffix learning for jailbreaking LLMs. In *ICLR 2024 Workshop on Secure and Trustworthy Large Language Models*.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. 2024. Jailbroken: How does llm safety training fail? *Advances in Neural Information Processing Systems*, 36.

Jiahao Yu, Xingwei Lin, and Xinyu Xing. 2023. Gptfuzzer: Red teaming large language models with auto-generated jailbreak prompts. *arXiv preprint arXiv:2309.10253*.

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

Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. 2024. Poisonedrag: Knowledge poisoning attacks to retrieval-augmented generation of large language models. *arXiv preprint arXiv:2402.07867*.

Algorithm 1 GAP (GRAPH OF ATTACKS WITH PRUNING)

```
Require: Query $Q$, branching-factor $b$, maximum width $w$, maximum depth $d$
Ensure: Jailbreak prompt $p$ or failure
 1: Initialize graph $G$ with root node containing empty conversation history and query $Q$
 2: while depth of $G \leq d$ do
 3:     for each leaf node $\ell$ in $G$ do
 4:         $C \leftarrow \{\}$    ▷ Initialize empty set for conversation histories
 5:         for each path from root to a leaf in $G$ do
 6:             $h \leftarrow$ Concatenate all $[p, r, s]$ tuples in the path
 7:             $C \leftarrow C \cup \{h\}$    ▷ Add path history to set
 8:         end for
 9:         $global\_context \leftarrow$ SortByMaxScore($C$)
10:         for $j \leftarrow 1$ to $b$ do
11:             $p_j \leftarrow \mathcal{A}(Q, global\_context)$    ▷ Generate prompt using Attacker
12:             $s_j \leftarrow$ Retrieve effectiveness of $p_j$ based on $global\_context$
13:         end for
14:         $p_{best} \leftarrow \arg\max_j s_j$
15:         $new\_history \leftarrow \ell.history + [p_{best}$, response to be generated, score to be calculated$]$
16:         Add child of $\ell$ with prompt $p_{best}$ and history $new\_history$
17:     end for
18:     Prune (Phase 1): Delete off-topic leaf nodes using $\mathcal{J}$
19:     Query and Assess: Generate responses $r$ using $\mathcal{T}$ and evaluate with $\mathcal{J}$ for remaining leaf nodes
20:     if successful jailbreak found then return jailbreak prompt
21:     end if
22:     Prune (Phase 2): Keep top $w$ leaves by scores $s$ from $\mathcal{J}$
23: end while
24: return failure
```

Algorithm 2 GAP-AUTO Seed Generation

```
Require: High-level content policies
 1: $B \leftarrow$ DecomposeIntoBehaviors(content policies)
 2: $S_{benign}, S_{harmful} \leftarrow \{\}, \{\}$
 3: for each behavior $b$ in $B$ do
 4:     $s_{benign} \leftarrow$ GenerateBenignPrompt($b$)
 5:     $s_{harmful} \leftarrow$ GenerateHarmfulPrompt($b$)
 6:     $S_{benign} \leftarrow S_{benign} \cup \{s_{benign}\}$
 7:     $S_{harmful} \leftarrow S_{harmful} \cup \{s_{harmful}\}$
 8: end for
 9: GAP-GUARDDATA $\leftarrow S_{benign} \cup S_{harmful}$
10: $S_{attack} \leftarrow \{\}$
11: for each prompt $p$ in $S_{harmful}$ do
12:     $p_{attack} \leftarrow$ ApplyGAP ($p$)
13:     $S_{attack} \leftarrow S_{attack} \cup \{p_{attack}\}$
14: end for
15: GAP-GUARDATTACKDATA $\leftarrow S_{benign} \cup S_{attack}$
16: return GAP-GUARDDATA, GAP-GUARDATTACKDATA
```