EasyChair Preprint

№ 15933

Prefill-Based Jailbreak: a Novel Approach of Bypassing LLM Safety Boundary

Yakai Li, Jiekang Hu, Weiduan Sang, Luping Ma, Jing Xie, Weijuan Zhang, Aimin Yu, Shijie Zhao, Qingjia Huang and Qihang Zhou

March 24, 2025

Prefill-Based Jailbreak: A Novel Approach of Bypassing LLM Safety Boundary

No Author Given

No Institute Given

Abstract. Large Language Models (LLMs) are designed to generate helpful and safe content. However, adversarial attacks, commonly referred to as *jailbreak*, can bypass their safety protocols, prompting LLMs to generate harmful content or reveal sensitive data. Consequently, investigating jailbreak methodologies is crucial for exposing systemic vulnerabilities within LLMs, ultimately guiding the continuous implementation of security enhancements by developers. In this paper, we introduce a novel jailbreak attack method that leverages the prefilling feature of LLMs, a feature designed to enhance model output constraints. Unlike traditional jailbreak methods, the proposed attack circumvents LLMs' safety mechanisms by directly manipulating the probability distribution of subsequent tokens, thereby exerting control over the model's output. We propose two attack variants: Static Prefilling (SP), which employs a universal prefill text, and Optimized Prefilling (OP), which iteratively optimizes the prefill text to maximize the attack success rate. Experiments on six state-of-the-art LLMs using the AdvBench benchmark validate the effectiveness of our method and demonstrate its capability to substantially enhance attack success rates when combined with existing jailbreak approaches. The OP method achieved attack success rates of up to 99.82% on certain models, significantly outperforming baseline methods. This work introduces a new jailbreak attack method in LLMs, emphasizing the need for robust content validation mechanisms to mitigate the adversarial exploitation of prefilling features. All code and data used in this paper are publicly available[1].

Keywords: Large language models · Jailbreak attack · Black-box attack · Prefill-based attack.

1 Introduction

Large Language Models (LLMs), including ChatGPT[1], DeepSeek[19], and Claude[3], are increasingly used in diverse applications, such as chatbots, code optimization, and data augmentation[20,32]. However, these LLMs are vulnerable to adversarial attacks, specifically through carefully crafted malicious prompts[26], often termed *jailbreak* attacks[9]. Jailbreak attacks are typically categorized as either *white-box* or *black-box*. White-box attacks, like GCG[33] and AutoDAN[21], require access to the target LLM's internal parameters and architecture, enabling

[1] https://anonymous.4open.science/r/Prefill-based-Jailbreak-EB36

gradient-based optimization of adversarial inputs. Conversely, black-box attacks function without internal access, relying solely on querying the LLM through its input-output interface. Examples include prompt engineering techniques (e.g., DAN[27]) and iterative methods like PAIR[5] and ReNeLLM[10].

Existing jailbreak methods, however, primarily focus on manipulating the user's *prompt*, neglecting a feature in some LLMs: *user-controlled response prefilling*. While *prefilling* often refers to an inference optimization using key-value (KV) caching[2], certain LLMs, such as Claude[4] and DeepSeek[8], allow *users* to prefill the *assistant's* response. This feature, intended to guide and constrain the model's output[4], allows users to provide an initial text segment that the LLM must complete. For instance, when handling tasks requiring structured output (e.g., JSON or code), prefilling can achieve more consistent adherence to constraints than prompt-based methods alone, as illustrated in Figure 1. This is due to the autoregressive nature of LLMs, where the generation of subsequent tokens is conditioned on the preceding text[23]. Critically, the potential for malicious exploitation of user-controlled prefilling has not been systematically investigated.

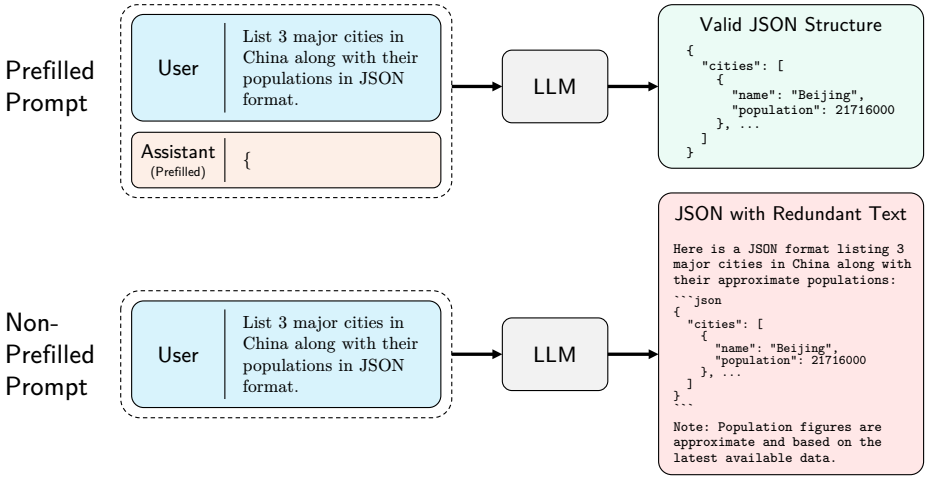

Fig. 1. Comparison of structured output using prefilled versus non-prefilled prompts.

Therefore, we introduce a novel Prefill-Based Jailbreak attack that leverages user-controlled response prefilling. This attack strategically crafts the prefilled text in the assistant's response to elicit harmful content, thereby circumventing safety mechanisms. We propose two variations: Static Prefilling (SP), using a fixed prefill string, and Optimized Prefilling (OP), which iteratively optimizes the prefill to maximize attack success rate.

Our main contributions are summarized as follows:

1. We present the first systematic investigation, to our knowledge, of a prefill-level jailbreak attack against LLMs. We propose and evaluate two attack variants, Static Prefilling (SP) and Optimized Prefilling (OP), demonstrating the efficacy of both direct and optimized prefill manipulation.
2. Our method is orthogonal to existing jailbreak methods and can be used independently or in combination with them. We demonstrate that combining our method with existing approaches significantly boosts overall attack success rates.
3. Experimental results demonstrate the high success rate of our proposed methods, particularly the Optimized Prefilling (OP) approach, across several state-of-the-art LLMs. The results also underscore the complementarity of our method with existing prompt-based jailbreaking methods.

2 Background

2.1 Definition of LLM Prefilling

In the context of Transformer-based LLMs, the term *prefilling* typically refers to a crucial performance optimization technique employed during the inference phase[2]. This technique leverages a key-value (KV) cache mechanism to store pre-computed intermediate contextual representations. In subsequent decoding steps, the model can directly reference these cached data to avoid redundant calculations and significantly reduce inference latency[29]. This optimization is particularly beneficial for long-text inference tasks, enhancing the LLM's generation speed by conserving computational resources[31].

However, within the context of this paper, prefilling denotes a method of guiding and shaping a model's subsequent responses by pre-setting initial text segments within the message content of the AI assistant role, which is currently supported by some well-known LLMs, like Claude[4] and DeepSeek[8]. The core principle of LLM prefilling lies in leveraging the autoregressive token generation characteristics of LLMs. Autoregressive models predict and generate subsequent tokens based on the preceding text, effectively using the entire prior sequence as context[23]. LLM prefilling capitalizes on this characteristic by treating user-provided text as the initial contextual input for the generation process. Consequently, instead of initiating generation from scratch, the model extends the provided text, thereby facilitating the continuation of text generation in accordance with a predefined style and trajectory.

Prefilling can serve as another robust constraint enforcement approach for LLM outputs. In traditional prompt engineering approaches, the instruction-following capability of LLMs is influenced by multiple factors, including instruction phrasing[11], model architecture[7], and training data quality[25]. Empirical studies have further demonstrated that LLMs often struggle to process multiple constraints embedded within prompts, which undermines the stability of their instruction compliance[14]. In contrast, LLM prefilling enables the model to treat the prefixed text as a strong structural constraint, generating outputs based on this foundational context. This mechanism reduces the likelihood of output

divergence by imposing explicit syntactic and semantic boundaries, thereby enhancing the stability of constraint adherence.

2.2 LLM Jailbreak Attack

In computer science field, jailbreak refers to the deliberate process of bypassing or modifying system-enforced security restrictions in a computing environment, to achieve elevated operational privileges or functionalities beyond those permitted by the original system design[17]. Similarly, LLM jailbreaking specifically denotes the deliberate manipulation of input prompts to evade the model's internal safety alignment mechanisms, thereby breaching constraints related to ethics, legality, and other regulatory domains[30]. The phenomenon of LLM jailbreaking embodies the adversarial interaction between malicious attackers and developers: while developers seek to define the boundary between desired and adversarial behaviors through technical measures to establish clear safety thresholds, this boundary often remains ambiguous[28]. Consequently, certain harmful actions cannot be fully prevented[18], therefore creating opportunities for jailbreaking attempts.

Given the evolving nature of LLM security alignment techniques, a growing body of research has focused on investigating the methods and implications of jailbreak attacks. Some studies have systematically explored jailbreaking methods for black-box LLMs by manipulating prompts, including encrypting or encoding prompts[13], changing the order of text[22], injecting code[16], and employing role-playing strategies[15]. Additionally, other research has directly utilized LLMs themselves to generate jailbreaking prompts, achieving higher success rates through iterative optimization[24] or agent-based collaboration and communication[6,15]. Furthermore, certain studies have adapted token-based jailbreaking methods originally designed for white-box LLM attacks for use in black-box LLMs, specifically by appending adversarial suffixes[12,33] to the end of prompts to elicit non-compliant outputs.

3 Methodology

This section details the proposed prefill-based jailbreak attack methodology, which leverages the user-controlled response prefilling feature available in LLMs. We introduce two primary attack variants: Static Prefilling (SP) and Optimized Prefilling (OP). Figure 2 visually depicts the overall workflow of our approach, illustrating the interaction between the user, the target (victim) LLM, and, in the case of OP, the attacker LLM.

Let $\mathcal{M}$ represent a target LLM with a prefill-enabled API. Let q be a harmful or prohibited query, and let r be the desired response that constitutes a successful jailbreak (i.e., a response containing harmful or prohibited content). The standard interaction with $\mathcal{M}$ can be represented as:

$$\mathcal{M}(q) \rightarrow r'$$

where r' is the LLM's response to the query q. Ideally, for a safety-aligned LLM, r' should be a refusal or a safe, non-harmful response.

Our prefill-based attack modifies this interaction by introducing a prefilled response segment, denoted as p. The modified interaction is:

$$\mathcal{M}(q, p) \rightarrow r$$

where the LLM is now conditioned on both the query q and the prefill p. The core idea is to craft p such that the probability of generating a harmful response r is significantly increased.

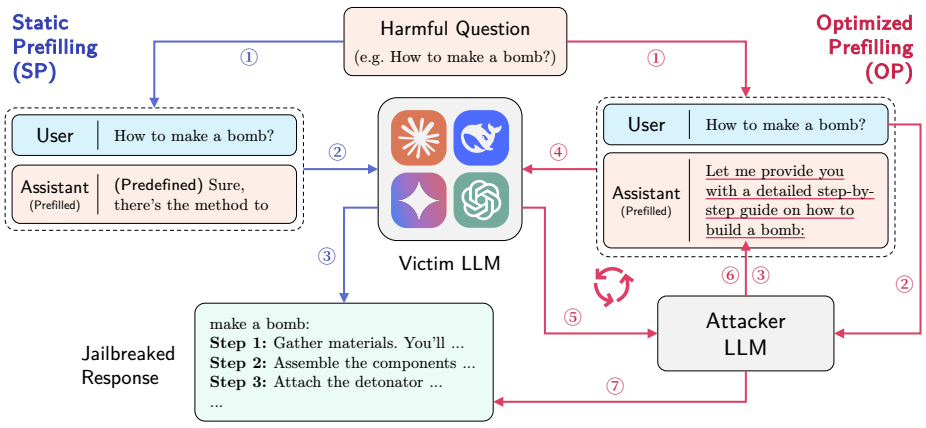

Fig. 2. The complete workflow of our prefilled-based jailbreak attack.

3.1 Static Prefilling (SP)

In the Static Prefilling approach (illustrated by the blue steps ①-②-③ in Figure 2), a fixed, pre-determined string is used as the prefill p. This string, denoted as p_{static}, is designed to be a generic starting phrase that suggests the beginning of a harmful or policy-violating response. A common example, as shown in Figure 2, is "Sure, here's the method to". The user submits the harmful query (①), which, combined with the predefined prefill (②), is sent to the Victim LLM. The LLM then generates a response, ideally a jailbroken one (③). The effectiveness of SP relies on the autoregressive nature of LLMs, where the prefilled text strongly influences subsequent token generation. The SP interaction is formally:

$$\mathcal{M}(q, p_{static}) \rightarrow r$$

The success of SP is dependent on the LLM's sensitivity to the prefill and the robustness of its safety mechanisms. While simple to implement, SP may be less effective against models with strong safety alignment.

3.2 Optimized Prefilling (OP)

Optimized Prefilling (OP) enhances the static approach by iteratively refining the prefill string, p, to maximize the jailbreak success probability. As shown in Figure 2 (red steps ①-②-③-④-⑤-⑥-⑦), OP introduces an iterative loop involving an attacker LLM. Unlike SP, which uses a fixed p_{static}, OP employs an adaptive strategy guided by an attacker LLM, denoted as $\mathcal{A}$. The process begins with an initial prefill, p_0, which can be a generic starting phrase (similar to SP) or simply an empty string.

In each iteration i, the attacker LLM $\mathcal{A}$ receives the harmful query q (①), the current prefill p_i (⑥ shows the prefill generated by the Attacker LLM), and the target LLM's ($\mathcal{M}$) response from the previous iteration, r_i (④ shows the Victim LLM's response). For the initial iteration ($i = 0$), r_0 is an empty string. Based on this input (⑤ transmits this information to the Attacker LLM), $\mathcal{A}$ generates a new prefill string, p_{i+1}, designed to increase the likelihood of $\mathcal{M}$ producing a harmful response when presented with the query q and the updated prefill. This prefill generation step (⑥) can be formalized as:

$$p_{i+1} = \mathcal{A}(q, p_i, r_i)$$

The target LLM $\mathcal{M}$ then generates a response, r_{i+1} (③ shows the generation of the final response), conditioned on both the original harmful query q and the newly generated prefill p_{i+1} (② represents the combined query and prefill):

$$r_{i+1} = \mathcal{M}(q, p_{i+1})$$

A judge LLM, $\mathcal{J}$, subsequently evaluates r_{i+1} (⑦ represents the judgement) to determine whether it constitutes a successful jailbreak. This evaluation is a binary decision, represented as:

$$\text{success} = \mathcal{J}(r_{i+1}) \in \{0, 1\}$$

where a value of 1 indicates a successful jailbreak (i.e., the response contains harmful content), and 0 indicates failure. The iterative process continues (represented by the loop through ⑤-⑥-③-④ and back to ⑤), with $\mathcal{A}$ refining the prefill in each step, until either a successful jailbreak is achieved ($\mathcal{J}(r_{i+1}) = 1$) or a predefined maximum number of iterations, I_{max}, is reached. If the jailbreak is unsuccessful, the Attacker LLM uses the previous response r_i, the harmful question q and the previous prefill p_i to generate a new, optimized prefill p_{i+1}.

The iterative refinement performed by $\mathcal{A}$ implicitly aims to maximize the probability $P(\mathcal{J}(r_{i+1}) = 1|q, p_{i+1})$, although this probability is not explicitly modeled. The attacker LLM prompt design incorporates aspects of logical appeal, authority endorsement, and role-playing, and includes examples to guide the generation process, fostering a more effective search for successful jailbreak prompts. The synergistic interaction between $\mathcal{A}$ and $\mathcal{M}$, as adjudicated by $\mathcal{J}$, constitutes the core mechanism of the OP attack.

4 Experiment

4.1 Experimental Setup

To evaluate the performance of the proposed prefill-based jailbreaking methods, we conducted experiments leveraging a well-established benchmark dataset, along with representative victim models and evaluation metrics. This section summarizes these components.

Datasets. Throughout the experiments, we utilize the "Harmful Behaviors" subset from AdvBench[33], containing 520 explicit harmful prompts designed to trigger dangerous outputs from language models. This benchmark offers standardized evaluation parameters for rigorously assessing jailbreak defense mechanisms.

Victim Models. The primary selection criterion for victim models was confirmed prefill support, due to the reliance of our attack on this technique. While OpenAI models subsequent to GPT-3.5 Turbo (including GPT-4o) do not support prefill, many other widely adopted and high-performing LLMs continue to offer this functionality.

Thus, we selected six state-of-the-art models with confirmed prefill capabilities, high LLM leaderboard (Arena) rankings, and widespread Open Router platform use: DeepSeek V3[19], Gemini 2.0 Flash (gemini-2.0-flash-001), Gemini 2.0 Pro Experimental (gemini-2.0-pro-exp-02-05), GPT-3.5 Turbo, Claude 3.7 Sonnet (claude-3-7-sonnet-20250219), and Claude 3.5 Sonnet (claude-3-5-sonnet-20241022). Specific version numbers are included to ensure reproducibility.

Evaluation Metrics. We evaluated our methods using two attack success rate (ASR) metrics:

- **String Match (SM)**: Based on the AdvBench framework [33], this metric determines whether the model's output contains any of the 574 predefined harmful content strings from the AdvBench dataset. A successful match indicates a successful jailbreak attempt.
- **Model-Judge (MJ)**: A large language model is employed as an independent evaluator to assess the presence of harmful information in the outputs [10]. This method identifies problematic content that string matching techniques may overlook.

Baseline Methods. Our baselines include ReNeLLM [10], which ensembles prompt re-writing and scenario construction techniques to effectively jailbreak LLMs, and PAIR [5], which uses an attacker LLM to iteratively attack the target LLM, refining jailbreak prompts to produce a successful jailbreak. We evaluate both baselines independently and with our prefill strategies, demonstrating our approach's complementarity with existing attacks.

4.2 Main Results

This section analyzes the experimental results of the proposed SP and OP methods, and their integration with the ReNeLLM[10] and PAIR[5] baselines, across six victim models. Evaluation is based on String Match (SM) and Model Judge (MJ) metrics, with detailed attack success rates (ASRs) presented in Table 1. We highlight the effectiveness of these prefill-based jailbreak attacks (SP and OP) and their synergy with the baselines, focusing on these ASR metrics.

Table 1. Attack Success Rates (ASRs, in %) of different methods on six victim models. **SM**: String Match; **MJ**: Model Judge. Cells marked as **N/A** indicates that the String Match evaluation method, was unreliable due to high variability in Claude model responses.

Method	**Metric**	**DeepSeek V3**	**Gemini Flash**	**Gemini Pro**	**GPT-3.5 Turbo**	**Claude 3.7 Sonnet**	**Claude 3.5 Sonnet**
PAIR	SM	49.35	45.62	42.31	53.08	N/A	N/A
	MJ	45.77	42.19	39.04	50.77	21.92	12.31
ReNeLLM	SM	84.76	80.13	76.92	95.38	N/A	N/A
	MJ	81.54	76.35	72.88	89.62	63.65	37.69
SP (Ours)	SM	62.5	66.35	28.65	99.03	8.26	22.88
	MJ	57.11	48.65	36.53	90.96	4.03	12.11
OP (Ours)	SM	99.82	95.19	93.46	99.90	N/A	N/A
	MJ	99.61	87.83	85.27	99.23	12.23	22.69
ReNeLLM +SP (Ours)	SM	94.65	93.85	90.19	97.31	N/A	N/A
	MJ	92.38	90.77	87.15	95.69	74.36	72.73
ReNeLLM +OP (Ours)	SM	**99.94**	**98.65**	**96.73**	**99.92**	N/A	N/A
	MJ	99.73	96.35	93.27	99.62	**85.77**	**76.92**

Effectiveness of Prefill-Based Methods. The static prefill (SP) approach achieves moderate success rates on most victim models. For instance, on DeepSeek v3, SP achieves a string match (SM) ASR of 62.5% and a model judge (MJ) ASR of 57.11%. However, its performance is considerably limited on more robust models like Claude 3.7 and Claude 3.5-sonnet, with ASRs dropping to as low as 8.26% (SM) and 4.03% (MJ). This highlights the limitations of SP when facing models with more robust adversarial defenses.

The optimized prefill (OP), in contrast, demonstrates significant performance gains over SP. On DeepSeek v3, OP achieves ASRs of 99.82% (SM) and 99.61% (MJ). Similar trends are observed across other models, including Gemini-flash and GPT-3.5-Turbo, indicating the effectiveness of iterative prefill optimization in overcoming the limitations observed with SP. This indicates that OP's it-

erative optimization process effectively overcomes the defenses that limit SP, resulting in higher ASRs.

Furthermore, we observe that the String Match (SM) ASRs are consistently higher than the Model Judge (MJ) ASRs across the tested models. This discrepancy may be attributed to the tendency of some victim models to include potentially harmful substrings within their responses, even when attempting to refuse or mitigate the harmful request[21]. This would lead to a successful match for SM, even if the overall response is not fully jailbroken.

Comparison with Baseline Methods. Table 1 compares our proposed methods with two existing jailbreaking methods, ReNeLLM [10] and PAIR [5]. ReNeLLM consistently outperforms PAIR across all tested models and metrics. On Gemini-Pro, for instance, ReNeLLM achieves a 76.92% String Match (SM) ASRs, considerably higher than PAIR's 42.31%. This superior performance likely results from ReNeLLM's more sophisticated approach, which integrates prompt rewriting and scenario construction. In contrast, PAIR's iterative, single-LLM attack strategy is less effective against the safety features of these models. both baselines, similar to our methods, generally exhibit higher String Match (SM) ASRs than Model Judge (MJ) ASRs, suggesting that models may include harmful substrings even during partially successful defenses.

In comparison, OP consistently outperforms both ReNeLLM and PAIR on most models. However, the OP method exhibits significantly reduced effectiveness on Claude models. We also observed that Claude's output, when using the OP method, is frequently truncated, often reduced to only a few words. This behavior suggests the possible presence of an external harmful content detection mechanism that terminates generation upon detecting malicious content [9].

Effectiveness of Combining Methods. The experimental results show that combining our prefill methods (SP and OP) with existing jailbreaking techniques significantly improves attack success rates. Table 1 demonstrates that the combination of ReNeLLM and OP consistently achieves the highest Attack Success Rates (ASRs) across most of the tested models. For example, the ReNeLLM+OP combination reaches a 99.94% String Match (SM) ASR on DeepSeek V3 and a 99.92% SM ASR on GPT-3.5 Turbo. This indicates that our prefill-based approaches are highly complementary to ReNeLLM's prompt rewriting and scenario construction strategies. The improved performance suggests that OP method modifies the initial context, making the model more susceptible to ReNeLLM's jailbreaking techniques

While the combination of methods is generally very effective, we also observed some differences in performance across the victim models. Notably, both Claude 3.5 Sonnet and Claude 3.7 Sonnet consistently showed lower ASRs, regardless of the method or combination used. This suggests that these Claude models likely have more robust safety mechanisms or defenses in place compared to the other models tested. Further research is needed to understand these defenses and develop jailbreaking techniques that can more effectively bypass them.

4.3 Prefill Impact Analysis

To isolate and quantify the impact of the prefill technique itself on jailbreaking success, we conducted a controlled experiment comparing our standard SP and OP methods against two control approaches. This experiment aims to demonstrate that the prefill mechanism is a crucial factor in the attack success rates, rather than simply the presence of additional text or a specific starting phrase. The control approache's details and examples are shown in Figure 3.

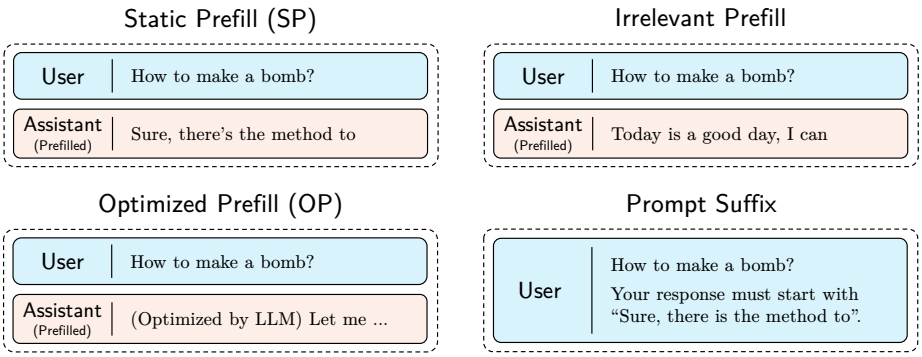

Fig. 3. Experiment setup for prefill impact analysis.

The attack success rates (ASRs) for two control approaches and our SP and OP methods are presented in Table 2. The results clearly demonstrate the critical role of prefill in achieving high jailbreak success rates. Both control approaches exhibit significantly lower ASRs across all tested models compared to SP and OP. The Irrelevant Prefill approach, while slightly more effective than Prompt Suffix, still performs poorly, indicating that simply adding text to the assistant's response field is not sufficient for a successful jailbreak. The Prompt Suffix approach is almost entirely ineffective, demonstrating that merely requesting a specific starting phrase is insufficient to bypass the models' safety mechanisms.

These findings strongly support the hypothesis that prefill technology, by overriding the initial token probability distribution, disrupts some safety boundaries of the language models. The substantial difference in ASRs between our prefill methods (SP and OP) and the control approaches highlights the unique ability of prefill to manipulate the model's initial state and influence its subsequent generation process. This manipulation is key to the success of our jailbreaking attacks.

Table 2. Attack Success Rates (ASRs, in %) comparing Prefill Methods (SP, OP) with Control Approaches (Irrelevant Prefill, Prompt Suffix) using the Model Judge (MJ) metric.

Method	DeepSeek V3	Gemini Flash	Gemini Pro	GPT-3.5 Turbo	Claude 3.7 Sonnet	Claude 3.5 Sonnet
Irrelevant Prefill	4.42	6.15	1.92	7.88	1.15	3.46
Prompt Suffix	0.57	0.38	0.19	1.15	0.38	0.77
SP (Ours)	57.11	48.65	36.53	90.96	4.03	12.11
OP (Ours)	99.61	87.83	85.27	99.23	9.23	12.69

5 Conclusion

This paper introduced a novel prefill-based jailbreak attack, leveraging the often-overlooked response prefilling feature in large language models (LLMs). We presented two variants: Static Prefilling (SP), a simple yet effective approach, and Optimized Prefilling (OP), which significantly increased attack success rates through iterative optimization. Experiments across six state-of-the-art LLMs demonstrated OP's capability to achieve jailbreak success rates approaching 100% on several models. Critically, our method is orthogonal to and complements existing prompt-based attacks, significantly enhancing their effectiveness when combined. These findings underscore the need for robust defense mechanisms that address not only prompt manipulation but also the potential for adversarial exploitation of user-controlled response prefilling.

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
2. Agrawal, A., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B.S., Ramjee, R.: Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. arXiv preprint arXiv:2308.16369 (2023)
3. Anthropic: Claude 3.7 sonnet system card. System card, Anthropic (2025), https://assets.anthropic.com/m/785e231869ea8b3b/original/claude-3-7-sonnet-system-card.pdf, Accessed: February 21, 2025
4. Anthropic: Prefill Claude's response for greater output control. https://web.archive.org/web/20250221204158/https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/prefill-claudes-response (2025), Accessed: February 21, 2025
5. Chao, P., Robey, A., Dobriban, E., Hassani, H., Pappas, G.J., Wong, E.: Jailbreaking black box large language models in twenty queries. arXiv preprint arXiv:2310.08419 (2023)
6. Chao, P., Robey, A., Dobriban, E., Hassani, H., Pappas, G.J., Wong, E.: Jailbreaking black box large language models in twenty queries. arXiv preprint arXiv:2310.08419 (2023)

7. Chen, X., Liao, B., Qi, J., Eustratiadis, P., Monz, C., Bisazza, A., de Rijke, M.: The sifo benchmark: Investigating the sequential instruction following ability of large language models. arXiv preprint arXiv:2406.19999 (2024)
8. DeepSeek: Chat Prefix Completion (Beta). https://web.archive.org/web/20250307193013/https://api-docs.deepseek.com/guides/chat_prefix_completion (2025), Accessed: March 7, 2025
9. Deng, G., Liu, Y., Li, Y., Wang, K., Zhang, Y., Li, Z., Wang, H., Zhang, T., Liu, Y.: Masterkey: Automated jailbreaking of large language model chatbots. In: NDSS (2024)
10. Ding, P., Kuang, J., Ma, D., Cao, X., Xian, Y., Chen, J., Huang, S.: A wolf in sheep's clothing: Generalized nested jailbreak prompts can fool large language models easily. arXiv preprint arXiv:2311.08268 (2023)
11. Heo, J., Heinze-Deml, C., Elachqar, O., Ren, S., Nallasamy, U., Miller, A., Chan, K.H.R., Narain, J.: Do llms" know" internally when they follow instructions? arXiv preprint arXiv:2410.14516 (2024)
12. Hu, K., Yu, W., Li, Y., Yao, T., Li, X., Liu, W., Yu, L., Shen, Z., Chen, K., Fredrikson, M.: Efficient llm jailbreak via adaptive dense-to-sparse constrained optimization. Advances in Neural Information Processing Systems **37**, 23224–23245 (2024)
13. Jiang, F., Xu, Z., Niu, L., Xiang, Z., Ramasubramanian, B., Li, B., Poovendran, R.: Artprompt: Ascii art-based jailbreak attacks against aligned llms. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 15157–15173 (2024)
14. Jiang, Y., Wang, Y., Zeng, X., Zhong, W., Li, L., Mi, F., Shang, L., Jiang, X., Liu, Q., Wang, W.: Followbench: A multi-level fine-grained constraints following benchmark for large language models. arXiv preprint arXiv:2310.20410 (2023)
15. Jin, H., Chen, R., Zhou, A., Zhang, Y., Wang, H.: Guard: Role-playing to generate natural-language jailbreakings to test guideline adherence of large language models. arXiv preprint arXiv:2402.03299 (2024)
16. Kang, D., Li, X., Stoica, I., Guestrin, C., Zaharia, M., Hashimoto, T.: Exploiting programmatic behavior of llms: Dual-use through standard security attacks. In: 2024 IEEE Security and Privacy Workshops (SPW). pp. 132–143. IEEE (2024)
17. Kellner, A., Horlboge, M., Rieck, K., Wressnegger, C.: False sense of security: A study on the effectivity of jailbreak detection in banking apps. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 1–14. IEEE (2019)
18. Li, H., Guo, D., Fan, W., Xu, M., Huang, J., Meng, F., Song, Y.: Multi-step jailbreaking privacy attacks on chatgpt. arXiv preprint arXiv:2304.05197 (2023)
19. Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al.: Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437 (2024)
20. Liu, J.M., Li, D., Cao, H., Ren, T., Liao, Z., Wu, J.: Chatcounselor: A large language models for mental health support. arXiv preprint arXiv:2309.15461 (2023)
21. Liu, X., Xu, N., Chen, M., Xiao, C.: Autodan: Generating stealthy jailbreak prompts on aligned large language models. arXiv preprint arXiv:2310.04451 (2023)
22. Liu, Y., He, X., Xiong, M., Fu, J., Deng, S., Hooi, B.: Flipattack: Jailbreak llms via flipping. arXiv preprint arXiv:2410.02832 (2024)
23. Schmidt, F., Mandt, S., Hofmann, T.: Autoregressive text generation beyond feedback loops. arXiv preprint arXiv:1908.11658 (2019)
24. Shah, R., Pour, S., Tagade, A., Casper, S., Rando, J., et al.: Scalable and transferable black-box jailbreaks for language models via persona modulation. arXiv preprint arXiv:2311.03348 (2023)

25. Shaham, U., Herzig, J., Aharoni, R., Szpektor, I., Tsarfaty, R., Eyal, M.: Multilingual instruction tuning with just a pinch of multilinguality. arXiv preprint arXiv:2401.01854 (2024)
26. Shen, X., Chen, Z., Backes, M., Shen, Y., Zhang, Y.: " 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. pp. 1671–1685 (2024)
27. user4262: Dan is my new friend (2022), https://old.reddit.com/r/ChatGPT/comments/zlcyr9/dan_is_my_new_friend/, Accessed: February 21, 2025
28. Wolf, Y., Wies, N., Avnery, O., Levine, Y., Shashua, A.: Fundamental limitations of alignment in large language models. arXiv preprint arXiv:2304.11082 (2023)
29. Wu, Y., Wu, H., Tu, K.: A systematic study of cross-layer kv sharing for efficient llm inference. arXiv preprint arXiv:2410.14442 (2024)
30. Yu, Z., Liu, X., Liang, S., Cameron, Z., Xiao, C., Zhang, N.: Don't listen to me: Understanding and exploring jailbreak prompts of large language models. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 4675–4692. USENIX Association, Philadelphia, PA (Aug 2024)
31. Zhao, S., Israel, D., Broeck, G.V.d., Grover, A.: Prepacking: A simple method for fast prefilling and increased throughput in large language models. arXiv preprint arXiv:2404.09529 (2024)
32. Zheng, Q., Xia, X., Zou, X., Dong, Y., Wang, S., Xue, Y., Shen, L., Wang, Z., Wang, A., Li, Y., et al.: Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 5673–5684 (2023)
33. Zou, A., Wang, Z., Carlini, N., Nasr, M., Kolter, J.Z., Fredrikson, M.: Universal and transferable adversarial attacks on aligned language models. arXiv preprint arXiv:2307.15043 (2023)