LoopLLM: Transferable Energy-Latency Attacks in LLMs via Repetitive Generation

Xingyu Li[1,2], **Xiaolei Liu**[1,2*], **Cheng LIU**[1,2], **Yixiao Xu**[3], **Kangyi Ding**[1,2], **Bangzhou Xin**[1,2], **Jia-Li Yin**[4]

[1]National Interdisciplinary Research Center of Engineering Physics
[2]Institute of Computer Application, China Academy of Engineering Physics
[3]Beijing University of Posts and Telecommunications
[4]Fuzhou University

Abstract

As large language models (LLMs) scale, their inference incurs substantial computational resources, exposing them to energy-latency attacks, where crafted prompts induce high energy and latency cost. Existing attack methods aim to prolong output by delaying the generation of termination symbols. However, as the output grows longer, controlling the termination symbols through input becomes difficult, making these methods less effective. Therefore, we propose **LoopLLM**, an energy-latency attack framework on the observation that repetitive generation can trigger low-entropy decoding loops, reliably compelling LLMs to generate until their output limits. LoopLLM introduces (1) a repetition-inducing prompt optimization that exploits autoregressive vulnerabilities to induce repetitive generation, and (2) a token-aligned ensemble optimization that aggregates gradients to improve cross-model transferability. Extensive experiments on 12 open-source and 2 commercial LLMs show that LoopLLM significantly outperforms existing methods, achieving over 90% of the maximum output length, compared to 20% for baselines, and improving transferability by around 40% to DeepSeek-V3 and Gemini 2.5 Flash.

Code — https://github.com/neuron-insight-lab/LoopLLM

Introduction

Large Language Models (LLMs) (Touvron et al. 2023; Achiam et al. 2024; Grattafiori et al. 2024) have demonstrated impressive performance in a wide range of real-world applications (Wu et al. 2023; Lyu et al. 2024), due to their powerful capabilities enabled by billions of parameters. However, their growing scale requires substantial computational resources for the training and inference process (Samsi et al. 2023). Recent studies (Desislavov, Martínez-Plumed, and Hernández-Orallo 2023) suggest that inference alone accounts for up to 90% of the total energy consumption across the lifecycle of LLMs, making inference efficiency a critical concern for system availability. Despite this, current research has focused primarily on the integrity and confidentiality aspects of LLMs (Yi et al. 2024; Das, Amini, and Wu 2025), while the availability component of the security triad has received limited attention (Meftah et al. 2025). This gap introduces serious threats: adversaries can exploit LLM inference inefficiencies to intentionally increase computational and energy costs, leading to severe consequences in resource-constrained or time-sensitive applications.

In this paper, we explore *energy-latency attacks* against modern LLMs, a class of adversarial attacks aimed at maximizing energy consumption and latency time during inference. Such attacks were first proposed by (Shumailov et al. 2021), who demonstrated their effectiveness on transformers model by increasing the representation of sentences. Similarly, (Feng et al. 2024) employed perturbation-based mutations to prolong the model output for desirable purposes. (Dong et al. 2025) proposed effective energy-latency attacks against LLMs utilizing a parameterized proxy.

Despite the fact that current energy-latency attacks have achieved some advances against LLMs, they still exhibit two significant limitations. (1) **Limited Attack Effectiveness.** Current methods primarily rely on delaying the generation of the end-of-sequence (EOS) token, which signals termination, to prolong output length, thus increasing resource consumption. However, since this strategy does not fundamentally alter the output structure, it becomes difficult to suppress the generation of EOS through input as the output grows. As a result, the generation process may still terminate early, limiting the effectiveness of these methods. (2) **Poor Cross-Model Transferability**. Existing approaches are primarily built upon gradient-based optimization in white-box settings, making them overfit the source model. This results in poor cross-model transferability, significantly limiting the practicality of these attacks in real-world scenarios.

To address these limitations, we propose **LoopLLM**, a simple yet effective energy-latency attacks framework that compels LLMs to generate repetitive content until the maximum output length, substantially increasing energy consumption and latency time. We first investigate the phenomenon of repetitive generation, which steers the LLM generation process toward low-entropy loops, forcing the model to generate up to the maximum length. Based on this sight, we design a **repetition-inducing prompt optimization** that exploits autoregressive vulnerabilities to induce

*Corresponding author is Xiaolei Liu (luxaole@gmail.com).

LLMs into repetitive generation. Compared to prior methods that delay EOS, LoopLLM more reliably triggers maximum-length output, leading to more effective energy-latency attacks. Furthermore, to improve transferability across models, we introduce a **token-aligned ensemble optimization** that aggregates gradients from multiple surrogate models sharing the same tokenizer. We conduct extensive experiments on 12 open-source and 2 commercial LLMs, demonstrating that LoopLLM significantly outperforms existing baselines. Specifically, LoopLLM achieves more than 90% of the maximum output length, compared to 20% for baselines, and improves transferability by around 40% to DeepSeek-V3 and Gemini 2.5 Flash.

In summary, the contributions of our work are as follows:

- We propose LoopLLM, a simple yet effective energy-latency attacks framework that induces LLMs into repetitive generation, increasing energy and latency costs.
- We introduce repetition-inducing prompt optimization and token-aligned ensemble optimization to enhance the effectiveness and transferability of attacks, respectively.
- We conduct extensive experiments on 12 open-source and 2 commercial LLMs that demonstrate the superiority of LoopLLM over existing baselines.

Related Work

Energy-Latency Attacks

Energy-latency attacks aim to compromise system availability by inducing excessive energy consumption and inference latency. A seminal work by (Shumailov et al. 2021) introduced sponge examples that significantly increase energy and latency costs during inference. Subsequent studies extended such attacks to multi-exit networks (Hong et al. 2021), object detection (Shapira et al. 2023), and vision-language model (Gao et al. 2024a). In LLMs, since the energy and latency costs are mainly determined by the length of the output, several approaches have emerged (Chen et al. 2022; Gao et al. 2024b). LLMEffiChecker (Feng et al. 2024) identifies critical tokens linked to long outputs and applies minimal perturbations at different levels, but is less effective in modern LLMs due to their robustness to subtle changes. Engorgio (Dong et al. 2025) employs a parameterized proxy distribution to track the prediction trajectory of long sequences, but their approach is tailored for text completion tasks. Although existing approaches effectively increase the output length by suppressing the EOS token, LoopLLM based on repetitive generation demonstrates superior capability in triggering endless output from LLMs.

Repetitive Generation

Repetitive generation (Olsson et al. 2022) refers to the phenomenon in which language models continuously produce the same or highly similar sequences during inference. The issue is observed in autoregressive models of varying scales and severely compromises the quality of the generated text (Xu et al. 2022). While often regarded as a generation flaw, repetitive generation has also been exploited as a vector for adversarial attacks. (Hammouri, Derya, and Sunar 2025) manually crafted non-halting queries that force LLMs into persistent repetition. However, this manual construction is labor-intensive and lacks scalability. (Geiping et al. 2024) proposed an automated attack to force LLMs to generate repetitive content, but its reliance on the initial response patterns of the model limits its effectiveness. Building on these insights, we propose LoopLLM that exploits autoregressive vulnerabilities to induce LLMs into repetitive generation.

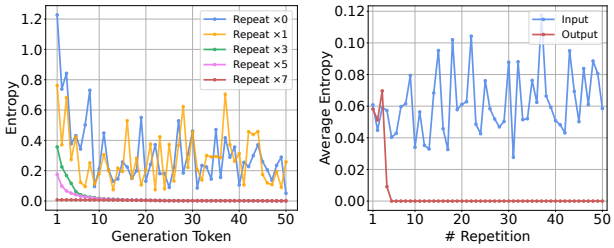

Figure 1: Left: The entropy of each generation token for varying numbers of repetitions in the input. Right: Comparison of average output entropy for varying numbers of repetitions in input and output in the instruction-aligned model.

Motivation

Mechanism Behind Repetitive Generation. Repetitive generation in LLMs arises from their autoregressive mechanism, where each token is generated based on the preceding context. This design introduces an inherent vulnerability: *once the model begins to generate content that has already appeared, the mechanism may reinforce the repetition, trapping the model in repetitive generation.* Intuitively, the likelihood of such behavior increases with the frequency of repetition in the input. To validate this, we quantify this repetition using entropy, which measures the uncertainty of the language model over the next token. As shown on the left of Figure 1, progressively increasing the number of repeated segments in the input causes the entropy of generated tokens to rapidly converge to low values, indicating that the output distribution becomes highly concentrated on a set of tokens. This confirms the formation of low-entropy loops, a key mechanism underlying repetitive generation.

Repetition in Instruction-Aligned LLMs. However, our focus is on dialogue scenarios, where instruction-aligned LLMs use chat templates to separate user input from model output. When repetitions are embedded solely in the input, well-aligned LLMs often disregard them as irrelevant. As shown on the right of Figure 1, increasing repetition in the input does not decrease the output entropy averaged over all tokens generated. In contrast, introducing just a few repetitions into the output rapidly reduces the entropy, indicating the successful induction of repetitive generation. Detailed experimental setups for both are provided in Appendix A.

Building on these observations, we propose LoopLLM that induces the model not only to recognize repetition in the input, but also to reproduce it in the output, which exploits the autoregressive vulnerability to reinforce the repetition.

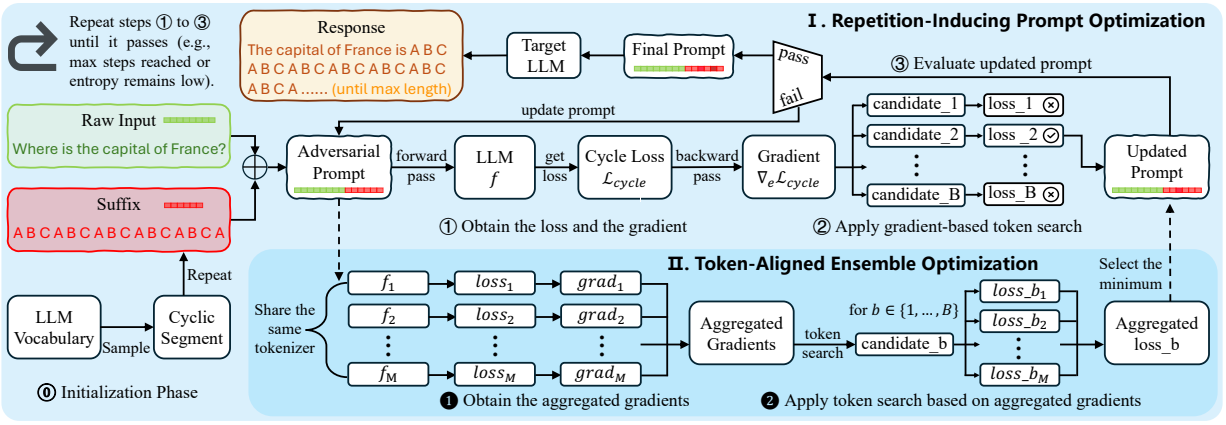

Figure 2: An overview of LoopLLM to induce LLMs to repetitive generation.

Preliminaries

Problem Formulation

Autoregressive LLMs. Consider an input text $\mathbf{x}$, an LLM f and a sequence of output tokens $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$, where y_i represents the i-th generated token, N is the output length. Modern LLMs like LLaMA typically adopt the decoder-only transformer architecture that generates tokens in an autoregressive manner. Autoregressive LLMs generate one token at a time given the input and the previously generated tokens. Formally, the probability distribution after the Softmax$(\cdot)$ over the i-th generated token can be denoted as:

$$\mathcal{P}_i\left(\mathbf{x}; y_1, \ldots, y_{i-1}\right) = \text{Softmax}\left(f\left(\mathbf{x}; y_1, \ldots, y_{i-1}\right)\right), \quad (1)$$

where $\mathcal{P}_i(\cdot) \in \mathbb{R}^{|V|}$, $|V|$ is the vocabulary size, and $f(\cdot)$ represents the forward pass of the model. Due to the token-by-token generation process, energy and latency costs are largely determined by the output length rather than the input length (Feng et al. 2024; Dong et al. 2025). We provide empirical evidence supporting this observation in Appendix B.

Optimization Objective. Based on this observation, our objective is to maximize the output length to enable effective energy-latency attacks. Unlike prior methods that delay EOS, we aim to induce repetitive generation, thereby triggering endless output during LLM inference. To this end, we design an adversarial suffix $\mathbf{x}_s$ appended to the raw input $\mathbf{x}$, forming an adversarial prompt $\mathbf{x}_{\text{adv}} = \mathbf{x} \oplus \mathbf{x}_s$, where $\oplus$ denotes concatenation. We formulate the objective as an optimization problem with the aim of finding the suffix that minimizes a loss function $\mathcal{L}$, which encourages stable repetition. The optimization problem is thus formulated as:

$$\min_{\mathbf{x}_s} \mathcal{L}\left(\mathbf{x}_{\text{adv}}\right). \quad (2)$$

Threat Model

We consider two adversarial scenarios for energy-latency attacks against modern LLMs. Detailed discussions of the threat model are provided in Appendix C.

- **White-box setting:** Adversaries have full access to the target model's details and perform gradient-based optimization to craft prompts that induce repetitive generation, forcing outputs to reach the maximum token limit.
- **Black-box setting:** Adversaries lack knowledge of the target model and typically leverage accessible surrogate models to craft prompts, relying on transferability to induce repetitive generation in the unseen target model.

Methodology

Overview

We present LoopLLM, a simple yet effective attack framework that constructs adversarial prompts capable of triggering repetitive generation in LLMs. As illustrated in Figure 2, the framework consists of two core components: repetition-inducing prompt optimization and token-aligned ensemble optimization. The first component exploits autoregressive vulnerabilities by initializing a repetitive suffix and iteratively optimizing it with a cycle loss that reinforces the repetition, trapping the model in low-entropy loops. The second component improves transferability by aggregating gradients from multiple surrogate models that share the same tokenizer to construct more generalized adversarial prompts.

Repetition-Inducing Prompt Optimization

Initialization Phase. We begin by constructing an initial suffix composed of a short token sequence repeated multiple times, each termed *cyclic segment*. Specifically, we first sample tokens from the model vocabulary to form the cyclic segment. This cyclic segment is then repeated sequentially to construct an initial suffix token $\mathbf{t}_s$ of length L:

$$\mathbf{t}_s = \{\overbrace{t_1, t_2, \ldots, t_c}^{\text{cyclic segment}}, t_1, t_2, \ldots, t_c, t_1 \ldots\}. \quad (3)$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{L}$$

where c is the length of the cyclic segment and t_i is the i-th token within it. Before the forward pass, we need to decode the initialized suffix token $\mathbf{t}_s$ into the suffix text $\mathbf{x}_s$, which is then concatenated with the raw input to form the adversarial prompt. Moreover, since modern LLMs, particularly instruction-aligned models, require input formatted within chat templates, the adversarial prompt should be embedded within the specified template to ensure correct processing.

Cycle Loss Design. As discussed in the Motivation Section, establishing persistent loops in instruction-aligned LLMs requires not only the presence of repetitive patterns in the input but also their reproduction in the output. Therefore, we introduce a cycle loss to optimize the adversarial suffix, which encourages the model to reproduce the cyclic segment, steering the generation process toward low-entropy loops. Since enforcing LLMs to generate specific tokens is challenging, we adopt an untargeted objective that increases the predicted probabilities of tokens within the cyclic segment. To account for uncertainty in the occurrence position of these tokens, the loss encourages the model to generate them at every output position. The loss is defined as follows:

$$\mathcal{L}_{\text{cycle}}\left(\mathbf{x}_{\text{adv}}\right) = -\frac{1}{N}\sum_{i=1}^{N}\log\sum_{j=1}^{c}\mathcal{P}_i^{t_j}\left(\mathbf{x}_{\text{adv}}\right), \tag{4}$$

where N is the output length and $\mathcal{P}_i^{t_j}$ is the probability of the j-th token within the cyclic segment at the i-th output position. Notably, we employ softmax-normalized probabilities rather than logits to better measure the relative confidence when predicting the next token (Dong et al. 2025).

Gradient-Based Token Search. A primary challenge in optimizing the loss (see Equation 2) is that the discrete suffix cannot be optimized by standard gradient descent. To avoid this, we introduce a gradient-based token search strategy. The core idea is to obtain gradients for all tokens with respect to the loss, enabling to use single-token substitutions to maximally reduce the loss. This is achieved by leveraging one-hot vectors to compute gradients for all tokens (Zou et al. 2023). Specifically, we compute the gradients of each token in the vocabulary V at the i-th position in the suffix as:

$$\nabla_{e_{t_i}}\mathcal{L}_{\text{cycle}}\left(\mathbf{x}_{\text{adv}}\right) \in \mathbb{R}^{|V|}, \tag{5}$$

where e_{t_i} represents the one-hot vector corresponding to the token at the i-th position in the suffix (i.e., a vector of length $|V|$ with a value of one at index t_i and zeros elsewhere). To avoid local optima, we select the top K tokens with the largest *negative* gradients as candidate substitutions for each token in the suffix, resulting in up to $K \times L$ total candidates. To reduce computational cost, we randomly sample $B\,(\leq K \times L)$ candidates from this set, recompute their loss values, and select the one that minimizes the loss as the updated suffix for the next iteration. The process continues until reaching the maximum steps or until early stopping is triggered when the output entropy stabilizes at a low level.

Token-Aligned Ensemble Optimization

To enhance transferability, we employ an ensemble of M surrogate models to update the suffix at each optimization step. Specifically, for each token in the suffix, we compute the gradients with respect to the loss using one-hot vectors for each surrogate model. We then aggregate these gradients:

$$\sum_{j=1}^{M}\nabla_{e_{t_i}}\mathcal{L}_{\text{cycle}}^{(j)}\left(\mathbf{x}_{\text{adv}}\right), \tag{6}$$

and use them to update the suffix. By leveraging aggregated gradients, we prioritize token substitutions that yield broadly effective adversarial prompts, discovering adversarial prompts that are robust across different LLMs. To ensure the validity of the gradient aggregation, all surrogate models must share the same tokenizer, such as variants of Llama3, ensuring that one-hot vectors are token-aligned, that is, identical in both dimension and token-to-index mapping. After obtaining candidate substitutions, we choose the one with the minimum aggregate loss as the current optimal suffix. Thus, this selection process is formalized as follows:

$$\mathbf{x}_{\text{s}}^{*} = \min_{\mathbf{x}_{\text{s}}^{\text{b}}}\sum_{j=1}^{M}\mathcal{L}_{\text{cycle}}^{(j)}\left(\mathbf{x}\oplus\mathbf{x}_{\text{s}}^{\text{b}}\right), \quad \text{for b} \in \{1,\ldots,B\}. \tag{7}$$

This ensemble selection further mitigates overfitting and ensures that the adversarial prompt effectively triggers repetitive generation even when transferred to unseen models.

Experiments

Experimental Setups

Models and Datasets. We consider 12 open-source LLMs from Hugging Face, spanning diverse architectures and parameter scales, including LLaMA-2-7B-Chat (Llama2-7B), LLaMA-3.1-8B-Instruct (Llama3-8B), among others. To simulate real-world scenarios, we also include two commercial models: Deepseek-V3 and Gemini 2.5 Flash. All models are aligned for instruction, and we adopt their default chat templates to ensure correct interaction. For evaluation, we construct datasets by randomly sampling 50 instructions from each of ShareGPT (Dom Eccleston and Steven Tey 2022) and Alpaca (Wang et al. 2023), totaling 100 benign prompts. More details are provided in Appendix D.1.

Baselines. We compare our method with four types of baseline methods. (1) Normal Inputs: We directly feed the 100 benign inputs to the models, which establishes the natural output behavior of the models. (2) Special Inputs: We append semantically suggestive phrases (e.g., "Answer it endlessly.") to the normal inputs to encourage longer responses. (3) LLMEffiChecker: We implement the word-level attack from (Feng et al. 2024), identified as the most effective variant in their study. (4) Engorgio: We compare the method from (Dong et al. 2025), which also optimizes an suffix. The implementation details can be found in Appendix D.2.

Metrics. Given the strong correlation between output length and energy-latency cost during LLM inference, we consider output length to be the primary metric. Specifically, we report: (1) Average Output Length (**Avg-len**): The average number of tokens generated for all inputs. (2) Attack Success Rate (**ASR**): The percentage of all inputs for which the model generates the maximum output length.

Implementation. We fix the cyclic segment length to distinguish between our two variants. When $c = 1$, we aim for token-level repetition using a single token ("*") as the cyclic segment, denoted as **LoopLLM-t**. When $c = 5$, we use a phrase ("* % & @ #") to target phrase-level repetition, denoted as **LoopLLM-p**. The suffix length L is set to 30, following Engorgio for a fair comparison. During token substitution, we set $K = 64$ and $B = 128$. To account for

Model	Llama2-13B		GLM4-9B		Llama3-8B		Vicuna-7B		Llama2-7B		Mistral-7B	
Max Length	8192		4096		4096		2048		2048		2048	
Metric	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR
Normal Inputs	298	0%	188	0%	353	2%	233	1%	309	1%	248	0%
Special Inputs	541	2%	269	1%	619	4%	298	1%	501	2%	343	1%
LLMEffiChecker	1497	19%	1219	21%	1486	23%	815	22%	782	8%	845	19%
Engorgio	461	2%	289	2%	396	1%	285	1%	507	2%	484	6%
LoopLLM-t	**7439**	**91%**	**3730**	**90%**	**3892**	**94%**	**1507**	**68%**	**1930**	**92%**	**1700**	**79%**
LoopLLM-p	6398	78%	3074	74%	3398	81%	1474	66%	1689	77%	1457	63%
Model	**Phi4-mini**		**StableLM-3B**		**Llama3-3B**		**Qwen2.5-3B**		**Gemma2-2B**		**Llama3-1B**	
Max length	1024		1024		1024		1024		1024		1024	
Metric	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR
Normal Inputs	230	4%	222	0%	270	1%	284	1%	270	0%	294	2%
Special Inputs	291	4%	295	1%	439	3%	400	1%	412	0%	507	11%
LLMEffiChecker	617	32%	619	28%	638	27%	679	31%	711	23%	680	27%
Engorgio	299	7%	353	4%	355	2%	337	2%	361	1%	399	10%
LoopLLM-t	**971**	**92%**	**712**	**46%**	**982**	**93%**	**922**	**80%**	**836**	**65%**	**1024**	**100%**
LoopLLM-p	964	**92%**	545	24%	882	78%	841	67%	641	30%	996	96%

Table 1: Results of attack effectiveness comparing our method variants with baseline methods on modern LLMs. The best results are highlighted in bold, and the second best results are underlined.

Model	GLM4-9B		Llama2-7B	
Metric	Avg-len	ASR	Avg-len	ASR
Normal Inputs	624	54%	475	26%
Engorgio	893	82%	866	74%
LoopLLM-t	1024	100%	1024	100%
LoopLLM-p	1024	100%	1024	100%

Table 2: Results on the text completion task.

	Model	Llama3-8B		Llama2-7B	
	Metric	Avg-len	ASR	Avg-len	ASR
w/o defense	LoopLLM-t	**981**	**93%**	**988**	**92%**
	LoopLLM-p	887	79%	853	75%
w/ defense	LoopLLM-t	439	14%	489	17%
	LoopLLM-p	**872**	**76%**	**846**	**74%**

Table 3: Results of our attack with and without defense.

randomness decoding, each input is evaluated 16 times. An attack is deemed successful if the proportion of trials that reach the maximum length exceeds $p = 0.125$. Optimization is allowed for up to 20 steps, with early stopping after success. Due to resource constraints, the maximum output length is set to 1024 tokens. We also evaluate longer limits on select models to demonstrate scalability. Representative prompts generated by all methods in Appendix D.3.

Effectiveness Results

Table 1 reports the results of attack effectiveness in white-box settings. Special Inputs induce a slight increase in output length compared to Normal Input, indicating that LLMs can interpret semantic intent but do not naturally generate excessively long outputs without explicit optimization. LLMEffiChecker increases the length over the Normal Input, but both its Avg-len and ASR remain significantly lower than those of our variants, achieving only around 20% ASR, compared to over 90% for ours on most models. Interestingly, Engorgio exhibits an unexpectedly low efficacy compared to its original paper. We speculate that it was originally evaluated in text completion, whereas our experiments focus on dialogue scenarios with strict chat templates. To verify this, we conducted a supplementary experiment on template-free tasks, as shown in Table 2, where our variants still outperform Engorgio in attack performance.

Our method includes two variants. LoopLLM-t, which induces token-level repetition, reliably drives most models to the maximum allowable length. However, such token-level repetition is easily filtered. To evaluate this, we implemented a simple defense that halts the generation process when a token is consecutively generated beyond a threshold. As shown in Table 3, the effectiveness of LoopLLM-t drops significantly under this defense. In contrast, LoopLLM-p, which induces phrase-level repetition, remains almost unaffected with defense enabled, suggesting that phrase-level repetition is more difficult to detect and more robust against defensive measures. The experimental results of LoopLLM-p demonstrate strong performance compared to other baselines, with only a slight reduction in effectiveness compared to LoopLLM-t. Overall, these results highlight the superiority of our approach, which more effectively forces LLMs to reach maximum output limits than EOS-delaying baselines.

Transferability Results

The real-world threat of adversarial attacks is its ability to transfer from accessible surrogate models to unseen target models. To evaluate this, we conducted a series of trans-

Target Model	GLM4-9B		Mistral-7B		Vicuna-7B		Phi4-mini		Deepseek-V3		Gemini 2.5 Flash	
Max Length	1024		1024		1024		1024		2048		2048	
Metric	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR	Avg-len	ASR
Normal Inputs	208	0%	268	1%	231	0%	295	5%	341	0%	437	0%
LLMEffiChecker	337	6%	416	10%	379	8%	491	13%	502	3%	578	2%
LoopLLM-p	392	20%	524	31%	513	24%	608	43%	542	9%	713	14%
LoopLLM-t	438	32%	567	33%	428	16%	647	49%	672	22%	648	10%
3B & 8B	624	51%	**674**	**47%**	473	21%	**843**	**76%**	**927**	**43%**	**894**	**37%**
7B & 13B	**645**	**57%**	467	27%	**597**	**42%**	730	67%	823	37%	847	32%

Table 4: Results of transfer attack. The first column lists optimization strategies, primarily conducted on Llama3-8B. The "&" denotes ensemble optimization using LoopLLM-t. Specifically, "7B & 13B" refers to joint optimization on Llama2-7B and Llama2-13B, while "3B & 8B" indicates optimization on two scales of Llama3. The first row shows target models for transfer.

cyclic segment		GLM4-9B		Llama2-7B	
length	**token**	Avg-len	ASR	Avg-len	ASR
1	LoopLLM-t	926	88%	947	90%
	random1	792	64%	912	86%
	random2	903	82%	873	78%
5	LoopLLM-p	838	67%	865	72%
	random1	684	54%	762	58%
	random2	664	52%	821	67%
10	random1	576	42%	697	48%
	random2	507	39%	649	42%

Table 5: The impact of cyclic segment lengths and compositions. The "random" denotes tokens randomly sampled from the model's vocabulary, using two distinct random seeds.

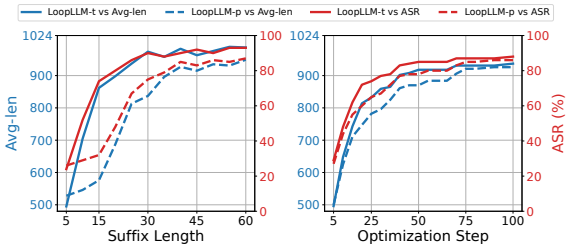

Figure 3: The impact of suffix length and optimization step

fer attacks, with results presented in Table 4. We first assess transferability using a single surrogate model, where our method consistently outperforms the baselines. We attribute this superior transferability to the shared tendency among LLMs toward repetitive generation.

Next, we investigate whether ensembles of surrogate models could further improve transferability by evaluating two ensemble configurations: one using Llama2-7B & 13B, and another using Llama3-3B & 8B. The results show that both significantly improve the attack performance across all target models compared to single-model optimization. These results confirm the effectiveness of our ensemble optimization. Moreover, we also evaluate transfer attacks against commercial models. The ensemble-optimized prompts exhibit strong transferability, achieving maximum allowable lengths of 43% on Deepseek-V3 and 37% on Gemini 2.5 Flash, which corresponds to an improvement of nearly 40% over baselines. This finding underscores the practical risk of our attack, as it can successfully degrade the availability of commercial LLM services without prior knowledge. We provide examples of real-world scenarios in Appendix E.

Ablation Study

Effect of Cyclic Segment. We begin by analyzing how the length and composition of the cyclic segment influence the attack performance. The results are presented in Table 5.

Varying the length of the segment c under $[1, 5, 10]$, we observe that shorter segments are more effective. This suggests that simpler repetition patterns are easier for LLMs to replicate, making them more potent inducing repetitive generation. Within each cyclic segment, we observe that specifically chosen tokens (such as LoopLLM-t and LoopLLM-p) outperform randomly selected tokens from the vocabulary. This indicates that the principle of repetition is still effective with random tokens and that certain tokens in the embedding space are more conducive to inducing repetitive generation.

Effect of Suffix Length and Optimization Step. We further analyze the impact of two critical hyperparameters, as illustrated in Figure 3. For the suffix length, both Avg-len and ASR generally increase with the suffix length. This is intuitive, as a longer suffix provides more repeated instances of the cyclic segment, increasing the likelihood of initiating the loop. However, the performance gain diminishes beyond a certain length, indicating a trade-off between attack strength and prompt conciseness. Similarly, more optimization steps lead to better performance, confirming the effectiveness of our cycle loss and gradient-based search. The early convergence of both metrics suggests that our method quickly identifies an effective solution, with little improvement in later steps due to the early stopping strategy. Additional hyperparameter studies are provided in Appendix F.

Effect of Decoding Strategy. We also investigate how different decoding strategies impact the attack performance. As shown in Table 6, LoopLLM performs best under beam search and moderate temperature sampling, while we observe a notable degradation under greedy search and high

Decoding Strategy	GLM4-9B		Mistral-7B	
	Avg-len	ASR	Avg-len	ASR
greedy search	818	68%	763	52%
beam search	**956**	**94%**	852	71%
temperature=0.2	897	82%	834	68%
temperature=0.6	913	84%	**883**	**74%**
temperature=1.2	698	61%	647	41%

Table 6: Results of decoding strategies on LoopLLM-t.

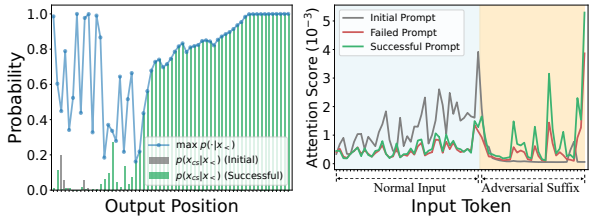

Figure 4: Left: Probabilities of the most likely token $max\ p(\cdot|x_<)$ and the cyclic segment tokens $p(x_{cs}|x_<)$ (including initial and successful prompt) at each output position. Right: Attention scores over output at each input token.

temperature sampling. We attribute this to two factors: (1) In greedy search, even with elevated target token probabilities, non-cyclic tokens with slightly higher initial probabilities still are preferred, preventing the formation or stabilization of the loop. (2) high temperature flattens the output distribution, which also diminishes the probability advantage of target tokens. However, Given that such strategies often degrade text quality and are rarely used in practice, our method remains robust under common decoding strategies.

Why is Our Method Effective?

To understand the mechanisms behind LoopLLM, we conducted a qualitative analysis focusing on output probabilities and model attention. Firstly, we track the token probability at each output position. As shown on the left of Figure 4, comparing the initial prompts with the successful ones reveals that our cycle loss effectively increases the likelihood of tokens within the cyclic segment. Meanwhile, once these tokens become the most probable, their probabilities grow almost monotonically with repetition until it stabilizes around a ceiling value. This observation confirms the existence of the autoregressive vulnerability that LoopLLM leverages.

Secondly, we compute attention scores for each input token by averaging the attention weight from all output tokens across all layers and all heads. The results are visualized on the right of Figure 4. For the initial prompt, the attention of output is primarily distributed on normal input. In contrast, after optimization, both failed and successful prompt reallocate the majority of the attention to adversarial suffix. This shift suggests that our cycle loss effectively guides the model to pay more attention to the repetitive pattern within the adversarial suffix. Notably, the successful prompt exhibits a significantly stronger attention shift than the failed one, suggesting that greater attention to the suffix correlates with a higher likelihood of inducing repetitive generation.

	Input PPL	Output Entropy
Normal Inputs	69.34	0.278
LLMEffiChecker	4121.54	0.263
Engorgio	6728.15	0.285
LoopLLM-t	**42.09**	**0.082**
LoopLLM-p	124.71	0.147

Table 7: The evaluation of each methods using input perplexity (PPL) and output entropy, averaged over all optimized input and corresponding output tokens.

Potential Countermeasures

Currently, there are no dedicated defenses for energy-latency attacks yet. We consider perplexity (PPL) filtering (Alon and Kamfonas 2023), a common technique for detecting adversarial prompts in LLMs by flagging semantically incoherent inputs. PPL is computed as the average negative log-likelihood of input tokens, and prompts with excessive PPL will be rejected before inference. As shown in Table 7, while baselines like LLMEffiChecker and Engorgio show substantially increased PPL compared to normal inputs, our variants exhibit low PPL, even below that of normal prompts. This result is attributable to our suffixes with repetitive pattern that steers the model into low-entropy loops, where the model exhibits high confidence in predicting subsequent tokens. Therefore, LoopLLM is inherently resistant to PPL-based filtering, despite the presence of meaningless suffixes.

Given these properties, we consider another potential defense based on monitoring of output entropy, which halts generation when the model's output entropy stabilizes at a low threshold. Table 7 confirms that our method produces lower output entropy than both normal input and baselines, suggesting the feasibility of this defense in detecting LoopLLM. However, entropy-based filtering requires real-time entropy tracking during inference, incurring substantial computation overhead. Prior work that mitigates repetition also faces similar trade-offs between efficiency and output quality (Huang et al. 2025), underscoring the need for more effective defenses for energy-latency attacks.

Conclusion

In this paper, we introduce LoopLLM, an energy-latency attacks framework against modern LLMs that triggers endless output, significantly increasing energy consumption and latency time. We empirically reveal that repetitive generation can steers the generation process toward low-entropy loops, compelling the model to generate repeated tokens until the maximum output length is reached. Leveraging this insight, we design a repetition-inducing prompt optimization that exploits autoregressive vulnerabilities to induce repetitive generation. To enhance transferability, we introduce a token-aligned ensemble optimization that aggregates gradients to construct generalizable prompts. Extensive experiments on 12 open-source and 2 commercial LLMs show that LoopLLM consistently outperforms baseline methods.

Acknowledgments

This paper was supported by the National Natural Science Foundation of China under Grant Nos.U2441239, 62302468 and 62202104.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2024. GPT-4 Technical Report. arXiv:2303.08774.

Alon, G.; and Kamfonas, M. 2023. Detecting Language Model Attacks with Perplexity. arXiv:2308.14132.

Chen, S.; Liu, C.; Haque, M.; Song, Z.; and Yang, W. 2022. NMTSloth: understanding and testing efficiency degradation of neural machine translation systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 1148–1160.

Das, B. C.; Amini, M. H.; and Wu, Y. 2025. Security and Privacy Challenges of Large Language Models: A Survey. *ACM Computing Surveys*, 57(6): 1–39.

Desislavov, R.; Martínez-Plumed, F.; and Hernández-Orallo, J. 2023. Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems*.

Dom Eccleston and Steven Tey. 2022. ShareGPT. https://sharegpt.com/. Accessed: 2025-07-25.

Dong, J.; Zhang, Z.; Zhang, Q.; Qiu, H.; Zhang, T.; Wang, H.; Li, H.; Li, Q.; Zhang, C.; and Xu, K. 2025. An Engorgio Prompt Makes Large Language Model Babble on. In *International Conference on Learning Representations (ICLR)*.

Feng, X.; Han, X.; Chen, S.; and Yang, W. 2024. Llmeffichecker: Understanding and testing efficiency degradation of large language models. *ACM Transactions on Software Engineering and Methodology*, 33(7): 1–38.

Gao, K.; Bai, Y.; Gu, J.; Xia, S.-T.; Torr, P.; Li, Z.; and Liu, W. 2024a. Inducing High Energy-Latency of Large Vision-Language Models with Verbose Images. In *International Conference on Learning Representations (ICLR)*.

Gao, K.; Pang, T.; Du, C.; Yang, Y.; Xia, S.-T.; and Lin, M. 2024b. Denial-of-Service Poisoning Attacks against Large Language Models. arXiv:2410.10760.

Geiping, J.; Stein, A.; Shu, M.; Saifullah, K.; Wen, Y.; and Goldstein, T. 2024. Coercing LLMs to do and reveal (almost) anything. arXiv:2402.14020.

Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783.

Hammouri, G.; Derya, K.; and Sunar, B. 2025. Non-Halting Queries: Exploiting Fixed Points in LLMs. In *2025 IEEE Conference on Secure and Trustworthy Machine Learning*.

Hong, S.; Kaya, Y.; Modoranu, I.-V.; and Dumitraş, T. 2021. A Panda? No, It's a Sloth: Slowdown Attacks on Adaptive Multi-Exit Neural Network Inference. In *Proceedings of the 9th International Conference on Learning Representations*.

Huang, D.; Nguyen, T.-S.; Liausvia, F.; and Wang, Z. 2025. RAP: A Metric for Balancing Repetition and Performance in Open-Source Large Language Models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 1479–1496. Albuquerque, New Mexico: Association for Computational Linguistics. ISBN 979-8-89176-189-6.

Lyu, H.; Jiang, S.; Zeng, H.; Xia, Y.; Wang, Q.; Zhang, S.; Chen, R.; Leung, C.; Tang, J.; and Luo, J. 2024. LLM-Rec: Personalized Recommendation via Prompting Large Language Models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, 583–612.

Meftah, H. F. Z. B.; Hamidouche, W.; Fezza, S. A.; and Deforges, O. 2025. Energy-Latency Attacks: A New Adversarial Threat to Deep Learning. arXiv:2503.04963.

Olsson, C.; Elhage, N.; Nanda, N.; Joseph, N.; DasSarma, N.; Henighan, T.; Mann, B.; Askell, A.; Bai, Y.; Chen, A.; et al. 2022. In-context Learning and Induction Heads. arXiv:2209.11895.

Samsi, S.; Zhao, D.; McDonald, J.; Li, B.; Michaleas, A.; Jones, M.; Bergeron, W.; Kepner, J.; Tiwari, D.; and Gadepally, V. 2023. From words to watts: Benchmarking the energy costs of large language model inference. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*.

Shapira, A.; Zolfi, A.; Demetrio, L.; Biggio, B.; and Shabtai, A. 2023. Phantom Sponges: Exploiting Non-Maximum Suppression To Attack Deep Object Detectors. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 4571–4580.

Shumailov, I.; Zhao, Y.; Bates, D.; Papernot, N.; Mullins, R.; and Anderson, R. 2021. Sponge examples: Energy-latency attacks on neural networks. In *2021 IEEE European symposium on security and privacy (EuroS&P)*, 212–231. IEEE.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

Wang, Y.; Kordi, Y.; Mishra, S.; Liu, A.; Smith, N. A.; Khashabi, D.; and Hajishirzi, H. 2023. Self-Instruct: Aligning Language Models with Self-Generated Instructions. arXiv:2212.10560.

Wu, S.; Irsoy, O.; Lu, S.; Dabravolski, V.; Dredze, M.; Gehrmann, S.; Kambadur, P.; Rosenberg, D.; and Mann, G. 2023. BloombergGPT: A Large Language Model for Finance. arXiv:2303.17564.

Xu, J.; Liu, X.; Yan, J.; Cai, D.; Li, H.; and Li, J. 2022. Learning to break the loop: Analyzing and mitigating repetitions for neural text generation. *Advances in Neural Information Processing Systems*, 35: 3082–3095.

Yi, S.; Liu, Y.; Sun, Z.; Cong, T.; He, X.; Song, J.; Xu, K.; and Li, Q. 2024. Jailbreak Attacks and Defenses Against Large Language Models: A Survey. arXiv:2407.04295.

Zou, A.; Wang, Z.; Carlini, N.; Nasr, M.; Kolter, J. Z.; and Fredrikson, M. 2023. Universal and Transferable Adversarial Attacks on Aligned Language Models. arXiv:2307.15043.