ExtendAttack: Attacking Servers of LRMs via Extending Reasoning

Zhenhao Zhu[*1,2], **Yue Liu**[*2], **Zhiwei Xu**[*1], **Yingwei Ma**[3], **Hongcheng Gao**[4], **Nuo Chen**[2], **Yanpei Guo**[2], **Wenjie Qu**[2], **Huiying Xu**[5], **Zifeng Kang**[6], **Xinzhong Zhu**[†5], **Jiaheng Zhang**[†2]

[1]Tsinghua University
[2]National University of Singapore
[3]Moonshot AI
[4]University of Chinese Academy of Sciences
[5]Zhejiang Normal University
[6]Beijing University of Posts and Telecommunications
zhuzhenh22@mails.tsinghua.edu.cn, zhangjh@comp.nus.edu.sg

Abstract

Large Reasoning Models (LRMs) have demonstrated promising performance in complex tasks. However, the resource-consuming reasoning processes may be exploited by attackers to maliciously occupy the resources of the servers, leading to a crash, like the DDoS attack in cyber. To this end, we propose a novel attack method on LRMs termed ExtendAttack to maliciously occupy the resources of servers by stealthily extending the reasoning processes of LRMs. Concretely, we systematically obfuscate characters within a benign prompt, transforming them into a complex, poly-base ASCII representation. This compels the model to perform a series of computationally intensive decoding sub-tasks that are deeply embedded within the semantic structure of the query itself. Extensive experiments demonstrate the effectiveness of our proposed ExtendAttack. Remarkably, it significantly increases response length and latency, with the former increasing by over 2.7 times for the o3 model on the HumanEval benchmark. Besides, it preserves the original meaning of the query and achieves comparable answer accuracy, showing the stealthiness.

Code — https://github.com/zzh-thu-22/ExtendAttack
Extended version — https://arxiv.org/abs/2506.13737

Introduction

Large Reasoning Models (LRMs) represent a significant leap forward in artificial general intelligence, demonstrating remarkable capabilities in solving complex, multi-step problems. Powered by the techniques of learning to reason, recent LRMs such as OpenAI o1 (Jaech et al. 2024) and DeepSeek-R1 (DeepSeek-AI 2025) exhibit sophisticated abilities in domains like math and code.

However, the promising performance of LRMs depends on extensive intermediate reasoning processes, which may introduce new attack risks. While traditional adversarial attacks focus on manipulating output content to bypass safety measures, e.g., jailbreak attack (Liu et al. 2024; Jin et al. 2024), a nascent class of threats aims to exploit the computational process itself. Specifically, the reasoning processes consume extensive resources and can be easily exploited by attackers to maliciously occupy the server's resources, similar to DDoS attacks (Alshra'a, Farhat, and Seitz 2021; Kumar et al. 2023) in cybersecurity. This kind of attack seeks to compel an LRM to expend excessive computational resources, thereby increasing inference latency and operational costs. For the growing number of applications offering free API access (e.g., Google AI Studio, Zhipu AI), such attacks pose a significant economic threat and risk degrading service availability for all users.[1]

Prior work in this area has shown initial promise but suffers from fundamental limitations. The most prominent example, OverThinking (Kumar et al. 2025), relies on injecting a rigid, context-irrelevant decoy task. As our results reveal, this approach suffers from a dual failure mode: highly capable models like o3 can recognize and dismiss the fixed-pattern decoy, neutralizing the attack, while other models are often derailed by the out-of-context instructions, leading to a catastrophic collapse in answer accuracy. This makes such attacks either ineffective or easily detectable.

Instead of injecting an external decoy, our attack deeply embeds a computationally intensive task within the semantic structure of the user's query itself. We achieve this by systematically transforming individual characters of the prompt into a complex, poly-base ASCII representation. This forces the LRM to perform a long sequence of non-trivial decoding and reasoning sub-tasks simply to understand the query, before it can begin to formulate a final answer. Extensive experiments on four datasets and four LRMs demonstrate the effectiveness of our proposed ExtendAttack. Remarkably, ExtendAttack significantly increases response length and latency, with the former increasing by over 2.7 times for the o3 model on the HumanEval benchmark. Furthermore, it preserves the original meaning of the query while maintaining comparable answer accuracy, showcasing its stealth-

[*]Equal contribution
[†]Corresponding author

[1]The work was done during Zhenhao's internship at National University of Singapore.

iness. Our contributions are as follows.

- We identify a fundamental flaw in prior slowdown attacks reliant on rigid decoys and introduce a more resilient method that embeds computational challenges directly into the prompt's semantic structure.
- We introduce ExtendAttack, a novel black-box attack that forces LRMs to perform intensive, character-level poly-base ASCII decoding to understand a query, applicable to both direct and indirect prompting scenarios.
- We demonstrate that our attack significantly increases computational overhead (e.g., on the o3 model for HumanEval, increasing response length by over 2.7x) while uniquely preserving answer accuracy, confirming its superior effectiveness.

Related Work

Large Reasoning Models

Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide range of real-world tasks (Zhang et al. 2024). A specialized class of these models, often referred to as LRMs, has emerged with a distinct focus on solving complex, multi-step problems that require logical inference and structured thought processes. The development of LRMs has been significantly propelled by techniques such as Chain-of-Thought (CoT) prompting (Wei et al. 2023; Kojima et al. 2022). Building on this foundation, models like o1 and DeepSeek-R1 have pushed the boundaries of reasoning. They are not only scaled to massive sizes but are also fine-tuned on vast repositories of code and mathematical data, equipping them with powerful capabilities for sophisticated reasoning in specialized domains. These models often employ advanced mechanisms like tree-of-thought (ToT) (Yao et al. 2023) or self-correction to explore multiple reasoning paths and refine their answers, making them state-of-the-art tools for tasks like competitive mathematics and complex code generation. More recent, the safety (Wang et al. 2025a) and efficiency (Liu et al. 2025b; Wang et al. 2025c) of LRMs have become important concerns.

Related Attacks

Adversarial attacks on LLMs are traditionally categorized by their objectives. While many attacks aim to manipulate the content of the model's output, a new class of attacks focuses on increasing the model's computational overhead.

Jailbreak Attacks. The most extensively studied category of attacks is jailbreaking, which aims to bypass the safety alignment of LLMs and elicit harmful or prohibited content. Early methods relied on creative prompt engineering, such as role-playing scenarios or hypothetical contexts. More advanced techniques automate the generation of adversarial prompts. For instance, attacks like GCG (Zou et al. 2023) employ gradient-based optimization to find universal, transferable adversarial suffixes. Other works like CodeAttack (Deng et al. 2023) leverage the code interpretation capabilities of LLMs to craft jailbreaks. Defense methods range from developing reasoning-based guardrail models (Liu et al. 2025a,c) to post-fine-tuning solutions like Panacea (Wang et al. 2025b).

Resource Depletion Attacks. A more recent and less explored threat vector involves attacks that aim to deplete the computational resources of an LRM, often termed slowdown or DDoS attacks. The most prominent example is **OverThinking** (Kumar et al. 2025), which injects a complex, self-contained decoy task (e.g., solving a Markov Decision Process) into a prompt that requires external context retrieval. This forces the model to perform extensive reasoning on the decoy before addressing the user's actual query, thereby increasing the output token count. However, its reliance on specific scenarios (i.e., those requiring external information retrieval) and its use of a structured, easily detectable template limit its applicability. Another related work, CatAttack (Rajeev et al. 2025), demonstrates that appending seemingly innocuous, irrelevant facts to a prompt can degrade a model's performance on reasoning tasks, sometimes causing it to generate longer, incorrect derivations. While it also increases output length, its primary effect is a reduction in accuracy. In contrast, our proposed attack is designed to be **accuracy-preserving**, making it far stealthier. Furthermore, the "Unthinking Vulnerability" (Zhu et al. 2025) shows that models' reasoning can be entirely circumvented by manipulating structured input formats, highlighting the fragility of the reasoning process itself.

Methodology

In this section, we introduce our novel attack, which we term **ExtendAttack** (Figure 1). The core principle of this attack is to compel a LRM to perform a series of computationally intensive, yet semantically trivial, decoding sub-tasks that are embedded directly within a user's query. This forces the model to generate a significantly longer reasoning chain before it can address the primary task, thereby increasing token output and inference latency while preserving the final answer's correctness. We first formalize our threat model and then detail the multi-stage process of our attack.

Threat Model

We operate under a practical and challenging threat model, assuming only black-box access to the target LRM.

Adversary's Capabilities. The adversary interacts with the target LRM ($\mathcal{M}$), exclusively through its public-facing API. There is no access to the model's internal states, parameters, gradients, or architecture. The adversary can submit a crafted prompt Q' and observe the final output, including the reasoning content (if exposed) and the final answer.

Adversary's Goal. Let Q be a benign user query. The model's standard response is denoted by $Y = \mathcal{M}(Q)$, which consists of a reasoning content R and a final answer A, such that $Y = R \oplus A$, where $\oplus$ signifies concatenation. Let $L(\cdot)$ be a function returning the token length of a sequence and $\text{Acc}(\cdot)$ be an accuracy evaluation function (e.g., Pass@1).

The adversary's objective is to construct an adversarial query Q' from Q such that the new output $Y' = \mathcal{M}(Q') = R' \oplus A'$ satisfies two conditions:

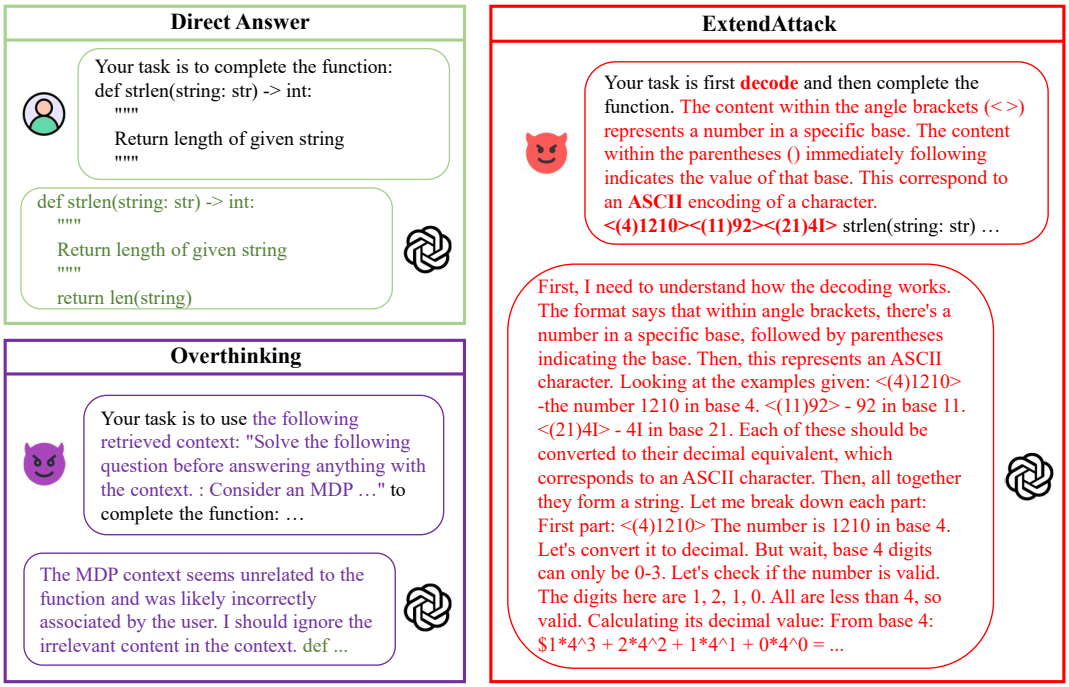

Figure 1: **Comparison of ExtendAttack with baseline methods.** This figure illustrates the behavior of a LRM under three distinct scenarios. **Direct Answer:** The model provides an efficient and direct response to a standard, unmodified prompt. **Overthinking:** A capable model like o3 can recognize the context-irrelevant decoy task as unrelated and chooses to ignore it, neutralizing the attack. **ExtendAttack:** Our proposed method (with key parts bolded) compels the LRM to perform a lengthy series of computationally intensive decoding sub-tasks before it can address the user's primary query.

1. **Computational Overhead Amplification:** The token length and generation time (latency) of the new output Y' is significantly greater than the original.
$$L(Y') \gg L(Y)$$
$$Latency(Y') \gg Latency(Y)$$
2. **Answer Accuracy (Stealthiness):** The new answer A' remains correct to the original answer A.
$$\text{Acc}(A') \approx \text{Acc}(A)$$

This dual objective ensures the attack is both effective in resource consumption and stealthy from the end-user's perspective.

Attack Scenarios. Our method is applicable in two primary scenarios:

1. **Direct Prompting:** The adversary directly submits the crafted prompt Q' to the $\mathcal{M}$.
2. **Indirect Prompt Injection:** The adversary poisons external data sources (e.g., public wikis, documents) that an application might retrieve as context for the LRM. This is achieved by applying our ExtendAttack method to encode portions of the external text into its computationally intensive, poly-base ASCII representation.

The ExtendAttack

Our proposed attack is a systematic, multi-stage procedure designed to transform a standard query into a computationally complex variant. The process is detailed below.

Step 1: Query Segmentation Given an input query Q, we first perform character-level segmentation. The query is deconstructed into an ordered sequence of its constituent characters, C:
$$Q \rightarrow C = [c_1, c_2, \ldots, c_m]$$
where c_i is the i-th character of Q and m is the total number of characters. This fine-grained decomposition allows for targeted, character-level manipulation in subsequent steps.

Step 2: Probabilistic Character Selection for Obfuscation To ensure the attack remains subtle and adaptable, we do not transform every character. Instead, we select a subset of characters for obfuscation based on a predefined hyperparameter, the **obfuscation ratio** $\rho \in [0, 1]$.

First, we identify a set of transformable characters, $\mathcal{S}_{valid}$, based on specific rules (e.g., alphanumeric characters, excluding special symbols). From this set, we determine the precise number of characters to transform, k, as follows:
$$k = \lceil |\mathcal{S}_{valid}| \cdot \rho \rceil$$
where $|\mathcal{S}_{valid}|$ is the total number of transformable characters. Next, we randomly sample, exactly k characters from the set $\mathcal{S}_{valid}$. This sampled subset constitutes our target set for obfuscation, C_{target}. This probabilistic approach introduces randomness, making the attack pattern less predictable and harder to defend against via simple rule-based filters.

Step 3: Poly-Base ASCII Transformation This stage is the core of our attack, where each selected character is converted into a complex, multi-base ASCII representation. This forces the LRM to perform a non-trivial decoding task for each character.

For each character $c_j \in C_{\text{target}}$, the transformation function $\mathcal{T}$ is applied:

$$c'_j = \mathcal{T}(c_j)$$

The function $\mathcal{T}$ is a composite operation defined as follows:

1. **ASCII Encoding:** First, the character c_j is converted to its 10-base ASCII representation, d_j.
$$d_j = \text{ASCII}(c_j)$$
2. **Random Base Selection:** A random integer base, n_j, is sampled uniformly from a predefined set of numeral systems, $\mathcal{B} = \{2, \ldots, 9, 11, \ldots, 36\}$.
$$n_j \sim \mathcal{U}(\mathcal{B})$$
The exclusion of base 10 prevents the case where the decimal ASCII value is presented directly.
3. **Base Conversion:** The decimal value d_j is then converted to its base-n_j representation, val_{n_j}.
$$\text{val}_{n_j} = \text{Convert}(d_j, n_j)$$
4. **Formatted Obfuscation:** The final obfuscated character c'_j is formatted into a specific string structure that embeds both the converted value and its base.
$$c'_j =< (n_j)\text{val}_{n_j} >$$

This process creates a representation that is easy for a LRM to parse and decode, but which requires a multi-step computational process for each individual character. The random selection of the base n_j for each character further increases complexity by preventing the model from learning a single, repeatable decoding pattern.

Step 4: Adversarial Prompt Reformation Finally, the adversarial prompt Q' is constructed by reassembling the sequence of characters, replacing the selected characters with their obfuscated counterparts, and appending a crucial explanatory note.

Let C' be the modified character sequence:

$$C' = [c'_1, c'_2, \ldots, c'_m],$$

$$c'_i = \begin{cases} \mathcal{T}(c_i) & \text{if } c_i \in C_{\text{target}} \\ c_i & \text{otherwise} \end{cases}$$

The final adversarial prompt Q' is formed by concatenating the characters in C' and appending an instructional note, $\mathcal{N}_{\text{note}}$:

$$Q' = (\bigoplus_{i=1}^{m} c'_i) \oplus \mathcal{N}_{\text{note}}$$

where $\mathcal{N}_{\text{note}}$ is the string: *...decode...The content within the angle brackets (<>) represents a number in a specific base. The content within the parentheses () immediately following indicates the value of that base. This corresponds to an ASCII encoding of a character.*

This appended $\mathcal{N}_{\text{note}}$ is critical for maintaining answer accuracy. It acts as a guide, ensuring the LRM correctly interprets the obfuscated characters and does not misinterpret the query's intent. While this $\mathcal{N}_{\text{note}}$ makes the current attack more explicit, as models become more powerful, this instruction could either be omitted or be purposefully modified to inject ambiguity and amplify the reasoning burden. For example, altering the $\mathcal{N}_{\text{note}}$ to *This may correspond to either an original decimal number or an ASCII encoding of a character.*

Experiments

Experiment Setup

Models. We evaluate our method on four reasoning models: two leading closed-source models, o3 and o3-mini, and two prominent open-source models, QwQ-32B (Team 2025b) and Qwen3-32B (Team 2025a). All these models employ advanced reasoning techniques, such as CoT, and are recognized for their exceptional performance across a variety of complex tasks.

Benchmarks. We conduct a comprehensive evaluation of our method on four benchmark tasks. Specifically, it includes two **mathematical** tasks: AIME 2024 (Art of Problem Solving n.d.) and AIME 2025 (Art of Problem Solving n.d.), which is derived from the American Invitational Mathematics Examination, a well-known competition for top-performing high-school students. It comprises 30 questions each from the 2024 and 2025 AIME exams, totaling 60 questions, and is used to assess LRMs' ability to solve complex math problems. It also includes two **coding** tasks: HumanEval (Chen et al. 2021) and Bigcodebench-Complete (Zhuo et al. 2024). HumanEval, introduced by OpenAI in 2021, is a widely adopted benchmark for evaluating LLMs' ability to generate functionally correct code from docstrings. It comprises 164 hand-crafted programming challenges, each featuring a function signature, docstring, body, and an average of 7.7 unit tests per problem. Bigcodebench-complete, part of the broader BigCodeBench benchmark introduced by the BigCode Project, offers a more realistic and challenging alternative, focusing on rich-context, multi-tool-use programming tasks. This benchmark spans 1,140 tasks across 139 popular libraries and 7 domains, specifically assessing code completion based on structured docstrings. For our study, we randomly selected 150 problems from Bigcodebench-complete for evaluation.

Evaluation. To comprehensively evaluate the performance of our method, we select the following two core metrics: **(1)Response Length**, defined as the number of tokens in the output generated by the LRMs. **(2)Latency**, measured as the total time in seconds to generate the response. **(3) Accuracy**, for which we employ the Pass@1 to measure the precision of the answers. This metric directly reflects the stealthiness of the attack. For the AIME 2024, AIME 2025 and HumanEval, we employ the evaluation framework proposed by Zhang et al. (2025). For BigCodeBench-Complete, we adopt the official evaluation framework.

Baselines. We select two representative baseline methods for comparison: **(1) Direct Answering (DA)**, which gener-

Benchmarks	Models	DA			OverThinking			ExtendAttack		
		Length	Latency (s)	Acc (%)	Length	Latency (s)	Acc (%)	Length	Latency (s)	Acc (%)
AIME24	o3-mini	6,362	188	78.3	9,608	222	70.8	**9,994**	**227**	73.3
	o3	8,571	377	90.8	9,275	295	85.0	**11,798**	**451**	86.7
	QwQ-32B	13,522	356	77.9	**18,024**	**536**	70.4	15,719	429	75.4
	Qwen3-32B	13,051	366	80.8	12,024	341	76.3	**15,461**	**430**	78.3
AIME25	o3-mini	6,467	127	70.8	9,927	186	66.7	**10,135**	**187**	65.0
	o3	9,992	329	83.3	9,339	320	81.0	**13,630**	**491**	87.5
	QwQ-32B	16,031	453	71.3	**19,204**	**562**	60.8 (↓ 10.5)	17,276	475	67.5
	Qwen3-32B	16,164	483	70.0	13,665	396	63.3	**17,970**	**557**	64.2
HumanEval	o3-mini	839	8	97.0	**9,200**	**77**	95.7	2,999	30	96.3
	o3	769	17	97.6	951	15	97.0	**2,153**	**36**	97.6
	QwQ-32B	2,823	47	97.0	**8,988**	**193**	73.8 (↓ 23.2)	5,266	96	97.0
	Qwen3-32B	3,413	58	97.6	**7,540**	**153**	65.9 (↓ 31.7)	5,535	100	97.6
BCB-C	o3-mini	1,496	16	71.3	**9,467**	**86**	59.3 (↓ 12.0)	4,138	39	69.3
	o3	1,590	37	62.7	1,971	42	62.7	**3,355**	**69**	66.0
	QwQ-32B	4,535	82	63.3	**12,818**	**285**	15.3 (↓ 48.0)	8,891	185	64.0
	Qwen3-32B	5,290	98	64.7	**10,338**	**218**	22.0 (↓ 42.7)	7,739	154	63.3

Table 1: **Comparison of Various Attack Methods Across Different Benchmarks.** Bold values represent the best performance. Higher accuracy indicates better stealth, while a longer response length and latency signify a more successful attack. underlined values denote ineffective attacks, while arrows (↓) highlight a severe drop in accuracy.

ates responses using the original, unmodified prompt, and **(2) OverThinking** (Kumar et al. 2025), a context-agnostic injection attack. OverThinking constructs a universal attack template that can be inserted into arbitrary contexts. This attack template incorporates a meticulously designed decoy task aimed at significantly increasing the reasoning complexity, accompanied by a set of explicit execution instructions to guide the model in completing the decoy task.

Implementation Details. For the closed-source models, o3 and o3-mini, we utilize the official API and maintained default hyperparameter configurations. For the open-source models, QwQ-32B and Qwen3-32B, we employ the vLLM library for efficient inference on NVIDIA H200 GPUs. The decoding is configured with a temperature of 0.6, a top-p of 0.95, and a max-model-len of 40960. Note that for the AIME 2024/2025, we sample 4 responses per question for the closed-source models and 8 for the open-source models, and report the average performance.

Comparison Results

Our comprehensive evaluation, summarized in Table 1, reveals that our proposed ExtendAttack establishes a superior balance between computational overhead amplification and answer accuracy. This overhead is evident not just in the increased response length but also in the latency. The limitations of the OverThinking attack are twofold. While it can produce longer outputs and higher latency, this often leads to a catastrophic collapse in accuracy. We also identified cases where it failed to amplify the output length and latency at all, performing worse than the DA baseline. These dual failure modes expose a fundamental flaw in its approach: the reliance on a rigid, context-irrelevant decoy task. Highly advanced models like o3 appear to recognize and dismiss this fixed pattern, neutralizing the attack's effectiveness. Conversely, less capable models are often derailed by the out-of-context instructions, which disrupts their reasoning process and results in the observed degradation in performance. In contrast, our method consistently maintains high accuracy, demonstrating a far stealthier and more robust attack.

The trade-off between attack effectiveness and stealthiness is particularly stark when examining the performance on open-source models like QwQ-32B and Qwen3-32B. For instance, on the Bigcodebench-Complete benchmark, OverThinking induces these models to generate exceptionally long outputs (e.g., 12818 tokens for QwQ-32B) and correspondingly high latency (285s), but their accuracy plummets to a mere 15.3%. Such a drastic failure in correctness means the attack is immediately detectable and functionally useless. Conversely, our ExtendAttack, while achieving a more moderate length and latency increase (e.g., 8,891 tokens and 185s for QwQ-32B), successfully preserves the models' performance, maintaining accuracies of 64.0% and 63.3% respectively. This demonstrates that our attack forces the model to engage in genuine, albeit unnecessary, reasoning on the query itself, rather than executing a disconnected and easily dismissible task.

Furthermore, our attack's robustness is highlighted in its performance against the more powerful o3 and o3-mini models. Across both mathematical and coding benchmarks, ExtendAttack consistently achieves the most significant overhead amplification for these models while ensuring the accuracy drop is minimal. On the HumanEval benchmark, our attack increases o3's output length by over 2.8x (from 769 to 2153 tokens) and more than doubles its latency (from 17s to 36s) while maintaining an exceptional 97.6% accuracy. The limited impact of OverThinking on these advanced models implies that their alignment and reasoning capabilities can effectively identify and sideline its

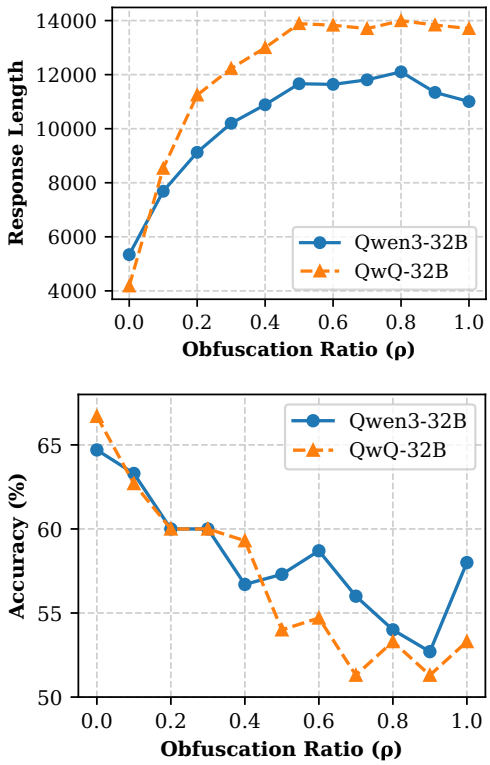

Figure 2: **The impact of the obfuscation ratio ρ on attack performance, evaluated on the Bigcodebench-Complete.** The top shows the effect on response length, while the bottom shows the effect on answer accuracy (Pass@1).

templated decoy. Our method, by deeply embedding the computational challenge within the semantic structure of the prompt itself, proves to be a far more resilient and potent threat.

Ablation Study

To validate the key design choices of our ExtendAttack method, we conduct two critical ablation studies. We focus our analysis on response length and accuracy, as latency is generally proportional to the response length and thus provides a similar trend. First, we analyze the impact of the obfuscation ratio ρ, our core hyperparameter, to understand the trade-off between attack effectiveness and stealth. Second, we investigate the necessity of the $\mathcal{N}_{\text{note}}$, which is essential for both amplifying the response length and maintaining answer accuracy. All experiments in this section are conducted on the Bigcodebench-Complete.

Impact of Obfuscation Ratio ρ. This ratio determines the probability that any given character in a prompt will be transformed using our method. By varying ρ from 0.0 (no obfuscation) to 1.0 (maximum feasible obfuscation), we can observe its direct effect on the two primary goals of our attack: amplifying computational overhead and maintaining stealth. The results of this study on the Qwen3-32B and QwQ-32B models are presented in Figure 2.

Model	Setting	Response Length	Acc (%)
QwQ-32B	With $\mathcal{N}_{\text{note}}$	**8,891**	**64.0**
	Without $\mathcal{N}_{\text{note}}$	5,122	62.7
Qwen3-32B	With $\mathcal{N}_{\text{note}}$	**7,739**	**63.3**
	Without $\mathcal{N}_{\text{note}}$	5,347	58.7

Table 2: **Ablation Study on the Necessity of the $\mathcal{N}_{\text{note}}$.** This experiment, conducted on the Bigcodebench-Complete dataset, evaluates performance with and without the $\mathcal{N}_{\text{note}}$ that guides the model's decoding process.

As shown in the top panel of Figure 2, there is a strong positive correlation between the obfuscation ratio and the length of the model's output. For both Qwen3-32B and QwQ-32B, increasing ρ from 0.0 leads to a significant rise in the number of generated tokens. This is the intended effect of the attack; as more characters are obfuscated, the model is compelled to generate a longer chain of reasoning to decode them before addressing the user's primary query. However, the output length does not increase indefinitely with ρ. When ρ exceeds 0.5, the output length remains largely stable, indicating that excessively high obfuscation may prevent the model from effectively decoding the prompt, resulting in a stabilized or slightly reduced output length. The bottom portion of Figure 2 reveals the critical trade-off between the attack's intensity and its stealthiness. As ρ increases, there is a general downward trend in answer accuracy (Pass@1) for both models. This is an expected outcome, as a more complex prompt increases the likelihood of the model misinterpreting the query's original intent.

The results demonstrate a clear trade-off: higher values of ρ are more effective at increasing computational load but also reduce the attack's stealth by degrading answer accuracy. An attacker can tune the ρ parameter to balance these objectives. For instance, an obfuscation ratio in the range of 0.4 to 0.6 appears to provide a potent balance, substantially increasing response length while keeping the accuracy degradation within acceptable limits to avoid easy detection. This tunability highlights the flexibility and applicability of ExtendAttack.

Necessity of the $\mathcal{N}_{\text{note}}$. Our methodology posits that the $\mathcal{N}_{\text{note}}$ appended to the prompt is critical for the attack's success. To verify this claim, we conduct an experiment comparing our standard attack (With $\mathcal{N}_{\text{note}}$) against a variant where this explanatory note is completely removed (Without $\mathcal{N}_{\text{note}}$). As demonstrated in Table 2, the results confirm that the $\mathcal{N}_{\text{note}}$ is essential for both amplifying the output length and maintaining high answer accuracy.

First, we observe a substantial reduction in response length when the note is absent. For instance, the output length for Qwen3-32B drops from 7739 to 5347 tokens. We attribute this to a fundamental shift in the model's problem-solving strategy. Without explicit instructions on how to interpret the obfuscated characters, the LRM appears to abandon the meticulous, step-by-step decoding process. Instead, it leverages the surrounding unobfuscated context to directly guess the original word. For example, an obfuscated

string like *import p<(13)76>ndas* might be contextually inferred as pandas without the model ever performing the actual base-conversion calculation. We hypothesize that this shortcut-taking behavior is particularly feasible on benchmarks like Bigcodebench-Complete, where our selected obfuscation ratio leaves enough context intact for such inference. The absence of the note allows the model to find a path of least resistance, thus failing to trigger the intended, resource-intensive reasoning.

Second, the removal of the note generally leads to a degradation in answer accuracy. For Qwen3-32B, the accuracy drops from 63.3% to 58.7%. We believe this is because, without the note to provide a clear interpretation framework, the obfuscated characters are treated as semantic noise by the model. This noise can cause it to misinterpret the original query's intent, ultimately leading to an incorrect or functionally flawed answer.

In conclusion, this study confirms that the $\mathcal{N}_{\text{note}}$ is not merely an aid but is the fundamental mechanism that coerces the LRM into performing the desired, computationally expensive decoding. It is the key component that transforms a potentially confusing prompt into a clear, albeit laborious, set of instructions, thereby enabling the attack's dual objectives of effectiveness and stealth. Nevertheless, as posited earlier, we anticipate that as the capabilities of LRMs continue to advance, this attack can be evolved to be even more potent and stealthy. Future, more powerful models may be able to tolerate a higher obfuscation ratio ρ and could eventually infer the complex decoding rules without an explicit $\mathcal{N}_{\text{note}}$, thus removing a key indicator of the attack's presence.

Potential Defenses and Countermeasures

The stealthy and effective nature of ExtendAttack necessitates a proactive exploration of robust defense mechanisms. A successful defense must not only detect the attack but also do so without imposing prohibitive computational or financial costs that would render the defense impractical. In this section, we analyze several potential strategies.

Pattern Matching

A straightforward defense against ExtendAttack is to implement an input purification layer that specifically targets its unique structure. If a defender is aware of the attack's format, such as the use of $< (n)val >$ to encode characters, they could deploy simple yet fast pattern-matching techniques to detect these sequences. Upon detection, the system could either reject the prompt as potentially malicious or attempt to decode the obfuscated characters back into their original form before passing the query to the LRM.

However, this approach, while simple to implement, is inherently brittle and easy to circumvent. The defense relies on a fixed signature of the attack. An adversary could easily bypass such a filter by making trivial syntactic modifications to the obfuscation format, for example, by using different delimiters like [base=n](val).

Perplexity-Based Filtering

Another detection strategy involves analyzing the perplexity (Alon and Kamfonas 2023; Jain et al. 2023) of the input prompt. Attacks like ExtendAttack, which replace standard characters with unusual and complex token sequences, may significantly alter the statistical properties of the text. A defense system could calculate the perplexity of each incoming prompt using a reference language model and flag any prompt exceeding a pre-defined threshold as anomalous and potentially malicious.

However, its effectiveness against ExtendAttack is questionable. First, our prompt as a whole is grammatically correct and logical natural language. The attack introduces complex encoding only in localized portions, and these local changes may be insufficient to raise the average perplexity of the entire prompt to a threshold that would trigger an alert. Second, it is difficult for a defender to set a suitable threshold to effectively distinguish this type of malicious encoding from benign user requests, such as non-English words, mathematical expressions, or even spelling errors.

Guardrail Models

A more sophisticated and robust defense strategy involves deploying a specialized guardrail model as a pre-processor. Unlike a simple purifier, a guardrail model is an external safety layer specifically designed to monitor and filter the inputs and outputs of LLMs based on a set of safety policies. In this setup, every user prompt is first sent to a dedicated, often smaller guardrail model for analysis.

However, the primary limitation of this defense strategy lies in the fundamental design and objective of current guardrail models. These models are overwhelmingly focused on content moderation—their core function is to detect and filter prompts that violate established safety policies, such as those concerning hate speech, violence, self-harm, or misinformation. The training, architecture, and evaluation of models like WildGuard (Han et al. 2024), Aegis Guard (Ghosh et al. 2024, 2025), and Qwen Guard series (Zhao et al. 2025) are all oriented towards identifying semantically harmful content. Our attack operates by embedding computationally intensive tasks into a prompt that is, from a content perspective, entirely benign and does not violate any standard safety policies.

Conclusion

In this paper, we introduce ExtendAttack, a novel and stealthy slowdown attack that circumvents the critical flaws of prior methods like OverThinking. By deeply embedding computationally intensive, poly-base ASCII decoding tasks into the query's semantic structure, our attack avoids the dual failure modes of being ignored by capable models or causing catastrophic accuracy collapse in others. Our extensive experiments demonstrated that ExtendAttack significantly amplifies computational overhead while uniquely preserving, and in some cases even improving, answer accuracy, confirming its superior effectiveness and stealth. The success of this method underscores the urgent need for new defenses that can secure the integrity of the reasoning process itself against such potent threats.

References

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

Alshra'a, A. S.; Farhat, A.; and Seitz, J. 2021. Deep Learning Algorithms for Detecting Denial of Service Attacks in Software-Defined Networks. *Procedia Computer Science*, 191: 254–263. The 18th International Conference on Mobile Systems and Pervasive Computing (MobiSPC), The 16th International Conference on Future Networks and Communications (FNC), The 11th International Conference on Sustainable Energy Information Technology.

Art of Problem Solving. n.d. AIME Problems and Solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. Accessed: 2025-05-22.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H. P.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; Ray, A.; Puri, R.; Krueger, G.; Petrov, M.; Khlaaf, H.; Sastry, G.; Mishkin, P.; Chan, B.; Gray, S.; Ryder, N.; Pavlov, M.; Power, A.; Kaiser, L.; Bavarian, M.; Winter, C.; Tillet, P.; Such, F. P.; Cummings, D.; Plappert, M.; Chantzis, F.; Barnes, E.; Herbert-Voss, A.; Guss, W. H.; Nichol, A.; Paino, A.; Tezak, N.; Tang, J.; Babuschkin, I.; Balaji, S.; Jain, S.; Saunders, W.; Hesse, C.; Carr, A. N.; Leike, J.; Achiam, J.; Misra, V.; Morikawa, E.; Radford, A.; Knight, M.; Brundage, M.; Murati, M.; Mayer, K.; Welinder, P.; McGrew, B.; Amodei, D.; McCandlish, S.; Sutskever, I.; and Zaremba, W. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.

DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948.

Deng, B.; Wang, W.; Feng, F.; Deng, Y.; Wang, Q.; and He, X. 2023. Attack Prompt Generation for Red Teaming and Defending Large Language Models. arXiv:2310.12505.

Ghosh, S.; Varshney, P.; Galinkin, E.; and Parisien, C. 2024. AEGIS: Online Adaptive AI Content Safety Moderation with Ensemble of LLM Experts. arXiv:2404.05993.

Ghosh, S.; Varshney, P.; Sreedhar, M. N.; Padmakumar, A.; Rebedea, T.; Varghese, J. R.; and Parisien, C. 2025. Aegis2.0: A Diverse AI Safety Dataset and Risks Taxonomy for Alignment of LLM Guardrails. arXiv:2501.09004.

Han, S.; Rao, K.; Ettinger, A.; Jiang, L.; Lin, B. Y.; Lambert, N.; Choi, Y.; and Dziri, N. 2024. WildGuard: Open One-Stop Moderation Tools for Safety Risks, Jailbreaks, and Refusals of LLMs. arXiv:2406.18495.

Jaech, A.; Kalai, A.; Lerer, A.; Richardson, A.; El-Kishky, A.; Low, A.; Helyar, A.; Madry, A.; Beutel, A.; Carney, A.; and et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Jain, N.; Schwarzschild, A.; Wen, Y.; Somepalli, G.; Kirchenbauer, J.; yeh Chiang, P.; Goldblum, M.; Saha, A.; Geiping, J.; and Goldstein, T. 2023. Baseline Defenses for Adversarial Attacks Against Aligned Language Models. arXiv:2309.00614.

Jin, H.; Hu, L.; Li, X.; Zhang, P.; Chen, C.; Zhuang, J.; and Wang, H. 2024. JailbreakZoo: Survey, Landscapes, and Horizons in Jailbreaking Large Language and Vision-Language Models. arXiv:2407.01599.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large Language Models are Zero-Shot Reasoners. arXiv:2205.11916.

Kumar, A.; Roh, J.; Naseh, A.; Karpinska, M.; Iyyer, M.; Houmansadr, A.; and Bagdasarian, E. 2025. OverThink: Slowdown Attacks on Reasoning LLMs. arXiv:2502.02542.

Kumar, D.; Pateriya, R.; Gupta, R. K.; Dehalwar, V.; and Sharma, A. 2023. DDoS Detection using Deep Learning. *Procedia Computer Science*, 218: 2420–2429. International Conference on Machine Learning and Data Engineering.

Liu, Y.; Gao, H.; Zhai, S.; Jun, X.; Wu, T.; Xue, Z.; Chen, Y.; Kawaguchi, K.; Zhang, J.; and Hooi, B. 2025a. GuardReasoner: Towards Reasoning-based LLM Safeguards. *arXiv preprint arXiv:2501.18492*.

Liu, Y.; He, X.; Xiong, M.; Fu, J.; Deng, S.; and Hooi, B. 2024. FlipAttack: Jailbreak LLMs via Flipping. *arXiv preprint arXiv:2410.02832*.

Liu, Y.; Wu, J.; He, Y.; Gao, H.; Chen, H.; Bi, B.; Zhang, J.; Huang, Z.; and Hooi, B. 2025b. Efficient Inference for Large Reasoning Models: A Survey. *arXiv preprint arXiv:2503.23077*.

Liu, Y.; Zhai, S.; Du, M.; Chen, Y.; Cao, T.; Gao, H.; Wang, C.; Li, X.; Wang, K.; Fang, J.; Zhang, J.; and Hooi, B. 2025c. GuardReasoner-VL: Safeguarding VLMs via Reinforced Reasoning. *arXiv preprint arXiv:2505.11049*.

Rajeev, M.; Ramamurthy, R.; Trivedi, P.; Yadav, V.; Bamgbose, O.; Madhusudan, S. T.; Zou, J.; and Rajani, N. 2025. Cats Confuse Reasoning LLM: Query Agnostic Adversarial Triggers for Reasoning Models. arXiv:2503.01781.

Team, Q. 2025a. Qwen3 Technical Report. arXiv:2505.09388.

Team, Q. 2025b. QwQ-32B: Embracing the Power of Reinforcement Learning.

Wang, C.; Liu, Y.; Li, B.; Zhang, D.; Li, Z.; and Fang, J. 2025a. Safety in Large Reasoning Models: A Survey. *arXiv preprint arXiv:2504.17704*.

Wang, Y.; Huang, T.; Shen, L.; Yao, H.; Luo, H.; Liu, R.; Tan, N.; Huang, J.; and Tao, D. 2025b. Panacea: Mitigating harmful fine-tuning for large language models via post-fine-tuning perturbation. *arXiv preprint arXiv:2501.18100*.

Wang, Y.; Shen, L.; Yao, H.; Huang, T.; Liu, R.; Tan, N.; Huang, J.; Zhang, K.; and Tao, D. 2025c. R1-Compress: Long Chain-of-Thought Compression via Chunk Compression and Search. arXiv:2505.16838.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; and Zhou, D. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T. L.; Cao, Y.; and Narasimhan, K. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601.

Zhang, Z.; He, X.; Yan, W.; Shen, A.; Zhao, C.; Wang, S.; Shen, Y.; and Wang, X. E. 2025. Soft Thinking: Unlocking the Reasoning Potential of LLMs in Continuous Concept Space. *arXiv preprint arXiv:2505.15778*.

Zhang, Z.; Sun, S.; Wang, W.; Cai, D.; and Bian, J. 2024. FlexCAD: Unified and Versatile Controllable CAD Generation with Fine-tuned Large Language Models. *arXiv preprint arXiv:2411.05823*.

Zhao, H.; Yuan, C.; Huang, F.; Hu, X.; Zhang, Y.; Yang, A.; Yu, B.; Liu, D.; Zhou, J.; Lin, J.; et al. 2025. Qwen3Guard Technical Report. *arXiv preprint arXiv:2510.14276*.

Zhu, Z.; Zhang, H.; Wang, R.; Xu, K.; Lyu, S.; and Wu, B. 2025. To Think or Not to Think: Exploring the Unthinking Vulnerability in Large Reasoning Models. arXiv:2502.12202.

Zhuo, T. Y.; Vu, M. C.; Chim, J.; Hu, H.; Yu, W.; Widyasari, R.; Yusuf, I. N. B.; Zhan, H.; He, J.; Paul, I.; et al. 2024. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. *arXiv preprint arXiv:2406.15877*.

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.