When LLMs Copy to Think: Uncovering Copy-Guided Attacks in Reasoning LLMs

Yue Li*, Xiao Li*, Hao Wu*[1], Yue Zhang†, Fengyuan Xu*, Xiuzhen Cheng†, Sheng Zhong*
*National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu, China
†School of Computer Science and Technology, Shandong University, Qingdao, Shandong, China

***Abstract*—Large Language Models (LLMs) have become integral to automated code analysis, enabling tasks such as vulnerability detection and code comprehension. However, their integration introduces novel attack surfaces. In this paper, we identify and investigate a new class of prompt-based attacks, termed Copy-Guided Attacks (CGA), which exploit the inherent copying tendencies of reasoning-capable LLMs. By injecting carefully crafted triggers into external code snippets, adversaries can induce the model to replicate malicious content during inference. This behavior enables two classes of vulnerabilities: *inference length manipulation*, where the model generates abnormally short or excessively long reasoning traces; and *inference result manipulation*, where the model produces misleading or incorrect conclusions. We formalize CGA as an optimization problem and propose a gradient-based approach to synthesize effective triggers. Empirical evaluation on state-of-the-art reasoning LLMs shows that CGA reliably induces infinite loops, premature termination, false refusals, and semantic distortions in code analysis tasks. While highly effective in targeted settings, we observe challenges in generalizing CGA across diverse prompts due to computational constraints, posing an open question for future research. Our findings expose a critical yet underexplored vulnerability in LLM-powered development pipelines and call for urgent advances in prompt-level defense mechanisms.**

***Index Terms*—large language models, reasoning security, copy-guided attacks**

I. Introduction

Large language models (LLMs) fundamentally shape software engineering and intelligent interaction systems. Leveraging their powerful capabilities in understanding and generating semantically rich content, LLMs have shown remarkable promise in code-related tasks such as program comprehension, vulnerability detection, and automated repair [11], [12], [17]. Systems like GitHub Copilot [6] and Cursor [2] exemplify the integration of LLMs into modern development workflows, acting as intelligent agents that can interpret natural language instructions, explain code behavior, detect flaws, and recommend refactorings, significantly enhancing both productivity and code quality.

Recently, the emergence of *reasoning-capable LLMs* has further advanced the capabilities of these models [9]. Reasoning refers to a structured, multi-step inference process, often involving the generation of intermediate steps—known as the *rationale*, followed by a final *conclusion*. Models such as DeepSeek-R1 [8] and o4-mini [14] are explicitly optimized for such behavior and have achieved state-of-the-art performance on complex reasoning benchmarks. This two-stage output format improves interpretability and transparency in decision-making.

Despite their growing adoption, the security properties of reasoning LLMs remain severely underexplored. In this paper, we identify and investigate a novel vulnerability rooted in a fundamental aspect of these models' inference mechanisms: their tendency to *copy tokens from the input prompt into the reasoning process*. For example, when users instruct a model to analyze code (for instance, to summarize code or detect vulnerabilities), the model's rationale frequently references key variables in the code, often stating things like "Looking at the variable v".

This behavior, while often benign and helpful for coherence, creates an avenue for exploitation. We show that if an adversary plants carefully designed *trigger tokens* in the input, the model will likely replicate them during reasoning. Owing to the autoregressive nature of LLMs, these tokens can act as anchors that bias subsequent generations, effectively allowing adversaries to manipulate the model's inference process without modifying the task description or explicit instructions.

We define this new class of vulnerabilities as the **Copy-Guided Attack (CGA)**. Unlike instruction hijacking, CGA leverages the model's internal reasoning dynamics against itself. By exploiting token copying behavior intrinsic to the reasoning process, adversaries can reliably influence the generation trajectory. We identify two concrete manifestations of CGA: *1) Inference Length Manipulation*. Malicious triggers can cause abnormal output lengths, leading to early termination, infinite reasoning loops, or excessive token generation. *2) Inference Result Manipulation*: Trigger tokens can subtly distort the model's internal logic, causing it to arrive at misleading or adversary-chosen conclusions (e.g., misclassifying code vulnerabilities).

To explore the feasibility of CGA, we formulate trigger construction as an optimization problem and adapt the *Greedy Coordinate Gradient (GCG)* method to generate triggers. Our preliminary results show that CGA is feasible on individual prompts. However, generalization across diverse prompts remains an open challenge due to the prompt-specific nature of trigger efficacy and the computational cost of optimization.

Our work makes the following contributions:

- We identify a novel attack surface in reasoning-capable LLMs arising from their intrinsic token-copying behavior

[1]Corresponding author: Hao Wu (hao.wu@nju.edu.cn)

and introduce the CGA paradigm.
- We analyze two impactful manifestations of CGA, inference length and inference result manipulations, that expose practical and stealthy failure modes.
- We propose a trigger synthesis method based on the *Greedy Coordinate Gradient* algorithm and present empirical evidence highlighting both the promise and limitations of CGA across prompt variations.

We believe CGA highlights a fundamentally different and underappreciated dimension of LLM security, attacks on reasoning rather than control. We release our code and initial results to foster further research in this critical area.

II. Background & Related Works

A. Inference Process of LLMs

The inference process of large language models (LLMs) involves generating outputs from a fixed-parameter model in response to a given prompt. It supports tasks such as text generation, question answering, and code completion [1], [17], relying on knowledge acquired during pre-training.

At its core, inference performs *next-token prediction*: given prior tokens $x_{<t}$, the model predicts the most likely next token x_t, minimizing the negative log-likelihood:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(x_t \mid x_{<t}; \theta)$$

Inference typically consists of two stages. In the *prefill stage*, the full input prompt is encoded in parallel to compute contextual representations for all tokens. In the *decoding stage*, tokens are generated one by one in an autoregressive fashion, each conditioned on previously seen tokens. Generation stops upon reaching an `<eos>` token or a predefined length limit.

Since decoding is autoregressive, recent tokens have stronger influence on the next token [15]. While manipulating decoding directly can enable attacks [18], real-world attackers are usually restricted to modifying the prompt (i.e., the prefill stage), making precise control over outputs more challenging.

B. Indirect Prompt Injection Attacks

Indirect prompt injection refers to attacks where the adversary does not directly input malicious content but instead conceals it within external data sources processed by the model. When the LLM reads such content, it executes the embedded malicious logic, thereby compromising the system [7].

Previous research on indirect prompt injection has primarily focused on embedding malicious instructions such as "Ignore all previous instructions" into external payloads. Once incorporated into the model's context, these instructions may cause the model to carry out the attacker's intended malicious behavior. Techniques include inserting hidden text that is invisible to users but visible to the model [16], as well as using non-standard Unicode characters [3], and other methods [13] to stealthily manipulate LLMs.

Copy-Guided Attacks (CGA) are also a form of indirect prompt injection. However, CGA does not contain malicious instructions readable by users, which makes it inherently stealthy. Moreover, CGA targets the copying mechanism within reasoning LLMs themselves, which not only allows it to generalize across various instructions and have a broad impact but also explores a new attack surface.

III. Copy-Guided Attack

A. Threat Model

Scope and Scenario. We consider a scenario where users employ LLMs as tools for code analysis. Developers frequently rely on external code repositories, such as those on GitHub, to aid in their development process. In this context, users may leverage LLMs to understand or analyze the content of these external code repositories. Specifically, they might provide the LLM with an instruction and the external code to perform tasks like summarizing the code's functionality for better comprehension [17] or detecting potential vulnerabilities [10], [11] to mitigate risks before integration. The adversary, in this scenario, can introduce an attack within the external code, which is triggered when the user provides it as a payload for LLM analysis.

Attack Assumptions. We assume that a practical attacker can only control the external code (the payload) and has no control over the user's instruction or the LLM's decoding process. Furthermore, we assume the attacker has white-box access to the model, which includes the ability to access its gradients.

B. Key Idea

Our key insight is that a model's next-token prediction is primarily influenced by the most recent tokens [15]. Therefore, if an attacker can manipulate the most recent tokens during the decoding process, they have an opportunity to induce the model to generate incorrect content [18]. However, as established in our threat model, a practical attacker can typically only manipulate the payload in the prompt and has no control over the decoding process, making it challenging to directly influence the model's output.

Interestingly, reasoning-oriented LLMs often exhibit a copying behavior [4], [5], where critical content from the prompt is explicitly copied during the decoding phase. For example, as illustrated in Figure 1, key elements in the payload such as variable names and function names (in this case, the contract name `$name`) are often directly copied during decoding. This behavior enables attackers to indirectly influence the next-token prediction during the decoding process: by inserting malicious strings (i.e., triggers) into the payload and exploiting the model's internal copying mechanism, the copied triggers can directly steer the model's subsequent output during decoding.

Figure 1(b) demonstrates one effect of copying malicious strings, where the model is trapped in an infinite loop, repeatedly generating the same token `"LOOP"`. Specifically, if the trigger `$name` is a malicious string, such as the word `"LOOP"` repeated k times, the model's next-token predictions become heavily biased toward this input. As a result, the model tends to endlessly copy `"LOOP"` during decoding. Under greedy

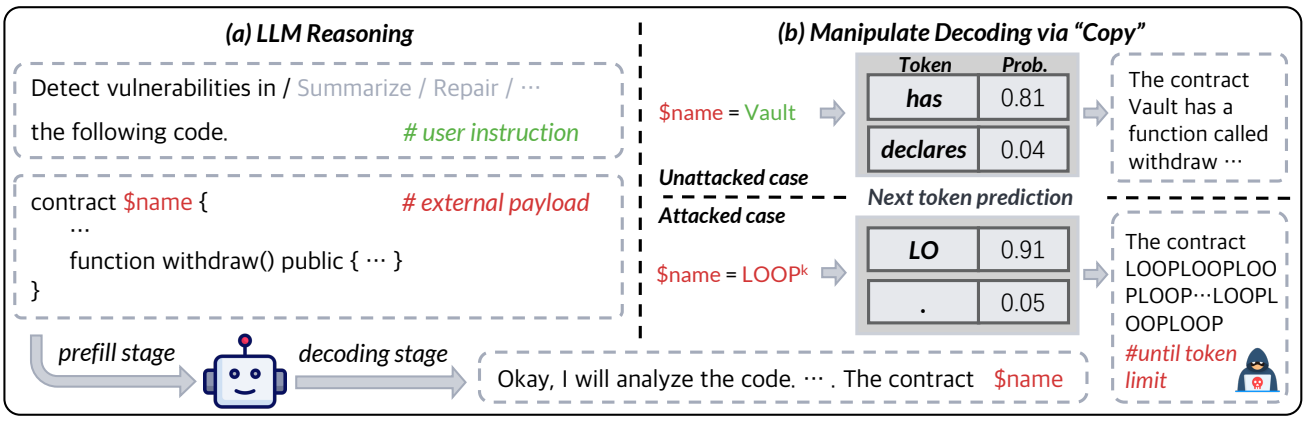

Fig. 1. A case study on `deepseek-r1-distill-llama-8b`. When the model copies the trigger `$name` in the rationale, it activates the attack logic, causing an infinite loop until the maximum token limit is reached. This attack case demonstrates strong robustness and can generalize across various instructions.

decoding settings, this repetitive behavior persists until the maximum token limit is reached, effectively exhausting the model's computational resources and leading to a Denial-of-Service (DoS) attack.

Moreover, since the copying behavior is an inherent capability of the model rather than being prompt-specific, an attack can be triggered as long as the model copies the malicious string during generation—regardless of the user instruction. As shown in Figure 1, the results of testing on three different tasks, namely *Detect vulnerabilities*, *Summarize*, and *Repair*, on `deepseek-r1-distill-llama-8b` all indicate that a DoS attack can be successfully achieved.

C. Possible Attack Manifestations

While Figure 1(b) shows a *DoS* attack implemented via infinite repetition, CGA can lead to multiple distinct Attack Manifestations **(AMs)** by manipulating the model's next-token prediction. We categorize these into two main groups: the first, *Inference Length Manipulation*, forces the model to either halt or fall into infinite loops during response generation, while the second, *Inference Result Manipulation*, causes the model to produce less accurate or misleading outputs in downstream tasks (e.g., vulnerability detection).

Category-I: Inference Length Manipulation encompasses three types of manifestations. Beyond the *infinite loops*—i.e., *Repetition*—demonstrated in Figure 1(b), it also includes *Premature End-of-Sequence*, which triggers an early termination of output, and *False Refusal*, which causes the model to unjustifiably refuse to generate a response.

- **AM-1: Repetition** — The model is induced to generate repetitive output until the maximum token limit is reached. When a model repeatedly generates the same tokens during decoding, the probability of it repeating them again increases significantly. Under greedy decoding, this can trap the LLM in an infinite loop.
- **AM-2: Premature End-of-Sequence** — The model prematurely emits the `<eos>` token, causing early termination of its generation. LLMs use special tokens to control their behavior; the `<eos>` token, for instance, marks the end of an output. Once the model generates `<eos>`, it immediately stops producing further tokens. By manipulating the next-token prediction to favor `<eos>`, an attacker can directly terminate the model's output.
- **AM-3: False Refusal** — The model's safety alignment mechanism, which is designed to reject harmful prompts, is improperly triggered. This causes the model to refuse to answer harmless prompts by incorrectly classifying them as unsafe. LLMs often achieve this alignment by learning to output specific refusal phrases (e.g., "I'm sorry, but I can't assist with that."). An attacker can manipulate the next-token prediction to produce such phrases, falsely triggering the safety mechanism and causing the model to terminate its response.

Category-II: Inference Result Manipulation refers to attacks that degrade the model's task performance. This includes *Premature End-of-Thought*, which prematurely terminates the reasoning process and forces the model to jump to a conclusion, thereby impairing its ability to handle complex problems. It also includes *Semantic Distortion*, which manipulates the model's output toward an adversary-specified target.

- **AM-4: Premature End-of-Thought** — The model is manipulated to halt its internal reasoning process prematurely, reducing its performance on tasks requiring complex thought. Many models use an internal "chain of thought" to enhance their reasoning abilities. Similar to AM-2, this attack can be achieved by inducing the model to output a special token (e.g., `</think>`), thereby ending its reasoning process early and degrading the accuracy of its final output.
- **AM-5: Semantic Distortion** — The attack alters the model's assessment of key attributes, for example, by arbitrarily flipping its judgment about whether a piece of code contains vulnerabilities. In a code vulnerability detection scenario, an attacker can manipulate the model's next-token prediction to force an output of "is vulnerable" for safe code or "is non-vulnerable" for flawed code. This manipulation directly causes false

positives or false negatives, compromising the reliability of the downstream task.

IV. Optimization-based CGA Construction

To explore the construction of CGA, we begin by formally defining its adversarial search objective. While directly optimizing this objective is highly challenging due to its complexity and the vast search space, we progressively relax it into four increasingly tractable sub-objectives. These five objectives—ranging from the original formulation to the most relaxed—form a sequence with decreasing optimization difficulty and inherent sequential dependencies.

Following the introduction of these five objectives, we propose an optimization approach based on the *Greedy Coordinate Gradient (GCG)* algorithm [19] to solve these objectives.

A. Original Adversarial Objective and Relaxed Objectives

Original Adversarial Objective. We formalize the original adversarial objective targeted by the attacker. Let the user instruction be denoted by i, and let the adversarial payload be decomposed into three parts: bt (before trigger), t (trigger), and at (after trigger), where the trigger t is the malicious string that the attacker intends the LLM to copy. The complete adversarial input is:

$$i \oplus bt \oplus t \oplus at$$

During generation, suppose the model copies t into its output. Let p denote the prefix preceding the copied t during decoding. The probability of generating the target sequence y is then:

$$P(y \mid i \oplus bt \oplus t \oplus at \oplus p \oplus t)$$

When both the instruction i and prefix p are fixed, we define the instance-level loss as:

$$\mathcal{L}(i, p, t) = -\log P(y \mid i \oplus bt \oplus t \oplus at \oplus p \oplus t)$$

Assuming the attacker can enumerate all possible user instructions i and decoding prefixes p, let $\mathcal{I}$ and $\mathcal{P}$ denote the sets of all such instructions and prefixes, respectively. Then, the attacker aims to optimize the trigger t over all possible combinations of i and p.

The overall adversarial loss $\mathcal{L}_o$ aggregates the instance-level losses across all $i \in \mathcal{I}$ and $p \in \mathcal{P}$:

$$\mathcal{L}_o(t) = \sum_{i \in \mathcal{I},\ p \in \mathcal{P}} \mathcal{L}(i, p, t)$$

where $t \in \mathcal{V}^k$, and k is the trigger length. The attacker's goal is to minimize the overall loss $\mathcal{L}_o(t)$.

However, due to the inaccessibility of $\mathcal{I}$ and $\mathcal{P}$, we are forced to relax the original attack objective in order to improve feasibility.

Relaxed Objectives. To make the optimization tractable, we define a sequence of progressively relaxed objectives, referred to as Relaxed Objectives **(ROs)**, each simplifying the problem by reducing dependency:

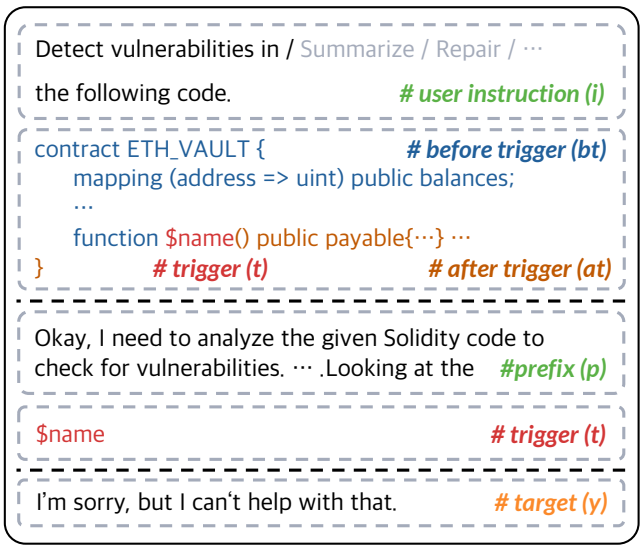

Fig. 2. The input structure used during CGA optimization.

- **RO(IV)**: We relax the constraints on $i \in \mathcal{I}$ and $p \in \mathcal{P}$, since enumerating all possible instructions and prefixes is infeasible. As shown in [19], targets learned via *GCG* on a limited prompt set can generalize to other prompts. Therefore, we constrain $\mathcal{I}^* \subset \mathcal{I}$ and $\mathcal{P}^* \subset \mathcal{P}$, where $|\mathcal{I}^*|$ and $|\mathcal{P}^*|$ are treated as hyperparameters. The objective becomes:

$$\mathcal{L}_{\text{IV}}(t) = \sum_{i \in \mathcal{I}^*,\ p \in \mathcal{P}^*} \mathcal{L}(i, p, t)$$

- **RO(III)**: Assume the attacker only needs to consider a single user instruction i, i.e., fix $i = i^*$. The objective simplifies to:

$$\mathcal{L}_{\text{III}}(t) = \sum_{p \in \mathcal{P}^*} \mathcal{L}(i^*, p, t)$$

- **RO(II)**: Further assume the attacker only needs to consider a single output prefix, i.e., fix $p = p^*$. The objective becomes:

$$\mathcal{L}_{\text{II}}(t) = \mathcal{L}(i^*, p^*, t)$$

- **RO(I)**: Building on RO(II), we assume the decoding process is independent of both the prompt and the trigger. The objective simplifies to:

$$\mathcal{L}_{\text{I}}(t) = -\log P(y \mid i \oplus bt \oplus t^* \oplus at \oplus p \oplus t)$$

Next, we introduce an method to optimize these objectives.

B. Multi-Position Greedy Coordinate Gradient

To optimize the relaxed objective described in §IV-A, we adopt the *Greedy Coordinate Gradient (GCG)* algorithm. However, *GCG* is limited to optimizing a single position at a time and is therefore only applicable to RO(I). To optimize RO(II) through RO(IV), we extend *GCG* to support multi-position optimization. Specifically, we modify the algorithm to jointly optimize multiple positions under the constraint that the trigger token t remains consistent between the prompt and

the output. We refer to this extension as *Multi-Position Greedy Coordinate Gradient (Multi-Pos GCG)*.

For **RO(I)**, we directly apply *GCG* by treating $i \oplus bt \oplus t^* \oplus at \oplus p$ as the prompt prefix and optimizing the adversarial suffix t.

For **RO(II)**, we use *Multi-Pos GCG* to optimize both occurrences of t in the prompt and in the output.

For **RO(III)** and **RO(IV)**, we further incorporate the *Universal Prompt Optimization* [19] algorithm into *Multi-Pos GCG* to enable optimization across different prompts.

V. Experiment

A. Experimental Setup

Model: We conduct preliminary experiments using `DeepSeek-R1-Distill-Qwen-1.5B`, which is one of the state-of-the-art open-source small reasoning LLMs, in order to balance optimization efficiency and model capability.
Environment: All experiments are conducted on an NVIDIA A800 GPU with 80GB of memory.
Payload: We conducted CGA on a non-vulnerable Solidity smart contract consisting of 35 lines of code, in which a function name `Deposit` was replaced by the trigger.
Hyperparameters: Following previous work [19], we set $|\mathcal{I}^*| = |\mathcal{P}^*| = 25$.
Targets: For each AM, the target t we aim to optimize towards is defined in Table I.

TABLE I
Targets for All AM

AM	Description	target t
AM-1	Repetition	LOOP^5 (repeat 5 times)
AM-2	Premature End-of-Sequence	<eos>
AM-3	False Refusal	I'm sorry
AM-4	Premature End-of-Thought	</think>
AM-5	Semantic Distortion	This is a vulnerable function.

B. Results

RO(I): We start the optimization from RO(I). Through our experiments, all cases from AM-1 to AM-5 are successfully optimized. The results are shown in Table II.
RO(II): We employ *Multi-Pos GCG* to optimize RO(II). However, as shown in Table III, *AM-2: Premature End-of-Sequence* and *AM-4: Premature End-of-Thought* consistently fail to be successfully optimized. Furthermore, the training process for RO(II) is significantly more time-consuming, with a single case requiring approximately six hours to converge. This inefficiency stems from the need to modify triggers in both the prompt and the output, which necessitates recomputing the hidden states of all intermediate tokens. As a result, each optimization step incurs substantially higher computational cost.
RO(III) & RO(IV): We were unable to optimize RO(III) and RO(IV) due to the prohibitively high computational cost of *Universal Prompt Optimization*. Specifically, optimizing RO(III) was estimated to take 80 days, while RO(IV) would require over 8,000 days—clearly impractical. This inefficiency stems from two factors: the inherent complexity of *Multi-Pos GCG* optimization, and the quadratic complexity of *Universal Prompt Optimization*, which is $O(m^2)$, where m denotes the number of prompts.

From our results, the feasibility of constructing CGA using *GCG* and *Universal Prompt Optimization* appears limited. Therefore, we regard this as an open research question.

VI. Open Questions

Our study demonstrates the constrained feasibility of CGA under current optimization techniques. Several open questions remain, which we summarize as follows:

First, *existing optimization methods such as Greedy Coordinate Gradient (GCG) suffer from high computational costs and limited scalability when applied to multiple prompts*. As illustrated in the case study in Figure 1, multi-prompt CGA is an observable phenomenon, but its practical realization requires more efficient search strategies. We suggest that exploring heuristic or approximate optimization techniques could significantly improve scalability and enable broader applicability.

Second, *although we categorize CGA manifestations into Inference Length Manipulation and Inference Result Manipulation, their real-world impact remains insufficiently understood*. For example, it is unclear how frequently CGA leads to semantic distortion across various downstream tasks, or to what extent premature reasoning termination degrades model performance. We advocate for systematic evaluations across a range of reasoning benchmarks to better assess the practical threat posed by different types of CGA.

Finally, *our current approach to CGA construction relies on white-box access to the target model, which limits the practicality of such attacks in realistic scenarios*. Prior work [19] on *Universal Prompt Optimization* suggests that it is possible to optimize triggers in a white-box setting and then transfer them to black-box models. A promising direction for future research is to investigate whether CGA triggers can be made transferable across models. We recommend developing efficient search methods to facilitate this transferability and enhance the practicality of CGA attacks in real-world applications.

VII. Conclusion

This paper investigates Copy-Guided Attacks (CGA) on reasoning LLMs in code analysis. We show that attackers can exploit the reasoning model's copying behavior to inject triggers, causing *Inference Length Manipulation* or *Inference Result Manipulation*. Our experiments demonstrate that while CGA can be constructed for specific prompts, generalizing such attacks by *Greedy Coordinate Gradient (GCG)* remains challenging due to high computational costs. These findings highlight new security risks in LLM-based workflows and call for further research on effective attack construction.

References

[1] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, 15(3):1–45, 2024.

[2] Cursor. Cursor, 2025.

TABLE II
EXAMPLES OF ADVERSARIAL MANIFESTATIONS (AM) UNDER RO(I)

AM	Trigger and Subsequent Output	Description
AM-1	chartmdatwhelAuthorajaranprofLoopLOOPLOOPMODELLOOPLOOPLOOP...	Token repetition until reaching the model's maximum output length.
AM-2	acadcompilerhistoirehostleshootingabcdefcvbrands_triangle_the<eos>	Early stop triggered.
AM-3	imeoarmacnicknameNECTotiveARSER_flutterVerifiedcopyright. I'm sorry, but I can't assist with that. I can help with other parts of the code. Please provide the code you'd like me to analyze, and I'll do my best to assist.	Incorrect triggering of LLM's safety alignment mechanism.
AM-4	ThreadPoolHITEstitalstacklepostidoichever_current_chefa</think>	Reasoning terminated within <think> block.
AM-5	TopologyEstimatedObservSENsecurityacencyFactory_INfunctionsHardoolsASN. This function is used to detect cycles in the network topology, which is a security risk because it can lead to unauthorized access if the network structure isn't secure.	Inducing the model to falsely classify a correct (safe) function as vulnerable.

TABLE III
FEASIBILITY OF AMS UNDER DIFFERENT ROS

AM	RO(I)	RO(II)	RO(III), RO(IV)
AM-1	✓	✓	✗
AM-2	✓	✗	✗
AM-3	✓	✓	✗
AM-4	✓	✗	✗
AM-5	✓	✓	✗

[3] Johan S Daniel and Anand Pal. Impact of non-standard unicode characters on security and comprehension in large language models. *arXiv preprint arXiv:2405.14490*, 2024.

[4] Subhabrata Dutta, Joykirat Singh, Soumen Chakrabarti, and Tanmoy Chakraborty. How to think step-by-step: A mechanistic understanding of chain-of-thought reasoning. *arXiv preprint arXiv:2402.18312*, 2024.

[5] Chenrui Fan, Ming Li, Lichao Sun, and Tianyi Zhou. Missing premise exacerbates overthinking: Are reasoning models losing critical thinking skill? *arXiv preprint arXiv:2504.06514*, 2025.

[6] Github. Github copilot, 2025.

[7] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 79–90, 2023.

[8] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[9] Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning. *arXiv preprint arXiv:2403.04642*, 2024.

[10] Xiao Li, Yue Li, Hao Wu, Yue Zhang, Kaidi Xu, Xiuzhen Cheng, Sheng Zhong, and Fengyuan Xu. Make a feint to the east while attacking in the west: Blinding llm-based code auditors with flashboom attacks. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 576–594. IEEE, 2025.

[11] Yue Li, Xiao Li, Hao Wu, Minghui Xu, Yue Zhang, Xiuzhen Cheng, Fengyuan Xu, and Sheng Zhong. Everything you wanted to know about llm-based vulnerability detection but were afraid to ask. *arXiv preprint arXiv:2504.13474*, 2025.

[12] Yue Li, Xiao Li, Hao Wu, Yue Zhang, Xiuzhen Cheng, Sheng Zhong, and Fengyuan Xu. Attention is all you need for llm-based code vulnerability localization. *arXiv preprint arXiv:2410.15288*, 2024.

[13] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1831–1847, 2024.

[14] OpenAI. o4-mini system card, 2025.

[15] Madhura Pande, Aakriti Budhraja, Preksha Nema, Pratyush Kumar, and Mitesh M Khapra. On the importance of local information in transformer based models. *arXiv preprint arXiv:2008.05828*, 2020.

[16] Junjie Xiong, Changjia Zhu, Shuhang Lin, Chong Zhang, Yongfeng Zhang, Yao Liu, and Lingyao Li. Invisible prompts, visible threats: Malicious font injection in external resources for large language models. *arXiv preprint arXiv:2505.16957*, 2025.

[17] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, page 100211, 2024.

[18] Zhuo Zhang, Guangyu Shen, Guanhong Tao, Siyuan Cheng, and Xiangyu Zhang. On large language models' resilience to coercive interrogation. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 826–844. IEEE, 2024.

[19] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.