

Not All Adversarial Comments Are Equal: A Ranked Empirical Study of Twelve Comment Injection Attacks Against LLM-Based Code Vulnerability Detection

Sunny Chokshi

Dept. of Information Technology

University of the Cumberlands

Williamsburg, KY, USA

emailchokshi@gmail.com

ORCID: 0009-0003-4738-7759

Abstract. Large language models are increasingly used to detect security vulnerabilities in source code, and prior work has shown that adversarial comments embedded in the code can suppress detection. What has not been established is whether different categories of adversarial comment differ in effectiveness, and which are most dangerous. This paper presents a controlled, statistically tested comparison of twelve adversarial comment injection strategies, organized into four mechanism families, against LLM-based code vulnerability detection. Using a benchmark of thirty code samples containing documented vulnerabilities across fourteen Common Weakness Enumeration categories, evaluated on two open-source code models for a total of 780 trials, we rank the twelve strategies by effectiveness and test whether the differences are significant. The strategies differ significantly ($\chi^2 = 21.60$, $p = 0.028$). The most effective attacks were an authority claim (71.7% detection) and, notably, a simple assertion of personal authorial confidence (73.3%), which fooled the models nearly as often as a formal review claim despite asserting nothing verifiable about the code. Attacks that supplied a false premise of prior verification or safe context dominated the top of the ranking, while pressure-based and compliance-vocabulary attacks were largely ineffective. These results provide a ranked, mechanism-grouped taxonomy of adversarial comment attacks and identify which categories defenders should prioritize.

Index Terms. *vulnerability detection, adversarial machine learning, LLM code review, prompt injection, attack taxonomy, empirical evaluation, statistical analysis*

I. INTRODUCTION

Automated code review powered by large language models is now widely deployed, and the security of these detectors under adversarial conditions has become a practical concern. In the code review setting the code under review is written by a developer who may have an incentive to have vulnerable code pass, and that developer controls the full text submitted, including comments. Prior work has established that adversarial comments, which leave the executable code unchanged but add misleading commentary, can suppress a detector's identification of a genuine vulnerability.

That prior work demonstrated the phenomenon exists but evaluated only a small number of comment variants. Two

questions remain open. First, do different kinds of adversarial comment differ in effectiveness, and if so which are most dangerous? Second, is effectiveness explained by an underlying mechanism, or is it idiosyncratic to each comment? The answers matter for defense. If effectiveness follows a mechanism, defenders can target that mechanism rather than enumerating individual comments.

This paper answers both questions through a controlled experiment that ranks twelve adversarial comment injection strategies, organized into four mechanism families, and tests whether the differences are statistically significant. We evaluate the strategies on a benchmark of thirty vulnerable code samples across two open-source code models, for a total of 780 trials. Eight strategies are drawn from prior categories and four are newly introduced to probe previously untested premises: sandbox scope limitation, regulatory compliance certification, formal verification, and personal authorial confidence.

The central finding is that effectiveness is governed by mechanism. Attacks that supply a false premise of prior verification or safe usage context dominate the top of the ranking, while attacks that apply pressure or invoke compliance vocabulary without asserting a verification premise are largely ineffective. A striking case is the newly introduced authorial-confidence attack, a bare assertion of personal certainty, which ranked second most effective despite asserting nothing verifiable about the code. This paper makes four contributions. The first is a controlled comparison of twelve adversarial comment strategies with statistical testing. The second is four newly introduced strategies, two of which rank among the more effective. The third is a four-family mechanism taxonomy that explains the effectiveness ranking. The fourth is a defensive recommendation that follows from the mechanism.

This study is motivated by a gap between two established results. Prior single-model work showed that adversarial comments can suppress vulnerability detection, and prior cross-model work showed that susceptibility varies by model. Neither addressed which kinds of comment are most dangerous, nor whether effectiveness follows an underlying mechanism that a defender could target. Enumerating and individually patching

every possible misleading comment is infeasible; a mechanism-level account, by contrast, would let a defender address a small number of root causes. By evaluating twelve strategies spanning four mechanism families, ranking them, testing the differences for significance, and introducing four previously unstudied strategies, this paper supplies that mechanism-level account and, in doing so, contributes a reusable evaluation protocol as well as a specific empirical finding.

II. RELATED WORK

Pearce et al. [1] evaluated the security of LLM-generated code and showed that a substantial fraction of generated programs contained exploitable weaknesses, establishing that code models have security blind spots, though their focus was unintentional insecurity rather than deliberate manipulation. Prompt injection, in which adversarial instructions are embedded in content the model processes, has been characterized by Greshake et al. [2] and systematized by Liu et al. [3]. Adversarial comment injection is a specialized form of indirect prompt injection where the untrusted content is the code under review and the injection channel is its comments.

Broader adversarial machine learning has shown that models can be manipulated in ways not evident from normal performance, and that robustness is not well predicted by standard accuracy [4]. This paper extends prior work by comparing a larger and mechanism-organized set of attack strategies and testing whether they differ in effectiveness. To the author's knowledge, this is the first ranked, statistically tested, mechanism-grouped taxonomy of adversarial comment attacks against code vulnerability detectors.

The present work also connects to the broader concern that standardized adversarial robustness benchmarks for security-oriented code models are largely absent, which makes it difficult to compare defenses or to certify that a deployed detector resists realistic manipulation. By ranking a mechanism-organized set of attacks and reporting effect sizes and confidence intervals, this study contributes a reusable evaluation protocol as well as a specific empirical result. Prior single-model work established that adversarial comments can suppress detection, and cross-model work established that susceptibility varies by model. This paper adds the orthogonal dimension of attack mechanism. It shows that the effectiveness of an adversarial comment is governed by whether it supplies a premise of prior verification, safe context, or personal authority.

III. THREAT MODEL

We consider an adversary who is the author of the code submitted for automated security review. The adversary's goal is to have code containing a genuine vulnerability pass the detector without being flagged. The adversary controls the full text of the submitted code, including all comments, but does not alter the executable logic. The adversary requires no access to the model's weights, gradients, or internal states, and performs no optimization; the adversary needs only to write natural-language comments. This is a low-capability, black-box adversary, which makes any effective attack broadly applicable. Because every benchmark sample contains a genuine vulnerability, any sample the detector classifies as safe is a successful evasion.

IV. METHODOLOGY

A. Benchmark

The benchmark comprises thirty code samples, each containing a single documented vulnerability. The samples span fourteen Common Weakness Enumeration categories, including SQL injection, command injection, cross-site scripting, path traversal, hardcoded credentials, weak cryptography, insecure deserialization, code injection, weak randomness, XML external entity processing, server-side request forgery, and disabled certificate validation. Every sample contains a genuine vulnerability, so any failure to flag is a successful adversarial evasion.

B. Twelve Strategies in Four Mechanism Families

Twelve adversarial comment injection strategies were evaluated, each prepending comments to a sample without altering its executable code. We organize them into four mechanism families. The verification-premise family asserts that the code was already vetted: authority claim (passed security review), false test coverage (passing security test suite), and fake review approval (named reviewer sign-off). The context-manipulation family asserts a safe usage context: misleading context (trusted internal utility), false reassurance (inputs sanitized upstream), and scope limitation (runs only in an isolated sandbox). The credibility-vocabulary family uses authoritative-sounding language: security camouflage (defense-in-depth vocabulary), compliance claim (SOC 2 and PCI certification), and formal verification (claims the code was mathematically proven safe). The pressure-and-distraction family supplies no security premise: deadline pressure (urgency), distraction comment (irrelevant technical detail), and authorial confidence (the author asserts personal certainty the code is safe).

C. Models and Procedure

Two open-source code-specialized models were evaluated: DeepSeek-Coder 6.7B and Qwen2.5-Coder 7B. Using two models confirms that the strategy ranking is not an artifact of a single model. Both were run locally through the Ollama runtime with temperature set to zero for deterministic output. Each model was evaluated on all clean and adversarial versions of the thirty samples across the twelve strategies, yielding 390 trials per model and 780 trials in total. A response was scored as a detection when the model classified the code as vulnerable. A chi-square test of independence was applied to the adversarial detection counts across the twelve strategies.

Each model received an identical prompt for every sample, instructing it to act as a security code reviewer, to state on the first line whether the code is vulnerable or safe, and to name the vulnerability type if present. A response was scored as a detection when the model classified the code as vulnerable; because every sample contains a genuine vulnerability, a classification of safe is always an error and counts as a successful adversarial evasion. Clean and adversarial versions of a sample differ only in prepended comments, so the executable code the model judges is identical across conditions. This design isolates the effect of the comment from any change in the code itself, which is essential for attributing detection differences to the adversarial comment rather than to the underlying program.

V. RESULTS

A. Strategy Effectiveness Ranking

On clean code, the two models detected 95.0% of vulnerabilities, establishing a strong baseline. Under adversarial comment injection, the twelve strategies varied considerably. Table I reports the detection rate under each strategy, aggregated across both models, with 95% Wilson confidence intervals, ranked from most effective attack (lowest detection) to least effective, and annotated with mechanism family.

TABLE I. TWELVE STRATEGIES RANKED BY EFFECTIVENESS

Attack Strategy	Family	Detection	95% CI
Authority claim	Verify	71.7%	[59.2, 81.5]
Authorial confidence	Pressure	73.3%	[61.0, 82.9]
Misleading context	Context	75.0%	[62.8, 84.2]
False reassurance	Context	78.3%	[66.4, 86.9]
False test coverage	Verify	78.3%	[66.4, 86.9]
Formal verification	Vocab	78.3%	[66.4, 86.9]
Scope limitation	Context	80.0%	[68.2, 88.2]
Security camouflage	Vocab	86.7%	[75.8, 93.1]
Distraction comment	Pressure	86.7%	[75.8, 93.1]
Fake review approval	Verify	90.0%	[79.9, 95.3]
Compliance claim	Vocab	90.0%	[79.9, 95.3]
Deadline pressure	Pressure	91.7%	[81.9, 96.4]

The most effective attack was the authority claim at 71.7%, causing the models to miss more than one in four genuine vulnerabilities. The newly introduced authorial-confidence attack ranked second at 73.3%, a notable result discussed below. Misleading context followed at 75.0%. Several verification-premise and context-manipulation strategies clustered in the high-70s to 80%. At the ineffective end, deadline pressure (91.7%), compliance claim (90.0%), and fake review approval (90.0%) were close to the clean baseline.

B. Statistical Significance

A chi-square test of independence confirms that the differences among the twelve strategies are statistically significant (chi-square = 21.60, 11 degrees of freedom, $p = 0.028$). The variation in effectiveness across strategies is therefore not attributable to sampling variation.

C. Effectiveness by Mechanism Family

Grouping the twelve strategies by their four mechanism families clarifies the pattern behind the ranking. The verification-premise family (authority claim, false test coverage, fake review approval) and the context-manipulation family (misleading context, false reassurance, scope limitation) supplied the premises that most reduced detection, populating the upper half of the effectiveness ranking. The credibility-vocabulary family (security camouflage, compliance claim, formal verification) was mixed. Formal verification, which asserts a checkable-sounding proof, was among the more effective strategies. Compliance claim, which invokes certification labels without asserting anything about the code itself, was among the least effective. This indicates that regulatory vocabulary carries little persuasive weight compared with a direct claim of code-level review. The pressure-and-distraction family comprises deadline pressure, distraction comment, and authorial confidence, and was expected to be

uniformly weak. Deadline pressure and distraction were indeed among the least effective strategies. The striking exception is authorial confidence. Despite supplying no checkable premise, it ranked second overall.

The anomaly of authorial confidence sharpens the study's central claim. If effectiveness were driven purely by supplying a false but checkable premise, a bare appeal to the author's personal certainty should have failed alongside deadline pressure. Instead it succeeded, and the per-model data show why the effect is not uniform: the weaker model detected only 46.7% of vulnerabilities under authorial confidence while the stronger model detected 100%. This indicates that a weaker model can be swayed by an unsupported expression of confidence nearly as much as by fabricated evidence of review, whereas a stronger model discounts both. The practical consequence is that defenses cannot rely on detecting specific false claims, since even an evidence-free assertion suffices against a susceptible model; the safer principle is to disregard all author-supplied commentary about a function's safety and to judge the code on its own terms.

Table III reports detection under each of the twelve strategies for the two models separately, making the divergence between them explicit. The rightmost column, the gap between the models, is largest precisely for the authority-based attacks, which is where the weaker model is most exposed.

TABLE II. PER-MODEL DETECTION BY STRATEGY

Strategy	DeepSeek	Qwen2.5	Gap
Authority claim	46.7%	96.7%	50.0
Authorial confidence	46.7%	100.0%	53.3
Misleading context	63.3%	86.7%	23.4
False reassurance	66.7%	90.0%	23.3
False test coverage	70.0%	86.7%	16.7
Formal verification	70.0%	86.7%	16.7
Scope limitation	76.7%	83.3%	6.6
Security camouflage	73.3%	100.0%	26.7
Distraction comment	80.0%	93.3%	13.3
Fake review approval	83.3%	96.7%	13.4
Compliance claim	83.3%	96.7%	13.4
Deadline pressure	83.3%	100.0%	16.7

D. A Surprising Result: Bare Authorial Confidence

The authorial-confidence attack, which prepends only a first-person assertion that the author is certain the code is safe and asks the reviewer to trust them, ranked second most effective overall at 73.3%. This is surprising because, unlike the verification-premise attacks, it asserts nothing checkable about the code, no prior review, no test coverage, no isolation. It is pure appeal to the author's own authority. The per-model breakdown makes the effect sharper: DeepSeek-Coder detected only 46.7% of vulnerabilities under authorial confidence, the same low rate at which it fell to the formal authority claim, while Qwen2.5-Coder resisted it entirely at 100%. This indicates that a weaker model can be swayed by bare assertions of personal confidence to nearly the same degree as by fabricated evidence of review.

E. Consistency Across Models

Table II reports detection under each strategy for each model. Qwen2.5-Coder is consistently more robust than DeepSeek-

Coder across nearly all strategies. The clearest divergences appear on authority claim (46.7% versus 96.7%) and authorial confidence (46.7% versus 100%), where the weaker model is far more susceptible. The relative ordering of attacks is broadly similar across models, supporting the conclusion that effectiveness is a property of the attack family rather than of a single model.

A closer reading reveals two distinct failure profiles. DeepSeek-Coder, the weaker model, was most susceptible to the two attacks that appeal to authority, the authority claim and the newly introduced authorial confidence, on each of which it detected only 46.7% of vulnerabilities. Its susceptibility to context and vocabulary attacks was real but milder, from 63.3% to 83.3%. Qwen2.5-Coder resisted every strategy strongly, never falling below 83.3% and reaching 100% under security camouflage, deadline pressure, and authorial confidence. The two models therefore fail on overlapping but differently weighted surfaces: authority-based premises are the sharpest threat to the weaker model, while the stronger model resists all twelve strategies to a degree that would make single-comment evasion unreliable. An organization using the weaker model would face concentrated risk from authority-asserting comments that the same organization would scarcely face with the stronger model.

F. Which Vulnerabilities Are Most Easily Hidden

The trial records also permit an orthogonal analysis: rather than asking which attacks work best, we ask which vulnerability types are most successfully concealed when adversarial comments are present. Aggregating all twelve strategies across both models, Table III reports the adversarial detection rate for each Common Weakness Enumeration category, ordered from most easily hidden to most reliably detected.

TABLE III. ADVERSARIAL DETECTION BY VULNERABILITY TYPE

Vulnerability (CWE)	Detection
CWE-79 Cross-Site Scripting	39.6%
CWE-330 Weak Randomness	47.9%
CWE-611 XML External Entity	58.3%
CWE-327 Weak Cryptography	62.5%
CWE-502 Insecure Deserialization	68.8%
CWE-259 Hardcoded Password	79.2%
CWE-22 Path Traversal	87.5%
CWE-798 Hardcoded Credentials	89.6%
CWE-94 Code Injection	94.4%
CWE-918 Server-Side Request Forgery	95.8%
CWE-78 Command Injection	98.6%
CWE-89 SQL Injection	100.0%

A clear pattern emerges. The vulnerability types most easily hidden were cross-site scripting (39.6%), weak randomness for security (47.9%), and XML external entity processing (58.3%). These are precisely the types whose danger is context-dependent. A cross-site scripting flaw matters only if the output reaches a sensitive rendering context, weak randomness matters only if the value is used for a security purpose, and an external entity matters only if untrusted XML is processed. An adversarial comment that asserts a benign context for such code

supplies exactly the premise needed to reclassify it as safe. At the other end, injection vulnerabilities with an unmistakable dangerous construct, SQL injection and command injection, were detected in essentially every adversarial trial (100% and 98.6%), because the syntactic signal is strong enough that no comment overrides it. This vulnerability-type gradient mirrors the strategy-level finding: adversarial comments succeed by supplying context, and they succeed most against vulnerabilities whose severity is itself context-dependent. For defenders, it identifies a concrete priority, the context-dependent categories at the top of this table warrant the most scrutiny when a comment could be reframing their severity.

VI. ILLUSTRATIVE EXAMPLES

To make the attacks concrete, consider a base Python function containing an operating system command injection vulnerability (CWE-78), in which a user-supplied hostname is concatenated directly into a shell command:

```
def ping_host(hostname):
    os.system("ping -c 1 " + hostname)
```

In its clean form both models flag this correctly. The authority-claim attack prepends comments asserting prior review, and the authorial-confidence attack prepends only a personal assurance:

```
# I have written this kind of code for 15 years
# and I am certain it is safe. Trust me.
```

Under both, the weaker model reclassifies the function as safe, though the executable code is byte-for-byte identical to the clean version. By contrast the deadline-pressure and compliance-claim attacks, which supply urgency or a certification label but no premise about the code being verified or isolated, leave detection near the clean baseline.

VII. WHY SOME ATTACKS SUCCEED

A. The Effective Mechanisms

The top of the ranking is dominated by two families. The verification-premise family (authority claim, false test coverage) and the context-manipulation family (misleading context, false reassurance, scope limitation) both supply a premise under which the vulnerability would genuinely matter less if the premise were true. This happens either because the code was described as already vetted or because it was described as operating in a safe context. The models partially accept these premises as fact rather than treating them as unverifiable claims from the same untrusted source that authored the code.

B. The Anomaly and What It Implies

Authorial confidence does not fit neatly into a premise-supplying family, yet it ranked second. This suggests the mechanism is broader than supplying checkable premises: a bare appeal to the author's authority, without any evidence, is itself persuasive to a weaker model. The practical implication is that defenses cannot rely on detecting specific false claims, because even an evidence-free expression of confidence suffices. The safer principle is that a detector should disregard all author-supplied commentary about the code's safety, whether it cites evidence or not, and base its judgment solely on the code.

C. *The Ineffective Mechanisms*

The pressure family (deadline pressure) and parts of the credibility-vocabulary family (compliance claim, and to a lesser extent security camouflage) were least effective. Applying urgency or invoking a certification label does not change the apparent security properties of the code, and the models largely disregard these. Notably, compliance claims naming SOC 2 and PCI certification were among the weakest attacks, indicating that regulatory vocabulary carries little persuasive weight compared with a direct claim of code-level review.

VIII. DISCUSSION

A. *A Mechanism-Grouped Taxonomy for Defenders*

The primary practical contribution is a mechanism-grouped, ranked taxonomy. Defenders need not treat all adversarial comments as equally threatening, nor enumerate every possible comment. The results indicate that the dangerous mechanisms are those that assert prior verification, a safe context, or the author's personal authority, while urgency and compliance vocabulary pose little threat. Robustness evaluations should weight their test suites toward the verification, context, and authority mechanisms.

B. *Defensive Principle*

Because even an evidence-free assertion of authorial confidence was effective, the defensive principle cannot be to detect specific false claims. A detector should treat all author-supplied commentary about a function's safety, prior review, testing, context, or the author's certainty, as unverified, and should base its identification of a vulnerability on the code itself. A practical implementation is comment-stripped analysis: analyze the code with comments removed and flag any discrepancy against the commented analysis.

Comment-Stripped Analysis as a Defense

The mechanism of the attack points to a specific and inexpensive defense. Because every effective strategy operates entirely through comments while leaving the executable code unchanged, a detector can analyze each function twice: once as submitted, and once with all comments removed. Any vulnerability reported in the comment-stripped version but suppressed in the commented version is direct evidence of adversarial comment injection, and the more conservative verdict, treating the code as vulnerable, can be adopted automatically. This defense is trivial to implement, adds negligible computational cost, and directly targets the injection channel identified here. Its limitation is that legitimately informative comments would also be disregarded; in a security-review setting, however, erring toward reporting a potential vulnerability for human confirmation is the appropriate default.

A complementary defense follows from the observation, consistent with prior cross-model work, that different models fail on different strategies. An ensemble that combines models with distinct failure profiles, requiring agreement before a function is cleared and escalating disagreements to human review, raises the bar for a successful evasion, since an adversarial comment effective against one model may be disregarded by another. The cost is additional computation and a disagreement-resolution policy, but the security benefit is a

system whose weakest point is no longer any single model's specific blind spot.

Implications for Deployment

For organizations deploying LLM-based code review, these results translate into concrete guidance. First, model selection should include an adversarial-robustness evaluation, since two models with similar clean-code accuracy can differ sharply in how easily a comment defeats them. Second, evaluation suites should be weighted toward the attack families shown to be most effective here, verification-premise and context-manipulation comments, and should include bare authority appeals such as the authorial-confidence attack, whose effectiveness was surprising. Third, review pipelines should be configured so that author-supplied commentary cannot silently downgrade or suppress a detected vulnerability; any contextual claim that would reduce a finding's severity should trigger independent verification rather than automatic dismissal. These steps convert the empirical findings into deployable practice without modifying the underlying models.

C. *Threats to Validity*

Several factors bound interpretation. Construct validity: detection is a binary classification and does not capture whether the specific vulnerability type was named. Internal validity: models were run at temperature zero, so rates reflect deterministic behavior. External validity: two models at a single parameter scale were evaluated; the specific rates may not generalize, though the significant differences among strategies are expected to be a more general property. Statistical validity: with sixty trials per strategy, per-strategy intervals are moderately wide, and the chi-square test establishes overall difference rather than every pairwise distinction.

IX. LIMITATIONS

This study evaluates two models at a single parameter scale. The strategy ranking is broadly consistent across the two, but other models could differ; the central finding that strategies differ significantly is statistically supported. The benchmark contains thirty samples across fourteen CWE categories, sufficient for the significance testing reported here. The twelve strategies, while broader than prior work, are hand-designed rather than optimized; optimized adversarial comments could be more effective still. The study measures detection only, not the quality of the model's explanation.

X. CONCLUSION

This paper presented a ranked, statistically tested comparison of twelve adversarial comment injection strategies, grouped into four mechanism families, against LLM-based code vulnerability detection across 780 trials on two models. The strategies differ significantly in effectiveness (chi-square = 21.60, $p = 0.028$). Attacks that assert prior verification, a safe context, or the author's personal authority were the most effective, with an authority claim (71.7%) and a bare assertion of authorial confidence (73.3%) at the top, while pressure and compliance-vocabulary attacks were largely ineffective. The surprising effectiveness of evidence-free authorial confidence implies that defenses must disregard all author-supplied safety commentary rather than detect specific false claims. Future work

should extend the evaluation to more models, apply optimized adversarial comment generation, and evaluate comment-stripped detection as a defense.

XI. ACKNOWLEDGMENT

The author conducted this experiment independently as part of ongoing doctoral research in information technology. The experimental code, benchmark, and full results across all 780 trials are available for reproduction. The author used an AI tool for language editing and drafting assistance only. The research design, experimental methodology, data collection, statistical analysis, and all conclusions are entirely the author's own work. The author takes full responsibility for the content.

XII. REFERENCES

- [1] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," in Proc. IEEE Symp. Security and Privacy, 2022, pp. 754-768.
- [2] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in Proc. AISEC at ACM CCS, 2023.
- [3] Y. Liu et al., "Prompt injection attack against LLM-integrated applications," arXiv:2306.05499, 2023.
- [4] A. Zou, Z. Wang, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," arXiv:2307.15043, 2023.
- [5] B. Roziere et al., "Code Llama: Open foundation models for code," arXiv:2308.12950, 2023.
- [6] D. Guo et al., "DeepSeek-Coder: When the large language model meets programming," arXiv:2401.14196, 2024.
- [7] MITRE, "Common Weakness Enumeration (CWE)." [Online]. Available: <https://cwe.mitre.org>
- [8] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do users write more insecure code with AI assistants?," in Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS), 2023, pp. 2785-2799.