Vulnerability Trends Across Open-Source AI-Attributed Code

Clarisa Caballero-Ignacio✉
Oregon State University
Corvallis, USA
clarisa.caballero@oregonstate.edu

Lucas Stephens
Oregon State University
Corvallis, USA
stephluc@oregonstate.edu

Josiah Sage
Oregon State University
Corvallis, USA
Sagejo@oregonstate.edu

Manish Motwani
Oregon State University
Corvallis, USA
manish.motwani@oregonstate.edu

Zane Ma
Oregon State University
Corvallis, USA
zane.ma@oregonstate.edu

Abstract

Recent advances in large language models (LLMs) have led to widespread adoption of AI-coding tools and increasing deployment of AI-assisted code. While AI's potential for programming productivity is becoming evident, the security of AI-attributed code is still poorly understood. In this work, we collect the largest set of in-the-wild AI-attributed code to date, examining 253,390 files containing AI-assisted code from seven tools and 9,544 human-written files from public GitHub repositories. Our analyses reveal significant differences in security-relevant CWE prevalence across programming languages and AI tools, with tool-specific differences varying by language. Additionally, human-written Python exhibited significantly higher security-relevant CWE prevalence than AI-attributed Python (21.08% vs. 11.15%) with higher CVSS severity. Ultimately, our results extend prior work showing that vulnerable AI-assisted code is making it into public repositories, indicating a need for stronger review practices and greater developer security awareness when using these tools.

CCS Concepts

• **Security and privacy → Software security engineering**.

Keywords

AI Code Generation, Static Analysis, Security

ACM Reference Format:
Clarisa Caballero-Ignacio, Lucas Stephens, Josiah Sage, Manish Motwani, and Zane Ma. 2026. Vulnerability Trends Across Open-Source AI-Attributed Code. In *Proceedings of the 2nd Workshop on Explainable and Reliable Software Systems (EXPRESS '26), October 4–9, 2026, Oakland, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3842650.3843174

1 Introduction

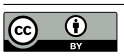
The advancement of large language models (LLMs) in recent years has led to an increase in generative artificial intelligence (Gen-AI) tools across many domains, including generating programming code. The AI tools that generate code are rapidly transforming software development. For example, a 2022 study reported that developers completed coding tasks up to 55% faster when using GitHub Copilot [16]. However, despite these productivity improvements and widespread adoption of AI coding tools, concerns have been raised about the reliability and security of AI-assisted code.

Recent studies have analyzed AI-generated code snippets, with Fu et al. identifying security weaknesses in 29.5% of Python and 24.2% of JavaScript [3]. Similarly, Armstrong et al. (2025) found that 30.6% of Copilot-generated code snippets contained at least one vulnerability [1]. These studies highlight a consistent pattern of vulnerabilities in AI-generated code, including a 2023 paper suggesting that ChatGPT is aware of potential vulnerabilities, but often generates source code that is not robust to certain attacks [4]. These findings demonstrate that AI-generated code introduces security risks.

Despite the growing adoption of AI coding assistants, there remains limited empirical evidence regarding the prevalence and severity of vulnerabilities in AI-generated code. Existing studies often focus on specific models, short observation periods, or *ex situ* lab environments.

To the best of our knowledge, the only study that has examined AI model security trends in real-world AI-assisted code is Schreiber et al. (2025). Given significant advances in AI tools since their 2024 data collection [13], we investigate whether AI-assisted code produced by more modern models is equally susceptible to security vulnerabilities. We expand on Schreiber et al. (2025) to cover a broader (253K files versus 7.7K files) and more current set of AI-coding models, and perform statistical analyses on the severity of CVSS scores. We analyze human-written and AI-attributed code from public GitHub repositories associated with AI tools: Generic AI, ChatGPT, Claude, Gemini, Cursor, GitHub Copilot, Amazon CodeWhisperer, and Tabnine. Using CodeQL static analysis for JavaScript, Python, and TypeScript, we evaluate whether vulnerabilities produced by different AI models vary in frequency and severity, as measured through Common Weakness Enumeration (CWE), Common Vulnerabilities and Exposures (CVE), and Common Vulnerability Scoring System (CVSS) metrics [8, 10, 11].

In our comprehensive analysis, we overcome several challenges, including accurately identifying AI-attributed repositories, which requires collecting and validating large volumes of code across multiple AI tools, programming languages, and time periods. In addition, repository discovery is constrained by platform search limitations and API restrictions, making large-scale data collection difficult. The rapid evolution of AI coding assistants can have



This work is licensed under a Creative Commons Attribution 4.0 International License.
EXPRESS '26, Oakland, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2968-3/2026/10
https://doi.org/10.1145/3842650.3843174

an impact on analysis, as vulnerability patterns and development practices can change over time.

We investigate the following research questions:

- **RQ1:** How does the prevalence of security-relevant CWEs in AI-attributed code compare across different AI tools and programming languages in public real-world repositories?
- **RQ2:** How does the security-relevant CWE prevalence in AI-attributed code compare to human-written code, and how do the severity profiles differ?
- **RQ3:** How has CWE prevalence in AI-attributed code evolved over time as AI tools have matured and adoption has grown?
- **RQ4:** What contextual factors, such as repository popularity and committer background, are associated with differences in CWE prevalence in AI-attributed code?

The remainder of this paper is organized as follows: Section 2 presents the background and related work; Section 3 describes the methodology, data collection, and analysis performed; Section 4 presents the results; Section 5 discusses the findings; Section 6 concludes the paper. The open-source repository collection and analysis framework used in this study is publicly available at https://anonymous.4open.science/r/vulnerability-trends-ai-code-C2ED.

2 Background and Related Work

One major use of large language models (LLMs) is the generation of source code in various programming languages. In this work, we use the term *AI-attributed* to describe the files in our dataset that contain *AI-assisted* code. AI-attributed refers to AI-generated and AI-assisted code. We use the term *AI-generated* to describe code that is known to have been generated entirely by AI.

Prior research investigating the security of AI-generated code has primarily focused on GitHub Copilot or a small subset of models [1, 3, 6, 12, 13]. Additionally, early studies have evaluated code generated in controlled environments rather than code generated in the wild [6, 12]. Across these studies, vulnerability rates varied, with Pearce et al. finding approximately 40% of Copilot-generated programs to be vulnerable and Majdinasab et al. reporting a lower rate of approximately 27.25% in a subsequent replication [6, 12]. Recent studies have begun to evaluate security vulnerabilities in AI-generated code in the wild [5, 7, 14]. Mao et al. (2026) examined code-level properties for AI-generated code in the wild, including complexity, structural characteristics, defect-related indicators, and commit-level characteristics. Their findings showed that the AI-generated JavaScript code had the highest number of CodeQL high-risk alerts per thousand lines of code (KLOC), while TypeScript demonstrated the lowest [7].

The dataset from Mao et al. (2026) includes several CWEs (e.g., CWE-20, CWE-400) that MITRE classifies as "discouraged" and unsuitable for mapping to real-world vulnerabilities [9]. In contrast, our analyses exclude CWEs classified as "discouraged" and those associated with < 10 CVSS observations to address sample size concerns with the CVSS severity scores. The CWEs that meet these conditions are referred to as *security-relevant*. Additionally, Mao et al. normalized the CodeQL metrics per KLOC to compare the vulnerability density between AI-generated and human-written code [7]. However, given that AI-generated code is more verbose, the vulnerabilities per KLOC may underestimate the vulnerability rate for equivalent AI-generated and human-written functionality. In contrast, our analysis measures CWE prevalence at the file level and examines security-relevant CWEs across AI tools and programming languages, while incorporating CVSS scores to evaluate vulnerability severity.

Compared to previous work analyzing vulnerabilities in AI-generated and human-written code, our study provides a large-scale empirical analysis using the largest dataset collected to date, covering the greatest diversity of AI coding tools and supporting our findings with statistical analyses.

3 Methodology

3.1 Data Collection

We collected AI-attributed and human-written files from public GitHub repositories. The AI-attributed dataset contains a diverse set of coding assistants, including Generic AI, ChatGPT, Claude, Gemini, GitHub Copilot, Amazon CodeWhisperer, Cursor, and Tabnine. These AI tools were selected to provide a representation of widely used AI-attributed code. The AI-attributed dataset includes repositories with last commit dates ranging from June 29, 2021 to February 2026.

A major challenge in this work is the reliable identification of AI-generated code. When querying GitHub, we deployed a pipeline similar to prior work by combining the following prefixes with tool names: "by", "with", "use", "used", "using", and "from" [3, 13]. Our approach improves on Schreiber et al. (2025) by including additional attribution phrases and AI models. We searched for code generally attributed to "AI" using the same prefixes except for "use" due to false positives, and included terms "vibe coded", "vibecoded", and "vibe-coded". We employed an iterative approach to search byte ranges instead of manual selection. After finding all relevant repositories, we performed a secondary query to gather additional metadata on the files. These changes allowed for a larger and richer dataset.

In our systematic approach, we applied six main filters to the data collected: (1) programming language, (2) removal of invalid file extensions, (3) file size between 150–384,000 bytes, (4) successful download and readability, (5) presence of the search keyword in a comment, and (6) duplicate removal. During the filtering stage, we categorized any general "AI" attributed files as Generic AI. The file counts after filtering AI tools and programming languages are shown in Table 1.

3.2 Human-Written Dataset

For comparison with AI-attributed files, we collected 9,544 human-written Python files. We utilized the GitHub Code Search API, limiting results to only Python files. To ensure that files are not assisted by AI, we only analyzed files with a commit date prior to the release of GitHub Copilot, June 29, 2021 [2]. This process is extremely challenging to perform with GitHub's API due to the lack of a date filter during the search. In order to ensure files are in the allotted date range, the repositories returned must be queried with GraphQL to return the commit date before file download. This restriction limits the human-written dataset in Python files, resulting in a dataset that is significantly smaller than the AI-attributed files. Human-written files have a size distribution similar to that of

Table 1: Distribution of AI tools and languages.

AI Tool	Pre-filter	Post-filter	%
Generic AI	359,028	118,873	46.91%
Claude	77,025	48,153	19.00%
Gemini	81,005	39,929	15.76%
ChatGPT	36,594	24,375	9.62%
Cursor	60,003	19,323	7.63%
GitHub Copilot	3,850	2,666	1.05%
Amazon CodeWhisperer	98	53	0.02%
Tabnine	51	18	0.01%
TOTAL	617,654	253,390	100.00%

Language	Pre-filter	Post-filter	%
Python	309,039	137,973	54.45%
TypeScript	141,807	68,346	26.97%
JavaScript	166,808	47,071	18.58%
TOTAL	617,654	253,390	100.00%

the AI-attributed Python files for comparison in our analysis. The files were processed through the same enrichment workflow and CodeQL analysis pipeline as the AI-attributed files.

3.3 Static Code Analysis

We performed static analysis using GitHub's CodeQL for its multi-language support and prevalence in the field. We analyzed complete source code files rather than sections with AI-attributed comments. The placement of attribution may not indicate the extent of AI assistance within a file. We observed cases where a comment referenced a single function despite placement at the beginning of the file, and cases where a comment located at the end indicated that AI assistance applied to the entire file. Extracting snippets and analyzing KLOC have inherent limitations. Code snippets are constrained by code length, limiting the ability to detect all vulnerabilities, while KLOC-based analysis may under-report vulnerabilities due to differences in function length and greater verbosity of AI-generated code relative to human-written code [7]. We mitigate these limitations by analyzing the complete source code of each file.

We map observed CWEs to associated CVEs and their corresponding CVSS scores to identify the potential severity of vulnerabilities present in the code. A key limitation of this analysis is the vulnerability mappings. CodeQL does not map all detected vulnerabilities to a CWE identifier, not all CWEs correspond to a documented CVE, and not all CVEs are assigned a CVSS score [11]. The CVSS v3.x severity scores are: None (0.0), Low (0.1-3.9), Medium (4.0-6.9), High (7.0-8.9), and Critical (9.0-10.0) [10].

Manual inspection revealed that some CWE severity scores were calculated from a single CVSS score, where the severity rating associated with that CWE was determined by a single report. Additionally, several tracked CWEs are classified as "discouraged" by MITRE and are not recommended for mapping or tracking real-world vulnerabilities. As a result, our analysis incorporates two restrictions. We exclude CWEs classified as "discouraged," and CWEs associated with fewer than 10 CVSS observations. With these filters, we narrowed the results to more security-relevant CWEs. In the analyses that follow, we indicate whether the results are based on all reported CWEs or only security-relevant CWEs.

4 Results

4.1 Prevalence Variations from CodeQL Analysis

A total of 253,390 AI-attributed files were analyzed using CodeQL, with 794,484 findings reported. Per language, the dataset contains TypeScript (84,355), JavaScript (151,917), and Python (558,212) findings. Of the total findings, 406,803 (51.20%) were mapped to CWEs. Overall, 77,619 of the analyzed files contained at least one CWE, representing 30.63% of the AI-attributed dataset. The language-to-tool breakdown is presented in Table 2, including file counts and prevalence percentages for files with ≥1 CWE and files with ≥1 security-relevant CWE.

Across AI tools, CWE prevalence by language followed a consistent pattern. Python showed the highest prevalence of files containing ≥1 CWE, while JavaScript showed the highest prevalence of security-relevant CWEs. TypeScript showed the lowest prevalence for both ≥1 CWE and security-relevant CWEs. However, the sample sizes for Amazon CodeWhisperer and Tabnine are so small that any conclusions drawn from these results should be interpreted with caution.

A total of 9,544 human-written Python files were analyzed, with 103,074 CodeQL findings retained. Overall, 5,824 of the analyzed

Table 2: Prevalence of files containing ≥ 1 CWE by AI tool and programming language, including security-relevant CWE counts (CVSS observation count ≥ 10; discouraged CWEs excluded).

AI Tool	Language	Files	Files w/ ≥ 1 CWE	Files w/ ≥ 1 Sec.–Rel. CWE
Generic AI	Python	52,721	24,440 (46.36%)	6,603 (12.52%)
	JavaScript	26,026	6,873 (26.41%)	4,041 (15.53%)
	TypeScript	40,126	4,691 (11.69%)	2,173 (5.42%)
ChatGPT	Python	15,937	6,162 (38.66%)	1,393 (8.74%)
	JavaScript	6,167	1,356 (21.99%)	727 (11.79%)
	TypeScript	2,271	104 (4.58%)	40 (1.76%)
Claude	Python	30,135	10,572 (35.08%)	2,044 (6.78%)
	JavaScript	5,008	1,343 (26.82%)	856 (17.09%)
	TypeScript	13,010	1,187 (9.12%)	704 (5.41%)
Gemini	Python	29,468	12,859 (43.64%)	4,644 (15.76%)
	JavaScript	4,561	1,500 (32.89%)	969 (21.25%)
	TypeScript	5,900	631 (10.69%)	347 (5.88%)
Cursor	Python	8,016	3,354 (41.84%)	615 (7.67%)
	JavaScript	4,872	1,349 (27.69%)	699 (14.35%)
	TypeScript	6,435	625 (9.71%)	201 (3.12%)
GitHub Copilot	Python	1,677	487 (29.04%)	79 (4.71%)
	JavaScript	426	69 (16.20%)	42 (9.86%)
	TypeScript	563	0 (0.00%)	0 (0.00%)
Amazon CodeWhisperer	Python	11	6 (54.55%)	0 (0.00%)
	JavaScript	3	1 (33.33%)	0 (0.00%)
	TypeScript	39	0 (0.00%)	0 (0.00%)
Tabnine	Python	8	5 (62.50%)	1 (12.50%)
	JavaScript	8	3 (37.50%)	2 (25.00%)
	TypeScript	2	2 (100.00%)	0 (0.00%)
Overall		**253,390**	**77,619 (30.63%)**	**26,180 (10.33%)**

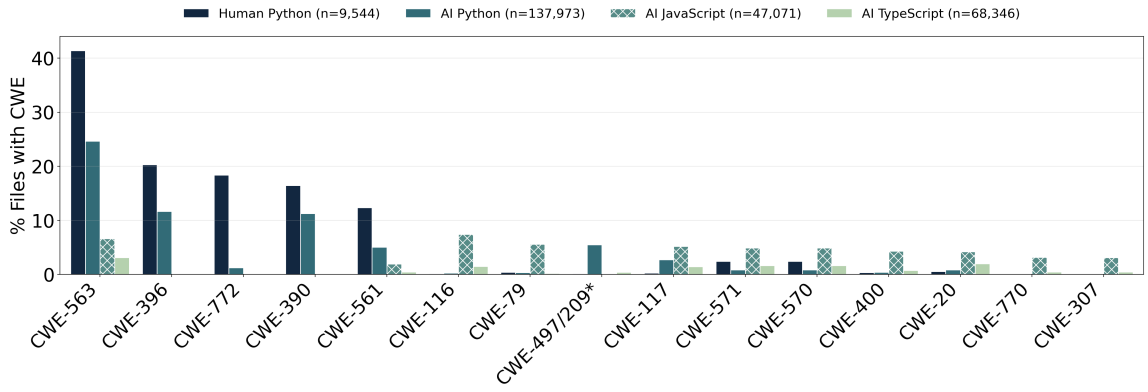


Figure 1: Top fifteen CWEs in human-written and AI-attributed code.

files contained at least one CWE, representing 61.02% of the human-written Python dataset. The fifteen most prevalent CWEs in human-written and AI-attributed code are shown in Figure 1; these represent general code weaknesses and are not restricted to the security-relevant CWE subset.

4.1.1 Fisher's Exact Test: AI Tools. Security-relevant CWE prevalence was compared between AI tools within each language using pairwise Fisher's exact tests. We applied the Bonferroni correction across 28 pairwise comparisons within each language. Significant results with $p_{\text{Bonf}} < 0.05$ are reported in Appendix C, excluding TypeScript comparisons involving GitHub Copilot because it had zero security-relevant CWE prevalence.

Thirty-one AI tool comparisons were significant after Bonferroni correction. JavaScript exhibited the highest overall security-relevant CWE prevalence across AI tools (9.86%–21.25%), with eleven significant pairwise comparisons. Python showed thirteen significant comparisons, with security-relevant CWE prevalence ranging from 4.71% to 15.76%, while TypeScript exhibited the lowest security-relevant CWE prevalence (1.76%–5.88%), with seven significant comparisons.

Among the significant comparisons, the largest prevalence difference in JavaScript was observed between Gemini (21.25%) and GitHub Copilot (9.86%). Python showed a similarly large difference between Gemini (15.76%) and GitHub Copilot (4.71%). In TypeScript, the largest difference reported was between Gemini (5.88%) and ChatGPT (1.76%). Although the differences involving GitHub Copilot were statistically significant, these results should be interpreted with caution due to their comparatively small sample size.

4.1.2 Fisher's Exact Test: Human-Written vs AI-attributed. Fisher's exact test with Bonferroni correction was used to determine whether the prevalence of security-relevant CWEs differed significantly between human-written and AI-attributed Python files. Fisher's exact test showed that AI-attributed Python files had lower odds of containing at least one security-relevant CWE than human-written Python files (OR=0.47, Bonferroni-adjusted $p_{\text{Bonf}} < 0.001$). Equivalently, the odds of a security-relevant CWE were approximately 2.13 times higher in human-written Python than in AI-attributed Python.

4.1.3 Temporal Trend. To examine temporal trends in CWE prevalence, we plotted the prevalence of CWEs in AI-attributed files at six-month intervals, as shown in Figure 2. The prevalence initially increased then declined between July 2021 and October 2022, following the release of GitHub Copilot. However, the relatively low attribution rates during this period may have contributed to the observed pattern as adoption increased and stabilized. The prevalence of both CWEs (42.8%) and security-relevant CWEs (14.4%) peaked in January 2022 and reached their lowest levels in January 2024, at 27.1% and 7.9%, respectively.

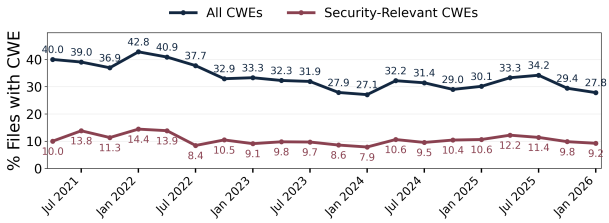


Figure 2: AI-attributed CWE prevalence timeline by last commit date.

4.1.4 Repository Popularity. We used repository star count as a proxy for project popularity and compared security-relevant CWE prevalence across repositories with different star counts to explore how security-relevant CWE prevalence varied across repository star counts. We observed a general decline in security-relevant CWE prevalence for AI-attributed Python and TypeScript files as the number of stars increased (Figure 3). JavaScript exhibited a different pattern, with relatively high CWE prevalence that fluctuated across AI tools. In contrast, human-written Python files showed the highest and relatively stable CWE prevalence.

4.1.5 Committer Email Domain. We collected the email domains associated with commit authors to characterize broad contributor backgrounds. We segmented these domains into six categories: (1) *Edu* includes domains ending with `*.edu` or international suffixes such as `.ac.uk`. (2) *Local Dev* includes domains such as `*.local` or `*.lan`, which typically represent development machine hostnames

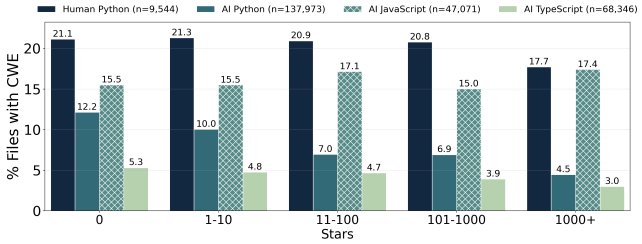


Figure 3: Security-relevant CWE prevalence by repository star count.

rather than public emails. (3) *Anonymous* includes emails such as users.noreply.github.com. (4) *Personal* includes domains identified from a manually curated list of 70 recognized email providers, such as `gmail.com` or `mail.ru`. (5) *Corporate Tech* includes domains associated with 48 well-known technology companies. (6) *Other* includes domains that could not be assigned to another category. This methodology is not exhaustive and some domains may be misclassified as *Other*.

The relationships between committer domains and security-relevant CWE prevalence are shown in Figure 4. Files with missing or unrecognized committer email domains were excluded, resulting in 9,257 human-written Python files, 136,473 AI-attributed Python files, 46,730 JavaScript files, and 67,706 TypeScript files. Overall, *Local Dev* email domains showed the highest security-relevant CWE prevalence in human-written Python and JavaScript code, while *Corporate Tech* showed the lowest across all groups. *Local Dev* domains are typically associated with local development or testing rather than publicly identifiable email domains. The higher prevalence observed in *Local Dev* may reflect the development or testing contexts associated with these domains.

Pairwise Fisher's exact tests with Bonferroni correction were performed to compare security-relevant CWE prevalence across email domains within each programming language. AI-attributed Python files showed the greatest variation across email domains, as shown in Table 3. *Corporate Tech* exhibited the lowest security-relevant CWE prevalence (7.60%), while Edu had the highest (11.82%). These results showed differences in security-relevant CWE prevalence across committer email domains in AI-attributed code.

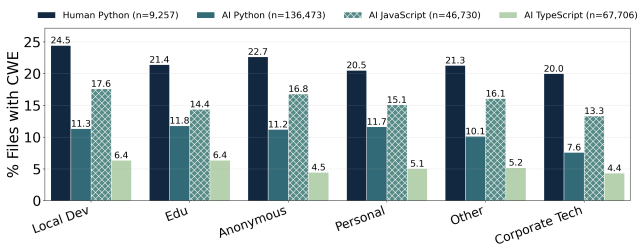


Figure 4: Security-relevant CWE prevalence by committer email domain category.

Table 3: Fisher's exact tests across email domain categories. Statistically significant comparisons with $p_{\text{Bonf}} < 0.05$ are shown.

Group A	Group B	Security-Relevant Prevalence (%)	Bonferroni p
AI Python			
Personal	Corporate Tech	11.65 vs 7.60	8.41×10^{-14}
Anonymous	Corporate Tech	11.23 vs 7.60	2.07×10^{-10}
Edu	**Corporate Tech**	**11.82 vs 7.60**	$\mathbf{3.64 \times 10^{-10}}$
Local Dev	Corporate Tech	11.34 vs 7.60	8.36×10^{-7}
Other	Corporate Tech	10.14 vs 7.60	1.26×10^{-5}
Personal	Other	11.65 vs 10.14	1.18×10^{-10}
Anonymous	Other	11.23 vs 10.14	1.23×10^{-3}
Edu	Other	11.82 vs 10.14	2.42×10^{-3}
AI JavaScript			
Anonymous	Personal	16.80 vs 15.12	7.04×10^{-3}
AI TypeScript			
Local Dev	Anonymous	6.37 vs 4.48	1.17×10^{-2}

4.2 Logistic Regression

We modeled whether each code sample contained at least one security-relevant CWE using a binomial generalized linear model with a logit link, i.e., a logistic regression. Programming language and AI tool were treated as categorical fixed effects, and the full model also included a language-by-AI-tool interaction:

$$\text{logit}[P(Y = 1)] = \beta_0 + \beta_{\text{language}} + \beta_{\text{tool}} + \beta_{\text{language}\times\text{tool}}$$

Because some categories produced sparse or perfectly separated language–tool cells, we excluded languages or AI tools that failed minimum support criteria before fitting the model. Specifically, we only retained categories with sufficient total observations and positive/negative outcomes, and retained language–tool cells were required to contain at least five samples and at least one observation of each outcome. Three languages (JavaScript, Python, and TypeScript) and five AI tools (ChatGPT, Claude, Cursor, Generic AI, and Gemini) satisfied the filtering criteria. We compared the interaction model with a reduced additive model using a likelihood-ratio test and similarly assessed the overall contributions of programming language and AI tool by comparing nested reduced models. For interpretation, fitted log-odds were transformed into predicted probabilities for each retained language–tool combination, with 95% confidence intervals. The full logistic regression model, including programming language, AI tool, and their interaction, provided a significantly better fit than an intercept-only model, $\chi^2(14) = 5629.39$, $p < 0.001$, and converged successfully. The interaction model also improved fit over the additive model, $\chi^2(8) = 441.87$, $p < 0.01$, indicating that differences in security-relevant CWE occurrence across AI tools varied by programming language.

The logistic regression results (Figure 5) indicate that Gemini models have significantly higher rates of security-relevant CWEs than other models across JavaScript, Python, and TypeScript. Other models had mixed results across languages. ChatGPT had the lowest JavaScript and TypeScript rates, while Claude had the lowest Python

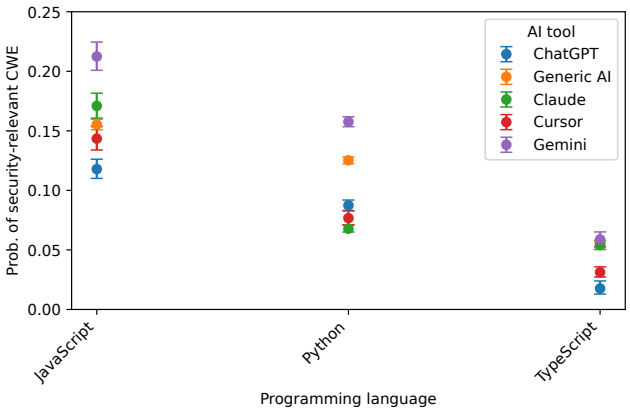


Figure 5: Logistic regression predictions with 95% confidence intervals.

rates. Generic AI was neither the best nor the worst model for any of the languages, which would align with a combination of the specifically attributed models.

4.3 CVSS Severity Analysis

Spearman correlation analysis was performed to evaluate the relationship between the prevalence of individual security-relevant CWEs and their CVSS severity scores. When CWE observations were pooled across all AI tools, weak negative correlations were observed between CWE prevalence and both mean CVSS scores ($\rho = -0.19$, $p = 0.001$) and median CVSS scores ($\rho = -0.22$, $p < 0.001$). Among AI-attributed languages, Python showed a significant moderate negative correlation between CWE prevalence and median CVSS scores ($\rho = -0.48$, $p = 0.013$), while the correlation with mean CVSS scores was not statistically significant.

Human-written Python showed significant moderate negative correlations between CWE prevalence and both mean CVSS scores ($\rho = -0.45$, $p = 0.025$) and median CVSS scores ($\rho = -0.47$, $p = 0.019$). These results indicate that within Python, more prevalent security-relevant CWE types tended to have lower CVSS severity scores in both AI-attributed and human-written code. The most prevalent security-relevant CWEs and their CVSS scores by language are reported in Table 6.

5 Discussion

5.1 Key Findings

5.1.1 ***RQ1*** *AI Tools and Programming Languages.* Pairwise Fisher's exact tests showed significant differences in the security-relevant CWE prevalence among AI tools within each programming language. Gemini showed the greatest pairwise prevalence difference across JavaScript, Python, and TypeScript. Similarly, our logistic regression results indicate that Gemini had the highest predicted probability of a security-relevant CWE across all programming languages, whereas the occurrence of security-relevant CWEs among the other AI tools varied by programming language.

The three most prevalent security-relevant weaknesses in AI-attributed Python are CWE-497/209 (exposure of sensitive system information and generation of error messages containing sensitive information), CWE-312 (cleartext storage of sensitive information), and CWE-532 (insertion of sensitive information into log files). Notably, the three most prevalent security-relevant CWE reporting groups in AI-attributed Python are the same as those observed for Gemini (Tables 5 and 6). These weaknesses primarily involve information exposure or insecure handling of sensitive data and have median CVSS scores between 5.3 and 6.5. While these weaknesses are relatively common, they are generally less severe than vulnerabilities that directly compromise a system.

In contrast, higher-impact injection weaknesses, including SQL injection (CWE-89), OS command injection (CWE-78), and Code injection (CWE-94), occur less frequently than these highly prevalent information-exposure and sensitive-data-handling weaknesses. This pattern suggests that frequently observed weaknesses may not be associated with the highest severity. A similar pattern is visible in JavaScript and TypeScript, where CWE-116 is among the most prevalent security-relevant weaknesses despite its moderate median CVSS score of 6.5.

These findings indicate that AI tools and programming languages are associated not only with differences in prevalence of security-relevant CWEs, but also with the type of CWE and the severity of weaknesses observed.

5.1.2 ***RQ2*** *Human-Written vs AI-attributed.* To assess differences between AI-attributed and human-written code, we compare CWE prevalence within Python files. We find that 41.95% of AI-attributed Python files contain ≥ 1 CWE, compared with 61.02% of human-written Python files. Human-written Python files also exhibited significantly higher security-relevant CWE prevalence than AI-attributed files (21.08% vs. 11.15%). Additionally, human-written Python had a higher median file-level CVSS score (6.66) than AI-attributed Python files (6.14), as shown in Table 4. These CVSS scores represent file-level severity among files containing security-relevant CWEs. Thus, prevalence captures how frequently security-relevant weaknesses occur, while CVSS captures the severity of weaknesses present in affected files. These results suggest that security-relevant CWE prevalence and CVSS severity capture distinct aspects of software security.

Fisher's exact test was used to determine whether the prevalence of security-relevant CWEs differed significantly between human-written and AI-attributed Python files. AI-attributed Python code had significantly lower odds of containing at least one security-relevant CWE than human-written Python (OR=0.47, $p_{\text{Bonf}} < 0.001$). In particular, CWE-772 (missing release of resource after effective lifetime) was highly prevalent in the human-written dataset, occurring in 30.1% of human-written Python files containing at least one CWE. The mean and median CVSS scores for CWE-772 are 6.66 and 6.50, respectively, corresponding to the medium severity range (4.0-6.9) as shown in the footnotes of Table 6. These findings suggest that AI-attributed and human-written Python code differ in both the prevalence and profile of security-relevant weaknesses.

The Spearman correlation analysis provides additional insight into the relationship between the prevalence and severity of individual CWE types. At the file level, human-written Python exhibited higher overall security-relevant CWE prevalence and higher median file-level CVSS severity than AI-attributed Python. At the

Table 4: Security-relevant CWE prevalence and CVSS severity by file type.

File Type	File Count (All)	File Count (≥ 1 CWE)	Security-Relevant File Count	Security-Relevant Prevalence %		CVSS Score	
				All Files	≥ 1 CWE Files	Mean	Median
Human Python	9,544	5,824	2,012	21.08%	34.55%	6.66	6.66
AI Python	137,973	57,885	15,379	11.15%	26.57%	6.23	6.14
AI JavaScript	47,071	12,494	7,336	15.58%	58.72%	6.74	6.69
AI TypeScript	68,346	7,240	3,465	5.07%	47.86%	6.70	6.69

Table 5: Most prevalent CWEs by AI tool (CVSS v3.x observation count ≥ 10; discouraged CWEs excluded).

AI Tool	CWE #	Files w/ CWE	AI Tool	CWE #	Files w/ CWE
	497/209*	3,986 (11.1%)		497/209*	2,555 (17.0%)
	116	2,941 (8.2%)		312	1,852 (12.4%)
Generic AI	312	1,985 (5.5%)	Gemini	532	1,729 (11.5%)
	79	1,915 (5.3%)		22	814 (5.4%)
	532	1,627 (4.5%)		23+73	811 (5.4%)
	772	645 (8.5%)		116	549 (10.3%)
	116	415 (5.4%)		79	433 (8.1%)
ChatGPT	312	397 (5.2%)	Cursor	312	195 (3.7%)
	79	381 (5.0%)		772	174 (3.3%)
	497/209*	317 (4.2%)		497/209*	173 (3.2%)
	497/209*	864 (6.6%)		772	23 (4.1%)
	312	819 (6.3%)		312	22 (4.0%)
Claude	532	734 (5.6%)	GitHub	532	19 (3.4%)
	116	504 (3.8%)	Copilot	497/209*	18 (3.2%)
	367	387 (3.0%)		116	17 (3.1%)

*CWE-497 and CWE-209 were combined due to identical file counts and their shared association with the exposure of sensitive information.
A "+" denotes CWEs tied at the same prevalence.
CWE Names:
CWE-22: Improper Limitation of a Pathname to a Restricted Directory
CWE-23: Relative Path Traversal
CWE-73: External Control of File Name or Path
CWE-79: Improper Neutralization of Input During Web Page Generation
CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page
CWE-116: Improper Encoding or Escaping of Output
CWE-209: Generation of Error Message Containing Sensitive Information
CWE-307: Improper Restriction of Excessive Authentication Attempts
CWE-312: Cleartext Storage of Sensitive Information
CWE-327: Use of a Broken or Risky Cryptographic Algorithm
CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)
CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition
CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere
CWE-532: Insertion of Sensitive Information into Log File
CWE-770: Allocation of Resources Without Limits or Throttling
CWE-772: Missing Release of Resource after Effective Lifetime

CWE-type level, however, more prevalent security-relevant CWE types were associated with lower CVSS severity scores in both datasets. This suggests that the CWEs observed most frequently within Python are not necessarily those with the highest severity. For example, in AI-attributed Python, CWE-497/209 was the most prevalent security-relevant CWE reporting group (13.1%) but had a median CVSS score of 5.30, whereas less prevalent CWE-22 (3.8%) and CWE-23 and CWE-73 (each 3.7%) had median CVSS scores of 7.50. The similar relationship observed in both datasets suggests that this is not unique to AI-attributed Python.

Table 6: Most prevalent CWEs by file type (CVSS v3.x observation count ≥ 10; discouraged CWEs excluded).

File Type	CWE #	Files w/ CWE	File Type	CWE #	Files w/ CWE
	116	3,494 (28.0%)		497/209*	7,585 (13.1%)
	79	2,636 (21.1%)		312	4,105 (7.1%)
JavaScript	770	1,490 (11.9%)	Python	532	3,861 (6.7%)
	307	1,482 (11.9%)		22	2,189 (3.8%)
	80	1,064 (8.5%)		23+73	2,142 (3.7%)
	116	1,030 (14.2%)		772	1,752 (30.1%)
	80	814 (11.2%)		312	96 (1.6%)
TypeScript	312	435 (6.0%)	Human	327	76 (1.3%)
	338	409 (5.6%)	Python	532	71 (1.2%)
	367	405 (5.6%)		22	64 (1.1%)

*CWE-497 and CWE-209 were combined due to identical file counts and their shared association with the exposure of sensitive information.
A "+" denotes CWEs tied at the same prevalence.
CVSS mean/median: CWE-22: 7.23/7.50; CWE-23: 7.47/7.50; CWE-73: 7.62/7.50; CWE-79: 5.76/6.10; CWE-80: 5.23/5.40; CWE-116: 6.77/6.50; CWE-307: 7.64/7.50; CWE-312: 6.37/6.50; CWE-327: 6.87/7.50; CWE-338: 7.24/7.50; CWE-367: 6.69/7.00; CWE-497/209: 5.57/5.30; CWE-532: 5.90/5.50; CWE-770: 6.57/6.50; CWE-772: 6.66/6.50.

5.1.3 ***RQ3*** *Temporal Trends in CWE Prevalence and AI Tool Adoption.* We observe a general decline in overall AI-attributed CWE prevalence in Figure 2, decreasing from approximately 40% in 2021 to 27% in early 2026. In contrast, security-relevant CWE prevalence fluctuated between 7.9% and 14.4%. Despite some variability, the observed decrease may suggest improvements in AI-generated code, particularly through reductions in more common CWEs, increased developer proficiency in identifying and preventing CWEs, or a combination of both. These findings coincide with studies identifying a decrease in vulnerability with model updates [6].

Our CWE prevalence findings align with those of Schreiber et al. (2025), while providing additional insight into the observed patterns. Their CWE prevalence of 12.1% is numerically similar to our security-relevant CWE prevalence of 10.33% [13]. There are a few factors to consider when comparing the CWE prevalence, including the increase in AI-assisted programming since their study, Generic AI attribution, and exclusion of "discouraged" CWEs to obtain a more accurate assessment of CWE severity.

Armstrong et al. reported that 30.6% of GitHub Copilot-generated code snippets contained at least one vulnerability, numerically similar to the 30.63% prevalence of files containing at least one CWE in our dataset [1]. However, their analysis was limited to GitHub Copilot, whereas our dataset spans multiple AI tools. In addition, they found a higher proportion of vulnerable LOC in JavaScript than in Python, while we observed a similar pattern in our security-relevant CWE prevalence results (Table 4). Notably, their unique

CWE count was substantially lower at 18 compared to 108 [1]. Although prior studies have used smaller datasets and evaluated fewer AI models, several of their findings show trends consistent with ours. Our broader dataset enables a more comprehensive comparison of vulnerability prevalence and severity across AI tools and programming languages.

5.1.4 ***RQ4*** *Repository Popularity and Committer Email Domains.* The repository star count and committer analyses suggest that security-relevant CWE prevalence in AI-attributed code may be associated with repository popularity and committer characteristics. Across most languages, security-relevant CWE prevalence generally decreased as repository popularity increased, as shown in Figure 3. These results suggest that popular repositories may have stronger practices for identifying or mitigating weaknesses in AI-attributed code, potentially because such code is more likely to be revisited and modified [7].

Across committer email domains, AI-attributed JavaScript exhibited higher security-relevant CWE prevalence than AI-attributed Python and TypeScript. Pairwise Fisher's exact tests with Bonferroni correction revealed that significant differences were concentrated in AI-attributed Python files, which showed the greatest variation across email domain groups. In particular, *Corporate Tech* domains consistently exhibited the lowest security-relevant CWE prevalence, while *Edu* domains exhibited the highest prevalence in Python. These differences may reflect variation in contributor populations, development contexts, or organizational practices, including greater student representation or higher rates of self-reporting in academia. Corporate contributors may also have greater access to organizational security tools and review practices. Although email domains provide a limited proxy for developer background, these findings are consistent with prior work showing that students using AI assistants often produce less secure code and exhibit lower security awareness, whereas stronger security awareness is associated with more secure outcomes [15].

5.2 Limitations

The dataset that we used has several potential biases. The first is reliance on self-reporting of AI attribution. Developers who self-report AI assistance or generation may be more mindful of code quality and security, potentially skewing vulnerability trends. We chose self-attribution as our measurement technique since it is intuitively reliable (excluding adversarial attribution) and can be applied at scale. Furthermore, automatic inference and attribution of AI-generated code remains an ongoing research problem and may be inherently difficult to resolve. The second dataset limitation is the limited scale and disjoint collection period for human-written code. To maximize the likelihood of human-written code, we restricted to commits made before the release of GitHub Copilot on June 29, 2021. Some variations we detect may be partially due to the temporal difference between our AI- and human-written datasets.

CWEs identify classes of software and hardware weaknesses that may reflect code quality or security risks, but they are not direct security indicators because they do not necessarily correspond to exploitable vulnerabilities. As a result, further analyses across CWE categories and using additional security-focused measures could provide greater insight into quality degradation most prevalent within code files.

6 Conclusion

In this study, we performed a comprehensive vulnerability analysis of AI-attributed files in the wild with the largest dataset to date. The security-relevant findings show that 10.33% of AI-attributed files contained at least one security-relevant CWE, providing a more focused measure of code weakness and potential vulnerability severity.

In conclusion, our findings show that human-written Python files exhibited significantly higher security-relevant CWE prevalence and higher file-level CVSS scores than AI-attributed Python files. Consistent with the prevalence difference, Fisher's exact test showed that AI-attributed files had significantly lower odds of containing at least one security-relevant CWE. In addition, CWE prevalence and CVSS severity capture distinct aspects of software security. Within both AI-attributed and human-written Python, more prevalent security-relevant CWE types were associated with lower CVSS severity scores, highlighting the importance of considering both the frequency and severity of security-relevant weaknesses.

Our logistic regression results indicate that Gemini had the highest predicted probability of a security-relevant CWE across JavaScript, Python, and TypeScript. However, differences in security-relevant CWEs across other AI tools varied by programming language. Fisher's exact tests showed that Gemini consistently had higher security-relevant CWE prevalence across programming languages. These findings suggest that, aside from Gemini's consistently higher prevalence, differences among AI tools may vary by programming language.

Increasing awareness of secure developer practices and strengthening security education may help reduce risk as the adoption of AI tools continues to grow. AI-generated code should not be assumed secure by default and should be reviewed for vulnerabilities and security risks. Additionally, integrating static analysis and automated vulnerability detection into AI-assisted development pipelines would help identify vulnerabilities before deployment.

7 Appendices

A Top Sixteen Most Prevalent CWEs in AI-attributed Code

CWE ID	CWE Name	File Count	% of ≥ 1 CWE CodeQL Files	% of All CodeQL Files	Count of CVSS Scores	CVSS Score	
						Mean	Median
CWE-563	Assignment to Variable without Use	39,281	50.61%	15.50%	0	*n/a*	*n/a*
CWE-396	Declaration of Catch for Generic Exception	16,079	20.72%	6.35%	1	5.90	5.90
CWE-390	Detection of Error Condition Without Action	15,551	20.04%	6.14%	0	*n/a*	*n/a*
CWE-561	Dead Code	8,166	10.52%	3.22%	0	*n/a*	*n/a*
CWE-209	Generation of Error Message Containing Sensitive Information	7,914	10.20%	3.12%	294	5.53	5.30
CWE-497	Exposure of Sensitive System Information to an Unauthorized Control Sphere	7,914	10.20%	3.12%	21	6.13	6.00
CWE-117	Improper Output Neutralization for Logs	7,224	9.31%	2.85%	3	6.03	5.00
CWE-312	Cleartext Storage of Sensitive Information	5,270	6.79%	2.08%	508	6.37	6.50
CWE-359	Exposure of Private Personal Information to an Unauthorized Actor	5,153	6.64%	2.03%	6	5.57	5.90
CWE-116	Improper Encoding or Escaping of Output	4,911	6.33%	1.94%	221	6.77	6.50
CWE-570	Expression is Always False	4,626	5.96%	1.83%	0	*n/a*	*n/a*
CWE-571	Expression is Always True	4,626	5.96%	1.83%	0	*n/a*	*n/a*
CWE-532	Insertion of Sensitive Information into Log File	4,566	5.88%	1.80%	661	5.90	5.50
CWE-20	Improper Input Validation	4,524	5.83%	1.79%	5,503	7.18	7.50
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	3,299	4.25%	1.30%	19,766	5.76	6.10
CWE-400	Uncontrolled Resource Consumption	3,137	4.04%	1.24%	1,126	6.69	7.50

This table provides mapping of CWE categories and their associated CVSS score.

B Top Three Security-Relevant CWEs by Language and AI Tool.

AI Tool	Rank 1 (Count %)	Rank 2 (Count %)	Rank 3 (Count %)
		Python	
Generic AI	CWE-497/209 (56.9%)	CWE-312 (20.6%)	CWE-532 (19.1%)
ChatGPT	CWE-772 (46.3%)	CWE-497/209 (21.9%)	CWE-312 (21.3%)
Claude	CWE-497/209 (40.0%)	CWE-312 (32.4%)	CWE-532 (30.5%)
Cursor	CWE-772 (28.3%)	CWE-497/209+CWE-312 (26.8%)	CWE-532 (25.0%)
Gemini	CWE-497/209 (54.3%)	CWE-312 (34.4%)	CWE-532 (33.2%)
GitHub Copilot	CWE-772 (29.1%)	CWE-312 (24.1%)	CWE-497/209+CWE-532 (22.8%)
		JavaScript	
Generic AI	CWE-116 (50.9%)	CWE-79 (37.5%)	CWE-770 (19.5%)
ChatGPT	CWE-116 (51.9%)	CWE-79 (45.4%)	CWE-307+CWE-770 (20.4%)
Claude	CWE-116 (27.5%)	CWE-367 (23.6%)	CWE-770 (22.9%)
Cursor	CWE-116 (67.0%)	CWE-79 (57.9%)	CWE-184 (16.5%)
Gemini	CWE-116 (35.2%)	CWE-307+CWE-770 (31.9%)	CWE-79 (26.9%)
GitHub Copilot	CWE-116 (35.7%)	CWE-79 (23.8%)	CWE-307+CWE-434+CWE-476+CWE-770 (14.3%)
		TypeScript	
Generic AI	CWE-116 (31.2%)	CWE-80 (23.9%)	CWE-338 (13.3%)
ChatGPT	CWE-307+CWE-770 (30.0%)	CWE-116 (17.5%)	CWE-367+CWE-79 (15.0%)
Claude	CWE-116 (28.6%)	CWE-367 (26.3%)	CWE-80 (24.6%)
Cursor	CWE-116 (33.8%)	CWE-80 (27.9%)	CWE-367 (11.4%)
Gemini	CWE-312 (32.6%)	CWE-532 (31.1%)	CWE-116 (21.9%)

*CWE-497 (Exposure of Sensitive System Information to an Unauthorized Control Sphere) & CWE-209 (Generation of Error Message Containing Sensitive Information) were combined due to identical file counts and their shared association with the exposure of sensitive information
A "+" indicates CWEs tied at the same rank.

C Fisher's Exact Tests Across AI Tools by Language.

Language	Tool A	Tool B	Security-Relevant Prevalence (%)	Bonferroni p
	Gemini	**GitHub Copilot**	**21.25 vs 9.86**	$\mathbf{6.60 \times 10^{-8}}$
	Gemini	Generic AI	21.25 vs 15.53	2.50×10^{-19}
	Cursor	Gemini	14.35 vs 21.25	4.98×10^{-17}
	ChatGPT	Generic AI	11.79 vs 15.53	9.27×10^{-13}
	ChatGPT	Gemini	11.79 vs 21.25	3.57×10^{-38}
JavaScript	ChatGPT	Cursor	11.79 vs 14.35	2.23×10^{-3}
	ChatGPT	Claude	11.79 vs 17.09	4.95×10^{-14}
	Claude	GitHub Copilot	17.09 vs 9.86	1.60×10^{-3}
	Claude	Gemini	17.09 vs 21.25	6.91×10^{-6}
	Claude	Cursor	17.09 vs 14.35	5.33×10^{-3}
	Generic AI	GitHub Copilot	15.53 vs 9.86	2.50×10^{-2}
	Gemini	**GitHub Copilot**	**15.76 vs 4.71**	$\mathbf{1.57 \times 10^{-42}}$
	Gemini	Generic AI	15.76 vs 12.52	3.23×10^{-36}
	Cursor	Generic AI	7.67 vs 12.52	5.15×10^{-38}
	Cursor	GitHub Copilot	7.67 vs 4.71	2.52×10^{-4}
	Cursor	Gemini	7.67 vs 15.76	5.53×10^{-84}
	ChatGPT	Generic AI	8.74 vs 12.52	1.40×10^{-39}
Python	ChatGPT	GitHub Copilot	8.74 vs 4.71	5.03×10^{-8}
	ChatGPT	Gemini	8.74 vs 15.76	2.32×10^{-102}
	ChatGPT	Claude	8.74 vs 6.78	1.53×10^{-12}
	Claude	Generic AI	6.78 vs 12.52	1.89×10^{-156}
	Claude	GitHub Copilot	6.78 vs 4.71	1.74×10^{-2}
	Claude	Gemini	6.78 vs 15.76	1.96×10^{-268}
	Generic AI	GitHub Copilot	12.52 vs 4.71	2.80×10^{-25}
	Gemini	**ChatGPT**	**5.88 vs 1.76**	$\mathbf{4.42 \times 10^{-16}}$
	Gemini	Cursor	5.88 vs 3.12	2.76×10^{-12}
	Cursor	Generic AI	3.12 vs 5.42	6.03×10^{-15}
TypeScript	Generic AI	ChatGPT	5.42 vs 1.76	1.63×10^{-16}
	ChatGPT	Cursor	1.76 vs 3.12	1.27×10^{-2}
	ChatGPT	Claude	1.76 vs 5.41	2.97×10^{-15}
	Claude	Cursor	5.41 vs 3.12	5.94×10^{-12}

Statistically significant comparisons with $p_{\text{Bonf}} < 0.05$ are shown.

References

[1] Nathan Armstrong and Thomas Hartley. 2025. Artificially Insecure: Examining GitHub Copilot's AI-based Vulnerability Prevention System. *2025 Silicon Valley Cybersecurity Conference (SVCC)* (2025), 1–6. doi:10.1109/SVCC65277.2025.11133629

[2] Nat Friedman. 2021. Introducing GitHub Copilot: your AI pair programmer. https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/

[3] Yujia Fu, Peng Liang, Amjed Tahir, Zengyang Li, Mojtaba Shahin, Jiaxin Yu, and Jinfu Chen. 2025. Security Weaknesses of Copilot-Generated Code in GitHub Projects: An Empirical Study. doi:10.48550/arXiv.2310.02059

[4] Raphaël Khoury, Anderson R. Avila, Jacob Brunelle, and Baba Mamadou Camara. 2023. How Secure is Code Generated by ChatGPT? *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (2023), 2445–2451. doi:10.1109/SMC53992.2023.10394237

[5] Yue Liu, Ratnadira Widyasari, Yanjie Zhao, Ivana Clairine Irsan, Junkai Chen, and David Lo. 2026. Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild. doi:10.48550/arXiv.2603.28592

[6] Vahid Majdinasab, Michael Joshua Bishop, Shawn Rasheed, Arghavan Moradidakhel, Amjed Tahir, and Foutse Khomh. 2023. Assessing the Security of GitHub Copilot Generated Code – A Targeted Replication Study. doi:10.48550/arXiv.2311.11177 arXiv:2311.11177 [cs.SE].

[7] Tianhao Mao, Dongfang Zhao, Haixu Tang, Xiaofeng Wang, and Hang Zhang. 2026. A Large-Scale Empirical Study of AI-Generated Code in Real-World Repositories. doi:10.48550/arXiv.2603.27130

[8] MITRE Corporation. 2024. CWE - About CWE. https://cwe.mitre.org/about/index.html

[9] MITRE Corporation. 2026. Common Weakness Enumeration (CWE). https://cwe.mitre.org/

[10] National Institute of Standards and Technology (NIST). 2024. NVD - Vulnerability Metrics. https://nvd.nist.gov/vuln-metrics/cvss

[11] National Institute of Standards and Technology (NIST). 2026. NVD - CVEs and the NVD Process. https://nvd.nist.gov/general/cve-process

[12] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2021. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. doi:10.48550/arXiv.2108.09293

[13] Maximilian Schreiber and Pascal Tippe. 2025. Security Vulnerabilities in AI-Generated Code: A Large-Scale Analysis of Public GitHub Repositories. Vol. 16219. 153–172. http://arxiv.org/abs/2510.26103

[14] Bin Wang, Wenjie Yu, Yilu Zhong, Hao Yu, Keke Lian, Chaohua Lu, Hongfang Zheng, Dong Zhang, and Hui Li. 2025. AI Code in the Wild: Measuring Security Risks and Ecosystem Shifts of AI-Generated Code in Modern Software. doi:10.48550/arXiv.2512.18567

[15] Jiawei Yuan and Yanyan Li. 2025. The Impact of Security Mindset on the Use of AI Assistants in Computing Education. *IEEE Transactions on Education* 68, 6 (Dec. 2025), 484–491. doi:10.1109/TE.2025.3610340

[16] Shuyin Zhao. 2023. GitHub Copilot Chat beta now available for all individuals. https://github.blog/news-insights/product-news/github-copilot-chat-beta-now-available-for-all-individuals/

Received 2026-06-27; accepted 2026-07-31