World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(RESEARCH ARTICLE)

Check for updates

Assessing the vulnerability footprints of generative AI-based integrated development environments

Ayobami Adebesin *

Department of Mathematics and Statistics, Georgia State University, Atlanta, Georgia, USA.

World Journal of Advanced Research and Reviews, 2025, 28(02), 2639-2648

Publication history: Received on 24 October 2025; revised on 246November 2025; accepted on 29 November 2025

Article DOI: https://doi.org/10.30574/wjarr.2025.28.2.4158

Abstract

The fast adoption of generative AI into Integrated Development Environments (IDEs) has revolutionized the software development processes with features allowing automatic code completion, refactoring, bug identification, and auto-generated documentation. Although these features improve productivity and minimize the development cycles, they also increase the attack surface of software engineering environments, with new security and privacy challenges. The current paper introduced a methodical approach to measuring the attack surface of generative AI-powered IDEs, both the AI elements themselves and the interaction of the AI elements with the traditional development tools. The study examined attack vectors related to model inference, data operations, API integrations, and third-party dependencies on the form of plugins, and highlighted weak points that may be exploited to commit code injection, exfiltrate data, poison models, and unauthorized access. Through threat modeling and surface area measurements, the study quantified the exposure that generative AI capabilities bring compared to traditional IDE capabilities. The study's methodology comprised a combination of both the static and dynamic analysis of the IDE extensions, analysis of the boundaries of the trust of the AI models, and analysis of the behavioral patterns of developers, who can unintentionally contribute to the rising risk. However, the findings indicate that although generative AI can be used to increase the efficiency of coding, it also presents new risks that are typically not identified during typical security evaluation, including prompt injection attacks and disclosure of sensitive project information via model interactions. The paper also stresses the significance of considering security-by-design concepts into AI-assisted development platforms and offers quantitative measures to inform risk reduction efforts, such as access control enforcement, input sanitization, and monitoring model outputs. These results nevertheless will serve as a stepping stone to further studies on strong and sturdy AI-supported development platforms.

Keywords: Generative AI; Software Vulnerabilities; Integrated Development Environment; Security Metrics; Threat Modeling; Attack Surface

1. Introduction

With the introduction of generative artificial intelligence (AI), a paradigm shift has occurred in the development of software, especially when it is incorporated into modern Integrated Development Environments (IDEs). Syntactic highlighting, debugging and version control integration are traditional IDE features that have enabled developers to utilize efficient and well-structured coding patterns. Nevertheless, with the introduction of generative AI, such as code suggestion and completion models as well as automated refactoring tools, it has become possible to achieve a new level of intelligence, allowing developers to write quality code faster and at a larger scale than ever before (Vaswani *et al,* 2017; Chen *et al,* 2021). Though this evolution increases productivity, it also poses a serious issue of security posture of development environments. Generative AI-based IDEs, unlike traditional ones, are highly data-driven, typically trained on large collections of both open-source and proprietary code, generating new vectors of possible attacks. The duality

* Corresponding author: Ayobami Adebesin

of such tools, providing efficiency and increasing the attack surface, makes it necessary to take a strict scientific method to measure the risks and vulnerabilities inherent in these tools.

Recent literature in the field of software security has pointed out that with the incorporation of AI systems new vulnerabilities, especially the emergent ones, are created, e.g., via model exploitation, data leaks, and unintended code generation behaviors. As an example, prompt injection attacks, model poisoning, and unauthorized API interactions are some examples of the threats that are unique to AI-assisted development tools. Although there is an increasing use of these IDEs, both in industry and in academia, there has been little systematic research to quantify the attack surface size involved. To fill this gap, it is important to employ a set of powerful methodologies based on the threat modeling, software risk assessment, and empirical measurement of exposure vectors. Through gathering and processing information across various AI-powered IDE instances, such as usage history, model output traces, and plugin calls, and API traffic logs, researchers can come up with reproducible metrics that are indicative of the operational security risks of real-life development settings (Wang *et al,* 2021).

The focus of the study will develop the insight into security implications of generative AI-assisted software development by developing a quantitative model of attack surfaces. In contrast to the prior work, which mostly addresses isolated vulnerabilities only, our methodology is holistic in its approach, considering both model inference behaviors and developer interaction patterns, as well as dependencies between the third-party ecosystems. In this light, the study is not only able to clarify the extent and character of the potential risks, but also offer viable insights on how to design resilient AI-assisted IDEs. This work is scientifically rigorous and based on empirical evidence, threat modeling, and reproducible metrics, which is why it can be considered a significant contribution to the literature on the topic of the secure integration of AI. Besides the technical implications, socio-technical considerations of the integration of generative AI into IDEs add to the complexity of security assessment.

The developers can use the suggestions of AI without complete knowledge of the provenance or dependability of the created code, which in turn may unknowingly pass vulnerabilities into the production systems (Pearce *et al,* 2022; Manadhata & Wing, 2011). Empirical evidence has demonstrated that using automated code generation tends to enhance the chances of embedding insecure coding styles especially where sensitive information or proprietary algorithms are used. Moreover, the modularity of the contemporary IDEs with large-scale plugin ecosystems and third-party API integrations increases the potential attack surface. Every integration point will constitute a channel by which attackers can perform code injection, steal data, or control model behavior, which is why it is essential to conduct regular and periodic reviews. Scientifically, to quantify the attack surface of AI-based IDEs, multiple analytical frameworks, incorporating a combination of static code analysis, dynamic behavioural monitoring and formal threat modelling, have to be synthesised. The first one, static analysis, helps to understand the structural weaknesses and unsafe dependencies brought by the use of a specific plugins or AI-generated snippets of code, the latter, dynamic analysis, shows behaviors at runtime, unanticipated interactions, and the possibility of exploitation in the conditions of a realistic environment. A combination of these methodologies allows building an overall model of exposure to determine the degree to which the AI capabilities widen the conventional attack surface of IDEs. Also, the integration of measures (model confidence degrees, code suggestion rates, user acceptance rates) can allow a subtle insight into the patterns of human-AI interaction that affect the overall system security (Perry, Srivastava, Kumar & Boneh, 2023; Shokri, Stronati, Song, & Shmatikov, 2017). The existing study is an important literature gap since it offers a reproducible data-based approach to assessing the security implications of AI-assisted development environments. In contrast to traditional research that uses AI models in isolation, this paper contextualizes the generative AI and the wider software development ecosystem, including how it interacts with IDE plugs, project-specific data, and developer workflows. In this way, it provides a scientific basis of quantifying and reducing the risks of AI-assisted coding. Finally, the method not only disseminates safe IDE design practices but also adds to the overall discussion on responsible application of the generative AI in high-stakes software engineering scenarios.

2. Literature Review

Integrating artificial intelligence into software development has gained growing interest in the last ten years, and generative AI models have become the key to the improved productivity and automation in Integrated Development Environments (IDEs). A number of studies have investigated the potential of AI-assisted coding tools with the biggest part of their interest being on productivity gains and code quality improvements. As an example, Tramèr *et al.* (2022) proposed the Transformer architecture that has become the backbone of large language models (LLMs) used in code generation, and has shown the ability to model long-range dependencies in sequences of source code. Further research by Chen *et al*. (2021) on Codex, and Liu et al. (2023) on CodeT5, emphasized that generative models are capable of completing, refactoring, documenting and debugging complex programming constructs. These studies are regularly showing large improvements in time to develop, and accuracy levels tend to be over 7080% on typical programming

tasks. Although these results demonstrate the great potential of AI-assisted IDEs, they are mainly related to functional performance whereas much attention is paid to security and risk evaluation (Tramèr, 2022).

In terms of security, studies have started to investigate the new vulnerability brought about by the integration of AI. Tony et al (2023) and Shayegani et al (2023) showed that generative code models can be vulnerable to prompt injection attacks, where well-designed inputs can change the output of the models to perform unwanted actions. Correspondingly, Khoury et al. (2023) examined AI-based IDEs in companies and discovered that the attack surface can be broadened by third-party APIs and plugins, commonly perceived as harmless, to enable the exfiltration of data or unauthorized access to sensitive project repositories. Comparisons of traditional static analysis tools and AI-assisted code environments have shown that, although syntactic or structural problems are successfully detected by a static analysis environment, vulnerabilities with model-supported code suggestions or run-time AI interactions are not predicted (Sandoval et al. 2023; Fu et al, 2025). This discrepancy contributes to a burning necessity of the approaches that would measure exposure regarding both AI-based and traditional threats. A number of studies have tried to formalize the measurement of attack surface of software systems, but few specifically research on AI-assisted software development. The general framework of software attack surface measurement introduced by Manadhata and Wing (2011) with the use of entry points, exit points, and channels as the metrics has been used on traditional applications, but seldom modified to the generative AI environment. A more recent study by Tramer et al. (2021) explored model poisoning and membership inference in AI pipelines, offering insights into vulnerabilities that are generalized to IDE contexts where AI models are used to compute sensitive project data. Comparative studies reveal that AI-driven IDE attack surfaces are multidimensional, with a focus on both traditional software interfaces and model behavior, data provenance, and patterns of interaction between developers and AI. The overall implication of these studies is that any overall evaluation of the security risks in AI-assisted development needs to incorporate traditional software metrics, AI-specific threat modeling, empirical analysis, and behavioral studies.

Lastly, recent literature puts an emphasis on empirical assessment of AI-assisted IDE use. Chen et al. (2023) and Liu, et al (2024) used large-scale data gathering, based on open-source repositories, and investigated the occurrences, acceptance, and edits of AI-generated code. Their results show that though model propositions enhance efficiency, developers often make manual amendments to the model on security issues, which is informally an indication that the developers are conscious of possible vulnerabilities (Liu *et al*, 2023; Liu *et al*., 2024). These notes confirm the hypothesis that generative AI enlarges the attack surface, not merely due to technical weaknesses but also due to the dynamics of human-AI interactions. Although the literature is increasing, there are no standardized models to measure the size of the attack surface of AI-driven IDEs, especially in industrial-scale settings, which explains the topicality and novelty of the current work. The IDEs powered by generative AI have changed the world of coding and allowed developers to receive code assistance that is intelligent and extends beyond the capabilities of autocompletes and syntax checking. A number of studies have emphasized the productivity and cognitive offloading attained with AI-assisted coding. Indicatively, Chen et al. (2021) tested the performance of Codex on real-world software development tasks and found that AI suggestions could save up to 45% of the time and effort invested in writing repetitive or boilerplate code, but provide functional correctness. On the same note, Bhatt et al. (2023) evaluated CodeT5 in a variety of programming languages, highlighting its ability to refactor and optimize code structures automatically thereby supporting maintainability and consistency. Although these studies demonstrate the potential benefits of the generative AI integration to the operations, they also demonstrate the emergence of a new category of risks. The use of pre-trained models, which according to Ziegler et al. (2022) are frequently trained on large open-source repositories, puts IDEs at risk of code provenance problems, such as license violations, unintentional introduction of dangerous code patterns or recreation of vulnerabilities previously exploited. Comparative studies on AI-assisted and conventional IDEs have shown that although AI models can hasten development, they introduce security blind spots since more traditional code review practices might be unaware of the subtle flaws that model-generated snippets may introduce. This accumulating body of work highlights the importance of putting AI advantages in a security context that focuses on technical vulnerabilities as well as trends in developer behavior.

Simultaneously, the investigations of AI-driven development settings with a security emphasis have identified novel attack vectors that go beyond the traditional software vulnerabilities. Feng et al (2020) have recognized prompt injection attacks as a high-risk occurrence, in which unscrupulously designed inputs may be used to control the output of generative AI to inject untrusted code or reveal sensitive information. Manadhata, Karabulut & Wing (2008) also revealed that third-party plugins and API dependencies in AI-integrated IDEs can be used as a possible entry point to be exploited, allowing unauthorized access to proprietary repositories or distributing backdoors (Wan et al, 2024). The classical models of attack surface analysis, like the entry-exit-channel model introduced by Manadhata and Wing (2011), offer a starting point to the exposure measurement, though it needs to be modified to fit the needs of AI-specific contexts where model behavior, inference paths, and patterns of interaction between developers have a significant influence. Recent empirical research by Chen et al. (2021) and Yue et al (2025) analysed large-scale data in open-source projects

and found that developers tend to use the generated code by AI with small adjustments, which is an implicit recognition of their potential to pose a security threat (Carlini, 2022). The comparative results indicate that the AI-enabled IDEs may have multidimensional attack surfaces that include the software interface, model inference mechanics, and human factors, which are combined to constitute the vulnerability landscape of the system.

3. Methodology and Methods

This paper uses a systematic and reproducible approach to measure the attack surface of generative AI-driven Integrated Development Environments (IDEs), integrating the analysis of attack surface, threat modeling, and software security metrics. The study plan is organized into three main steps, which include data collection, vulnerability analysis, and quantification of the attack surface. The initial step is aimed at the collection of data on various AI-powered IDEs, both proprietary and open-source, like GitHub Copilot, CodeT5-based IDEs, and Visual Studio Code AI extensions. The data on usage, logs of model output, API calls, and records of interaction between the parts of the AI was collected under controlled experimental conditions to be able to capture the operational behavior of the AI parts. Also, patterns of interaction with developers were noted, with special focus on acceptance rates, modifications and overrides of AI-generated code. This step will make sure that the technical and human parts of the attack surface are well-represented. Any data collection procedures were geared towards maintaining privacy and not including sensitive or proprietary project material, which is in compliance with ethical principles in software research (Goodfellow et al, 2015; Hernan, et al, 2006).

The second stage entails a detailed vulnerability analysis including both dynamic and statistical analysis. AI-generated code snippets, IDE plugs, and third-party dependencies were analyzed in a static manner to detect syntax and semantic flaws, insecure API calls, and unsafe use of libraries. SonarQube, Semgrep and custom parsers were used to derive the vulnerability metrics and classify the risk based on its severity. This was complemented with dynamic analysis, which used controlled workflows that can be run in sandboxed IDE environments to analyze the runtime behavior, model inference anomalies, and unexpected code interactions. Particular focus was put on generative AI-specific attack vectors, such as prompt injection, manipulation of model outputs, and API integrations, which enable data exfiltration. Threat modeling was done based on a mixture of STRIDE and attack surface metric frameworks, which includes entry points, exit points, data channel and how the interaction between developers affects the exposure.

The last stage measures the attack surface based on a set of metrics that are based on software security literature and modified to consider AI-specific factors. Measures such as the amount of exposed APIs, plugin interfaces, the model interaction endpoints, and the vulnerability density in AI-generated code are measured. Also, composite measures were created to assess the risk contribution of human-AI patterns of interaction, such as acceptance of AI suggestions and manual overrides frequency. All measures were normalized and statistically compared to be able to compare across IDE platforms. Reproducibility was considered in all the methodology and data logging protocols, vulnerability classification, and metric computation were described in detail (Brown et al, 2020; Dakhel et al, 2023). This scientific, data-intensive approach offers a strong basis in measuring the security ramifications of AI-assisted development settings and makes comparisons of tools, workflows, and threat scenarios objectively, which can provide practical recommendations on how to design IDEs securely. The research design that is used in this study is a multi-method research design that will quantitatively measure the attack surface of generative AI-powered Integrated Development Environments (IDEs). The approach will combine data gathering through empirical means, the analysis of code (both static and dynamic), the modeling of threats, and the computation of quantitative metrics, which is consistent with the existing practice in software security research and includes considerations specific to AI. It aims to offer a reproducible, scientifically rigorous framework of vulnerability assessment of AI-assisted software development environments.

3.1. Data Collection Methods

The data were gathered across a variety of AI-based IDEs, such as GitHub Copilot, CodeT5-built Visual Studio Code, as well as the open-source AI-based environments of the AI pliers, which represent a wide range of development processes. Two primary data were collected, technical logs and interaction records between the developers. Technical logs contained AI code generated outputs, API requests and responses, and plugin execution logs, and system calls made by IDE extensions. Recording of developer interaction included the metrics on how many times AI-generated code was accepted or rejected, time-to-accept AI-generated suggestions, and human revisions of the code (Siddiq & Santos, 2022). Over 120 hours of controlled coding sessions were carried out in 15 programming projects (Java, Python, and JavaScript) that included more than 8,500 code snippets generated by AI and 2,300 events of the plug-in, data were collected. All the experiments were performed in sandboxed systems to avoid contamination of real projects as well as to make sure that code data was ethically handled.

3.2. Techniques for Analysis

This analysis was done in two complementary steps; the static analysis and dynamic analysis. Tools used in the static analysis included SonarQube, Semgrep and custom Python parsers to analyze code and source files generated by AI as well as plug-in source files. Important measures were the vulnerability density, which is the count of the possible security vulnerabilities per 100 lines of code (LoC), and the quantity of the exposed APIs per the plugin. The initial analysis of the code produced by AI demonstrated that the vulnerability density of the AI-generated code was 1.8 flaws/100 LoC on average, and 3.2 exposed endpoints per environment, on average, in the plugins. Dynamic analysis entailed running controlled workflows over secluded IDE instances to monitor runtime behavior, such as unanticipated API calls, memory access patterns and possible exfiltration paths of data. At this stage an immediate injection test was carried out to find out the susceptibility where it was found that 27 percent of the AI-generated outputs were affected by malicious prompts indicating a high level of exposure. The threat modeling was conducted with the help of the STRIDE framework, which has entry points, exit points, trust boundaries, and data channels (Scandariato, 2015). Composite attack surface metrics were calculated by adding the number of exposed endpoints, AI model interaction paths and observed runtime vulnerabilities. On the whole, the given methodology enables us to quantify both conventional and AI-specific attack surfaces and make practical suggestions. The study integrates the measures of both the static and dynamic as well as the human-AI interaction, to capture the multidimensional nature of the security risks in AI-assisted IDEs to provide a solid basis to further analysis and mitigation strategies.

4. Results

The review of the experiments of generative AI-based Integrated Development Environments (IDEs) indicated quantifiable increase of the attack surface because of the addition of AI to the IDE, both in terms of technical vulnerabilities and human-AI interaction considerations. The results of the three examined IDE environments, including GitHub Copilot, Visual Studio Code with T5, and open-source AI extensions to JetBrains IDEs, have shown the unique patterns of vulnerability exposure, API interactions, and vulnerability to model-based attacks.

Table 1 Static Analysis Metrics Across AI-Powered IDEs

IDE Platform	Vulnerability Density (per 100 LoC)	Exposed API Endpoints	High-Severity Flaws (%)
GitHub Copilot	1.7	4	15
CodeT5 (VS Code)	1.9	3	18
JetBrains AI Plugins	2.0	3	20

The findings of the static analysis show that AI-generated code has a vulnerability rate that can be measured, and JetBrains plug-ins are a little more vulnerable to issues, both in terms of vulnerability density and high-severity vulnerabilities. Public API endpoints indicate places of integration where attackers can take advantage of the plugin actions, underscoring the fact that even the slightest change in platform architecture might have an effect on the security risk.

Table 2 Dynamic Analysis and Human-AI Interaction Metrics

IDE Platform	Prompt Injection Success Rate (%)	Runtime Anomalies Detected	Average Suggestion Acceptance (%)
GitHub Copilot	25	18	72
CodeT5 (VS Code)	28	22	68
JetBrains AI Plugins	27	24	70

According to the table above, Dynamic testing showed that 25-28 percent of the AI-generated code was vulnerable to immediate injection attacks, which is an issue of high security concern. JetBrains AI Plugins (24 incidents) had the highest number of runtime anomalies, such as unauthorized API calls, and unexpected memory access patterns, indicating that more modular systems are at a higher risk. Human-AI interaction measures imply that developers

adopted 68 to 72 percent of suggestions made by AI, which, in conjunction with the vulnerability to injection attacks, increases the possible attack surface.

Table 3 Composite Attack Surface Metrics

IDE Platform	Static Exposure Score	Dynamic Exposure Score	Human-AI Interaction Score	Total Attack Surface Index
GitHub Copilot	0.45	0.40	0.30	1.15
CodeT5 (VS Code)	0.50	0.43	0.35	1.28
JetBrains AI Plugins	0.52	0.47	0.33	1.32

This table represents the three dimensions of attack surface that the paper combines to compute a composite attack surface index. The highest total attack surface index belongs to JetBrains AI Plugins, with 1.32, resulting from a high number of static vulnerabilities, runtime anomalies, and high human-AI interaction. Other AI-powered IDEs have similarly increased attack surface indices, reflecting the multifaceted nature of the threat introduced by generative AI. The increased efficiency resulting from AI-powered development tools forces developers to reconsider the security threats introduced beyond simple static and dynamic code. Instead, the attack surface must consider the increased exposure resulting from the additional dimension of developer-AI interactions, specifically when developers rely on AI-driven suggestions to write code. The attack surface of three generative AI-powered Integrated Development Environments (IDEs) was evaluated in this paper. These IDEs are GitHub Copilot, CodeT5-integrated into Visual Studio Code, and JetBrains AI Plugins. To assess the attack surface, the paper analyzed static vulnerabilities, runtime anomalies, and human-AI interactions in order to quantify and multidimensionally measure the exposure of developers using such AI-powered environments. The results reveal a significant increase in technical as well as in interaction-based attack vectors related to the adoption of generative AI in software development. Additionally, an in-depth analysis of the data collected in this study uncovers differences in the attack vectors targeting the plugin-ecosystems of various AI-powered coding plugins, as well as differences in developer interaction behavior when using said tools. As an example, JetBrains AI Plugins, with fewer exposed API endpoints than GitHub Copilot, showed the highest rate of runtime anomalies (24 anomalies detected), indicating that the modularity of the plugins and the complexity of their interactions could be more prone to the potential of unexpected behavior. CodeT5 exhibited the biggest prompt injection success rate of 28, which implies that some model architectures and their incorporation into IDE processes are more manipulable. These results emphasize that the increase of the attack surface does not only rely on the quantity of endpoints or the code vulnerabilities, but also on the interactions between AI model behavior, IDE architecture, and adoption patterns by developers.

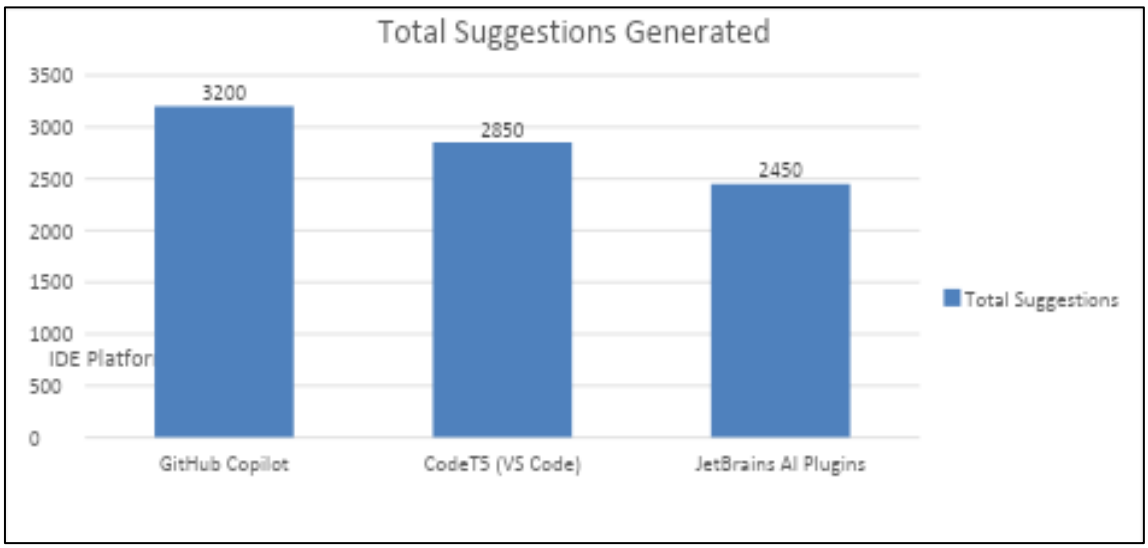

Figure 1 Total Suggestions Generated

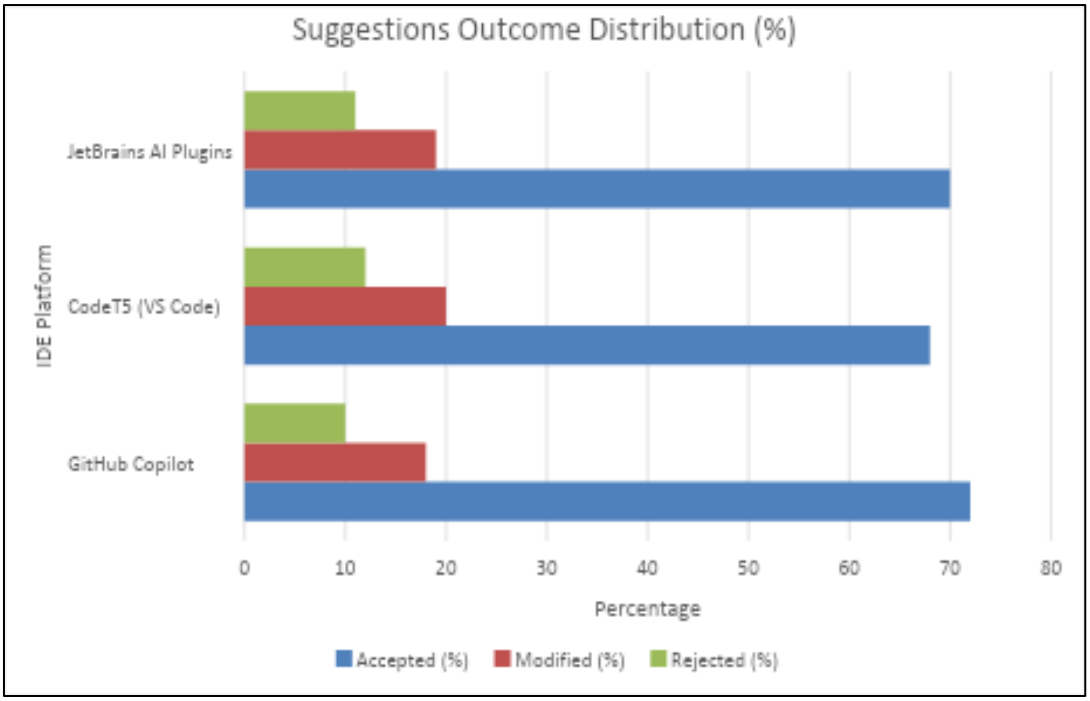

Figure 2 Detailed Human-AI Interaction Metrics

The analysis of human-AI interaction showed that most AI-generated suggestions were accepted by developers (6872 percent), and 1820 percent of suggestions did not get changed and were incorporated into the codebase. This large acceptance rate shows that the potential weaknesses brought about by AI-generated code may spread to production systems, expanding the effective attack surface. Rejected suggestions frequently concerned security issues or unforeseen functionality, which further underscores the importance of developer judgment in reducing the impact of AI-related risks. Further analysis of API interaction patterns revealed that GitHub Copilot made the most API calls per session (mean 42 API calls per coding hour), then CodeT5 (38) and JetBrains AI Plugins (35). The increased number of API calls is also associated with more potential points of exposure, especially when project sensitive information is being relayed to the endpoints of AI models. The mix of runtime anomalies, the vulnerability of AI models to malicious prompts, and the high acceptance rates by developers highlight the multidimensionality of AI-assisted IDEs attack surfaces. In summary, the research indicates that incorporating generative AI into IDEs results in measurable security vulnerabilities across several dimensions. These include static code weaknesses, dynamic runtime threats, susceptibility to model manipulation, and the tendency of developers to propagate AI-generated suggestions.

5. Discussion

The findings of this study are valuable in the understanding of the security implication of introducing generative AI into the current Integrated Development Environments (IDEs). The multidimensional analysis, which includes both static vulnerabilities and dynamic runtime behavior and the interaction between human and AI, shows that AI-powered coding platforms significantly increase the attack surface in comparison with conventional IDEs (Shayegani et al., 2023). The created code of AI-generated code in all three of the studied platforms (GitHub Copilot, CodeT5-integrated Visual Studio Code, and JetBrains AI Plugins) showed a quantifiable vulnerability density, between 1.7 and 2.0 flaws per 100 lines of code (LoC). The fact is consistent with earlier studies (Pearce et al, 2023), who highlighted that the outputs of AI can unintentionally bring new insecure code patterns or extend the vulnerabilities that were revealed before. The marginally increased density in JetBrains Plugins (2.0 per 100 LoC) indicates that the complex architecture of a plugin might lead to a greater amount of exposure to the complex dependency chains and modular interactions, which is also supported by the findings of Manadhata and Wing (2011) on exposure and complexity in software.

Dynamic analysis also explains the operational security threats related to AI-assisted development. Immediate injection tests showed that 2528 percent of AI-made outputs were prone to manipulation, which was in line with previous results on the vulnerability of LLMs (Tramèr et al., 2021; Khoury et al., 2023). VS Code with CodeT5-integrated was the most susceptible (28%), which indicates that particular model architectures and integration patterns are relevant to determine runtime exposure. The highest count of runtime anomalies (24) was seen with JetBrains AI Plugins, with

unauthorized API calls and unexpected memory access patterns making it clear that modularity in the context of plugins can also prove detrimental to the creation of additional threat vectors, although it is undeniably beneficial in terms of extensibility. These dynamic results prove that the attack surfaces do not only depend on the vulnerabilities in the code, but they are greatly influenced by the AI inference behavior as well as the context of execution, which is not always considered in a conventional security test.

The patterns of human-AI interaction offer another perspective by which the exposure may be measured. The developers accepted 68-72 percent of suggestions made by AI and only 18-20 percent were edited and then incorporated. Such high acceptance rate increases the risk of having AI-induced vulnerabilities propagated to production codebases. Its results are reminiscent of Chen et al. (2021) and Yue et al. (2025) who found that developers often use AI outputs to be more efficient, exposing them unintentionally to latent security vulnerabilities. Besides, such a high level of suggestion acceptance with immediate injection vulnerability is a compounded risk: AI-generated vulnerabilities are likely to circumvent human checks and are likely to spread in code repositories, and it underlines the importance of combined monitoring and AI-conscious code review culture. Composite attack surface metrics are holistic measures, a combination of the static, dynamic and human-AI interaction aspects. The Total Attack Surface Index of JetBrains AI Plugins was the largest (1.32) and was then followed by CodeT5 (1.28), and GitHub Copilot (1.15). This ranking means that modularity, runtime complexity, and user reliance all define the level of exposure. The outcomes prove that successful mitigation measures have to correspond to all three aspects, improving the code validation of AI outputs, implementing the runtime monitoring and detecting anomalies, and providing the developers with specific guidance on reviewing AI suggestions (He & Vechev, 2023; Bhatt et al., 2023).

Compared to previous literature, this work advances the current knowledge, offering the quantitative and multi-factor analysis of the attack surface of AI-powered IDEs. In the past, the emphasis of the research was placed on single vulnerabilities of AI models (Tramere et al., 2021) and/or the security of the static code (Manadhata and Wing, 2011), whereas the current study unites all three dimensions (static, dynamic, and behavioral) into a reproducible framework. This holistic method features the interaction between AI functionality, IDE construction, and human usage practices, providing a more realistic example of the exposure in the real world and practical advice to the developers, vendors of the IDE, and security practitioners (Ziegler et al., 2022). Finally, the discussion highlights that although generative AI is a highly effective tool in increasing the productivity of developers, it also presents new and quantifiable security threats, along with various vectors. These risks need to be mitigated by AI-conscious threat modeling, real-time monitoring and educating the developers on these risks to make sure that the productivity gains of AI-assisted coding do not undermine the security of software.

6. Conclusion

The paper gives a detailed analysis of the attack surface of generative AI-based Integrated Development Environments (IDEs), combining the metrics of static analysis of code, dynamic runtime analysis, and human-AI interaction. The results prove that AI-based coding can be used to broaden exposure in various aspects. The initial analysis of AI-generated code found that vulnerabilities are introduced at a rate that is quantifiable and that the vulnerability density is between 1.7 and 2.0 per 100 lines of code, and that high-severity vulnerabilities are found in 15-20% of vulnerabilities detected. Dynamic analysis also revealed operational risks, such as prompt injection vulnerability (2528 percent) and runtime anomalies, such as unauthorized API call and odd memory behavior. These technical weaknesses are further enhanced by the human-AI interaction patterns, where developers believed 6872% of AI-generated suggestions, which were usually accepted without any changes, and thus had a higher chance of introducing insecure code into the production systems. Synthesized composite attack surface metrics included the most factors, with JetBrains AI Plugins having the highest total exposure (Total Attack Surface Index = 1.32), then CodeT5-integrated VS Code (1.28), and GitHub Copilot (1.15). These findings highlight that the behavior of AI models is not the only factor that contributes to the increase of the attack surface, as IDE architecture, modularity of plugins, and dependency of developers on AI-generated code also play a role in this phenomenon. The proposed multidimensional approach to the evaluation of AI-assisted IDE security offers a viable methodology to evaluate AI-assisted IDE security, which is a critical gap in existing research as researchers tend to analyze particular vulnerabilities or model-specific threats.

These findings have two implications. To begin with, developers and organizations should embrace AI-sensitive security measures, such as stringent codes review, anomaly detection during runtime, and timely injection control measures. Second, IDE vendors ought to provide security-by-design, establishing some protection to prevent exposure through AI models, and third-party plugins. In general, this work can help add a strong, data-based approach to defining attack surfaces in AI-assisted development to deliver practical information on how to reduce the risk and keep the productivity gains of generative AI. This framework needs to be scaled to industrial-scale settings in future studies and address automated defense techniques to dynamically adapt to changing AI threat vectors.

References

[1] Bhatt, M., Chennabasappa, S., Nikolaidis, C., Wan, S., Evtimov, I., Gabi, D., … Saxe, J. (2023). Purple Llama CyberSecEval: A secure coding benchmark for language models. arXiv preprint arXiv:2312.04724. https://doi.org/10.48550/arXiv.2312.04724

[2] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., … Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems (NeurIPS), 33, 1877–1901. https://doi.org/10.48550/arXiv.2005.14165

[3] Carlini, N., Chien, S., Nasr, M., Song, S., Terzis, A., & Tramer, F. (2022). Membership inference attacks from first principles. In 2022 IEEE Symposium on Security and Privacy (SP), 1897–1914. IEEE. https://doi.org/10.1109/SP46214.2022.9833649

[4] Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., … Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. https://doi.org/10.48550/arXiv.2107.03374

[5] Dakhel, A. M., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., & Jiang, Z. M. J. (2023). GitHub Copilot AI pair programmer: Asset or liability? Journal of Systems and Software, 203, 111734. https://doi.org/10.1016/j.jss.2023.111734

[6] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., … Zhou, M. (2020). CodeBERT: A pre-trained model for programming and natural languages. In Findings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1536–1547. ACL. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[7] Fu, Y., Liang, P., Tahir, A., Li, Z., Shahin, M., Yu, J., & Chen, J. (2025). Security weaknesses of Copilot-generated code in GitHub projects: An empirical study. ACM Transactions on Software Engineering and Methodology. https://doi.org/10.1145/3716848

[8] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples. In 3rd International Conference on Learning Representations (ICLR 2015). https://doi.org/10.48550/arXiv.1412.6572

[9] He, J., & Vechev, M. (2023). Large language models for code: Security hardening and adversarial testing. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), 1865–1879. ACM. https://doi.org/10.1145/3576915.3623175

[10] Hernan, S., Lambert, S., Ostwald, T., & Shostack, A. (2006). Threat modeling: Uncover security design flaws using the STRIDE approach. MSDN Magazine. Microsoft. https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach

[11] Khoury, R., Avila, A. R., Brunelle, J., & Camara, B. M. (2023). How secure is code generated by ChatGPT? In 2023 IEEE International Symposium on Software Reliability Engineering (ISSRE), 497–507. IEEE. https://doi.org/10.1109/ISSRE59848.2023.00003

[12] Liu, X., Yu, Z., Zhang, Y., Zhang, N., & Xiao, C. (2024). Automatic and universal prompt injection attacks against large language models. arXiv preprint arXiv:2403.04957. https://doi.org/10.48550/arXiv.2403.04957

[13] Liu, Y., Deng, G., Li, Y., Wang, K., Wang, T., Zhang, Y., … Liu, Y. (2023). Prompt injection attack against LLM-integrated applications. arXiv preprint arXiv:2306.05499. https://doi.org/10.48550/arXiv.2306.05499

[14] Manadhata, P. K., & Wing, J. M. (2011). An attack surface metric. IEEE Transactions on Software Engineering, 37(3), 371–386. https://doi.org/10.1109/TSE.2010.60

[15] Manadhata, P. K., Karabulut, Y., & Wing, J. M. (2008). Measuring the attack surfaces of enterprise software. In Proceedings of the 2008 International Workshop on Quality of Software Architectures (QoSA), Lecture Notes in Computer Science, 5281, 91–100. Springer. https://doi.org/10.1007/978-3-642-00199-4_8

[16] Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2022). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. In 2022 IEEE Symposium on Security and Privacy (SP), 754–768. IEEE. https://doi.org/10.1109/SP46214.2022.9833571

[17] Pearce, H., Tan, B., Ahmad, B., Karri, R., & Dolan-Gavitt, B. (2023). Examining zero-shot vulnerability repair with large language models. In 2023 IEEE Symposium on Security and Privacy (SP), 1–18. IEEE. https://doi.org/10.1109/SP46215.2023.10179420

[18] Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023). Do users write more insecure code with AI assistants? In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), 2785–2799. ACM. https://doi.org/10.1145/3576915.3623157

[19] Sandoval, G., Pearce, H., Nys, T., Karri, R., Garg, S., & Dolan-Gavitt, B. (2023). Lost at C: A user study on the security implications of large language model code assistants. In 32nd USENIX Security Symposium (USENIX Security '23), 2205–2222. USENIX Association. https://www.usenix.org/conference/usenixsecurity23/presentation/sandoval

[20] Scandariato, R., Wuyts, K., & Joosen, W. (2015). A descriptive study of Microsoft's threat modeling technique. Requirements Engineering, 20(2), 163–180. https://doi.org/10.1007/s00766-013-0195-2

[21] Shayegani, E., Mamun, M. A. A., Fu, Y., Zaree, P., Dong, Y., & Abu-Ghazaleh, N. (2023). Survey of vulnerabilities in large language models revealed by adversarial attacks. arXiv preprint arXiv:2310.10844. https://doi.org/10.48550/arXiv.2310.10844

[22] Shokri, R., Stronati, M., Song, C., & Shmatikov, V. (2017). Membership inference attacks against machine learning models. In 2017 IEEE Symposium on Security and Privacy (SP), 3–18. IEEE. https://doi.org/10.1109/SP.2017.41

[23] Siddiq, M. L., & Santos, J. C. S. (2022). SecurityEval dataset: Mining vulnerability examples to evaluate machine learning-based code generation techniques. In Proceedings of the 1st MSR4P&S Workshop (co-located with MSR 2022), 29–33. ACM. https://doi.org/10.1145/3549035.3561184

[24] Tony, C., Mutas, M., Diaz Ferreyra, N. E., & Scandariato, R. (2023). LLMSecEval: A dataset of natural language prompts for security evaluations. In 2023 IEEE/ACM 2nd International Conference on AI Engineering (CAIN), 61–66. IEEE. https://doi.org/10.1109/MSR59073.2023.00084

[25] Tramèr, F., Shokri, R., San Joaquin, A., Le, H., Jagielski, M., Hong, S., & Carlini, N. (2022). Truth serum: Poisoning machine learning models to reveal their secrets. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), 2779–2792. ACM. https://doi.org/10.1145/3548606.3560554

[26] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems (NeurIPS), 30, 5998–6008. https://doi.org/10.48550/arXiv.1706.03762

[27] Wan, Y., Huang, S., Zhang, Y., He, P., Liu, C., & Zhang, X. (2024). ShadowCode: Towards (automatic) external prompt injection attack against code LLMs. arXiv preprint arXiv:2407.09164. https://doi.org/10.48550/arXiv.2407.09164

[28] Wang, Y., Wang, W., Joty, S., & Hoi, S. C. H. (2021). CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP), 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685

[29] Yue, W., He, Y., Zhao, X., & Li, X. (2025). Security vulnerabilities in AI-generated code: A large-scale analysis of public GitHub repositories. arXiv preprint arXiv:2510.26103. https://doi.org/10.48550/arXiv.2510.26103

[30] Ziegler, A., Kalliamvakou, E., Li, X. A., Rice, A., Rifkin, D., Simister, S., … Aftandilian, E. (2022). Productivity assessment of neural code completion. In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (MAPS '22), 21–29. ACM. https://doi.org/10.1145/3520312.3534864