Federal Office
for Information Security

RÉPUBLIQUE
FRANÇAISE
Liberté
Égalité
Fraternité

AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION
ANSSI

AI Coding Assistants

Published by

Bundesamt für Sicherheit in der Informationstechnik
53133 Bonn, Germany
Phone: +49 (0) 228 99 95820
E-Mail: bsi@bsi.bund.de

Source:

Federal Office for Information Security (BSI)
P.O. Box 20 03 63
53133 Bonn, Germany
Phone: +49 (0) 228 99 95820
E-Mail: ki-kontakt@bsi.bund.de

Agence nationale de la sécurité des systèmes d‘information
Secrétariat général de la défense et de la sécurité nationale
51, boulevard de La Tour-Maubourg
75700 Paris 07 SP, France
Phone: +33 (0)1 71 76 85 85
E-Mail: communication@ssi.gouv.fr

Last updated: September 2024

Executive Summary

This report provides recommendations for a secure use of AI coding assistants compiled by the French Cybersecurity Agency (Agence nationale de la sécurité des systèmes d'information, ANSSI) and the German Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik, BSI). Within the document, perspectives are given on the opportunities that arise through the use of AI coding assistants as well as risks associated with the technology. Concrete mitigation measures are outlined.

Opportunities

AI coding assistants can be utilized in several different stages of the software development process. While the generation of source code is the key functionality, these LLM-based AI systems can also help developers to familiarize themselves with new projects by providing code explanations. Furthermore, AI coding assistants can support the code development process by automatically generating test cases and ease the burden of code formatting and documentation steps. The functionality to translate between programming languages can simplify the maintenace efforts by translating legacy code into modern programming languages. Additionally, the assistive nature of the technology can help increase the satisfaction of employees.

Risks

One important issue is that sensitive information can be leaked through the user inputs depending on the contract conditions of providers. Furthemore, the current generation of AI coding assistants cannot guarantee to generate high quality source code. The results have a high variance in terms of quality depending on programming language and coding task. Similar limitations can be observed for the security of generated source code. Mild and severe security flaws are commonly present in AI-provided code snippets. Moreover, using LLM-based AI systems during software development allows for novel attack vectors that can be exploited by malicious actors. These attack vectors include package confusion attacks through package hallucination, indirect prompt injections and poisoning attacks.

Recommendations

Building on the identified risks, BSI and ANSSI formulate the following key recommendations for the use of AI coding assistants (more detailed recommendations can be found within the document):

- AI coding assistants are no substitute for experienced developers. An unrestrained use of the tools can have severe security implications.
- A systematic risk analysis should be performed before introducing AI tools (including an assessment of the trustworthiness of providers and involved third-parties).
- Gain in productivity of development teams must be compensated by appropriate scaling measures in quality assurance teams (AppSec, DevSecOps).
- Generated source code should generally be checked and reproduced by the developers.

Table of Contents

1 Introduction

1.1 What are AI Coding Assistants?

In recent years, generative artificial intelligence (AI) has attracted significant media and social attention. These are AI models that create content such as texts, images or videos based on an input. Triggered by the great progress in the field of text generation, a large number of AI coding assistants based on large language models (LLMs) have been recently developed for the (partial) automation of source code generation. These are models that, depending on the approach, have either been trained on large amounts of text and then fine-tuned using source code, or have been trained directly on a large amount of source code. In application, these models can be used similarly to a chatbot. The users give the model a prompt, which can either be a description of the desired functionality or a (commented) source code skeleton. The output is source code with the desired functionality in a programming language chosen by the user. The current generation of models generates several alternatives in addition to a top suggestion - this is the suggestion that the model rated as most likely to be correct. Users can select one of these suggestions and adopt it into their current software project. These AI coding assistants are often accessed through a plug-in for integrated development environments (IDEs). Additionally, general chatbots, which are hosted in the cloud or locally, are also used by developers for programming.

1.2 Objective of the Document

In this publication we examine the opportunities and risks that arise from the use of AI coding assistants. The use of AI coding assistants is already widespread in many organizations and will become an integral part of software development in the future. Therefore, we propose concrete recommendations for managers and developers on how to handle this technology. This publication is intended to contribute to responsible and safe use of AI coding assistants.

The publication focuses on the use of AI coding assistants for professional software development. The closely related topics of no-/low-code applications are not addressed here. While many of the following chapters are also relevant to general chatbots based on LLMs, they fall outside the scope of this publication. For general opportunities and risks associated with LLMs, we refer to the publications from ANSSI (ANSSI, 2024) and BSI (BSI, 2024) on generative AI. Non-LLM-based coding assistants relying on other forms of AI, such as rule-based systems and traditional machine learning algorithms also fall outside the scope of this publication.

2 Opportunities for AI Coding Assistants

2.1 Generation of Code

Based on inputs in natural language or source code, coding assistants can generate methods, classes, and prototypes. This enables AI-powered coding assistants to take over recurring tasks in the software development process, such as creating frequently occurring code blocks and functions. Examples of basic algorithms that provide important functionalities for many areas of a software project include sorting or graph algorithms. The creation and adaptation of these building blocks to specific contexts can be (partially) automated by AI coding assistants. A study showed that current AI coding assistants are capable of correctly implementing such algorithms. Furthermore, it was found that the proposed implementation suggestions did not only have functional correctness but also optimal runtime performance (Dakhel, et al., 2023). The success rate of the technology decreases for more complex tasks but can still reach usable performance levels (Yeadon, et al., 2024). AI coding assistants can furtber be used for code completion, e.g. filling out missing parameter for a function call, code adaption to specific contexts or refactoring, e.g. restructuring code without changing its functionality. However, they are limited in solving issues in real-world software engineering szenarios with large codebases (Jimenez, et al., 2024).

2.2 Debugging

Debugging, which translates to "finding and fixing errors in source code," is a time-consuming task for developers. A report from 2018 suggests that developers dedicate more than 17 working hours per week to this task (Stripe, 2018). Using AI to support debugging could reduce the workload in software development. For specific programming languages, such as JavaScript, it has already been shown that AI models can be trained to automatically detect errors and function effectively (Pradel, et al., 2018; Berabi, et al., 2021). Furthermore, AI assistants not only detect but also directly fix errors (Allamanis, et al., 2021). The developments in recent years indicate that AI coding assistants have the potential to reduce the workload related to correction and maintenance of software, often referred to as "Technical Debt" (Popper, 2023).

2.3 Generation of Test Cases

The generation of test cases is another area where coding assistants can offer support. They can automatically propose test cases and unit tests that also cover edge cases by analysing the existing code and understanding the program logic (Smith, 2024). Such automation alleviates developers from implementing a large number of unit tests themselves, which can increase code coverage, a term that refers to the amount of source code covered by unit tests. Additionally, applying test-driven development (TDD) could be simplified. Here, developers could use plain text to prompt AI coding assistants with the desired functionality of the code and relevant parameters such as input and output formats and the assistants automatically generate the required test cases. This way, the benefits of this software development paradigm (e.g. focus on design and functionality (Mayr, 2005), reduced debugging effort (Williams, et al., 2003) or better modularity (Madeyski, 2010)) can be utilized while the necessary additional workload is significantly reduced.

2.4 Code Explanation

A common task that occurs in practice for software developers is familiarizing themselves with "foreign" source code. When new employees join a company or switch to new tasks within the same organization, developers may come into contact with large amounts of unfamiliar code. AI coding assistants could reduce the effort required for familiarization by allowing developers to ask specific questions in natural language about the code and receive answers in natural language as well (Bansal, et al., 2021). This could also be beneficial for source code audits, e.g. in the context of open source projects, and the Q&A functionality might allow professional auditors to find critical code parts in less time.

2.5 Code Formatting, Commenting and Documentation

Coding assistants offer support with formatting and documenting code. They can apply coding styles automatically to improve the standardisation, readability and maintainability of the code across larger teams, for example by setting correct indentation, spacing, and parentheses, restructuring the code or unifying variable names. Additionally, they can assist in generating automated comments and documentation blocks (Doerrfeld, 2023). This can promote uniformity of code and documentation, which is especially important in large projects and teams. Furthermore, a comprehensive and understandable documentation is highly valuable for maintenance and modernization of software products.

2.6 Automated Code Translation

Software and IT products often contain important building blocks that were developed in older programming languages (such as Cobol) and are referred to as legacy code. A problem arises when expertise in these languages is lost due to employee fluctuation, and the maintenance of business processes is jeopardized by a lack of (security) updates. In the future, AI coding assistants could help, but current large language models cannot accurately translate older programming languages yet. A study (Pan, et al., 2024) shows that syntactic, semantic, dependency, and logical errors occur in translations. To improve future results, more training data is needed in both source and target languages. However, another study (Weisz, et al., 2022) demonstrates that developers could also benefit from incorrect AI suggestions when translating source code, albeit only for common languages. Future developments in generative AI could enable the translation of outdated languages through specific fine-tuning. There are already first initiatives adressing the translation of memory unsafe languages to memory safe languages, see e.g. the TRACTOR call by DARPA aiming to translate C code to Rust in order to make it more secure (DARPA, 2024).

2.7 Increased Productivity and Employee Satisfaction

Providers often advertise potentially increased productivity when using AI programming assistants. Although productivity in software development is difficult to quantify (Forsgren, et al., 2021), a self-assessment by software programmers can provide insights into employee satisfaction (Meyer, et al., 2021). The analysis of a survey conducted among users by an AI coding assistant provider found that developers felt more productive the more suggestions from the coding assistant they adopted (Ziegler, et al., 2022). A further examination of the data showed that mental workload was reduced and employee satisfaction increased mainly through reducing repetitive tasks (Github, 2024). Another study found no significant shortening of processing time for programming tasks through the use of an AI coding assistant but participants generally preferred it over a traditional autocomplete function (Vaithilingam, et al., 2022). Specifically, the replacement of a laborious online search was highlighted as an advantage by participants. Preliminary results suggest that the use of AI coding assistants can have a positive impact on employee satisfaction. Currently, there is a lack of conclusive evidence regarding productivity enhancement.

3 Risks associated with AI Coding Assistants

A selection of relevant security risks and suitable mitigation strategies is presented below. There are further risks, such as potential licensing violations when using AI-generated code, which have no direct security connection and are not addressed in the following.

3.1 Missing Confidentiality of Inputs

AI coding assistants typically use machine learning for training and fine-tuning. Depending on the product and deployment model, confidential information entered by users may unintentionally flow into the assistant's training data. Although some providers emphasize that they do not systematically store or directly use user input for improving models, this is partly dependent on the monetization model chosen by users. Moreover, it is possible that sensitive code, login credentials, and API keys stored in public registries could be fed into the training (Niu, et al., 2023). Furthermore, there is a possibility that such sensitive information can also be extracted through supply-chain attacks by attackers (see 3.4).

Possible mitigation measures:

- The use of uncontrolled access to coding assistants (e.g. through private accounts of employees) in the cloud should be prohibited for business software development.
- By creating legitimate offers with clear usage rules, shadow IT within the company can be reduced. Using securely configured enterprise accounts is preferred over uncontrolled access.
- When using cloud services, the contractual conditions should be carefully examined, particularly with regard to the further use of your own data for AI training.
- Developers and companies should carefully consider what information they disclose in such tools and, if necessary, anonymize or outsource sensitive data to maintain its confidentiality. This particularly applies to sensitive data such as API or cryptographic keys, but also detailed information about the infrastructure and resource configuration can be sensitive. The last part is particularly relevant if approaches like Infrastructure as Code (IaC) are used. Clear rules in this regard should be defined and communicated within the company in usage guidelines.
- Employees, stakeholders, and committees (e.g., personnel/works council) within companies should be sensitized about the issue.

3.2 Automation Bias

Large language models typically generate linguistically flawless and convincing text, which can foster an impression of human-like abilities and overconfidence in their statements (Automation Bias), leading users to draw false conclusions or accept unverified statements (BSI, 2024). For beginners, AI coding assistants can provide valuable learning resources by offering insights into proven solutions, but they also pose the risk of excessive dependence and uncritical acceptance of generated code, as even flawed solutions are well-worded (Dakhel, et al., 2023; Kabir, et al., 2024). The output based on learned statistical distributions can lead to illogical or impractical suggestions, which only experienced developers can recognize and evaluate (Dakhel, et al., 2023; Chen, et al., 2021). Studies show a cognitive bias when using AI coding assistants, as many developers perceive them as secure, although security vulnerabilities are regularly identified. Some developers even bypass security guidelines to use these assistants (Snyk Limited, 2023; Perry, et al., 2023). The quality of solutions generated through AI assistants depends heavily on the clarity of prompts, with unclear prompts leading to less useful results (Dakhel, et al., 2023). Therefore, the use of AI in programming tasks requires caution and awareness of the system boundaries (Kabir, et al., 2024). Another point is that programmers relying too much on AI assited tools might loose over time the necessary skills to thouroughly evaluate and review source code. Programmers should be able to continue business operations without AI assistance (e.g. in case of incident).

Possible Mitigation Measures:

- Employees within companies should be sensitized to the problem.
- Employees should be trained in "prompting" and "challenging" answers.
- It might be beneficial to "deconstruct" AI-generated code and the used prompts in public code reviews within the company. This allows knowledge and experiences to be shared across teams.

3.3 Lack of Output Quality and Security

AI coding assistants can provide code suggestions based on best practices, but there is a significant risk of generating incorrect, insecure, or inefficient code. A study by Purdue University compared the solutions of a language model for coding tasks with answers from a developer forum and concluded that the model provided incorrect answers 50% of the time (Kabir, et al., 2024). Another study by New York University analysed the security of programs generated by coding assistants and found that about 40% of the programs had security vulnerabilities (Pearce, et al., 2022). When multiple suggestions were generated by the AI for the same prompt, many of the top suggestions (about 39%) were insecure. This is problematic because inexperienced users often trust the top suggestions. One cause of these security flaws is the use of outdated programs in the training data of the AI models, leading to the suggestion of outdated and insecure best practices. This is evident, for example, in password encryption, where insecure methods such as MD5 or a single iteration of SHA-256 are still often used. A study by Stanford University confirmed these security concerns and found that insecure libraries were suggested even when their documentation flagged security concerns (Perry, et al., 2023). Although first mitigation techniques are getting proposed, these techniques are currently limited in their capabilities and lack the generalization guarantees that would allow them to be widely applicable (He, et al., 2023). Moroever, explanations, comments or documentation generated by an AI assistant can be incorrect or completetly hallucinated, which might lead to security issues and reduce code maintenability.

Possible mitigation measures:

- Generated content, in particular source code, should generally be reviewed and understood by the developers.
- Automatic function tests should be employed.
- Additional code reviews by security teams can be another measure to reduce the risk. However, it should be noted that AI-assisted developers might produce more code, so security teams may need to scale accordingly.
- It might be beneficial to "deconstruct" AI-generated code and the used prompts in public code reviews within the company. This allows knowledge and experiences to be shared across teams.
- Automated vulnerability scanners or approaches like chatbots that critically question the generated source code ("source code critics") can reduce the risk.
- It might be beneficial to flag AI generated code blocks and to document the used AI tools. This might help security testing and it can also be a useful information for third-party auditors, e.g. in the context of a security certification process.
- Employees in companies should be made aware of these issues.

3.4 Supply Chain Attacks and Malicious Code

AI coding assistants suggest program code to developers, including appropriate libraries. Attackers can manipulate or exploit coding assistants with the goal of having them generate malicious code, invoke specific malicious libraries, or disclose sensitive information from the prompt. Three examples of such attacks are outlined in the following.

3.4.1 Hallucinations of Methods and Packages

AI coding assistants can use autocompletion to suggest methods and classes to developers that do not exist for the package in question. This is called hallucinating because the training data likely contained methods or classes with similar names but for different packages. When an AI coding assistant generates code that references or recommends non-existent packages or libraries, it is known as package hallucinations. These hallucinations can lead to so-called package confusion attacks (Spracklen, et al., 2024). Attackers exploit the hallucinations by creating a package with the same name as the hallucinated package and tagging it with malicious code. If developers use this incorrect package, it can lead to the entire software supply chain being compromised. Attacks can lead to a variety of consequences, from subtle biases in AI decisions to major malfunctions in critical systems. In one study, researchers examined 16 different code generation models in Python and JavaScript (Spracklen, et al., 2024). Across models, 576,000 code examples were generated with a total of 2.23 million imported packages. They found that 19.7% of imported packages were hallucinated. Depending on the choice of hyperparameters (e.g. temperature) and programming language, there are very large differences between the models in terms of the number of hallucinated packets.

Possible mitigation measures:

- Generated source code should generally be checked and reproduced by the developers.
- Unknown libraries should be checked for plausibility, e.g. when they were created, how commonly they are used or how active a source code repository is.
- If there are guidelines in the company as to which packages can be used as part of a development and which cannot, a whitelisting of permitted packages could be carried out.
- The impact of malicious libraries (such as included ransomware) can be reduced by sandboxing the development environment.
- The creation of a Software Bill of Materials (SBOM) allows you to retrospectively understand whether vulnerable libraries were used and enables a targeted response if a vulnerability of certain components becomes known.
- Employees in companies should be made aware of the problem.

3.4.2 Indirect Prompt Injections

Indirect Prompt Injections are attacks where malicious inputs are inserted into requests to AI models by third parties. These inputs can manipulate the behaviour of AI models and generate undesired outputs. Such attacks aim to cause the AI to perform certain actions or disclose information that would normally be inaccessible (BSI, 2024).

Since AI coding assistants are based on LLMs, prompt injections can pose significant security risks. For example, attackers can write malicious instructions into the documentation of software packages. If these end up in the prompt (possibly in the context window of the LLM, which may not be visible to users), the LLM might respond to the commands and suggest malicious commands or code snippets to developers as part of completion attacks (Greshake, et al., 2023). Another example of prompt injection attacks could come from untrustworthy third-party LLM providers repackaging a legitimate LLM with malicious prompts (Zhang, et al., 2024). This can lead to faulty or harmful code. Targeted prompt injections can also be used by attackers to extract sensitive data by, for example, generating a link that, when accessed, transfers sensitive information to a prepared server. Displaying Markdown images is also a common way to exfiltrate sensitive information in a successful attack (Rehberger, 2024).

Possible Mitigation Measures:

- Providers and manufacturers are particularly responsible for mitigating indirect prompt injections.

  - For instance, they can restrict the display of images to trusted sources to make it harder to exfiltrate information this way.
  - Providers of IDEs with AI plugins should clearly communicate which information can end up in the prompt (context window) of the AI model and allow exceptions (e.g. certain folders).
  - Users should ask their provider what mitigation measures they have taken regarding this attack vector.
- When using regular chatbots, sensitive information (such as API keys) should, if possible, not end up in the prompt.
- Employees in companies should be made aware of the problem, in particular that clicking on AI-generated links may lead to an outflow of sensitive information.
- Generated source code should generally be checked and reproduced by the developers.

3.4.3 Data and Model Poisoning

AI coding assistants are often trained on large source code repositories, like e.g. GitHub or HuggingFace. Attackers might therefore publish code at such plattforms aiming at poisoning the training data, which might lead to generation of insecure code. The poisoning possibilities of AI code assistants are an active research area (Yan, et al., 2024; Oh, et al., 2023; Aghakhani, et al., 2024; Cotroneo, et al., 2024).

Attackers may also have the possibility to poison the model weights directly via supply-chain compromission for example (Li, et al., 2024) .

Possible Mitigation Measures:

- Manufacturers are responsible for mitigating posioning attacks by carefully selecting, securing and analyzing training data.
- If a poisoning attacks becomes public, e.g. due to an incident, documentation of used AI coding assistants (asset management) and libraries (SBOM) might help to identifiy, whether one is affected or not.
- The mitigation measures from Sections 3.3 and 3.4.1 are also applicable in this case.

3.4.4 Extensions for Coding Assistants

Modern coding assistants can often be augmented with extensions which can take actions on behalf of programmers. In this case, employees should pay attention to the security of these extensions and their interactions with applications, especially when they have access to internal resources (e.g. logs, mails).

Possible Mitigation Measures:

- Limit the use of extensions.
- Audit and anticipate impacts of the interactions of these extensions with development, production and CI/CD environments.

3.5 Misuse for Attacks

The ability of LLMs to generate program code can also be used by attackers to generate, modify or improve malicious code. Moreover, the technology can assist in decompiling binaries. A detailed discussion can be found in the respective BSI and NCSC publications (BSI, 2024), (BSI-2, 2024) and (NCSC, 2024).

4 Conclusion and Recommendations

It is generally recommended that companies and developers carefully weigh the opportunities and risks associated with AI coding assistants, adopt a responsible approach to implementing and utilizing them, and focus on continuous improvement and collaboration. This will contribute to maximizing the benefits of this technology while minimizing potential drawbacks. Based on the identified opportunities and risks when using current-generation AI coding assistants, specific recommendations can be formulated for various organizational levels for a responsible and secure handling of the technology. The following lists are not exhaustive but provide baseline suggestions that should be supplemented by company- and situation-specific measures.

4.1 Management

On the management level, the following points should be considered when using an AI coding assistant:

- AI coding assistants are no substitute for experienced developers. An unrestrained use of the tools can have severe security implications.
- Offers should be made to developers to prevent the risk of a shadow IT. For example, local hosting of open-source models can prevent the loss of sensitive information. When using cloud services, controlled and configurable company accounts with clear contract conditions are better than employees using private access. It is recommended to review the provider's contract conditions, particularly regarding data usage (e.g., whether the data are used to retrain and optimize the AI model).
- A systematic risk analysis should be performed before introducing AI tools. This should particularly consider how secrets (e.g., API keys) are currently managed and how they will be protected from leakage in the future.
- Security guidelines should be established in the company and appropriate mitigation measures should be taken. Examples are given in Chapter 3.
- Developers can potentially write program code faster with the help of AI. It is important that other teams in the company, such as quality assurance (AppSec, DevSecOps, etc.), also benefit from similar productivity gains through AI or are scaled up personnel-wise.
- Employees must be sensitized to risks and trained in handling the AI tools.
- Employees must have clear guidelines on which tools they can use with what data for what purposes.
- It is recommended to conduct an evaluation phase with monitoring of relevant key figures (e.g., number of new checked-in code or security team workload), measurable goals, and corresponding success controls to better understand the impact of technology implementation and react to any issues that arise.

4.2 Development

At the employee level, the following points should be taken into account when using an AI coding assistant:

- The responsible use of AI coding assistants requires knowledge of the limitations and security concerns of the systems. The use of AI tools must be carried out in a thoughtful manner, with particular attention to the protection of sensitive information.
- Generated source code should generally be checked and reproduced by the developers. A critical review should be carried out particularly with regard to hallucinations and security risks.
- The employees need to have clear instructions on what tools they are allowed to use with which data and for what purposes. Employees should feel encouraged to seek out the dialogue with colleagues, if they are uncertain about the rules.

- Further training on the effective use of AI coding assistants (keyword prompt engineer) should be attended.
- There should be a lively exchange with colleagues on the topic in order to ensure effective and secure use in the company (consideration of human factors) and to share best practices.

4.3 Research Agenda

Large language models are known for reflecting the problems within their data sets. This also applies to AI coding assistants. By training and fine-tuning on large sets of open source programs, many problems commonly encountered in such programs are reflected in the outputs of the models (Siddiq, et al., 2022; Pearce, et al., 2022). A research goal should therefore be to increase the quality of the training data, with particular attention to reducing security vulnerabilities. Furthermore, data sets are required that are specifically designed for translating from one programming language to another in order to ensure automated source code translation (Pan, et al., 2024). Here, the languages to be translated must be taken into account. The exclusive use of publicly available source code carries the risk that translation models will develop a bias in favor of programming paradigms and syntax of modern programming languages. This bias arises because modern programming languages constitute by far the largest portion of public source code (Github-2, 2022). To reliably translate outdated programming languages today, it is necessary to invest effort in creating meaningful training data. Furthermore, research in the area of automated quality and security control of software must be advanced. If AI coding assistants lead to an increased output of source code, such scaling must also take place within security teams to counteract a significant spread of security vulnerabilities. Finally, it must be stated that with current research results it is difficult to determine whether AI coding assistants can really contribute to increasing productivity in software development. A comprehensive, meaningful and independent study is necessary to examine this claim from the providers.

Bibliography

Aghakhani, Hojjat, et al. 2024. *TROJANPUZZLE: Covertly Poisoning Code-Suggestion Models.* s.l. : arXiv, 2024.

Allamanis, Miltiadis, Jackson-Flux, Henry and Brockschmidt, Marc. 2021. Self-supervised bug detection and repair. *Proceedings of Advances in Neural Information Processing Systems.* 2021.

ANSSI. 2024. RECOMMANDATIONS DE SÉCURITÉ POUR UN SYSTÈME D'IA GÉNÉRATIVE. [Online] 29 04 2024. https://cyber.gouv.fr/publications/recommandations-de-securite-pour-un-systeme-dia-generative.

Bansal, Aakash, et al. 2021. A neural question answering system for basic questions about subroutines. *Proceedings of IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER).* 2021.

Berabi, Berkay, et al. 2021. Tfix: Learning to fix coding errors with a text-to-text transformer. *Proceedings of the International Conference on Machine Learning.* 2021.

BSI. 2024. *Generative AI Models: Opportunities and Risks for Industry and Authorities.* Bonn : Bundesamt für Sicherheit in der Informationstechnik, 2024.

BSI-2. 2024. *How is AI changing the cyber threat landscape?* Bonn : Bundesamt für Sicherheit in der Informationstechnik, 2024.

Chen, Mark, et al. 2021. Evaluating Large Language Models Trained on Code. *arXiv.* 2021.

Cotroneo, Domenico, et al. 2024. *Vulnerabilities in AI Code Generators: Exploring Targeted Data Poisoning Attacks.* s.l. : arXiv, 2024.

Dakhel, Arghavan Moradi, et al. 2023. GitHub Copilot AI pair programmer: Asset or Liability? *Journal of Systems and Software.* 2023.

DARPA. 2024. [Online] Defense Advanced Research Projects Agency, 2024. https://www.darpa.mil/program/translating-all-c-to-rust.

Doerrfeld, Bill. 2023. How Generative AI Can Streamline Code Documentation. *DevOps.com.* [Online] 5 December 2023. [Cited: 13 June 2024.] https://devops.com/how-generative-ai-can-streamline-code-documentation/.

Forsgren, Nicole, et al. 2021. The SPACE of Developer Productivity: There's more to it than you think. *Queue.* 2021, Vol. 19, 1.

Github. 2024. Research Quantifying Github Copilots Impact on Developer Productivity and Happiness. [Online] 21 May 2024. [Cited: 23 May 2024.] https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/.

Github-2. 2022. The top programming languages. [Online] 2022. [Cited: June 03, 2024.] https://octoverse.github.com/2022/top-programming-languages.

Greshake, Kai, et al. 2023. *More than you've asked for: A Comprehensive Analysis of Novel Prompt Injection Threats to Application-Integrated Large Language Models.* s.l. : arXiv, 2023.

He, Jingxuan and Vechev, Martin. 2023. Large Language Models for Code: Security Hardening and Adversarial Testing. *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security.* 2023.

Jimenez, Carlos E, et al. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues? *The Twelfth International Conference on Learning Representations.* 2024.

Kabir, Samia, et al. 2024. Is Stack Overflow Obsolete? An Empirical Study of the Characteristics of ChatGPT Answers to Stack Overflow Questions. *Conference on Human Factors in Computing Systems.* 2024.

Li, Yige, et al. 2024. *BackdoorLLM: A Comprehensive Benchmark for Backdoor Attacks on Large Language Models.* arxiv : s.n., 2024.

Madeyski, Lech. 2010. *Test-driven development: An empirical evaluation of agile practice.* Heidelberg : Springer, 2010.

Mayr, Herwig. 2005. *Projekt Engineering Ingenieurmässige Softwareentwicklung in Projektgruppen.* München : Fachbuchverlag Leipzig im Carl-Hanser-Verlag, 2005. ISBN 978-3446400702.

Meyer, André N., et al. 2021. Today Was a Good Day: The Daily Life of Software Developers. *Today Was a Good Day: The Daily Life of Software Developers.* 2021, Vol. 47, 5.

NCSC. 2024. *The near-term impact of AI on the cyber threat.* s.l. : National Cyber Security Centre UK, 2024.

Niu, Liang, et al. 2023. CodexLeaks: Privacy leaks from code generation language models in GitHub copilot. *32nd USENIX Security Symposium (USENIX Security 23).* 2023.

Oh, Sanghak, et al. 2023. *Poisoned ChatGPT Finds Work for Idle Hands: Exploring Developers' Coding.* s.l. : arXiv, 2023.

Pan, Rangeet, et al. 2024. Lost in Translation: A Study of Bugs Introduced by Large Language Models while Translating Code. *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering.* 2024.

Pearce, Hammond, et al. 2022. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. *IEEE Symposium on Security and Privacy.* 2022.

Perry, Neil, et al. 2023. Do Users Write More Insecure Code with AI Assistants? *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security.* 2023.

Popper, Ben. 2023. Self-healing code is the future of software development. *Stack Overflow.* [Online] 28 Dezember 2023. [Cited: 14 Juni 2024.] https://stackoverflow.blog/2023/12/28/self-healing-code-is-the-future-of-software-development/.

Pradel, Michael and Sen, Koushik. 2018. Deepbugs: A learning approach to name-based bug detection. *Proceedings of the ACM on Programming Languages.* 2018.

Rehberger, Johann. 2024. [Online] 14 June 2024. [Cited: 23 May 2024.] https://embracethered.com/blog/posts/2024/github-copilot-chat-prompt-injection-data-exfiltration/.

Siddiq, Mohammed Latif, et al. 2022. An Empirical Study of Code Smells in Transformer-based Code Generation Techniques. *Proceedings of IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM).* 2022.

Smith, Craig S. 2024. The 15 Top AI-Powered Tools For Automated Unit Testing. *Forbes.* [Online] 7 April 2024. [Cited: 13 Juni 2024.] https://www.forbes.com/sites/technology/article/unit-testing/.

Snyk Limited. 2023. AI Code, Security, and Trust: Organizations Must Change Their Approach. 2023.

Spracklen, Joseph, et al. 2024. *We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs.* 2024.

Stripe. 2018. *The Developer Coefficient.* 2018.

Vaithilingam, Priyan, Zhang, Tianyi and Glassman, Elena L. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Model. *CHI EA.* 2022.

Weisz, Justin D., et al. 2022. Better together? an evaluation of ai-supported code translation. *27th International conference on intelligent user interfaces.* 2022.

Williams, Laurie, Maximilien, Michael E and Vouk, Mladen. 2003. Test-driven development as a defect-reduction practice. *14th International Symposium on Software Reliability Engineering.* 2003.

Yan, Shenao, et al. 2024. *An LLM-Assisted Easy-to-Trigger Backdoor Attack on Code Completion Models: Injecting Disguised Vulnerabilities against Strong Detection.* arXiv : s.n., 2024.

Yeadon, Will, Peach, Alex and Testrow, Craig P. 2024. A comparison of Human, GPT-3.5, and GPT-4 Performance in a University-Level Coding Course. *arXiv preprint arXiv:2403.16977.* 2024.

Zhang, Rui, et al. 2024. Instruction Backdoor Attacks Against Customized LLMs. *Proceedings of the 33rd USENIX Security Symposium.* 2024.

Ziegler, Albert, et al. 2022. Productivity Assessment of Neural Code Completion. *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (MAPS '22).* 2022.