AI-assisted code generation tools

University of Oulu
Information Processing Science
Master's Thesis
Joonas Nygård
2024

Abstract

The rapid advancement of Artificial Intelligence (AI) has significantly impacted various domains during the recent years. In the field of software engineering, AI-assisted code generation tools have been revolutionizing conventional practices. This thesis delves into the impact of these tools on software engineers' work. The goal was to address a gap in existing literature by capturing the users view on these tools by analysing the user reviews for AI-assisted code generation tools. Data collection and analysis for Socio-Technical Grounded Theory (STGT) method was followed to extract categories from the user reviews.

The findings of this study support the understanding that these tools improve software professionals' productivity, learning and even satisfaction in work. From quality aspect, the users value the usability and correctness, while unsuitable business models or high price can cause deep resistance in the userbase. From different features, the users valued most the code completion, chat assistant, and documentation features.

This study provides valuable insights into users' perspective on AI-assisted code generation tools, offering new paths for future research. Notably, the research highlights the potential influence of these tools on software professionals' well-being and satisfaction, as well as the emerging discourse surrounding their business and ethical implications.

Keywords
AI, Artificial Intelligence, ML, Machine Learning, LLM, Large language model, Big Code, OpenAI, GPT, CGT, Generative pre-trained transformer, Codex, Code Generation, Test Generation, Code explanation, Exercise generation, Program synthesis, Pair programming, Generative AI, GitHub Copilot, Amazon CodeWhisperer, Tabnine, Codeium, Socio-Technical Grounded Theory, STGT

Supervisor
Professor Burak Turhan

Abbreviations

AI	Artificial Intelligence
APR	Automated Programming Repair
CPU	Central Processing Unit
DCA	Data Collection and Analysis
GPT	Generative Pre-trained Transformer
GT	Grounded Theory
IDE	Integrated Development Environment
ML	Machine Learning
NLP	Natural Language Processing
PBE	Programming By Example
RAM	Random-Access Memory
SE	Software Engineering
STGT	Socio-Technical Grounded Theory

Foreword

I had become familiar with the workflow of programming and debugging from personal hobby projects all the way from year 2015. When I started my journey as a software developer in 2020, I had no clue how much the workflow I had learned could change during the next few years due to rapid advancement of AI. AI assistants were also a hot topic at work around 2022, and employers were providing training and licenses for these tools to improve productivity.

Although there was a lot of hype around the topic, the people's adaptation to start using these tools seemed slow in my view. I had also my suspicions and it took way too long before I even tried some of the AI-assisted code generation tools. To get rid of these prejudices, I wanted to make thesis about this topic. I wanted to gain my, and my collogues knowledge, on the different ways these tools could help us, and what are the strengths and deficiencies these tools have. Some single negative comment had most likely caused my prejudices. That's why I thought it would be important to capture the users collective view by using a large sample.

I want to thank my supervisor, Professor Burak Turhan for the feedback and guidance in the decisions I had to make in the turning points of the research process. Thanks also for my fellow students for reviewing my thesis draft and giving valuable feedback for improvements on multiple occasions. Thanks for my collogues for sparking up the interest towards the subject and supporting the topic selection. Also, thanks for the software developer community for writing the user reviews, which were the core of this thesis. Lastly, I want to thank my family and friends. Completing studies and writing thesis can be lonely at times, and I want to thank you for your presence and support during these times.

Joonas Nygård, 7.6.2024

Contents

Abstract 2
Abbreviations 3
Foreword 4
1. Introduction 6
2. Previous literature 9
2.1 Impact of LLMs on society 9
2.2 Code Correctness, Complexity and Quality 9
2.3 Interaction 10
2.4 Education 11
2.5 Security, Privacy, and Ethics 11
3. Research method 13
3.1 Socio-Technical Grounded Theory 14
3.2 Research questions 15
3.3 Data collection 15
3.4 Data analysis 16
4. Results 18
4.1 Categories in numbers 20
4.1.1 Impact on programming 21
4.1.2 Satisfaction on tool quality 23
4.1.3 Features 28
4.2 Tool specific differences 29
4.3 Summary 32
5. Discussion 34
5.1 Impact on programming 34
5.2 Tool quality 35
5.3 Features 36
5.4 Limitations 37
5.5 Implications 37
6. Conclusion 39
References 40
Appendix A. Positive category occurrences 44
Appendix B. Negative category occurrences 45

1. Introduction

In recent years, AI generation tools leveraging LLM (large language models) and NLP (natural language processing) have increasingly emerged in our daily lives. The enabling factor for this has been the transformer architecture, introduced in 2017 by Google's researchers (Vaswani et al., 2017). Shortly, the first version of GPT-1 (Generative Pre-trained Transformer) was released in 2018 by OpenAI. Transformer architecture significantly advanced the field of NLP. Professionals across various fields are increasingly recognizing the integration of AI-based tools into their daily workflows.

For majority of the software developers the most concrete example of this change is probably the AI-assisted code generation tools. Developers' opinions on these tools differ, some say that they have increased their productivity massively while others see that they only cause additional work by generating buggy code. While new innovations may have their drawbacks, the benefits often become so evident that the majority eventually adopts them. AI-assisted code generation tools can help programmers to improve their performance by not having to concentrate on writing trivial code manually (Kazemitabaar et al., 2023), but also help to make better architectural decisions, write less buggy and secure code, (Asare et al., 2023) or even automatically generate documentation or automated tests. Number of AI-assisted code generation tools have been introduced during the last few years, and their capabilities have been researched from various angles. To gain insight on this emerging phenomenon from users' perspective, I will take a closer look on the user reviews these tools have received in the IDE plugin marketplaces by following the guidelines of Socio-Technical Grounded Theory (STGT) (Hoda, 2022).

To first understand how AI could help software developers daily work, it's useful to understand what the typical workflow of software developers programming work is. Software developers encounter daily numerous amounts of small problems. Fortunately, the solutions can be usually found quite easily after conducting a fast internet search. Internet has gained over the years of its existence a massive amount of open-source code, code snippets and code examples on various websites. The problems developers' encounter in their daily work are usually non-unique. When a programmer encounters a problem and conducts a web search, they usually find a discussion about someone else struggling with the same issue. Sometimes there is an open-source library which developer can just import into the project, or someone has posted a code snippet, which developer just copies into code editor with minor changes. This phenomenon has name called *Example Embedding* or *Example-centric Programming*. In empirical studies it has been discovered that manual searching from web can take up to 35% of software developers' worktime. (Heyman et al., 2021)

AI-assisted code generation tools could potentially automate the process of *example embedding* and thus improve developers' productivity. Term "Big Code" refers to large-scale collection of online software artifacts such as massive collections of source code (Wong et al., 2023). Big Code data can be used as training data for machine learning techniques such as LLM and NLP. With combination of these techniques, we can construct probabilistic models of extensive codebases, which can calculate the most probable solution for a specific problem (Wong et al., 2023). Software engineers can utilize AI-tools based on these techniques for example to automatically complete and generate code which will admittedly increase their productivity. GitHub Copilot, Amazon CodeWhisperer, Tabnine and Codeium are popular implementations of such AI-assisted

code generation tools which are available as a plugin for major IDEs such as Visual Studio, Visual Studio Code and JetBrains IDEs.

- **GitHub Copilot** was released by GitHub and OpenAI in June 2021 as IDE plugin for Visual Studio Code, JetBrains and NeoVim. Copilot offers features such as code auto completion and code generation. It is powered by OpenAIs Codex AI model, which parses natural language and generates code as a response. Codex is built on OpenAIs GPT-3 model. GitHub states that Copilot is trained on all languages that can be found in the public repositories, so it should support majority of the programming languages, although more popular languages will have better support, as they have more training data. Copilots basic version subscription costs 10$/month for one user, but students can get it for free. (*GitHub Copilot · Your AI Pair Programmer · GitHub*, 2024)
- **CodeWhisperer** by Amazon was released in April 2023. It is included in AWS Toolkit plugin for VS Code and JetBrains. Amazon promotes that its prominent feature is its ability to provide suggestions while coding. CodeWhisperer can also perform security scans for your project files. The LLM CodeWhisperer uses was trained on billions of lines of open-source code and code by Amazon. Amazon states that in terms of quality of the training data CodeWhisperer supports best following programming languages: Java, Python, JavaScript, TypeScript, C#, Go, PHP, Rust, Kotlin, SQL. Basic version of CodeWhisperer for individuals is free. (*AI Code Generator - Amazon CodeWhisperer - AWS*, 2024)
- **Tabnine** released its first version of AI coding assistant plugin for VS Code in October 2018. Tabnine assists programmer with code completion and chat interface that can automate various software development tasks. Tabnine can write new code, accelerate unit testing, extend, and refactor existing code, understand existing legacy code, generate documentation, and generate boilerplate code. Tabnine states that it supports all the major programming languages like JavaScript, Python, Java, Typescript c/c++. Tabnine is available as plugin for Visual Studio, Visual Studio Code, JetBrain IDEs and Eclipse. Tabnine has basic version which is free for everyone. (*Tabnine AI Code Assistant | Private, Personalized, Protected*, 2024)
- **Codeium** was released in October 2022. Codeium offers code auto completion, chat feature, and it has also a full repository awareness. Codeium provides a comprehensive list of 70 programming languages it considers having a good performance. Codeium promotes that it has support for 40+ IDEs including VS Code, JetBrains, NeoVim and Xcode. Codeium is free for individual developers. (*Codeium · Free AI Code Completion & Chat*, 2024)

AI-assisted code generation applications and their features can be divided into three main paradigms: *description-to-code, code-to-description,* and *code-to-code*. Applications under *description-to-code* paradigm generates code based on non-code input. This paradigm can be divided into four categories based on the input: NL description, PBE (Programming by example), images and other structured data inputs. *Code-to-description* paradigm in most cases means automated documentation generations based on code input. Applications under *code-to-code* paradigm generate code based on existing code. This paradigm can be divided into four categories: APR (automated program repair), Cross-PL (programming language) translation, refactoring, and code completion. (Dehaerne et al., 2022)

Various studies evaluating the AI-assisted code generation tools, and the models behind them have been made to understand the capabilities of these technologies on different aspects. The correctness and complexity of the output generated by tools have been studied by Chen et al., 2021; Finnie-Ansley et al., 2022; Nguyen & Nadi, 2022; Wermelinger, 2023. The usability and interaction between the tool and programmers have been studied by Barke et al., 2023; Liu et al., 2023; Prather et al., 2023; Vaithilingam et al., 2022. Security aspects have been addressed in studies by Asare et al., 2023; Pearce, Ahmad, et al., 2021; Pearce, Tan, et al., 2021; Schuster et al., n.d. Effect and opportunities of generative AI for teaching programming and generating programming exercises have been studied by Becker et al., 2023; Kazemitabaar et al., 2023; Sarsa et al., 2022. Automatic documentation generation have been studied by Khan & Uddin, 2022. Comprehensive literature reviews about Code Generation and AI-assisted programming has been made by Dehaerne et al., 2022; Wong et al., 2023.

Although there are papers that address the usability aspect taking users opinions into account in smaller setting, no papers can be found that would have studied which are the main themes that arise in wide scale from the massive amount user reviews found from the internet. This thesis strives to answer the following research questions by analysing user reviews posted on the IDE plugin marketplaces for AI-assisted code generation tools:

- RQ1: What effects do the AI-assisted code generation tools have on software engineers work?
- RQ2: Which quality factors influence users' opinions on the tools?
- RQ3: What features do users value most in the tools?

The rest of the thesis continues following: In the second chapter we will take brief a look on the previous literature. In chapter three I will introduce the research methods and in chapter four the results are presented. In chapter five I will discuss the findings and limitations and conclude the thesis in chapter six.

2. Previous literature

A lean literature review was conducted following the STGT process to understand the current state of research and identify any existing gaps. While AI-driven code generation is a relatively new phenomenon, there is a considerable amount of literature on the subject, approaching the topic from different perspectives with different research methods.

2.1 Impact of LLMs on society

The goal of the first research question is to gain the understanding on how AI-assisted code generation tools impact on software engineers' work. To get touching point on the subject, we should look on a broader level how AI and LLMs are impacting the society and the way of working in different domains. Raiaan et al., 2024 conducted a review of recent changes in LLMs, covering 135 papers. Their paper provides a comprehensive summary and overview of LLMs history, architectures, training methods, applications, challenges and impact on society. They summarized the impacts of LLMs on society to and divided them into positive and negative sides. Positive impacts were advancements in NLP, automation and efficiency, creative tools, virtual assistant and chatbots, medical and scientific research, accessibility, personalization, education and skill development. On the negative side they listed ethical concerns, misinformation & disinformation, job displacement, data privacy, economic impact, regulation and accountability. These categories can be referenced when forming the categories from the user reviews.

Impact on efficiency and productivity is one of the main positive impacts LLMs have had and will have on society. Various roles on different domains include tasks that are perhaps not that complex, but time-consuming and labour-intensive. LLMs can automate these time-consuming and labour-intensive tasks that were previously had to be done manually by humans. Industries like customer support, content generation and data analysis are few examples of domains which have benefited greatly from LLMs in terms of productivity. On the other side of the coin is the job displacement, as there is no need for human workers for those tasks that can be handled fully by AI (Raiaan et al., 2024). This requires adaptation from the people who working in these positions. These are impacts that can be applied also for the field of software engineering. As the nature of software engineering changes and productivity of the developers utilizing AI increases, it might become difficult to keep up with the competition without adopting AI in the workflow.

2.2 Code Correctness, Complexity and Quality

The code correctness, with the complexity and quality are perhaps the most studied areas of the AI-driven code generation tools, with no excuse. Without correct outputs the tools would be completely useless. When Codex was released, the team of OpenAI (Chen et al., 2021) evaluated Python code generated by early version of Codex with 164 programming problems. 28,8% of the problems were solved correctly within the first attempt and with repeated sampling, using 100 samples they got at least one correct solution with rate of 70,2%. They noted also that Codex had difficulties with binding operations to variables and with docstrings describing long chains of operations.

Nguyen & Nadi, 2022 evaluated correctness and cyclomatic complexity of solutions Github Copilot generated for 33 LeetCode questions. They performed the study with four

programming languages: Python, Java, JavaScript, and C. The correctness was evaluated by LeetCodes automated tests and the complexity by SonarQube's cyclomatic complexity and cognitive metrics. Java suggestions had the highest correctness percentage (57%) and JavaScript had the lowest correctness (27%). Complexity of the suggested solutions was low for all the four programming languages. As potential shortcomings they noted that some of the solutions Copilot created were simplified and relied on undefined helper methods.

Yetistiren et al., 2022 conducted an experimental setup to evaluate code quality generated by GitHub Copilot in terms of validity, correctness, and efficiency. By providing docstring and function prototype as input, Copilot was able to generate valid code for 91,5% of the problems in HumanEval dataset. 28,7% of the solutions were correct, 51,2% were partially correct and 20,1% were incorrect. They also evaluated the impact of given input parameters for Copilot. When only function and parameter names were provided without docstring, the validity rate dropped to 79,3, correctness to 19,5%, partially correct to 26,8% and incorrect solutions increased to 53,7%. With a docstring and dummy function names, the validity rate was 84,2%, correctness rate 26,8%, partially correct 51,2% and incorrect 19,7%. Regarding efficiency, the solution Copilot created was mostly as efficient as the canonical solution in terms of time & space complexity.

Finnie-Ansley et al., 2022 studied how Codex performs on typical introductory programming problems. They used two sets of CS1 problems in python. As the results by Codex were compared on results made by students answering the same questions, the Codex seemed to perform better than students. For the first set, out of 23 problems it solved 10 problems within the first attempt and 4 problems it could not solve at all. For the second set with rainfall problems Codex solved six out of seven problems within 50 attempts. They noted also that there was much variation in the generated solutions by Codex, as identical input could lead to very different solutions.

Wermelinger, 2023 performed a qualitative study to understand GitHub Copilots limitations. They compared how Copilots responses differed from the responses of Codex's most performant model; DaVinci. They also reported on experiences using Copilot to explain code, generate tests and fix bugs. They noted that Copilot performed poorer in all areas compared to Codex and Copilots answers often were wrong. They however stated that Copilot can provide helpful starting point for solving the problem, but the user still needs to understand the languages syntax and semantics well in order to fix the incorrect suggestion Copilot generates.

2.3 Interaction

Vaithilingam et al., 2022 conducted user study with 24 programmers to understand how they use GitHub Copilot and does it improve their productivity. They found out that use of Copilot did not actually improve their task completion time or success rate. However, programmers still preferred using Copilot, as it provided starting point for the solution without additional online search. The reason for poor performance with Copilot was that the participants had issues on understanding, debugging, and editing the code generated by Copilot.

Barke et al., 2023 conducted a grounded theory analysis to find out how programmers interact with Copilot. They observed 20 programmers with different amount of experience in using Copilot. The main finding of the study was that the participants used Copilot mainly in two modes: *acceleration* and *exploration.* In *acceleration mode*

participants knew what code should be written and they just used Copilots automation to complete the implementation faster. In *exploration mode* programmers were unsure what should be done, and they used Copilot to explore the options Copilot suggested for them to get started.

Prather et al., 2023 studied how novice programmers interact with code generation tools by observing students using Copilot on introductory programming assignment. Students had issues understanding and using Copilot, but they thought it would be useful for them in the future.

2.4 Education

The effect of AI-driven code generation on programming education is also a much-discussed topic. How should the teachers take it into account the possible challenges it brings and is there some value including it into the education. Becker et al., 2023 position paper argues that community needs to decide quicky what opportunities should be leveraged from it. Also, preparation for the possible challenges should be done. Their primary concern is that without quick and concrete efforts, educators will lose the advantage of shaping opportunities and face challenges.

Sarsa et al., 2022 explored NLPs capabilities by generating learning resources for programming courses with OpenAI:s Codex. The learning resources included programming exercises and code explanations. The result of the study was that majority of the generated learning resources were sensible and novel and some even ready to use as is. Their analysis suggests that there is significant value in the generative machine learning models as a tool for instructors, although the quality of the generated content should be always monitored and reviewed before using it as actual learning material.

Kazemitabaar et al., 2023 conducted a controlled experiment with 69 novice programmers to learn about the implications that AI code generators have on introductory programming. Half of the research subjects were able to use Codex to solve their programming tasks, while the other half could not. The results of the study showed that Codex significantly increased the code authoring performance. The increase in completion rate was 1.15x and 1.5 in scores. They also conducted a post-test evaluation week later, and the group with access to Codex performed lightly better on that also, although the difference was not statistically significant.

2.5 Security, Privacy, and Ethics

Multiple studies have addressed the security of AI based code generation tools. Code written by humans often contains bugs, and this code is then used as training data for the AI models. This leads to concern that AI tools probably will make similar mistakes as humans do, which could lead to unsecure vulnerable software.

Pearce et al., 2021 studied if GitHub Copilot suggest insecure code and how prevalence insecure suggestions are. They used Copilot in 89 scenarios relevant to high-risk cybersecurity weaknesses and resulted in 40% of the solutions being vulnerable. AI based tools like Copilot will definitely increase software developers' productivity, but the developers should not trust blindly the code Copilot generates. Software developers should remain awake and ideally use security-aware tooling with Copilot to minimize the

risk of introducing new security vulnerabilities (Pearce, Ahmad, et al., 2021). Pearce, Tan, et al., 2021 researched also if LLM tools could be used to fix security vulnerabilities in software. They performed a study with five black box LLMs and came to result that LLMs did collectively repair 100% of their synthetically generated and hand-crafted scenarios.

Asare et al., 2023 study aimed to determine if Copilot is as bad as human developers from a security perspective. They found out that Copilot replicated the original vulnerabilities in 33% of the cases and fixed the code in 25% of the cases, although the behaviour was not consistent. They noticed also that Copilot more likely introduces some types of vulnerabilities more likely than others. Their conclusion was that Copilot is not as bad as human developers in security perspective.

The ethical aspect of LLM training data is largely discussed topic regardless the domain. The LLMs AI-assisted code generation tools use requires a large amount of training data as source code. Open-source code is mainly used to train these models. This has raised some concerns and dissatisfaction in open-source community, as companies benefit financially from these tools by charging a fee from individual users. Also, security and privacy issues are present when open-source data is used to train LLMs.

Al-Kaswan & Izadi, 2023 discussed the security, privacy and licensing implications considering the use on open-source code to train LLMs. They argued that use of copyleft code to train LLMs is legal and ethical dilemma. They also provided recommendations to address these issues. They argued that massively mined code datasets are not sanitized and thus can contain large number of biases. Privacy concerns also arise as the open-source repositories can contain private information which has accidentally slip to public repository. This information can be deleted from the public repository, but it is much harder to delete when it has been used as training data for LLM. With query access to these models, adversary could potentially extract this private data.

Licensing issues are also reality, and lawsuits have been filed against companies. Open-source-code is licensed under two types of licenses. Permissive licenses allow to use, modify, and distribute software for any purpose with no requirement the user to share their work. Non-permissive licenses, which are also called as “copyleft” licenses, require that the user freely shares their own software under the same license when distributing software. Thus, creating closed or commercial software based on open-source code made under this license is unethical and possible illegal. (Al-Kaswan & Izadi, 2023)

To mitigate these ethical issues Sun et al., 2022 argued that there is need for mechanism for protecting open-source code from being exploited by these deep learning models. They designed a prototype called CoProtector, which defends source code repositories from such exploits using data poisoning techniques.

3. Research method

The main research method followed in this thesis is Socio-Technical Grounded Theory (STGT), which is a modern socio-technical version of traditional Grounded Theory (GT) method (Hoda, 2022). The traditional GT is a research method which goal is a theory that captures the key patterns, concepts, and relationships in a set of data. We can find thousands of user reviews for various AI-assisted code generation tools, and it would be valuable to capture the common thoughts the software engineers share on this emerging phenomenon changing completely their way of working. GT method is well suited for this kind of broad topics, as it progressively narrows down and captures the key phenomena from a large dataset with its systematic and rigorous data analysis process.

Grounded Theory has become a popular research method in the field of Software Engineering especially to investigate complex phenomena in human and social aspects. However, GT studies in SE have received critique about being low in quality. In most of the cases the issue has been that GT has not been used correctly as the understanding of the method has been lacking (Stol et al., 2016). One reason for the poor application of GT method in SE is that the GT was originally designed by sociologists to study social phenomena, while SE studies are done by socio-technical researchers to study socio-technical phenomena. (Hoda, 2022)

This study falls also under the frames of sample study, which is one of the most used research strategies in SE research. Sample study is one of the eight archetypal research strategies of ABC framework for SE. Sample study aims for generalizability over a certain population of actors, and it has the potential to maximize generalizability to a population. In sample studies with human respondents, the data is usually collected through questionnaires or websites and the researcher does not have any control over the respondent (Stol & Fitzgerald, 2018).

Figure 1 provides an overview of the main stages of the research process and data analysis followed in this thesis. In stage 1 the basic stage of STGT is followed to extract the categories from the user reviews and in stage 2 the frequencies of the categories are counted from the sample. As a result, we have categories which can be arranged by the attention they received among the reviewers.

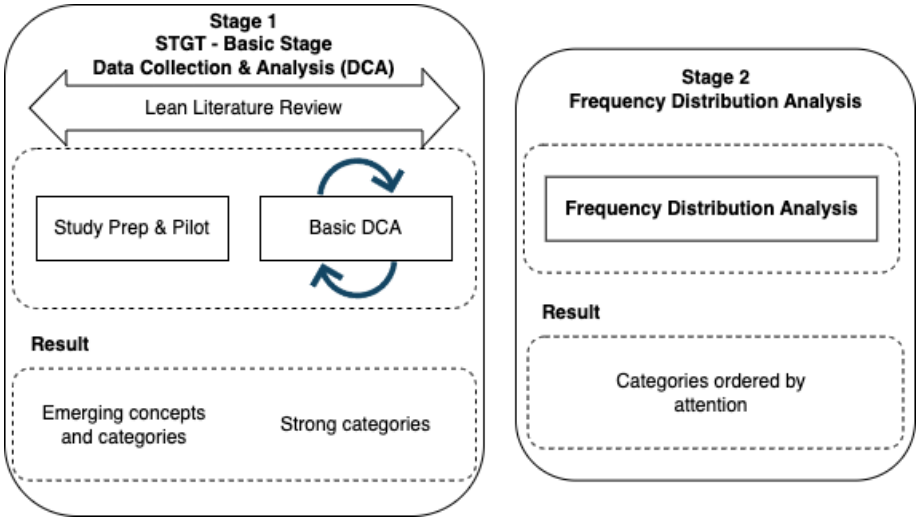

Figure 1. Overview of the research process

3.1 Socio-Technical Grounded Theory

STGT is a variation of the original GT method, which is customized specifically for socio-technical research to study socio-technical phenomena. STGT is easier to understand and apply by SE researchers and it offers a simple visualized process diagram for following the method. A full STGT study consists of basic data collection, analysis and advanced theory development phases which leads to a mature theory. STGT is iterative and incremental, and the data collection and analysis is done in cycles. The outcome of STGT study is novel, useful, parsimonious, and modifiable theory. STGT, as well as the traditional GT is inductive research method, so the theory is built from bottom to up although it also acknowledges deduction and abduction. Although STGT method is considered to be easier to follow by novice researchers, conducting research with it requires a set of skills; Philosophical foundations, qualitative research skills, theoretical sensitivity, socio-technical sensitivity, and domain knowledge. (Hoda, 2022)

STGT study has two stages: Basic stage for data collection and analysis (DCA) and an optional Advanced stage for theory development. **Basic stage** starts with a lean literature review. It is lightweight, and its purpose is to identify gaps in the research and motivate a need for a study. Literature review is followed by a study prep and a pilot. Already at this stage, the initial concepts and categories should start to stand up from the data. The next step is the basic Data Collection and Analysis (DCA) where data is iteratively collected and analysed in cycles. Each cycle consists of following steps: 1. (Theoretical) Sampling 2. Basic data collection 3. Basic data analysis (Open coding & constant comparison) 4. Basic memoing. During this step strong concepts and categories start to take shape. The iterative nature in data collection and analysis gives the researcher freedom to decide what data would be important to collect next, based on the analysis done for the data collected earlier. The final output of the basic stage are the concepts and categories once the saturation is reached, and there is no point to continue the DCA cycle anymore. (Hoda, 2022)

In the **advanced stage** there is two options for the theory development: **1. Emergent mode** and **2. Structured mode**. Emergent mode fits for broad topics and if the theoretical structure is still unclear after the basic stage. In emergent mode you continue conducting targeted DCA iteratively until theoretical saturation is achieved. Targeted DCA includes steps: 1. Theoretical sampling, 2. Targeted data collection, 3. Targeted data analysis, 4. Advanced memoing. The final step in emergent mode is theoretical structuring step where the final outcome is structured in a presentable format, e.g. a diagram. (Hoda, 2022)

Structured mode fits better for narrow and specific topics, or if the structure of the theory is already clear after the basic stage in a sense that the concepts and categories will not change anymore. However, there might still lot more to learn about each of these concepts or categories. In structured DCA you will study in depth each of the categories and saturate each of the categories separately. The final step in structured mode is theoretical integration. Whether the emergent or structured mode is used, the outcome of the study will be a mature theory. A targeted literature review is final part of the advanced stage of STGT. In the targeted literature review, only the literature related to the categories and concepts of your theory is reviewed. It gives a chance to compare your findings with the existing literature and situate the findings into a larger context. (Hoda, 2022)

3.2 Research questions

The categories that emerged from the data analysis led to the following research questions:

- RQ1: What effects do the AI-assisted code generation tools have on software engineers work?
  - RQ1.1 How do they improve the software engineering work?
  - RQ1.2 What possible negative effects the tools have on software engineering?
- RQ2: Which quality factors influence users' opinions on the tools?
  - RQ2.1 Which quality factors cause positive reviews?
  - RQ2.2 Which quality factors cause negative reviews?
- RQ3: What features do users value most in the tools?
  - RQ3.1 Which features got most positive feedback?
  - RQ3.2 Which features users think the tools are missing?

The RQ1 has been addressed in past studies, but it would be valuable to find out if the larger sample could capture something new related to this question. The results on RQ2 and RQ3 would be beneficial especially for the developers of AI-assisted code generation tools, as they address the things that users are currently satisfied with the tools, and the things which users find problematic with the tools.

3.3 Data collection

The data collection started from two major IDE marketplaces; 1. Visual Studio Marketplace and 2. JetBrains Marketplace, as both had reasonable number of reviews for all the major tools available. Both marketplaces also provided an API where the reviews for each plugin could be easily retrieved in JSON format, although both APIs were limited to fetching only 100 items in a single request. All reviews for each tool in each marketplace were fetched and pasted into a single JSON file and then imported into Excel to wait for further analysis.

Both marketplaces provided at least following valuable information considering the study: **1. Review date, 2. Comment, 3. Rating from 1 to 5**. Visual Studio marketplace reviews also contained information about product version, which could also be possibly considered as valuable information. Visual Studio marketplace provided separate plugins and plugin reviews for Visual Studio and Visual Studio Code while JetBrains marketplace provided single plugin which was compatible with majority of the JetBrains family IDEs. Tools selected for the analysis were GitHub Copilot, Tabnine, Amazon Codewhisperer and Codeium as these plugins had the most installations and reviews in the marketplaces. (Visual Studio Marketplace, 2024; IntelliJ IDEs Plugin | Marketplace, 2024).

Table 1 The number of collected reviews for each plugin

Tool	IDE	Reviews
GitHub Copilot	Visual Studio	179
GitHub Copilot	Visual Studio Code	999
GitHub Copilot	JetBrains IDEs	577
Tabnine	Visual Studio	11
Tabnine	Visual Studio Code	548
Tabnine	JetBrains IDEs	179
Amazon CodeWhisperer	Visual Studio	17
Amazon CodeWhisperer	Visual Studio Code	63
Amazon CodeWhisperer	JetBrains IDEs	119
Codeium	Visual Studio	23
Codeium	Visual Studio Code	815
Codeium	JetBrains IDEs	373
Total		**3903**

Visual Studio marketplace provides plugins for Visual Studio and Visual Studio Code IDEs and JetBrains Marketplace provides plugins for several JetBrains IDEs. All selected JetBrains plugins were compatible with following IDEs: IntelliJ IDEA (Ultimate, Community), Android Studio, AppCode, Aqua, CLion, Code With Me Guest, DataGrip, DataSpell, GoLand, JetBrains Client, MPS, PhpStorm, PyCharm (Professional, Community), Rider, RubyMine, RustRover, WebStorm). In addition, Tabnine, Amazon Codewhisperer, and Codeium were compatible with JetBrains Gateway.

The reviews collected were posted between timeframe of November 2018 and February 2024. All the reviews for the selected tools on the marketplaces were included in the sample. The oldest reviews from 2018 were posted for Tabnines VS Code plugin and AWS Toolkits JetBrains plugin. The first review for GitHub Copilot was posted on January of 2021 for VS Code plugin. For Codeium the first review can be found from November of 2022.

3.4 Data analysis

The reviews for each plugin were sorted in an Excel based on the ratings, as it would ease capturing separately the positive and negative experiences with the tools. In the first-round open coding was applied. Reviews were read one by one and whenever something unique was pointing out in the comment, it was added into list of codes. At some point there was no new unique data popping up in the comments, meaning theoretical saturation was achieved. At this point the initial codes were refined, and axial coding was conducted

by placing the emerged categories under broader parent categories. During the analysis, the reviews without comments, or spam message were ignored, and reviews written in foreign language were translated with Google Translate.

To give an example how the categories were identified in practice, I'll describe how I came up with category *Increased productivity*. This category was such strongly represented in the reviews that I came up with it already in the study preparation & pilot phase. As I was skimming the reviews on the marketplace, productivity increase was popping up frequently in the comments, so I added it to the list of initial categories. The subcategories for this category were included at later phase when conducting open coding and constant comparison for the positive reviews in Excel. *Less typing* was the strongest category that got identified first, and shortly after that category *Provides ideas* was also identified. These themes were popping up repeatedly in the reviews mentioning increase in productivity, so I added them under *Increased productivity* as subcategories. *Automated information search* was also mentioned in three or more reviews, so I decided to add also it under *Increased productivity* as a subcategory. All these three subcategories had the common factor that they were mentioned commonly together with productivity increase, and they were all practical examples how the tool increases productivity.

To credibly justify the answers for the research questions, it was essential to gain also quantitative data about the attention the different categories had received in the reviews. Thus, a frequency distribution analysis was conducted in Excel. Frequency distribution is simply a table used to organize data (Newbold et al., 2007). All the possible categories are listed in the left column, while the frequencies for each of these categories are listed in the right column. Frequency distribution analysis made it possible to illustratively compare the categories based on the attention they had received. In practice each review was analysed and linked to the categories the review seemed to fit in. If detailed information was obtained from the review, subcategories were also assigned for the review. This was done by adding the formed categories and subcategories as new columns to the review tables in Excel, so that each review could be assigned to one or more categories. To give an practical example about the process, I came up with an example of a typical positive review:

> *"Awesome plugin! Huge productivity boost. Less time wasted on writing mundane code. Also helps me with the syntax when working with programming languages I'm not yet that fluent."*

This review would have been assigned to category *Increased productivity* and its subcategory *Less typing.* Also, category *Helps learning programming* and subcategory *Teaches language syntax* would have been assigned.

In addition, a trend analysis on the ratings for each tool was conducted in order to identify potential events affecting users' views on the tools. Aside, a time-based frequency distribution analysis was conducted to find possible anomalies in the number of reviews given at different times. In practice, this was done by counting the monthly reviews of each plugin. This was helpful for example to identify machined fraudulent review campaigns falsely increasing or decreasing the rating of the tool.

4. Results

In total 3903 reviews were included in the analysis. From these reviews 19 main categories were extracted. For majority of the categories' more detailed subcategories were also identified. Categories were divided into positive and negative sides. Nine of the categories belonged to positive observations and 10 of the categories in negative observations. In total 53 subcategories were identified, from which 21 belonged to positive observations and 33 to negative observations. Categories were further divided into three groups based on the three research question: *1. Impact on programming, 2. Tool quality* and *3. Tool features*. Table 2 summarizes the grouping and represents how many categories were identified for each group. Tables 3, 4 and 5 present the categories and subcategories for each group.

Table 2 The number of categories and subcategories identified for each group.

Group	Positive Categories	Positive Subcategories	Negative Categories	Negative Subcategories
1. Impact on programming	4	8	3	3
2. Tool Quality	4	5	6	22
3. Tool Features	1	8	1	8
Total	9	21	10	33

Table 3 The categories identified for group *1. Impact on programming*

Positive categories	Negative categories
Increased productivity - Less typing - Provides ideas - Automated information search	**Decreases productivity** - Reviewing & fixing generated code takes more time than writing it from scratch
Increases code quality - Less human errors - Suggests code with good coding conventions	**Decreases code quality** - Too much trust in poor generated code
Helps learning programming - Teaches good coding conventions - Teaches language syntax - Explains code	**Harmful for learning programming** - Novices don't have to understand the generated code to get functioning solutions
Makes programming more enjoyable	

Table 4 The categories identified for group *2. Tool Quality*

Positive categories	Negative categories
Good correctness - Accurate suggestions - Adaptive coding style - Understands context	**Poor correctness** - Incorrect suggestions - Context ignored - Framework ignored - Correctness decreases over time - Inconsistency in suggestions - Poor language support
Good performance - Fast responses - Efficient resource usage	**Performance issues** - High RAM usage - High CPU usage - Freezing and slowing down the system - Latency with responses
Good usability	**Usability issues** - Bugs - Suggestions override/conflict with IDEs own suggestions - Have to login repeatedly - Unavailability due to frequent updates
Satisfied with price and business model	**Dissatisfaction with price and business model** - Expensive - Limitations - Ads
	Security concerns - Tool reads unrelated files. - Telemetric data sent
	Ethical concerns - Open-source data used for training - AI replacing developers - Not available in certain countries

Table 5 The categories identified for group *3. Tool Features*

Positive categories	Negative categories
Tool has useful features - Documentation/Comment generation - Code explanation - Code completion - Chat assistant - Boilerplate code generation - Refactoring - Debug - Test generation	**Tool is missing features** - Documentation/Comment generation - Code explanation - Chat assistant - Refactoring - Debug - Test generation - Commit message generation

4.1 Categories in numbers

In total reviews contained 3049 observations which were placed under positive categories and 1305 observations that were placed under negative categories in the frequency distribution analysis. The complete results are available in Appendix A for positive categories and in Appendix B for negative categories. The occurrence of positive observations per category is presented in Figure 2. *Increased productivity* was the most popular category with value of 557 occurrences. The second place was taken by *Useful features* with 420 occurrences and *Good correctness* was third with 299 occurrences. The number of negative observations per category is presented in Figure 3. In the negative side *Usability issues* was the most reported category with value of 548 occurrences. This was followed with *Dissatisfaction with price/business model* with value of 222 occurrences and *Poor correctness* with value of 185 occurrences.

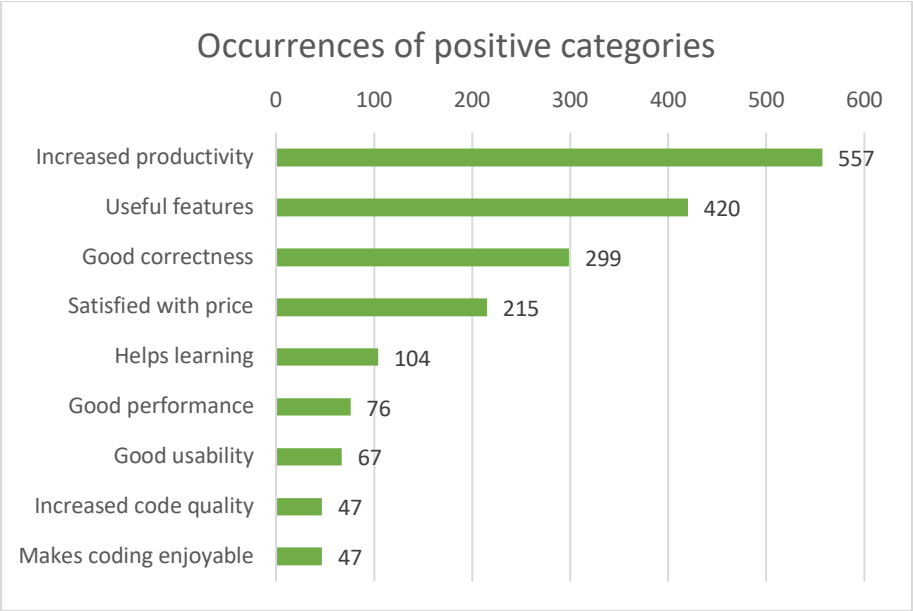

Figure 2. Categories for positive observations by occurrence

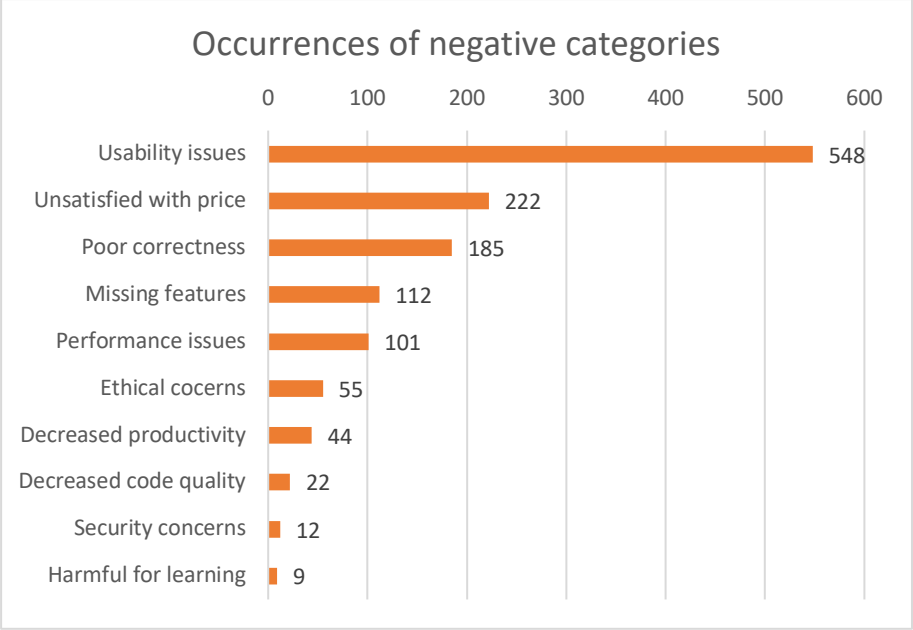

Figure 3. Categories for negative observations by occurrence

4.1.1 Impact on programming

Regarding the categories related to impact on general programming experience, positive categories gained more attention compared to their negative counterparts (see Fig. 4).

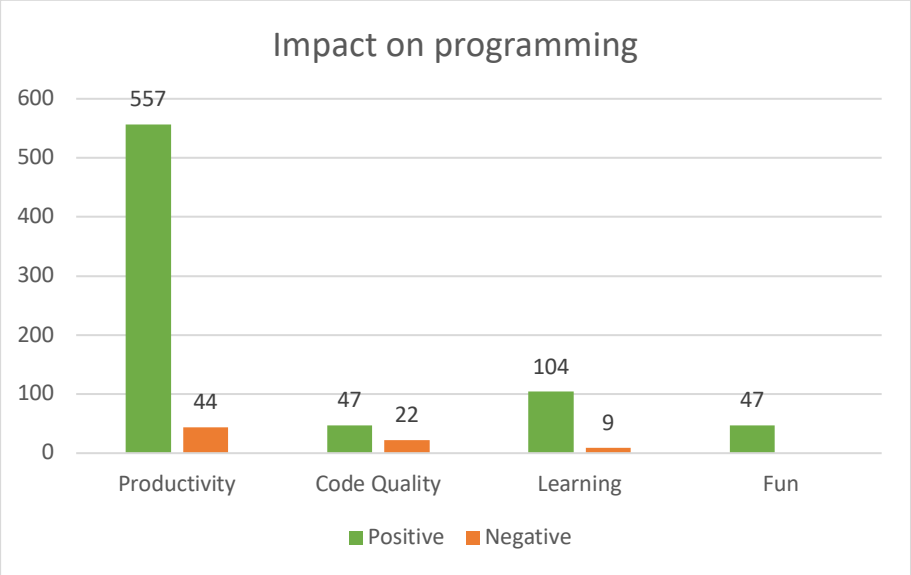

Figure 4. Occurrence of categories related to tool impact on programming.

Impact on Productivity

Increased productivity was the most popular category with 557 reviewers reporting the tool having positive impact on productivity while only 44 reported negative impact on productivity (see Fig. 4).

Three subcategories were identified for *Increased productivity* category, the distribution of which is shown in Figure 5. Majority of the reviews reporting increased productivity reported that the tool saves time in typing. Autocompletion and generation of boilerplate code and other repetitive tasks were reported. Tool was reported also providing different ideas on solving the issues. Autocompletion providing different solutions was seen increasing productivity, as well as ability to ask questions from chat, if stuck. Information search was also reported to increase productivity, as additional web search was automated with the tools. IDE plugins were considered better than separate AI tools like ChatGPT in that sense, that there is no need to jump between different applications and copy paste. Also, IDE plugin can take advantage of the project context in providing the suggestions.

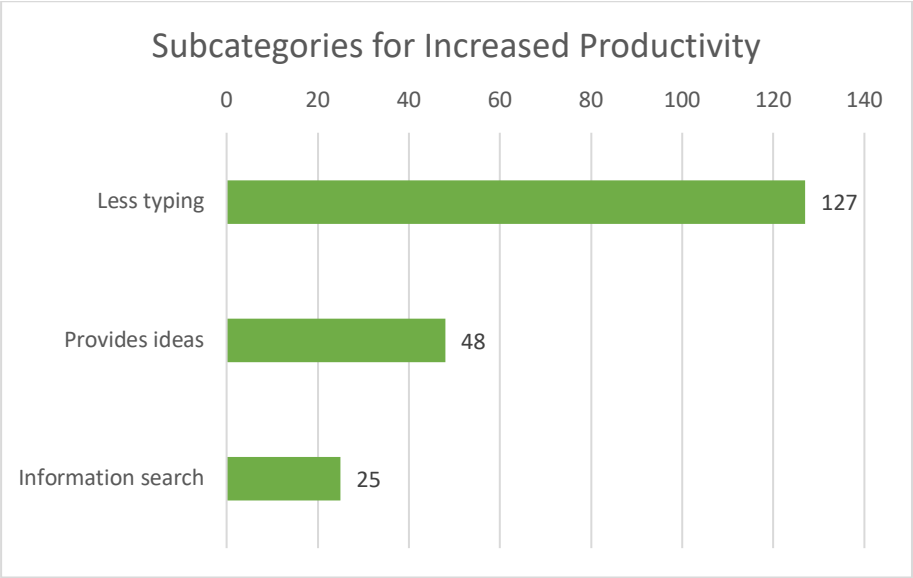

Figure 5. Occurrences of the subcategories for category *Increased productivity*.

Reviews reporting decreased impact on productivity were mainly caused by tool generating unwanted and buggy solutions. These reviews reported that it takes more time reviewing and fixing the generated code than writing the code from scratch.

Impact on Code Quality

A greater part of the reviewers reported tools having positive effect of code quality rather than negative. 47 reviews reported increased code quality and 22 decreased code quality (see Fig. 4).

From the reviews reporting increased code quality, 16 reported tool suggesting code with good coding conventions and 14 reported the code decreasing human errors. The tools were reported to decrease human errors, as AI generated code should not contain those. Some reviews reported the tool suggesting code with better solutions, that the reviewer thought of first writing. Also, some tools were reported to have functionality reporting known bugs and suggesting refactoring on the existing code. Negative impacts on code quality were mostly related to reviewers having concern that programmers trust too much on the generated code, and skip reviewing the code if it works. Many of these comments wanted to just remind their colleagues to not trust too much on the AI.

Impact on Learning

Considering AI code generation tools impact on learning, the reviewers had solid view that tools are more helpful for learning coding, rather than being harmful. 104 reviewers reported the tools helping learning programming while only nine reported they are being harmful for learning (see Fig. 4).

In Figure 6 we can see that from the reviews considering the tool being helpful for learning, 25 reported the tool helping in learning syntax of a new programming language. 20 reported the tool helping them learning better coding conventions. Eight reported the tool teaching programming by explaining the code for them.

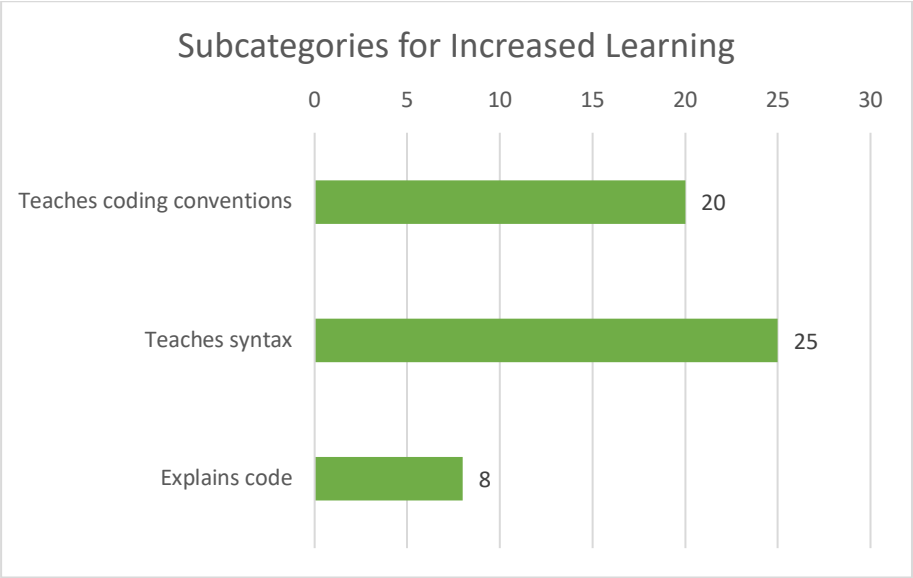

Figure 6. Occurrences of the subcategories for category *Increased Learning*.

The reviews that reported tool being harmful for learning programming were worried about the tool being able to generate functioning code without need for understanding the code. This was seen as potential threat as novice programmers and students can theoretically rely on the AI generated code without really understanding it.

Makes programming more enjoyable.

Considerable amount of 47 reviews mentioned also that the tool makes programming more fun or enjoyable (see Fig. 4).

4.1.2 Satisfaction on tool quality

Majority of the critical and negative comments were related to tool specific quality issues. For majority of the categories under this group the negative side overcame the positive counterpart (see Fig. 7).

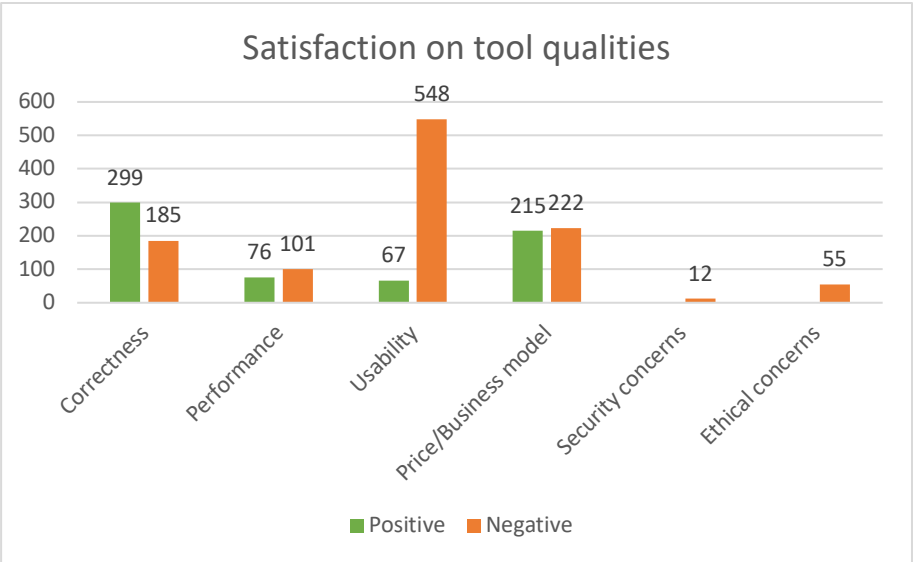

Figure 7. Occurrence of categories related to satisfaction on tool quality

User experience

Majority of the reviews commenting the tools usability and user experience were on the critical side. *Usability issues* was the most popular category on the negative side of the observations. 548 observations felled into category of usability issues while only 67 reported good usability (see Fig. 7).

Figure 8 presents the distribution of usability issues under subcategories. Majority of the bad user experiences were caused by different kind of bugs with occurrence of 283. Single specific issue reported with majority of the tools was the Suggestion override with 77 occurrences in total. The users were annoyed that the tool did override, or conflict with the IDEs default suggestions. Some reviews proposed a switch for disabling the plugin or configurability on which suggestions are prioritized. Some comment suggested that there should be mechanism that prioritizes plugins suggestion over IDEs suggestion only if the confidence of the plugin's suggestion is over some threshold, e.g. 80% confidence. In addition, also having to repeatedly login was mentioned 25 times and frequent updates blocking the usage of the tool were reported six times.

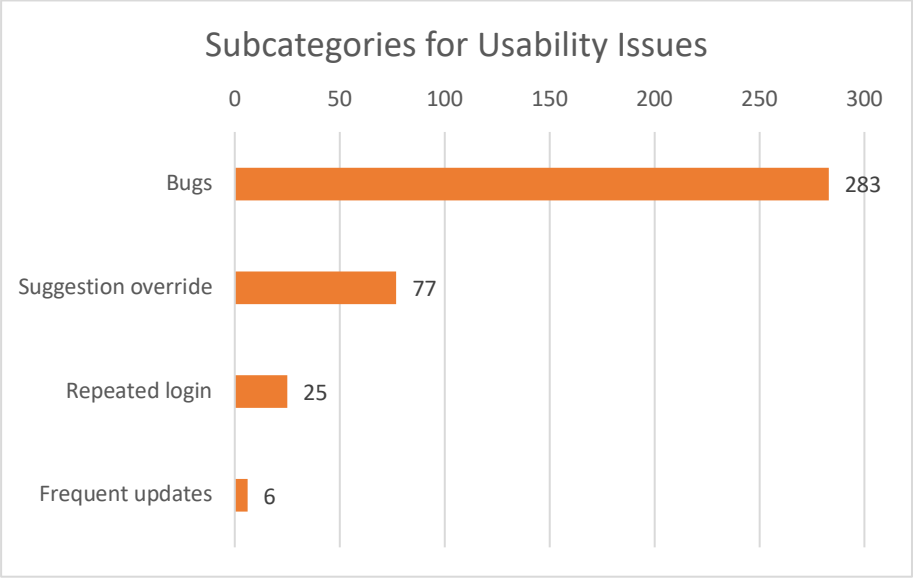

Figure 8. Occurrences of the subcategories for category *Usability issues*.

Correctness

Majority of the reviews considered the tool to have good correctness, while there was variation between the tools. 299 reviews reported good correctness, while 185 reported poor correctness (see Fig. 7).

Figure 9 presents the distribution of subcategories for good correctness. 215 of the reviews reported the tool providing accurate suggestions. 74 reported that the got better over time and learned their programming style. 44 reported that the tool takes the context into account when generating code.

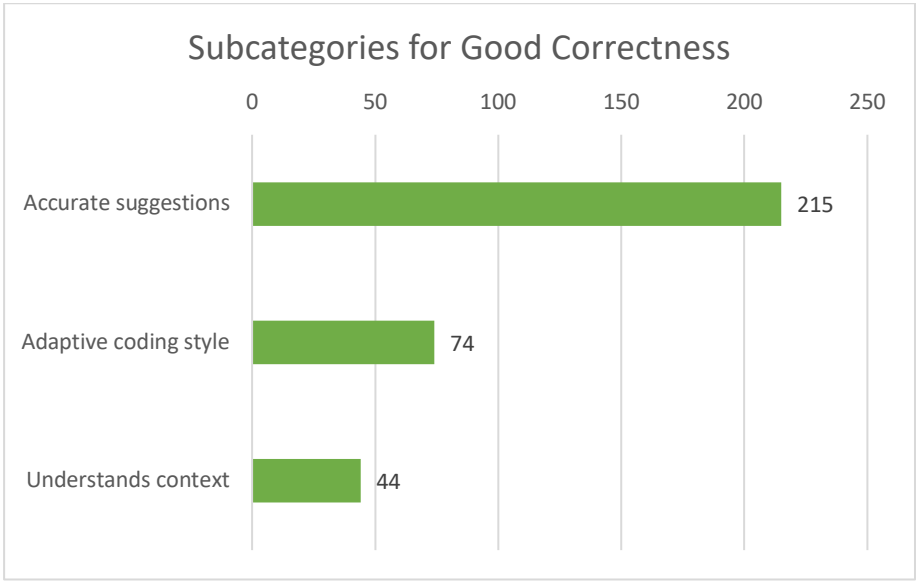

Figure 9. Occurrences of the subcategories for category *Good correctness*.

In Figure 10 we can see that 139 of the reviewers reported the tool providing incorrect suggestions. 25 reported that tool ignores the project context when generating code. 19 reviews issued that tool did not have proper support for programming language they were using. 16 reviews stated that the correctness of the suggestions had dropped over time. Nine reported that there was inconsistency in the suggestions.

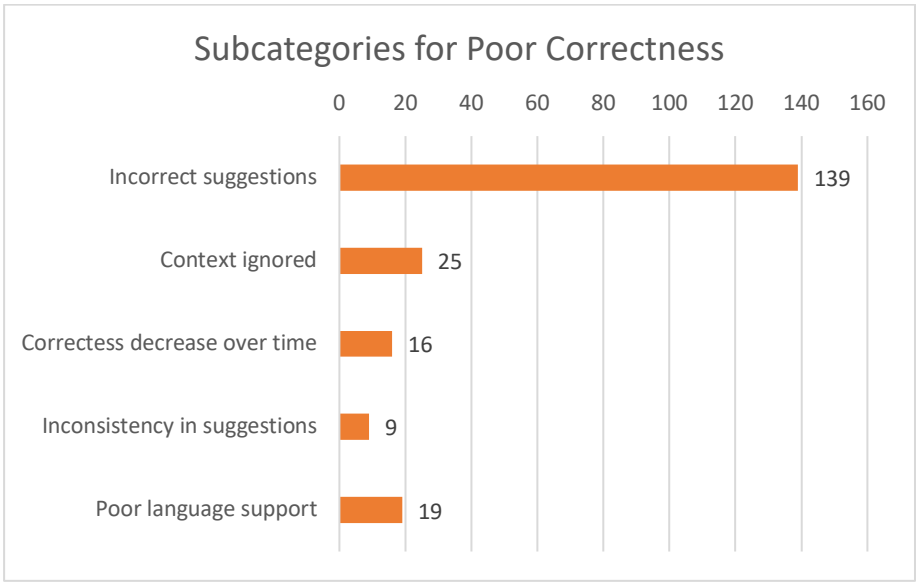

Figure 10. Occurrences of the subcategories for category *Poor Correctness*.

Performance

Regarding performance of the tools, 76 reported good performance, while 101 reported performance issues with the tool (see Fig. 7). About those reporting good performance, 67 were impressed about the fast responses the tool provided. 5 mentioned that tool used system resources efficiently (see Fig. 11).

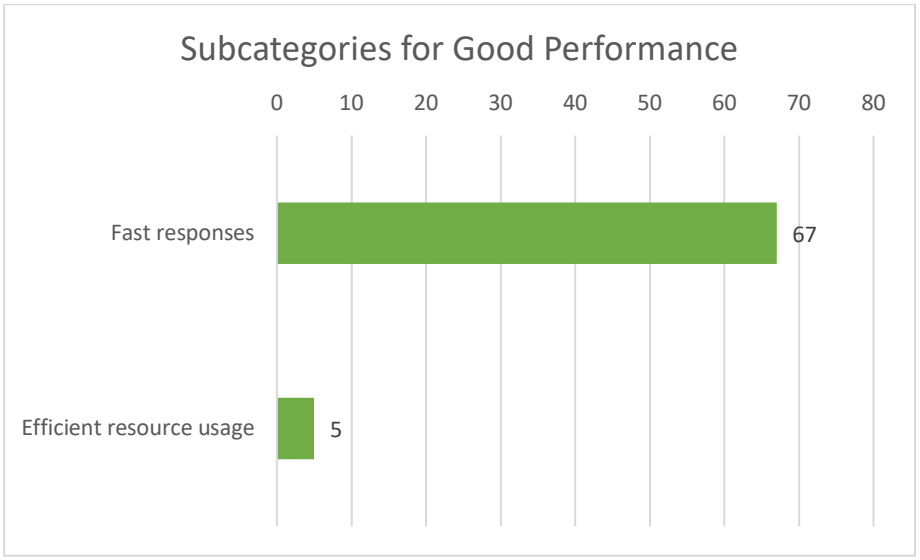

Figure 11. Occurrences of the subcategories for category *Good Performance.*

Regarding the reviews reporting performance issues with the tool, 36 reported that tool had too much latency when suggesting and generating code. 32 reported that plugin caused the system or IDE to freeze. 24 reported about high RAM usage and 16 about high CPU usage (see Fig. 12).

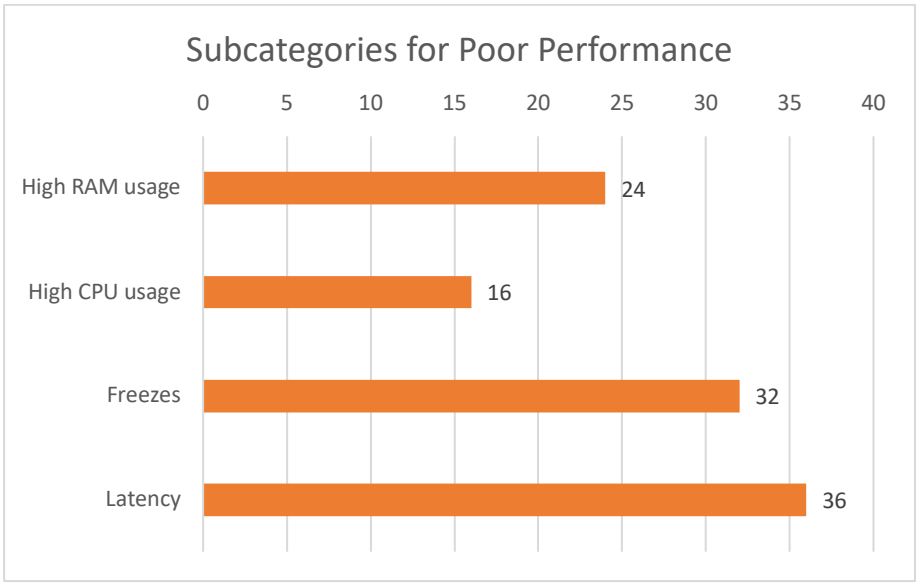

Figure 12. Occurrences of the subcategories for category *Poor Correctness*.

Concerns with price, business model, ethics, and security

215 reviewers were satisfied with the price/business model of the tool while 222 were not satisfied with the price/business model. 55 reviewers had ethical concerns and 12 reported security concerns (see Fig. 7).

In Figure 13 we can see that 195 reviewers thought that the tool was not worth of the price. 18 complained about the limitations the free version of the tool had and 12 complained about ads. 24 reviewers criticized about tool using open-source data for training its AI model while demanding fee for using it. 20 reviews complained that tool was not available in their country. 12 reviewers mentioned fear of AI taking over their job. Security related subcategories received least attention, with concern of tool reading unrelated files on their system with six mentions and concerns in telemetric data sent with three mentions.

Figure 13. Occurrences of the subcategories for categories regarding business model, ethical and security concerns.

4.1.3 Features

420 of the reviews reported that tool had useful features and 112 reported that tool is missing some feature they would like to have. 278 reviews reported that they are using code completion. 79 users were using the chat feature, 69 used documentation/comment generation. 35 used the tool for boilerplate generation, 26 code explanation, 19 debugging, 18 refactoring and five test generation (see Fig. 14).

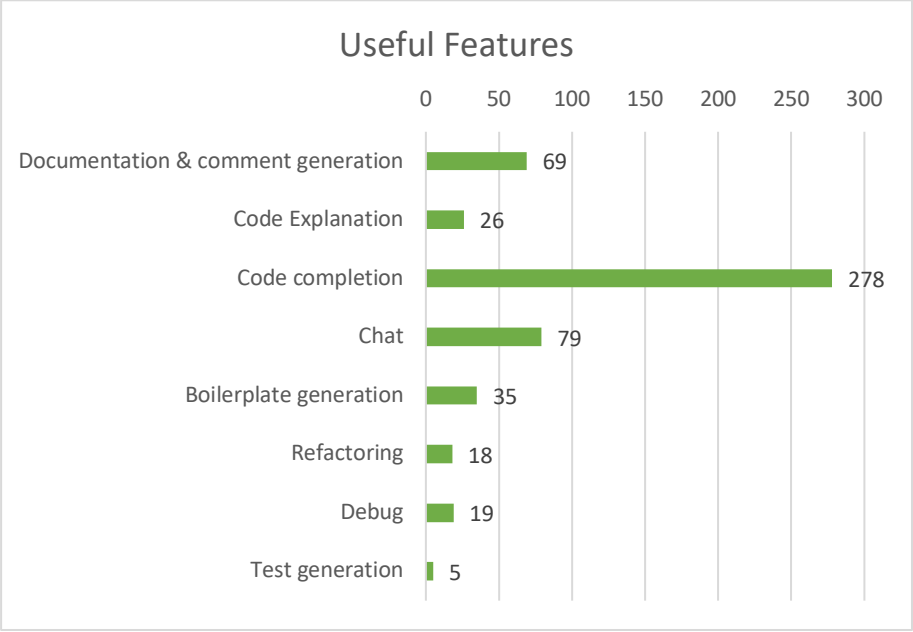

Figure 14. Occurrences of the subcategories for category *Useful Features*.

Chat feature was clearly single most wanted feature which was missing from some of the tools. 53 reported that the tool is missing chat feature. 10 reviewers missed refactoring feature. Eight would have liked the tool to be able to generate GIT commit messages. Five reviewers wanted documentation / comment generation, four wanted debugging feature, three code explanation and two test generation (see Fig. 15).

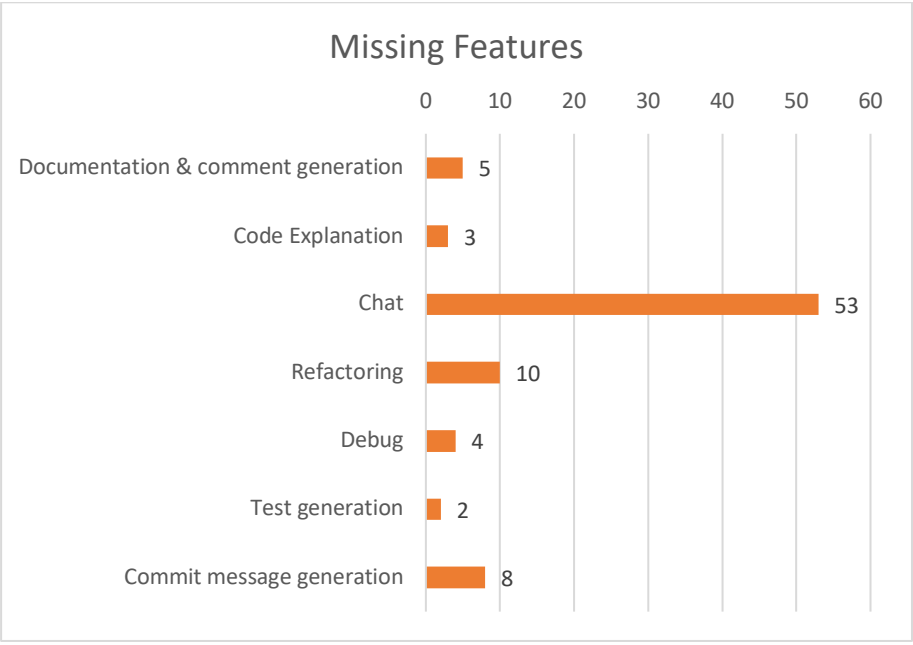

Figure 15. Occurrences of the subcategories for category *Missing Features*.

4.2 Tool specific differences

The ratings and reasons for different ratings varied between different tools and IDEs so it is necessary to open the matter in more detail to better understand the users' preferences. The average rating was 3,3 for GitHub Copilot, 4,1 for Tabnine, 2,4 for AWS Toolkit and 4,8 for Codeium. The detailed ratings for each tool and each IDE are presented in Figure 16.

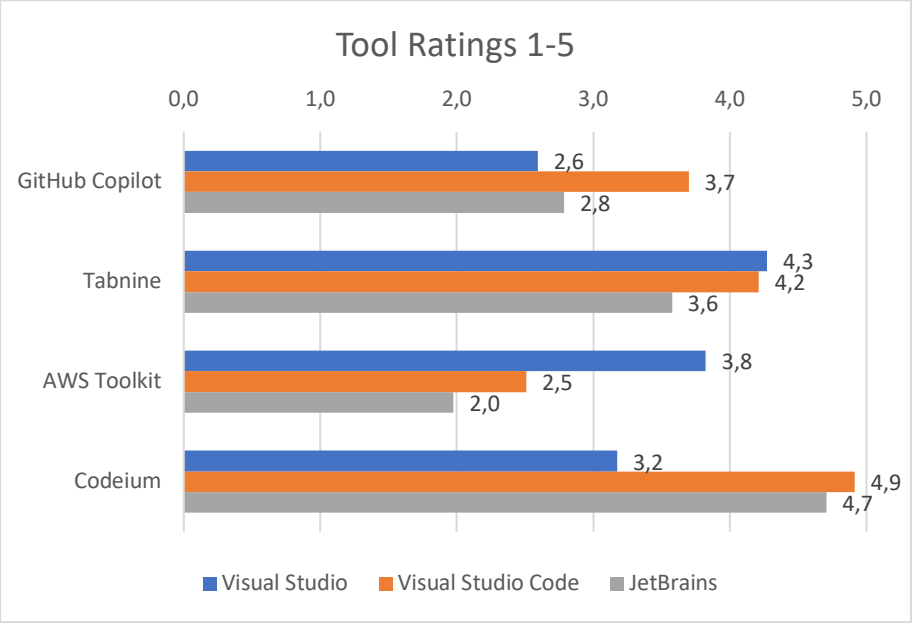

Figure 16. Average ratings for each plugin.

GitHub Copilot

GitHub Copilot had rating of 3,7 for VS Code plugin, while JetBrains plugin got 2,8 and Visual Studio 2,6. Majority of the positive comments mentioned about increased productivity, useful features, and good correctness. Reasons for negative reviews were usability issues, unsatisfaction with price and poor correctness. Bugs were the most common reason for negative reviews. Bugs contained issues preventing the usage like login issues, downtime, and IDE incompatibilities after updates. The second most common specific reason for negative review was the monthly fee, as the tool was at first free and then it changed to have 10$ monthly fee. There was no free version of it and especially the users from third-world countries saw the price as a big issue. Geographical pricing was suggested so that developers around the world could utilize the tool. Regarding correctness there was more negative feedback about the correctness than positive. Many users reported that they noticed that the accuracy of the suggestions has decreased over time. For usability, the way that plugin override the IDEs suggestions was also highly criticized similarly to the other tools.

Ethical issues were also notable discussed in GitHub Copilots reviews. Many reviewers were not happy with the fact that open-source code was used to train Copilots AI model and then users were charged for using the tool. Some reviewers reported and criticised that Copilot was not accessible in their countries without using VPN. Majority of these reviews mentioned that it is not available in Russia.

The plugin for VS Code got the best ratings, while plugins for Visual Studio and JetBrains got considerably lower rating. Reviews for the plugin for Visual Studio and JetBrains often compared the plugin for VS Codes corresponding plugin stating that the plugin for VS Code was much better. Usability, correctness and performance of these both plugins were described worse compared to VS Code plugin. Missing chat feature caused a lot of negative feedback especially for the JetBrains plugin. Some users criticized that the effort put for the JetBrains plugin was not as great as the effort put to Visual Studio plugins, as it got more features, frequent updates and better usability. The price was also criticised, as the price is same for all IDEs, although other have fewer features and worse user experience. The use of tab key to trigger the suggestions was criticised on Visual Studio plugin, as in VS Code the suggestions are automatically showed when you write code. Incompatibility issues and not supporting ARM architecture was also criticised for Visual Studio plugin.

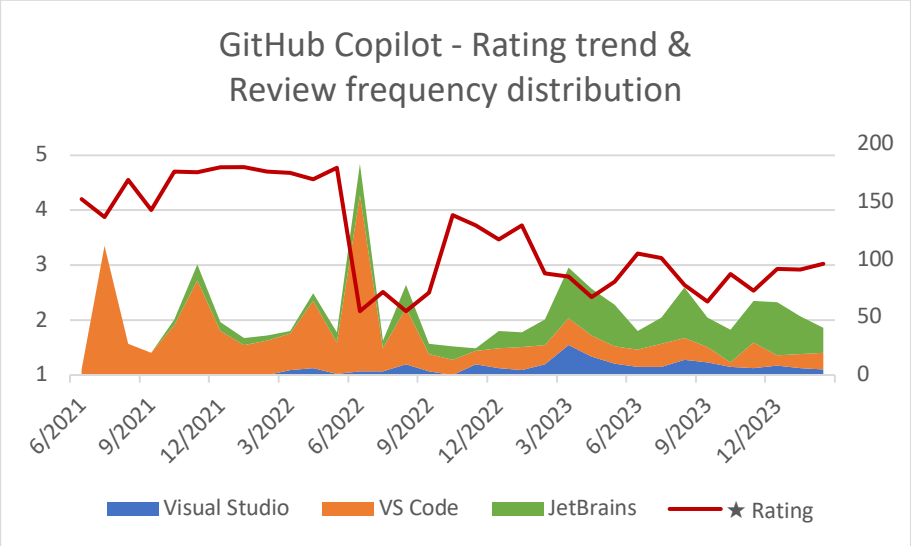

Figure 17. Rating trend and review frequency distribution for GitHub Copilot

From Figure 17 we can see that majority of the reviews from 2021 to end of 2022 for GitHub Copilot were posted for VS Code plugin. After 2023 reviews for JetBrains plugin took over VS Code in frequency. Clear peak in review frequency can be detected in summer 2022. At the same time the average rating dropped dramatically. The reason for the sudden negative change in reviews at this point was the end of free beta testing for GitHub Copilot. Only verified students had free access to GitHub Copilot after this point of time.

Tabnine

Tabnine for Visual Studio got rating of 4,3, while VS Code version got 4,2 and JetBrains 3,6. The correctness of the tool was most criticized quality aspect, while it also still got more positive than negative feedback about the correctness. Its support for some programming languages was criticized as well as context awareness. Also, the correctness rate of the tool was reported to decrease over time similarly to Copilot. Some reviewers were speculating that free version was worsened on purpose so that people start purchasing the paid version. Tabnine got still relatively the largest amount of positive feedback about its adaptive coding style. Reviewers noticed how the tool learned over time their coding style and got better suggestions as time passed.

Performance issues were overrepresented in Tabnine reviews. High CPU and RAM usage were reported as well as freezing and latency in responses. Some reviewers reported that the performance issues were solved when changing the local serving mode to cloud. Tabnines price/business model was criticized as the free version had limited amount of code completions which made the tool unusable regarding users. Also, ads in the free version were considered annoying. Tabnines support team however received particularly lot of positive feedback for their fast and helpful responses.

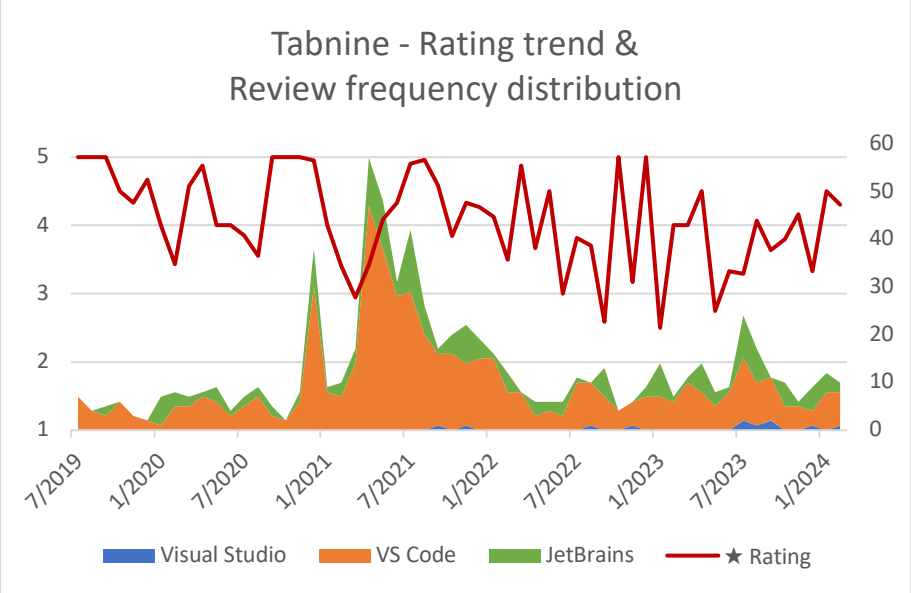

Figure 18. Rating trend and review frequency distribution for Tabnine

From Figure 18 we can see that VS Code plugin have always received majority of the reviews for Tabnine. In summer 2021 we can see a major peak in review frequency which can be caused by multiple factors. We can see that there is also clear decrease in rating at the same time. One clear reason for the increase of negative feedback was caused by the limitations Tabnine added for the free plan. Another reason for the increase in overall reviews is probably caused by the release of GitHub Copilot, as its beta release gained much attention, and people started seeking alternative solutions for it.

Amazon CodeWhisperer / AWS Toolkit

Amazon CodeWhisperer was not available as individual plugin for any of the IDEs, but it was included in AWS Toolkit plugin. The number of reviews was also smallest. These facts made analysing CodeWhisperer more difficult than the other tools. AWS Toolkit for Visual Studio got rating of 3,8 while VS Code got 2,4 and JetBrains only 2. Usability issues got most attention with AWS Toolkit. Users didn't seem to like that multiple tools are included in a single plugin. Some users wanted to use only CodeWhisperer, and others wanted to disable the features they did not need. The lack of configurability and ability to disable some of the features seemed to bother users. Also, lots of incompatibility and unitability due to IDE crashing were reported. CodeWhisperer, especially for VS Code collected relatively lots of usability complains about having to login constantly. Performance issues also gained some attention, mostly about latency in the responses.

Codeium

Codeium got the best overall rating. Plugin for VS Code got rating of 4,9 while JetBrains version got 4,7 and Visual Studio 3,2. In the reviews Codeium was often compared to GitHub Copilot and titled as free alternative for GitHub Copilot. The satisfaction about the tools basic version being free was expressed in a large part of the reviews. Overall, the tool received very little critical or negative reviews. Its correctness and performance were seen great, while usability got some critique. The suggestion override was the biggest reason for critical feedback in addition to bugs. Displaying suggestions too quickly was also criticised by some reviewers.

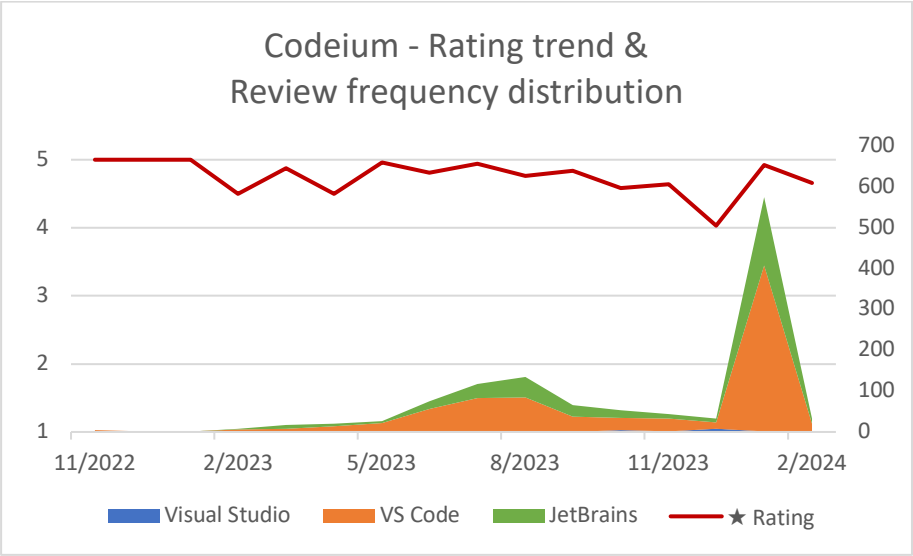

Figure 19. Rating trend and review frequency distribution for Codeium

From Figure 19 we can see that the rating for Codeium has remained good and stable for its whole lifetime. There is however a strangely large peak in review frequency in January of 2024. VS Code and JetBrains plugins received 576 reviews together in January 2024, which is more than any other tool has received reviews in such a short timeframe. When taking a closer look, 219 reviews were posted on 20.1.2024 only, with almost all of them being five-star reviews. This raised a possible concern about fraudulently generated reviews. A web search for that particular date with search word "Codeium" did not either provide clear reason for the peak in reviews. When analysing the content of these reviews, there was however nothing particularly suspicious nor abnormal about these reviews. The reason behind this peak remained unknown.

4.3 Summary

Based on the results obtained from the data analysis, the results can be summarized by answering the three research questions. The research questions were following:

- RQ1: What effects do the AI-assisted code generation tools have on software engineers work?
- RQ2: Which quality factors influence users' opinions on the tools?
- RQ3: What features do users value most in the tools?

The answers are listed in Table 6, and arranged based on the attention they received in the frequency distribution analysis.

Table 6 Summary of the answers for the research questions

Research question	**Answer**
RQ1: What effects do the AI-assisted code generation tools have on software engineers' work?	1. Increased productivity 2. Helps learning programming 3. Makes programming more fun and enjoyable 4. Increased code quality
RQ2: Which quality factors influence users' opinions on the tools?	1. Usability 2. Correctness 3. Business model and price 4. Performance 5. Ethical aspects 6. Security aspects
RQ3: What features do users value most in the tools?	1. Code completion. 2. Chat 3. Documentation and comment generation 4. Code explanation 5. Refactoring 6. Boilerplate generation 7. Refactoring 8. Commit message generation 9. Test generation

5. Discussion

The analysis conducted in this study for the data collected from the IDE plugin marketplaces supports the findings on the subject made in prior literature. However, new findings that had not been discussed in previous studies were also discovered.

5.1 Impact on programming

The goal of first research question in this study was to capture the positive and negative impacts the AI-assisted code generation tools have on software engineers work from the point of view of the user of these tools. Based on the data analysis conducted in this study, we can summarize that AI-assisted code generation tools have most impact on following three aspects:

1. Increased productivity
2. Helps learning programming.
3. Makes programming more fun and enjoyable.

All the categories on the negative side considering the first research question received marginal amount of attention compared to their positive counterpart. Thus, negative impact regarding the first research question could not be captured in this study. The category receiving most negative attention regarding this area was decrease in productivity. The main reason behind these reviews were tool specific usability issues and poor accuracy of the suggestions. The smallest difference in attention received between the positive and negative counterparts was observed with the code quality aspect. The main reason for the negative reviews was the concern of programmers trusting too much on the generated code and skipping the review, which would potentially lead to decreased code quality. This concern was noted also in the prior literature (Pearce, Ahmad, et al., 2021).

Productivity increase was clearly the most often reported positive impact on programming in the reviews. In some domains the productivity increase has been such great that roles have been fully automized and replaced by AI (Raiaan et al., 2024). This was however not a concern among the reviewers of AI-assisted code generation tools, as only small group of reviewers mentioned being afraid of losing their job over AI. Also, some of these comments were partly joking referring to the good accuracy of the tool. The decreased typing was most recognized reason for increased productivity. The ideas it suggested and automated information search were also identified as elements that improved the productivity. The studies by Vaithilingam et al., 2022 and Barke et al., 2023 support the findings made in this study. Barke et al., 2023 found out that the participants in their study used AI-assisted code generation tools in *acceleration* or *exploration* mode. In acceleration mode the participants used the tool to complete their code faster when they knew what they need to write. In exploration mode the participants were unsure what should be done, so tool was used to explore the options. In this study, the users who reported using the tool for information search and ideas, were using the tool in exploration mode, while the users reported about reduced writing were using the tool in acceleration mode.

Regarding learning, the tools' ability to teach programming language syntax received most attention. Both, novices, and more experienced programmers reported that the tool was a great help when working with programming languages that they were not that

familiar. Many reviewers reported also that the tool teaches them better coding conventions. Novices and students were also reporting that code explanation feature was useful. Tool being harmful for learning did not receive much attention in the reviews.

Impact on code quality received only little attention in the reviews. Majority of the reviewers considering code quality thought that the tool increased code quality instead of decreasing it. Some reviewers noticed that the tool suggested better solutions and coding conventions that they were unaware or would not even think about without the tool. Reduction of human errors were also noticed.

Interesting discovery was that so many of the reviews mentioned that the tool made their work more enjoyable and fun than working without it. What made the discovery surprising was that this aspect was not discussed in the previous literature at all. Some reviewers considered the tool to be good alternative for pair programming if human co-worker was not available. No wonder these tools are often titled as AI pair programming tools.

5.2 Tool quality

The second research question strived to find out which quality factors affect the users' opinions about the tool. From the results we can summarize three quality factors that affect most on users' opinions:

1. Usability
2. Correctness
3. Business model and price

Previous literature did not address the usability of the code generation tools very deeply, although usability issues caused majority of the negative feedback in the reviews collected in this study. The vast majority of the reviews under this category was related to different bugs decreasing the user experience and, in some cases even preventing the usage completely. Usability of the tool got only small amount of positive feedback compared to the negative feedback. This is expected as people give feedback usually if something does not work as expected rather than when everything works as expected.

One clear specific usability issue which was present in all tools was the suggestion override. IDEs own suggestions were overridden or conflicted with the plugins suggestions which was seen problematic by considerable number of users. Simple solutions to solve this issue were suggested by the reviewers: A switch for disabling plugins suggestion or configuration for prioritising IDEs suggestions over plugins suggestion were proposed by reviewers. Also, more sophisticated mechanism that would prioritize the plugins suggestion over IDEs suggestion only if the confidence of the plugin's suggestion is over specified threshold.

The correctness of the code generated by AI-assisted code generation tools is a lot studied area in prior literature as generating correct solutions for problems is the main function of these tools. The previous studies have provided mainly laboratory experiments where tools ability to generate solutions for programming tasks was benchmarked and evaluated with automated unit tests. This study captured the users' subjective views on how spot on the generated solutions actually were.

Majority of the overall reviews commenting correctness of the tool were positive, although the amount of negative feedback was also relatively high. Accurate suggestions and incorrect suggestions were the subcategories that gained most attention. The accuracy of the suggestions is essential, as with poor suggestions the tool is just unusable. On the other side the accurate suggestions amaze the users. Several reviews stated that they were amazed about the accuracy of the suggestions, feeling that it can read their mind.

Poor context awareness and poor language support were the next more specific reasons for negative feedback related to correctness. On positive side adaptive coding style and context awareness were the most commented factors. The context awareness and adaptive coding style were factors that increased the wow effect towards of the tool.

Majority of the positive observations about performance in the reviews were amazed about the fast response of the tool. Respectively on the negative side the latency in responses gained majority of the attention. Performance issues like freezing and high RAM and CPU usage also gained lots of negative attention. Performance is obviously essential factor for tool of any kind.

The feedback on the tools price and business model varied a lot between different tools. When initially free tool is changed to have a monthly fee, a flood of negative feedback is guaranteed. Majority of the negative feedback under this category came from users not happy with the high price of the tool. Many reviewers from third world countries were upset about the monthly fee, which was unreasonably high compared to the income level in their country. Geographical pricing was suggested to solve this issue. Free tools like Codeium gained a lot of positive feedback about being free while performing as well or even better as its paid counterparts. Codeium have stated that they will stay free forever. A question arises, how are they going to get money? Well, they have paid enterprise version of the tool with extra features which make sense to be included only on team or organization level (*Codeium · Free AI Code Completion & Chat*, 2024).

The monthly fee led to ethical discussion in the reviews. Is it ethical to charge individual users a monthly fee for utilizing a tool which AI model is trained on open-source code? Scientific articles regarding this issue were introduced in the second chapter (Al-Kaswan & Izadi, 2023; Sun et al., 2022). The ethical aspect of using publicly available data to train LLMs is tricky. Without data we can't train the models which are required by these tools which improve our work in various aspects. Al-Kaswan & Izadi, 2023 made various recommendations for lawmakers, ML community and software engineering community to address this issue. For software engineering community, Al-Kaswan & Izadi recommended that developers should make informed decision and clearly denote if their source code can be used to train AI models. LLMs are most likely here to stay and change the way software is being developed, but should developers have opportunity to give a separate permission for their code to be used as training data for these LLMs? Also, should developers be credited and compensated somehow when their code is user to train LLM? These are questions that the software engineering community should address.

5.3 Features

Previous literature has studied mainly code generation from prompts and autocompletion, while software engineering has lot of other areas that could use help of generative AI. Dehaerne et al., 2022 divided AI-assisted code generation applications into three main paradigms: description-to-code, code-to-description, and code-to-code. The tools included in this study provided features for all these paradigms. The most obvious

examples for these paradigms would be code generation from natural language description, autocompletion and comment generation from code.

The goal of third research question was to gain understanding which tools the users of AI-assisted code generation tools value the most. From the results we can list the top three features' users valued most:

1. Code completion.
2. Chat
3. Documentation and comment generation.

Code completion was clear winner in the category of useful features. All the tools had code completion, so no one missed it in the reviews. Chat was the second most mentioned feature on the useful features category. Some of the plugins were missing it and its absence caused large amount of negative feedback. The documentation/comment generation received third most attention regarding the features. This feature was also missing from some of the tools, which caused small amount of negative feedback.

Commit message generation was one surprising subcategory which was identified for the side of missing features. No review mentioned tool having feature for commit message generation. This could be potential feature to be included for these tools, as it has some demand for it.

5.4 Limitations

The sample analysed in this study contained only user reviews from IDE plugin marketplaces. The reviews were limited to four tools which were all IDE plugins. There is wide variety of other tools for AI-assisted code generation which were not included in this study. While lot of these tools are available as plugins for IDEs, there are also independent AI applications for code generation and chat applications like ChatGPT that are able to generate code.

Large portion of the reviews were short or were lacking detailed information. Especially the positive reviews usually did not contain any specific reason why the tool was considered to be so great. Negative feedback had usually more details, as it is much easier to identify the things that are not working, rather than the things that work especially well. People also tend to give feedback more often when something is not working as expected, or when encountering negative feelings.

There is also always possibility for fraudulently generated reviews, as anyone can post reviews on the plugin marketplaces without strong identification. These kinds of fraudulent reviews can be either positive or negative. In this study there was one suspicious anomaly detected in review frequency for one particular day, for which the reason stayed unknown.

5.5 Implications

One of the major advantages of this study, compared to previous literature, was its large sample size and the inclusion of several different tools. This study brought up the voice of the users of AI-assisted code generation tools. Subjects that had not been addressed in

previous literature were captured from the user reviews. The findings made in this study provide various new research paths for future. For example, a considerable number of reviewers reported that the tool made their work more enjoyable. It would be interesting to study further how AI-assisted code generation tools affect the well-being and satisfaction of software professionals at work. Additionally, the business model and ethical aspects of AI-assisted code generation tools gathered significant attention among the reviewers, yet only a few scientific publications on these topics were found.

6. Conclusion

We live in a time where AI is affecting people's life almost on every domain. On the field of software engineering, the AI-assisted code generation tools are changing the way of working. To gain more knowledge on this current topic, and narrowing down the gaps in previous literature, in this thesis I strived to find answer for three research questions. RQ1: What effects do the AI-assisted code generation tools have on software engineers work? RQ2: Which quality factors influence users' opinions on the tools? RQ3: What features do users value most in the tools? To capture the users, view on these questions, user reviews of four popular AI-assisted code generation tools were analysed.

The findings of the analysis were mostly in line with previous literature while new findings were also made. Regarding the first research question, users reported the most that the tool have increased their productivity, helped learning programming and made programming more enjoyable. For the second research question, users reported mostly about usability issues with the tool. Correctness of the code gathered also a lot of positive attention, although many tools got also lot of criticism regarding poor correctness. Business model and price of the tool received also a lot of feedback, mainly due to the fees for individual users. For the third research question, code completion was mentioned most often in the sample. Chat feature was also appreciated and needed feature according to users. The thirdly most commented feature was documentation and comment generation feature.

This study contributes to the existing knowledge base of AI-assisted code generation by providing overview of the users' direct thoughts on these tools without the intervention of a researcher. One of the major advantages of this study compared to the previous literature is that this study used a large sample including four different tools. This study provides ideas and paths for further research. For example, considerable number of reviewers reported that the tool made their work more enjoyable. It would be interesting to study more how the AI-assisted code generation tools impact software professionals' well-being and satisfaction on work. The business-model and ethical aspects of the AI-assisted code generation tools were also topics gaining lot of attention among the reviewers while only few scientific publications regarding these topics were found.

Regarding the limitations of this study, the sample used in this study captures only the reviews of four AI-assisted code generation tools for three IDEs. In total, reviews of twelve plugins were included in the sample, containing in total 3903 reviews. Large portion of the reviews however did not contain much detailed information. Also, some reviews were missing the verbal description completely, which meant these reviews had to be excluded from the analysis.

Although many software professionals criticize the AI-assisted code generation tools and weight their negative impacts over positive, the research proves the opposite. We are still in the beginning of this change on AI revolutionizing the way how software engineers perform their work. The tools are still not perfect, but they still manage to impress their users and increase the productivity notably.

References

AI Code Generator - Amazon CodeWhisperer - AWS. (2024). https://aws.amazon.com/codewhisperer/

Al-Kaswan, A., & Izadi, M. (2023). *The (ab)use of Open Source Code to Train Large Language Models*. http://arxiv.org/abs/2302.13681

Asare, O., Nagappan, M., & Asokan, N. (2023). Is GitHub's Copilot as bad as humans at introducing vulnerabilities in code? *Empirical Software Engineering*, *28*(6). https://doi.org/10.1007/s10664-023-10380-1

AWS Toolkit - Amazon Q, CodeWhisperer, and more - Visual Studio Marketplace. (2024). https://marketplace.visualstudio.com/items?itemName=AmazonWebServices.aws-toolkit-vscode&ssr=false#review-details

AWS Toolkit for Visual Studio 2022 - Visual Studio Marketplace. (2024). https://marketplace.visualstudio.com/items?itemName=AmazonWebServices.AWSToolkitforVisualStudio2022&ssr=false#review-details

Barke, S., James, M. B., & Polikarpova, N. (2023). Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proceedings of the ACM on Programming Languages*, *7*(OOPSLA1). https://doi.org/10.1145/3586030

Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023). Programming Is Hard - or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, *1*, 500–506. https://doi.org/10.1145/3545945.3569759

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. de O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021). *Evaluating Large Language Models Trained on Code*. http://arxiv.org/abs/2107.03374

Codeium · Free AI Code Completion & Chat. (2024). https://codeium.com/

Codeium - Visual Studio Marketplace. (2024). https://marketplace.visualstudio.com/items?itemName=Codeium.CodeiumVS&ssr=false#review-details

Codeium: AI Coding Autocomplete and Chat for Python, Javascript, Typescript, Java, Go, and more - Visual Studio Marketplace. (2024). https://marketplace.visualstudio.com/items?itemName=Codeium.codeium&ssr=false#review-details

Dehaerne, E., Dey, B., Halder, S., De Gendt, S., & Meert, W. (2022). Code Generation Using Machine Learning: A Systematic Review. In *IEEE Access* (Vol. 10, pp. 82434–82455). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2022.3196347

Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., & Prather, J. (2022). The robots are coming: Exploring the implications of OpenAI codex on introductory programming. *ACM International Conference Proceeding Series*, 10–19. https://doi.org/10.1145/3511861.3511863

GitHub Copilot - Visual Studio Marketplace. (2024a). https://marketplace.visualstudio.com/items?itemName=GitHub.copilotvs&ssr=false#review-details

GitHub Copilot - Visual Studio Marketplace. (2024b). https://marketplace.visualstudio.com/items?itemName=GitHub.copilot&ssr=false#review-details

GitHub Copilot · Your AI pair programmer · GitHub. (2024). https://github.com/features/copilot

Heyman, G., Huysegems, R., Justen, P., & Van Cutsem, T. (2021). Natural language-guided programming. *Onward! 2021 - Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Co-Located with SPLASH 2021*, 39–55. https://doi.org/10.1145/3486607.3486749

Hoda, R. (2022). Socio-Technical Grounded Theory for Software Engineering. *IEEE Transactions on Software Engineering*, *48*(10), 3808–3832. https://doi.org/10.1109/TSE.2021.3106280

Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., & Grossman, T. (2023, April 19). Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. *Conference on Human Factors in Computing Systems - Proceedings*. https://doi.org/10.1145/3544548.3580919

Khan, J. Y., & Uddin, G. (2022). *Automatic Code Documentation Generation Using GPT-3*. https://doi.org/10.1145/3551349

Liu, M. X., Sarkar, A., Negreanu, C., Zorn, B., Williams, J., Toronto, N., & Gordon, A. D. (2023, April 19). "What It Wants Me To Say": Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models. *Conference on Human Factors in Computing Systems - Proceedings*. https://doi.org/10.1145/3544548.3580817

Newbold, P., Carlson, W. L., Thorne, B. M., Columbus, B., New, I., San, Y., Upper, F., River, S., Cape, A., Dubai, T., Madrid, L., Munich, M., Montréal, P., Delhi, T., São, M. C., Sydney, P., Kong, H., Singapore, S., & Tokyo, T. (2007). *Statistics for Business and Economics*.

Nguyen, N., & Nadi, S. (2022). An Empirical Evaluation of GitHub Copilot's Code Suggestions. *Proceedings - 2022 Mining Software Repositories Conference, MSR 2022*, 1–5. https://doi.org/10.1145/3524842.3528470

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2021). *Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions*. http://arxiv.org/abs/2108.09293

Pearce, H., Tan, B., Ahmad, B., Karri, R., & Dolan-Gavitt, B. (2021). *Examining Zero-Shot Vulnerability Repair with Large Language Models*. http://arxiv.org/abs/2112.02125

Prather, J., Reeves, B. N., Denny, P., Becker, B. A., Leinonen, J., Luxton-Reilly, A., Powell, G., Finnie-Ansley, J., Powell, G.-R., Santos, E. A., Denny, ; P, Leinonen, J., Luxton-Reilly, A., Finnie-Ansley, J., Becker, B. A., & Santos, E. A. (2023). ``It's Weird That it Knows What I Want'': Usability and Interactions with Copilot for Novice Programmers. , *1*(4). https://doi.org/10.1145/3617367

Raiaan, M. A. K., Mukta, M. S. H., Fatema, K., Fahad, N. M., Sakib, S., Mim, M. M. J., Ahmad, J., Ali, M. E., & Azam, S. (2024). A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges. *IEEE Access*, *12*, 26839–26874. https://doi.org/10.1109/ACCESS.2024.3365742

Reviews: AWS Toolkit - Amazon Q, CodeWhisperer, and more - IntelliJ IDEs Plugin | Marketplace. (2024). https://plugins.jetbrains.com/plugin/11349-aws-toolkit--amazon-q-codewhisperer-and-more/reviews

Reviews: Codeium: AI Autocomplete and Chat for Python, JS, Java, Go... - IntelliJ IDEs Plugin | Marketplace. (2024). https://plugins.jetbrains.com/plugin/20540-codeium-ai-autocomplete-and-chat-for-python-js-java-go--/reviews

Reviews: GitHub Copilot - IntelliJ IDEs Plugin | Marketplace. (2024). https://plugins.jetbrains.com/plugin/17718-github-copilot/reviews

Reviews: Tabnine: AI Code Completion & Chat in Java JS/TS Python & More - IntelliJ IDEs Plugin | Marketplace. (2024). https://plugins.jetbrains.com/plugin/12798-tabnine-ai-code-completion--chat-in-java-js-ts-python--more/reviews

Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022). Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. *ICER 2022 - Proceedings of the 2022 ACM Conference on International Computing Education Research*, *1*, 27–43. https://doi.org/10.1145/3501385.3543957

Schuster, R., Tech, C., Song, C., & Tromer, E. (n.d.). *You Autocomplete Me: Poisoning Vulnerabilities in Neural Code Completion You Autocomplete Me: Poisoning Vulnerabilities in Neural Code Completion Vitaly Shmatikov*. https://www.usenix.org/conference/usenixsecurity21/presentation/schuster

Stol, K. J., & Fitzgerald, B. (2018). The ABC of software engineering research. *ACM Transactions on Software Engineering and Methodology*, *27*(3). https://doi.org/10.1145/3241743

Stol, K. J., Ralph, P., & Fitzgerald, B. (2016). Grounded theory in software engineering research: A critical review and guidelines. *Proceedings - International Conference on Software Engineering*, *14-22-May-2016*, 120–131. https://doi.org/10.1145/2884781.2884833

Sun, Z., Du, X., Song, F., Ni, M., & Li, L. (2022). CoProtector: Protect Open-Source Code against Unauthorized Training Usage with Data Poisoning. *WWW 2022 - Proceedings of the ACM Web Conference 2022*, 652–660. https://doi.org/10.1145/3485447.3512225

Tabnine AI Assistant - Autocomplete And Chat - Visual Studio Marketplace. (2024). https://marketplace.visualstudio.com/items?itemName=TabNine.tabnine-visual-studio&ssr=false#review-details

Tabnine: AI Autocomplete & Chat for Javascript, Python, Typescript, PHP, Go, Java & more - Visual Studio Marketplace. (2024). https://marketplace.visualstudio.com/items?itemName=TabNine.tabnine-vscode&ssr=false#review-details

Tabnine AI code assistant | Private, personalized, protected. (2024). https://www.tabnine.com/

Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022, April 27). Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. *Conference on Human Factors in Computing Systems - Proceedings*. https://doi.org/10.1145/3491101.3519665

Vaswani, A., Brain, G., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention Is All You Need*.

Wermelinger, M. (2023). Using GitHub Copilot to Solve Simple Programming Problems. *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, *1*, 172–178. https://doi.org/10.1145/3545945.3569830

Wong, M. F., Guo, S., Hang, C. N., Ho, S. W., & Tan, C. W. (2023). Natural Language Generation and Understanding of Big Code for AI-Assisted Programming: A Review. In *Entropy* (Vol. 25, Issue 6). MDPI. https://doi.org/10.3390/e25060888

Yetistiren, B., Ozsoy, I., & Tuzun, E. (2022). Assessing the quality of GitHub copilot's code generation. *PROMISE 2022 - Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering, Co-Located with ESEC/FSE 2022*, 62–71. https://doi.org/10.1145/3558489.3559072

Appendix A. Positive category occurrences

Positive categories	GitHub Copilot	Tabnine	AWS Toolkit	Codeium	Total
Increased productivity	**175**	**113**	**3**	**266**	**557**
- Less typing	45	29	0	53	127
- Provides Ideas	21	4	0	23	48
- Information search	15	2	0	8	25
Increased code quality	**11**	**8**	**0**	**28**	**47**
- Less human errors	4	5	0	5	14
- Suggest code with good conventions	6	1	0	9	16
Helps learning programming	**36**	**19**	**0**	**49**	**104**
- Teaches coding conventions	7	2	0	11	20
- Teaches language syntax	13	3	0	9	25
- Explains code	1	2	0	5	8
Makes programming more enjoyable	**14**	**16**	**0**	**17**	**47**
Good correctness	**80**	**91**	**0**	**128**	**299**
- Accurate suggestions	65	65	0	85	215
- Adaptive coding style	14	34	0	26	74
- Good context awareness	16	4	0	24	44
Good performance	**4**	**7**	**0**	**65**	**76**
- Fast responses	5	7	0	55	67
- Efficient resource usage	0	0	0	5	5
Good usability	**7**	**9**	**0**	**51**	**67**
Satisfied with price and business model	**15**	**9**	**0**	**191**	**215**
Tool has useful features	**83**	**84**	**1**	**252**	**420**
- Documentation/Comment generation	23	6	0	40	69
- Code explanation	1	2	0	23	26
- Code completion	50	72	0	156	278
- Chat assistant	9	7	0	63	79
- Boilerplate code generation	13	3	0	19	35
- Refactoring	1	1	0	16	18
- Debugging	1	1	1	16	19
- Test generation	0	2	0	3	5

Appendix B. Negative category occurrences

Negative categories	GitHub Copilot	Tabnine	AWS Toolkit	Codeium	Total
Decreased productivity.	**20**	**19**	**1**	**4**	**44**
- Review & fix takes time	12	3	0	2	17
Decreased code quality	**13**	**5**	**0**	**4**	**22**
- Too much trust in generated code	11	3	0	4	18
Harmful for learning programming	**6**	**0**	**0**	**3**	**9**
- Novices don't have to understand the generated code to get functioning solutions	5	0	0	2	7
Poor correctness	**101**	**65**	**6**	**13**	185
- Incorrect suggestions	76	47	5	11	139
- Poor context awareness	15	7	0	3	25
- Correctness decrease over time	10	6	0	0	16
- Inconsistency in suggestions	5	2	2	0	9
- Poor language support	10	9	0	0	19
Performance issues	**34**	**43**	**10**	**14**	**101**
- High RAM usage	1	21	1	1	24
- High CPU usage	1	11	1	3	16
- Freezing/slowing system	14	10	2	6	32
- Latency with responses	19	7	5	5	36
Usability issues	**382**	**54**	**41**	**71**	**548**
- Bugs	236	9	12	26	283
- Suggestion override/conflict	27	28	10	12	77
- Repeated login	9	2	13	1	25
- Frequent updates	6	0	0	0	6
Dissatisfaction with price/business model	**182**	**39**	**0**	**1**	**222**
- Expensive	178	17	0	0	195
- Limitations	0	18	0	0	18
- Ads	0	12	0	0	12
Tool is missing features	**87**	**4**	**3**	**18**	**112**
- Documentation/Comment generation	3	1	0	1	5
- Code explanation	1	1	0	1	3
- Chat assistant	48	0	0	5	53
- Refactoring	7	0	0	3	10
- Debugging	2	1	0	1	4
- Test generation	0	1	0	1	2
- Commit message generation	7	0	0	1	8
Security concerns	**6**	**4**	**1**	**1**	**12**
- Tool reads unrelated files	4	2	0	0	6
- Telemetric data sent	2	0	0	1	3
Ethical concerns	**54**	**0**	**0**	**1**	**55**
- Opensource code used for training	24	0	0	0	24
- AI replacing developers	12	0	0	0	12
- Not available in certain countries	19	0	0	1	20