ChatGPT and AI for software optimization

A bachelor thesis on current practices, opinions, and views

Lappeenranta–Lahti University of Technology LUT

Bachelor's thesis

2024

Jaakko Lipponen

Examiner: Associate Professor Ari Happonen

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Science

Software Engineering

Jaakko Lipponen

ChatGPT and AI for software optimization

Bachelor's thesis, software engineering

2024

34 pages, 7 figures and 2 tables

Examiner: Associate Professor Ari Happonen

Keywords: ChatGPT, generative AI, software development, automation, code optimization, AI applications, coding with AI

This thesis investigates the application of ChatGPT and other generative AI tools in software optimization, focusing on their role in enhancing software development. Specifically, this thesis explores their impact on code generation, test automation, and programming optimization. The study is based on a literature review, a comparison of commercial AI tools, and interviews with software professionals.

The findings demonstrate that ChatGPT can notably enhance the efficiency of software development processes by automating repetitive tasks and elevating code quality. Generative AI has been widely found to benefit developers in their work, for example, using ChatGPT for code generation and optimisation can speed up development processes and improve job satisfaction. However, the views of the local professionals interviewed were cautious, especially regarding the reliability of the generated code and the specific requirements of the industry.

Moreover, the study reveals that current AI tools like ChatGPT and GitHub Copilot excel in generating commonly used code and unit tests but require further development for generating more complex and customized solutions. Future research could focus on the long-term implications of AI tools for software development, as well as their ethical and labour market implications.

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUT Teknis-luonnontieteellinen

Tietotekniikka

Jaakko Lipponen

ChatGPT ja tekoäly ohjelmistojen optimointiin

Tietotekniikan kandidaatintyö

2024

34 sivua, 7 kuvaa ja 2 taulukkoa

Tarkastaja(t): Tutkijaopettaja Ari Happonen

Avainsanat: ChatGPT, generatiivinen tekoäly, ohjelmistokehitys, automaatio, ohjelmoinnin optimointi, tekoälyn sovellukset, koodin optimointi tekoälyllä

Tämä kandidaatintutkielma tutkii ChatGPT:n ja muiden generatiivisten tekoälytyökalujen käyttöä ohjelmistojen optimoinnissa. Työssä tarkastellaan tekoälyn roolia ohjelmistokehityksessä, erityisesti koodin generoinnissa, testauksen automatisoinnissa ja ohjelmoinnin optimoinnissa. Tutkimus perustuu kirjallisuuskatsaukseen, kaupallisten tekoälytyökalujen vertailuun, sekä ohjelmistoalan ammattilaisten haastatteluihin.

Tutkimuksen tulokset osoittavat, että ChatGPT voi merkittävästi tehostaa ohjelmistokehityksen prosesseja automatisoimalla yksitoikkoisia tehtäviä ja parantamalla koodin laatua. Generatiivinen AI on yleisesti todettu hyödyttävän kehittäjiä työtehtävissään esimerkiksi ChatGPT:n käyttö koodin generoinnissa ja optimoinnissa voi nopeuttaa kehitysprosesseja ja parantaa työtyytyväisyyttä. Haastateltujen paikallisten ammattilaisten näkemykset olivat kuitenkin varovaisia, erityisesti generoidun koodin luotettavuuden ja teollisuuden erityisvaatimusten osalta.

Työssä havaittiin myös, että nykyiset tekoälytyökalut, kuten ChatGPT ja GitHub Copilot, ovat erityisen hyödyllisiä yksinkertaisten koodimallien ja yksikkötestien generoinnissa, mutta vaativat edelleen kehitystä monimutkaisempien ja asiakaskohtaisten ratkaisujen tuottamisessa. Tulevaisuuden tutkimus voisi keskittyä tekoälytyökalujen pitkän aikavälin vaikutuksiin ohjelmistokehityksessä, sekä niiden eettisiin- ja työmarkkinavaikutuksiin.

SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
CI/CD	Continuous Integration and Continuous Deployment
GPT	Generative Pre-trained Transformers
IaC	Infrastructure as Code
IDE	Integrated Development Environment
IT	Information Technology
LLM	Large Language Model
NLP	Natural Language Processing
VCS	Version Control System

Table of contents

Abstract

1 Introduction 6
2 Background 8
2.1 Historical Timeline of Development Tools 8
2.1.1 Early days of SWE tools -1980s 9
2.1.2 Modern tools and testing automation 1990-2000s 10
2.1.3 CI/CD and DevOps during the 2010s 11
2.1.4 Recent Advancements in software development 2020- 12
2.1.5 Technological Foundation of ChatGPT 13
3 Applying generative AI into software engineering 15
3.1 Current research 15
3.1.1 Leetcode solution generation and debugging 15
3.1.2 Related opinions and effects of usage 18
3.1.3 The Efficacy of ChatGPT in Automated Test Generation and Optimization 20
3.2 Comparison of commercial AI programming automation tools 21
3.2.1 User experience and interface 22
3.2.2 Technology, capabilities, and limitations 23
3.2.1 Claims of AI code generation tools. 24
4 Professional views on coding with AI 25
4.1 Background 26
4.2 Technical Questions 27
4.3 Perspectives on optimization 27
4.4 Implications of the results 28
5 Conclusions 30
References 31

1 Introduction

The incorporation of Artificial Intelligence (AI) across diverse sectors is the start of a pivotal shift in the landscape of technology and business across different fields. Advanced data analysis, data-based forecasting (Ghoreishi and Happonen, 2020), design, development, automation and optimization benefit greatly from the employment of AI (Tamasiga et al., 2023).

Generative natural language models like ChatGPT have gathered significant attention, while bringing new possibilities for various sectors, including software development. ChatGPT, developed by OpenAI, represents the cutting-edge of language models that can generate coherent and contextually relevant text over extensive user prompts. ChatGPT is claimed to be especially great at writing and manipulating text (Sarrion, 2023).

With the recent release of GPT-4-Turbo model, significant improvements are seen in the LLM's technical capabilities: it can now handle up to 300 pages long text as a single input prompt, the information is processed much faster than with GPT-4, and the price has gone down significantly. With OpenAI API, input to GPT-4-Turbo is three times cheaper and output is two times cheaper ("GPT-4 Turbo in the OpenAI API | OpenAI Help Center," 2024). Along with these upgrades, GPT's coding abilities have seen an upgrade to code accuracy and completeness and error reductions ("Microsoft Makes Powerful GPT-4 Turbo Free for All Copilot Users," 2024). A chart of some of the key differences is listed in the comparison chapter.

The impact of AI extends further into software development and automation, where it has become a catalyst for innovation and efficiency. AI-driven development tools are transforming the software creation process, automating mundane tasks, optimizing code, and reducing error rates. This shift in approach has significantly enhanced both the efficiency and quality standard of software development (Kortelainen et al., 2019). From extensive use of AI also rises some concerns with ethics and difficulties with control (Usmani et al., 2023a).

This thesis seeks to highlight the potential benefits and challenges of integrating ChatGPT-based tools into software development cycles. This paper contains the following sections:

literature review, comparing commercial solutions, and surveying professionals in the field, this research will offer a comprehensive view into the current state of software development, using generative tools based on OpenAI's GPT-models.

The focus of this thesis will remain on the perspective of automating software development processes with ChatGPT, and not delve too deeply into the more technical part of how ChatGPT and generative AI function. Currently the newest available version of ChatGPT, GPT-4, will be examined and compared in short, while focusing mainly on the tool in general for this use case. In addition, the thesis will briefly compare other similar tools, such as GitHub Copilot, Amazon Code Whisperer and TabNine for development automation with other AI tools.

Integrating one of the previously mentioned generative AI tools into software development workflows holds the potential to significantly improve efficiency, streamline processes, and enhance productivity. The rapid rise in the popularity of ChatGPT, as evidenced by its record-breaking milestone of attracting 100 million users in just two months, highlights the timeliness of examining its impact and utility in software development (Chow, 2023; Edwards, 2023).

2 Background

This section will cover background information on the history of some of the most significant development tools related to this topic, tracing their evolution from the early days of software development to the technology we now have currently. It highlights significant moments and advancements in the field to provide a clear understanding of how software development practices have developed over time.

2.1 Historical Timeline of Development Tools

Better automation, like unit testing robotization for written code, can improve developer work experience, while offering other important benefits, such as potentially improving code quality and quantity simultaneously. Resources can then be allocated more effectively on more important matters with better automation and tools. ChatGPT, Software development automation, and optimization tools are core to the subject. Models like ChatGPT, are prime examples of generative AI in action. Automation in software development streamlines multiple stages of the software development lifecycle, including coding, testing, deployment, and maintenance. Generative AI can significantly enhance these processes. Optimization tools improve the efficiency and performance of software. Generative AI can play a key role here by analysing existing codebases and suggesting optimizations or even generating more efficient code.

These can best be understood while being aware of their technological backgrounds. In this chapter, light is shed on the evolution of development automation, and the foundational technologies supporting generative AI's like ChatGPT, will be unpacked in more detail.

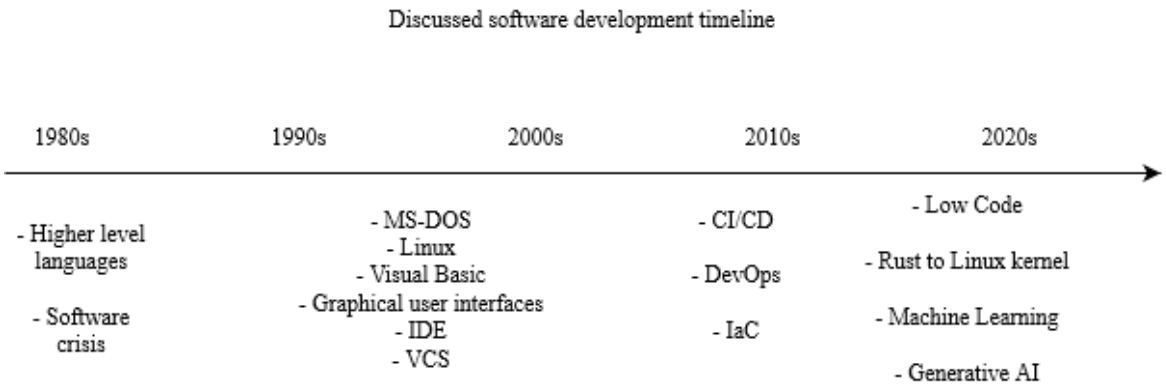


Fig. 1: Discussed software development topics timeline

2.1.1 Early days of SWE tools -1980s

The beginning of software development automation tools can be seen to be the creation of Fortran in the 1950s, the first high-level programming language. Fortran sparked the transition from writing manual machine code to automatic programming with a high-level language. It did solve a lot of issues at the time, but most notably it was a common language that could run on multiple machines when compiled for them. Before high-level languages and compilers, programs had to be specifically made for each machine it needed to run on (Backus, 1978; Wirth, 2008).

From there on, in the 1970s to -80s along came the so-called software crisis. This was mainly caused by the lack of knowledge in the field to scale smaller projects to larger projects. Common development method used back then was to read requirements, code a feature and then fix the code, this was repeated until the project was completed. This caused severe issues in large projects, such as missing deadlines, exceeding budgets, and unnecessary complexity in the resulting product. In 1987, Fred Brooks “No silver bullet” recognized that there would not be a singular tool or fix that would solve everything, but instead little improvements everywhere would amount to a bigger change for the better (Tasnim, 2023; University of Cape Town, 2011). Computers became much smaller and more common during this period, causing personal computers to become more common, which increased the demand for software development (Wirth, 2008).

2.1.2 Modern tools and testing automation 1990-2000s

Along with personal computers, Microsoft's operating system, known as MS-DOS, became increasingly popular. First versions of Windows were initially built to work on a MS-DOS base system. Windows provided a graphical user interface to text-based MS-DOS. With the rise of graphical user interfaces, also graphical development became of interest, resulting in visual development environments. One of the first ones was Microsoft's Visual Basic, which released in 1991. With graphical development tools, the developers could drag and drop user interface elements into place, instead of just programming them and checking if it worked. These graphical user interface elements could then be hooked up to different kinds of functions and methods to handle functionality (Intersimone, 2010; "What is IDE or Integrated Development Environments?," n.d.).

From this more Integrated Development Environments (IDE) followed, like Microsoft's Visual Studio and Eclipse in 2001 for Java, backed by IBM. A modern IDE usually has everything one would need to develop and test programs in appropriate environments or platforms. This typically involves a code editor, compiler, debugger, and a graphical user interface that integrates these tools and their functionalities (Heaton et al., n.d.).

Version control systems (VCS) gained popularity with better development environments and development practises. Version control systems can be traced back to 1970s with Source Code Control System, developed by Bell Labs. VCS allows for the tracking and management of changes to code, documents, or any set of files, thereby providing a historical record and facilitating collaboration among contributors (Ruparelia, 2010).

The primary purpose of version control systems is to ensure code integrity, manage multiple versions of the project, and enable collaboration between multiple developers. VCS gained popularity in early 2000s with Subversion holding most of the market share. Git, which released in 2005, was developed by Linus Torvalds. Git was created to replace the existing VCS Torvalds was using to develop the Linux kernel. Git is now the most used VCS out there, and the use of Git is recommended in every project, small or large. Nowadays, implementing VCS is fundamental in software development projects and collaborative document management ("Git - A Short History of Git," n.d.).

2.1.3 CI/CD and DevOps during the 2010s

One of the next major leaps in software development automation after IDEs were Continuous Integration and Continuous Deployment/Delivery (CI/CD). CI/CD practices became fundamental in modern software development projects quickly. CI/CD has the potential to enhance organizational productivity, efficiency, and streamline workflows through automation, testing, and collaboration with version control systems.

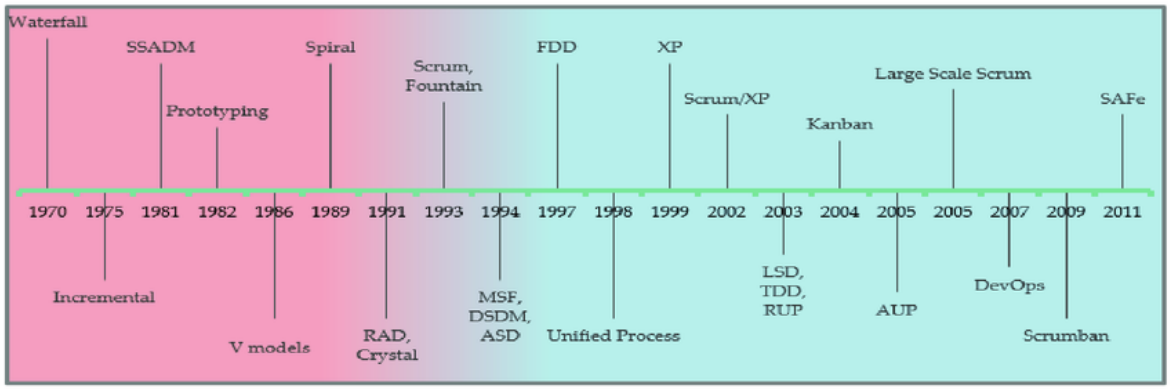


Fig. 2: Common development practises and their evolution timeline from: ("Figure 1. Software development methodologies timeline.," n.d., p. 1)**(**CC-license**)**

As software projects scale, the features of CI/CD become increasingly beneficial, aiding in quicker and more reliable delivery of newer versions. It achieves this objective by starting the CI pipeline of integration processes with first completing automated testing, each time a merge or a change is made in the main branch of the VCS repository. Code validation processes can involve automatic code analysis and testing, then after testing the branch will be automatically compiled and packaged, ready for delivery to the users. After the CI pipeline begins the Continuous Delivery pipeline consisting of delivery or deployment processes. CD processes ensure that the latest packaged version has everything needed to be deployed. CD can be made to manage the exact deployment environment's infrastructure automatically for the package to be as needed, making sure that the software can be provided to users whenever the developers wish to. Continuous Deployment differs from Continuous Delivery in that it allows for automatic publication of the software, while Continuous Delivery needs the developers to manually publish the software (Rossel, 2017; "What is CI/CD?," n.d.).

Down the line, from the relatively narrow set of CI/CD practices and agile development, evolved a much broader set of practices called DevOps (Development & Operations), combining fast paced lean and agile development, quality assurance, CI/CD, and many other practices into one set. The four fundamental principles of DevOps include automating processes, encouraging collaboration and communication, continuously improving and minimizing waste, and prioritizing user needs with rapid iteration cycles. With DevOps, the same team responsible for developing the software, deploys, and maintains the software. This brings efficiency and quality gains when the team with best knowledge on the software is responsible for its functionality for its entire lifecycle. DevOps, in the early phases of the software lifecycle, is designed to shorten the time spent on development and testing, while increasing organizations pace of production and quality (Ebert et al., 2016; "What is DevOps?," n.d.).

After development, DevOps can make deployment easy and quick with the addition of cloud-based infrastructure as code (IaC) into the workflow. IaC allows for automation on provisioning and managing infrastructure, which the software will be deployed on. Compared to manual platform configuration, IaC brings great improvements in efficiency, consistency, optimization, and cost savings. With IaC organizations can easily set up, modify, and replicate their infrastructure. IaC has the potential to provide monetary benefits with freeing up time from repetitive tasks and enabling better control, and monitoring over resource usage and allocation ("Infrastructure as Code | IBM," n.d.; "What does infrastructure as code mean?," n.d.).

2.1.4 Recent Advancements in software development 2020-

From the year 2020, automation in software development has been advancing swiftly, with new practises and methods being increasingly embraced into workflows. One of these is the emergence of Low code and even No code development platforms, such as Microsoft's Power Apps and Google's AppSheet. Low code development usually is done by drag-and-dropping elements to their desired locations, customizing their features, and finally adding functionality. Therefore, low code development platforms do not require traditional coding knowledge for the most part or at all, making developing simple applications accessible to almost anyone ("Business Apps | Microsoft Power Apps," n.d.; Woo, 2020).

Manual programming has also evolved with new languages taken to professional use, most significant of these being Rust-language. The importance of Rust can be highlighted by it being the only other language to be added to the Linux kernel in addition of C in 2022 with kernel version 6.1. Rust shines in comparison to the competing languages with its unique combination of safety, security, ease of use, while offering performance similar to C and C++(Bugden and Alahmar, 2022). Rust's automatic memory safety features can reduce up to 70% of security vulnerabilities in comparison to languages like C. This collaterally improves the pace of development and security of programs written in Rust, as most security bugs take advantage of memory exploits (Bugden and Alahmar, 2022; Proven, 2022).

2.1.5 Technological Foundation of ChatGPT

One of the most popular technologies as of late is Generative AI (Edwards, 2023), which has been making headlines, with the release of OpenAI's GPT models. ChatGPT in its current form is a web application, in which a user can have a realistic conversation with GPT-models. An answer will be provided to the user based on their input. These text responses by ChatGPT can include anything, which can be processed in text form. For example even creative task involving writing, translation, summarizing text and answers to math questions are valid (UCA, 2023; "What is GPT AI?," n.d.).

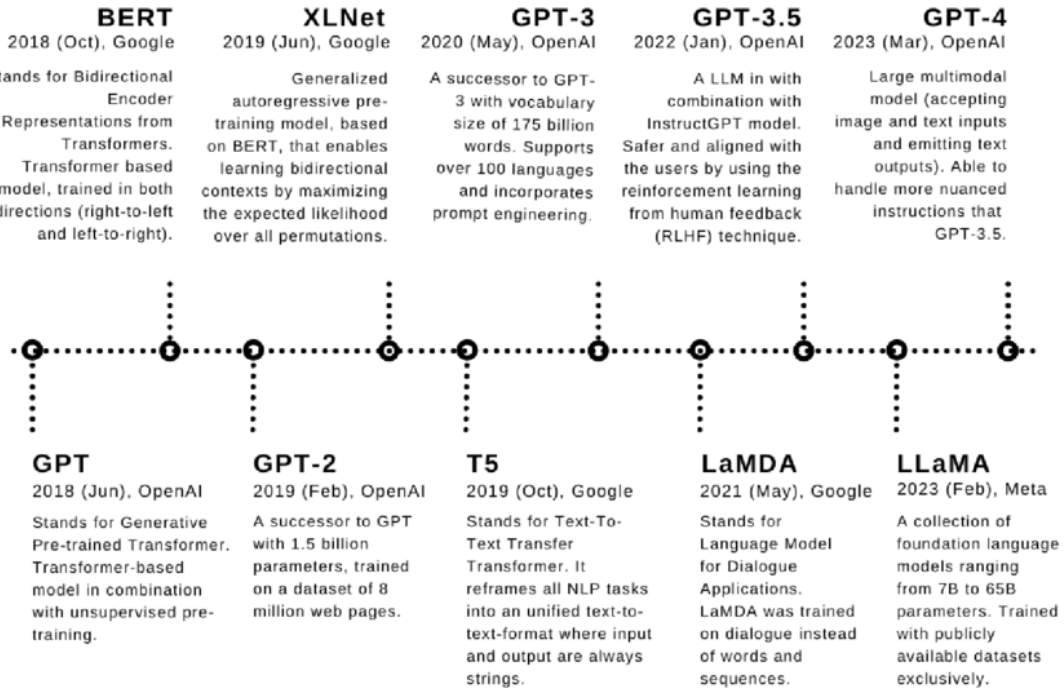


Fig. 3: Modern LLMs release timeline diagram (CC-license) (“Figure 1. Modern LLMs timeline diagram,” 2024)

ChatGPT, short for generative pre-trained transformer, is a language model with access to a large pool of training data and processing parameters, commonly referred to as a Large Language Model (LLM). OpenAI develops ChatGPT and other models for affordable public access. The company seeks to provide a highly accessible natural language processing model with billions of parameters to the general public, developers and corporations to use freely for a relatively affordable fee (“OpenAI Platform,” 2023).

Large Language Models belong to Natural language Processing (NLP) branch of artificial intelligence. This branch of AI aims to enable human-computer interaction with natural languages. NLP combines rule-based linguistics modelling and machine learning to achieve this goal (“What is Natural Language Processing?,” 2023). Large Language Models like ChatGPT are built on state-of-the-art neural networks that use the transformer architecture. Transformer models are efficient in gathering information from sequential data streams. A good example of this would be the common flow of natural human languages represented in text form. The meaning of words is learnt through the context in which it is used, from this a multi-dimensional vector is constructed, other words with similar meaning are placed near each other in the vector space. From this, the relationships between different words is mapped, and the words can now be applied to output the desired answer to the user’s prompt (Merritt, 2022; “What are Large Language Models?,” n.d.).

3 Applying generative AI into software engineering

3.1 Current research

This chapter reviews prior research on the topic and on related subjects. While the application of generative AI in software engineering support is novel, yet it has attracted significant scholarly attention, as there are many different domains to consider, including safety and AI control (Usmani et al., 2023a), security (Usmani et al., 2023b) and ethics (Usmani et al., 2023c) point of views. Various studies have investigated the efficacy of AI integration in development processes. This section will discuss selected publications and their findings and possible research gaps.

3.1.1 Leetcode solution generation and debugging

ChatGPT's code generation skills were tested by (Sakib et al., 2023). The paper examined the viability of using ChatGPT to solve different programming questions. It offers insights into the capabilities of ChatGPT in generating code, which is essential for evaluating its effectiveness as a programming aid. The research was conducted by examining and comparing the resulting code for correctness and performance in time and memory complexity. ChatGPT was prompted with 128 coding problems involving algorithms and data structures, from the website "LeetCode", ranging from easy, medium, to hard, with varying LeetCode acceptance rates for a wide range of cases.

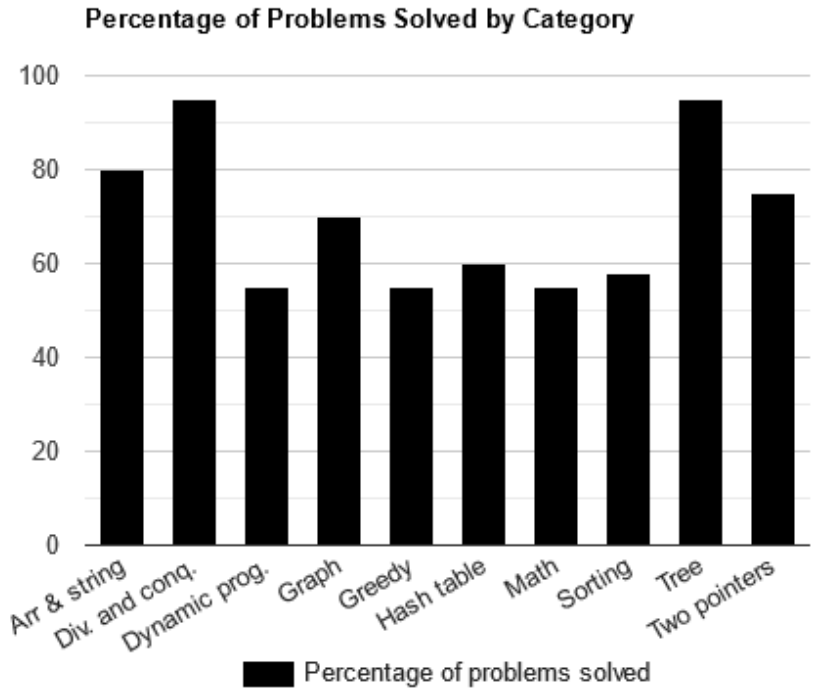


Fig. 4: GPT's performance by category in LeetCode. Based on fig. 4/table 1 (Sakib et al., 2023).

The results showed that ChatGPT succeeded in about 72% of the coding problems it was tasked with. The highest success was found in problems, that follow clear guidelines to solve them, such as tasks where it is feasible to divide the main task into smaller sub-problems, then combining the sub-problems to solve the main task.

In coding problems, which require non-structured solutions, specifically in greedy- and dynamic programming related questions, ChatGPT's success rate dropped significantly. These two were the problem domains with the lowest success rates, whilst the highest success rates were seen in "tree" and "divide-and-conquer" -type problems.

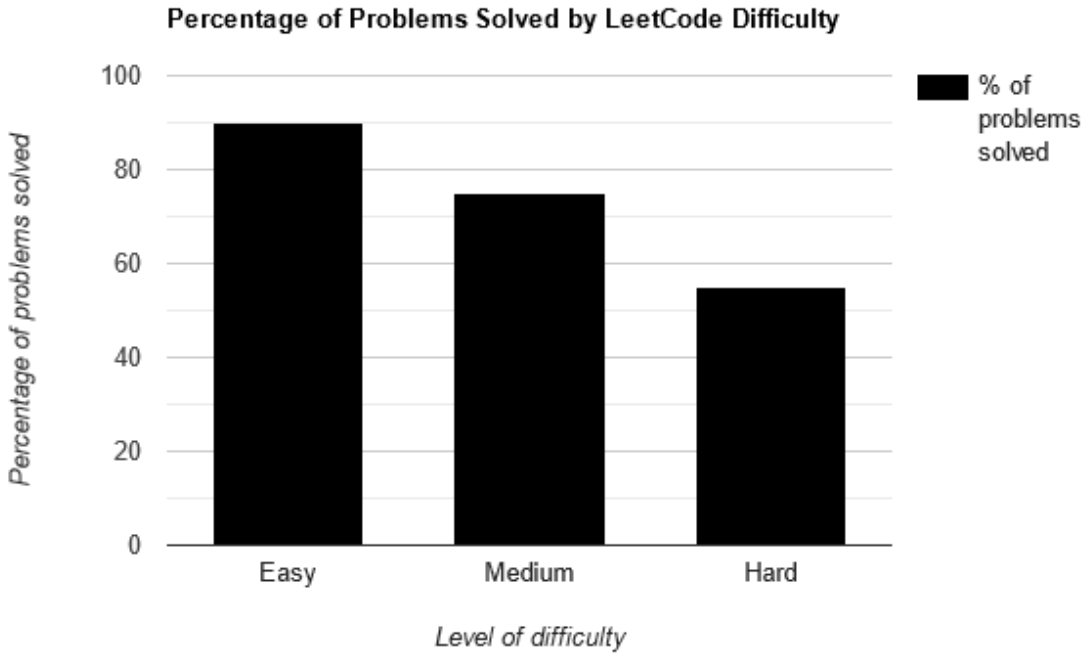


Fig. 5: Acceptance rate of GPT's solutions by level of difficulty. Based on fig. 6 (Sakib et al., 2023)

In the paper, ChatGPT's success rates showed correlation with the task's difficulty and acceptance rate. In tasks categorized as "hard", ChatGPT's success rate was reduced from the average of 72% to 55%. Similarly, in tasks categorized as "easy", the success rate rose to as high as 90%. ChatGPT's debugging skills were also put to the test. In about 30% of the cases, where ChatGPT failed to produce a valid code solution, it was given feedback from LeetCode on why it was not a valid solution. Despite this, only 36% of the updated solutions improved and passed LeetCode's tests, while most of the time (in 67% of the cases) the solutions updated with feedback, performed worse in LeetCode's tests.

This demonstrates the practical capabilities of AI in a real-world setting, and identifies specific areas, where AI tools like ChatGPT excel, and where they need improvement. The insight from these aspects is important for understanding AI technologies capabilities for software development, and for informing users about the realistic expectations and applications of these tools.

3.1.2 Related opinions and effects of usage

The usefulness and riskiness of Copilot and ChatGPT depends on the users experience level and how it is utilized. (Bull and Kharrufa, 2023) explore how generative AI affects students and people working in unfamiliar areas, depending on how it is used, in their paper "Generative AI Assistants in Software Development Education". Instead of trying to prevent students from using generative tools at all, it would be more beneficial to think of how the tools should be accounted for and integrated into the assignments. (Bull and Kharrufa, 2023) share recommendations from their future vision of teaching programming. Generative AI should be used to help software engineering students become problem solvers with support from AI tools. Tasks given to students must collaterally see significant changes to account for generative AI. Shifting the assignments from simpler types to needing more applied knowledge on the topic, and requiring full context, will reduce the risk of generative AI tools to be used for generating full solutions. Integrating code reviews into courses would help in developing the problem solving, and code quality detection skills of students.

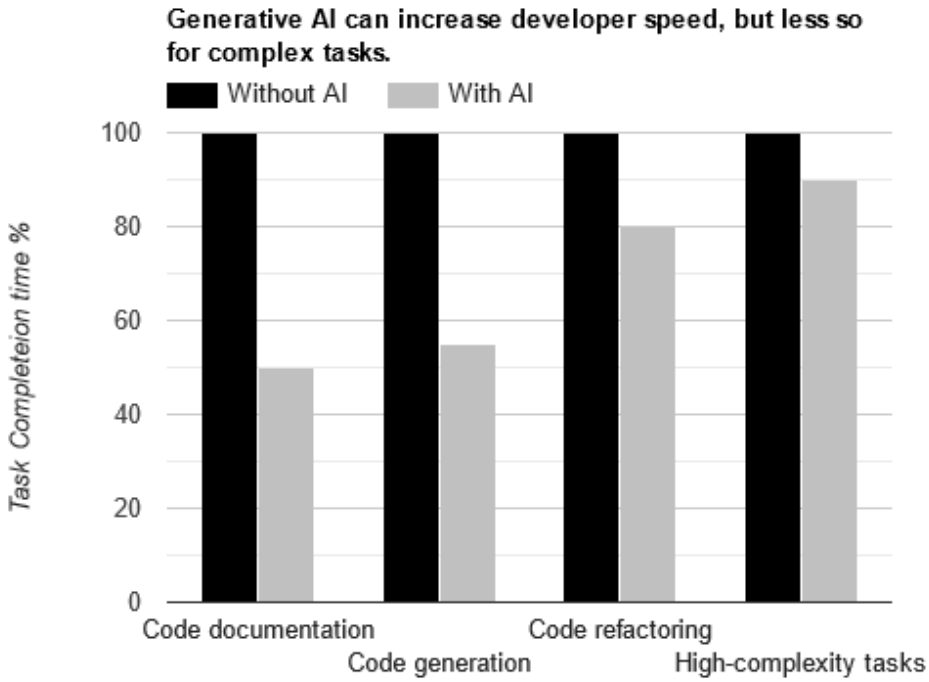


Fig. 6: Efficiency improvements with and without AI compared. Based on Exhibit 1("Unleash developer productivity with generative AI | McKinsey," 2023)

(Bull and Kharrufa, 2023) interviewed five software development professionals from the field, working in various environments with different levels of experience, on their opinions and practices related ChatGPT and Copilot. Some used generative AI tools more than others, most being exited for the future potential of these tools being integrated into development workflows. ChatGPT was generally preferred over Copilot in most cases, with Copilot gathering criticism for it lacking project context, and its inability to provide feedback, or have access to previous information. ChatGPT's conversational model was its greatest strength when compared to Copilot, which functions more like a code-complete system in an IDE suggesting code inside the editor. ChatGPT could generate the code, then explain it to the user, causing the user to learn more on how the code functions. ChatGPT has access to the full conversation, having more context and access to information which the user has given previously.

ChatGPT can be used in a business environment more than Copilot, mainly because the user must explain their code and request in the chat. This allows for abstraction and not giving the LLM full access to the actual codebase. Copilot is integrated into the IDE, and it will learn from the edited codebase, which in a corporate environment is highly unwanted. The five professionals were also asked what features they would like to see in the future being implemented into ChatGPT and/or Copilot. Key highlights given in the paper were: understanding of the business logic and file structure, the ability to run the tools offline, and including diagrams, citing used sources, and crawling the project codebase to suggest optimizations and improvements on its own.

The study provides a perspective on the adoption of generative AI tools in education and professional practice. Highlighting the potential of these tools to transform learning and work, while also showing the importance of strategic implementation and continuous evolution in response to user needs and industry demands. This paper not only informs educators and professionals about the effective use of AI in software development but also contributes to the broader discourse on the future of AI-assisted education and professional development.

3.1.3 The Efficacy of ChatGPT in Automated Test Generation and Optimization

ChatGPT and other similar LLM-based tools, hold the potential to improve even more traditional development strategies, like test driven development. Currently test-driven development uses search, and limitation -based automatic test generation tools. ChatGPT's potential for automatic test generation is investigated closely in a paper by (Yuan et al., 2023). ChatGPT's proficiency in test generation is compared to Evosuite – a more traditional tool to automate test generation – and AthenaTest, a competing learning-based test generation tool. The code used in the paper's research is taken from a large dataset consisting of different kinds of java projects that use maven for testing. The generated tests were then evaluated by a few criteria: correctness, coverage sufficiency and user readability and usability.

Next the results were examined: with basic generation prompts, only one fourth of the tests generated by ChatGPT successfully met all criteria and were valid, about 58% had compilation errors, and about 17% compiled successfully, but failed when the tests were run. One notable issue with ChatGPT is that it doesn't try to automatically fix the faults in its generated tests, like the more traditional tool Evosuite does. However, when given a more optimized prompt with context on the project and adding post-generation validation by giving ChatGPT a chance to try iterating on the failed tests, the success rate of the generated tests rose significantly to 41% passing, and 73% compiling, with 100% syntactical correctness. (Yuan et al., 2023) The paper calls this method of iterative test generation "ChatTester".

Most of the test which failed to compile might be caused by ChatGPT's lack of deeper knowledge on the code and the project. Errors, which occur during execution are caused by ChatGPT's lack of understanding of the method in question, which the test is written for.

Compared to AthenaTest, a learning-based tool, findings point to ChatGPT generating more correct tests with better coverage of the code. The working tests were found to be comparable to manually written tests with 82.3% coverage, compared to 84.2% for manual tests, while the competing tools achieved 65-68% coverage (Yuan et al., 2023).

Another important criterion for tests generation is user readability and usability. For this a small team of 5 professionals were asked to rate the tests generated by the tools. Human

feedback -based learning is suspected to help ChatGPT write more human-like tests with good readability, on par with manually written tests, outshining the competing tools. In usability ChatGPT's tests held up well being even preferred over manually written tests in some cases. According to the paper, ChatGPT seems to be able to follow good testing practices.

COVERAGE OF GENERATED TESTS

Coverage (%)	ChatGPT	AthenaTest	Evosuite	Manual
Statement	82.3	65.5	68.0	84.2
Branch	65.6	56.2	61.2	68.9

Table 1: Generated tests which pass execution. Based on Table V (Yuan et al., 2023)

The research by (Yuan et al., 2023) demonstrates that ChatGPT and similar AI tools have significant potential to enhance traditional software development practices like Test-driven development. The ability of ChatGPT to generate high-quality, human readable, and usable tests, especially when given optimized prompts, aligns with the objectives of this thesis. It highlights the transformative impact of AI on software optimization, not just in automating tasks, but in elevating the quality and efficiency of essential development practices. This chapter provides compelling evidence of the beneficial integration of AI in software development, a key theme of the thesis.

3.2 Comparison of commercial AI programming automation tools

The following section of the paper contains a detailed comparison of commercially available generative software development tools. A general comparison between the tools will be conducted for each separately. A special case will be made for ChatGPT, as it has two available versions to the public to use, whereas other tools do not. ChatGPT's two available versions GPT-3.5 and GPT-4.0 will be compared in short and the general short-term evolution of the tool will be highlighted. In addition to ChatGPT, Google Bard, LLAMA 2.0 and GitHub Copilot will be compared to each other. Tools which are based on OpenAI's models like Codium AI will not be discussed as they are very similar to GitHub Copilot and ChatGPT in terms of functionality. The new GPT-4-Turbo model is assumed to be faster, more comprehensive, and more accurate with software development tasks ("Microsoft

Makes Powerful GPT-4 Turbo Free for All Copilot Users," 2024). GPT-4-Turbo is generally seen as an incremental upgrade to GPT-4 in speed and cost, more details listed below:

Feature	GPT-4	GPT-4-Turbo	GPT-4o
Processing Power	Standard	Increased	wider support for input combinations (text, audio, image, video)
Knowledge Base	Relatively wide range of information	Wider and more code-specific	Up-to date and optimized.
Code Completion	Suggests relevant code	Suggests more accurate and complete code	Suggests more accurate and complete code
Error Reduction	Limited errors	Reduces errors further	Reduces errors further
Creative Exploration	Allows for new functionalities	Prioritizes new functionalities in "Creative" mode	Prioritizes new functionalities in "Creative" mode

Table 2: GPT-4 compared to GPT-4-Turbo and GPT-4o in coding according to ("Hello GPT-4o," 2024; "Microsoft Makes Powerful GPT-4 Turbo Free for All Copilot Users," 2024).

The new multi-modal "GPT-4omni" model shows promise in OpenAI's development towards more human-like AI model interactions, accepting any combination of human input with text, audio, video, photos and speech. The natural feeling fast paced speech dialogue with the user is heavily emphasised in this model's feature set. ("Hello GPT-4o," 2024).

3.2.1 User experience and interface

The most common interface for generative AI tools is a web-based chat messaging interface. ChatGPT and Bard have a similar chat interface accessible on the internet. Using the models

through the web interfaces is simple and easily accessible to beginner users. Google Bard offers additional tuning for the responses in terms of tone, length, and additional automatic fact checking from Google search. Copilot differs slightly having the chat interface integrated into a Visual Studio Code plugin. Meta's LLAMA is open source with the source code available in a Meta-Research GitHub Repository (meta-llama, 2024). However, to get the LLAMA model functioning, approval from Meta's website is needed to download the model weights and tokenizer (Meta-AI, 2024). Unofficial LLAMA chat interfaces do already exist for public use on the internet.

In addition to the web-based interfaces, ChatGPT has an open API accessible to anyone for a relatively low cost. Google Bard's API is not accessible to the public yet. After getting approval for the LLAMA source code, it can be deployed as needed locally. Copilot has additional programming focused features such as workspace analysis, and code generation without the need to use the chat feature, like intelligent autocompletion and quick fix integration inside of Visual Studio Code ("GitHub Copilot · Your AI pair programmer," 2024).

3.2.2 Technology, capabilities, and limitations

ChatGPT utilized OpenAI's GPT-3 and GPT-4 with GPT-4 according to estimates having 1.7T parameters, which is over 10 times more than GPT-3 (Schreiner, 2023). Parameters in transformer models are internal variables that the model learns through training. They are numerical values (weights) within the neural network, and they determine how the model processes and generates language. In short, parameters reflect the model's capacity or complexity (Radford et al., 2018).

Meta's largest Llama 2 is relatively small with only up to 70B parameter model available currently(Meta-AI, 2024). GitHub Copilot according to latest information is still using the latest revision of GPT-3 based model named codex. Codex is trained specifically on code from selected GitHub repositories ("OpenAI Codex," n.d.). Codex is most skilled in python, and other languages like JavaScript. Codex is enhanced with a larger memory size for programming, having 14KB of memory to use for code, whereas regular GPT-3 only has 4KB available for information about the coding project.

Google Bard is built on the current LaMDA model with estimated 137B parameters (Thoppilan and et al, 2022). Both are technically using a similar transformer-based model; however, some differences do remain. Google's model is specially designed around conversationality and enhanced factuality, being trained with filtered training data and integrated fact checking (Singh et al., 2023). Bard also has access to the current information on the internet, whereas ChatGPT-3 does not (Singh et al., 2023). ChatGPT-4 has recently gotten access to current internet data, but the model currently behind a subscription paywall.

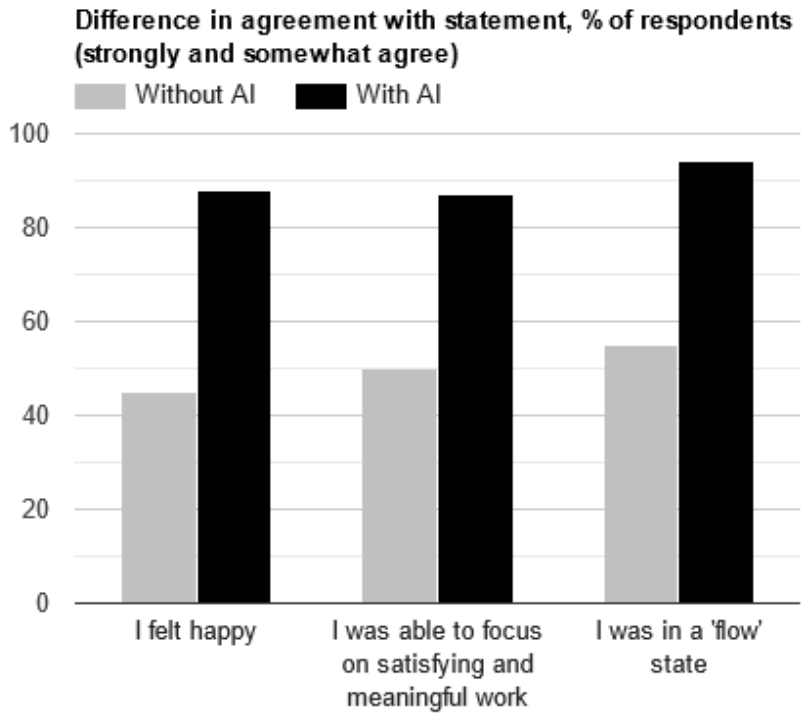
3.2.1 Claims of AI code generation tools.

Generative AI tools generally claim sizeable efficiency gains, with tools designed for specific use cases, like GitHub Copilot, introduced as "Your AI pair programmer," claims 55% faster coding ("GitHub Copilot · Your AI pair programmer," 2024). Tools with broader applications, such as ChatGPT, claim high adaptability across various domains and subjects, serving to augment productivity and assist in creative tasks ("Introducing ChatGPT," n.d.). Google's Bard claims high factual consistency and accurate information dissemination, setting it apart from its counterparts (Pichai, 2023). Leveraging Google's vast data repositories, Bard aims to provide users with not just linguistically coherent, but also factually reliable content. This feature is particularly crucial in domains where the accuracy of information is important. Nevertheless, the factual correctness of AI-generated content is an ongoing challenge with bard too, continuous updates and rigorous data validation are essential to maintain the integrity of these tools.

Meta's LLaMa stands out by presenting a generative AI model that is not only more open and efficient but also more compact, having fewer parameters than its rivals, without compromising on efficiency and accuracy (Touvron et al., 2023). The leaner design means it uses fewer computational resources, which could make advanced AI tools more accessible to a broader audience. However, it's important to consider the balance between the model's size, its efficiency, and its ability to handle a wide range of information.

4 Professional views on coding with AI

To gain a deeper insight into the impact of utilizing generative AI tools in programming-related tasks, a structured interview was carried out with software developers from two distinct, Lappeenranta-based IT and software development firms. One developer from each company volunteered to answer a series of questions related to the themes of this paper. The two participating developers will bring a relatively diverse view on AI tools due to their differing levels of experience and seniority. First developer has 3-5 years of experience, and the second developer has over 15 years of professional experience. For confidentiality reasons, both the companies and the participating individuals have chosen to remain anonymous in this publication and will be referred as developer 1 and developer 2 from here on out.



Generative AI tools have the potential to improve developers' working experience. Based on exhibit 3 ("Unleash developer productivity with generative AI | McKinsey," 2023)

4.1 Background

This interview was conducted remotely with pre-written text questions, the questions were provided to two medium-to-large sized local firms' selected employees. To ease the process, a Google Forms document was used to present the interview questions and record the answers. Google Forms provides many benefits over a more traditional way of interviewing, such as being able to answer the questions easily remotely, with minimal friction, and during the most convenient timeframe for the respondent. The forms-based interview consisted of three parts: 1. background questions, 2. technical questions and 3. opinions on the outlook of AI. The first set of questions were meant to provide context into the volunteer as a professional, their knowledge and experience in AI and in software development. In this chapter, we will be focusing on the first set of questions and the second set of questions will be discussed in more detail during chapter 4.2 and finally the third set of questions is discussed in chapter 4.3 with conclusions and implications in chapter 4.4.

First two questions were "Your role at the company" and "Years of professional experience in software development". The interviewed answered to the questions with the role of "Software developer", with 3-5 years of experience, and over 15 years of experience, working on software development projects. Context on AI tool knowledge was required to provide solidity on the opinionated answers. The interviewed were asked: which AI development tools they were familiar with, their confidence to use the AI tools effectively to solve solutions, and their enthusiasm level on generative AI coming more common place in software developers' daily lives. These questions were answered on a scale from 1 to 5, with higher being better, apart from AI tools question being a multiple-choice question with 8 most popular publicly available tools, custom-built tools, and none as options.

ChatGPT's version 3.5 and GitHub Copilot were selected as the preferred tools the first interviewee with relatively good confidence to be using them effectively. The second developer has a more diverse tool selection with Custom AI tools, chatgpt-3, Copilot, TabNine and Google Bard, with relatively good confidence to be using them effectively. For the enthusiasm question, the developers were low- to indifferently enthusiastic on the impact of AI tools for software development. Lastly a common consensus question on their opinion on AI being generally ready to be integrated into development was asked to base the next set of questions. This question was also given a 1-5 scale, with higher being better. Both

developers choosing to answer with a 2 out of 5 for readiness to fully integrate generative AI tools to development workflows.

4.2 Technical Questions

This section will detail set 2 of the interview questions, focusing on more technical aspects of generative AI tools. These questions are designed to provide insight into when, what, and how generative AI tools are used in the line of professional software development work.

First question of set 2 was a follow up on the tools previously inquired about: “Which generative AI tools do you or your company currently prefer to use for work”. Also, a reasoning of this choice was asked in short free form text. For the first developer, for the two tools mentioned (ChatGPT and copilot) only Copilot was used as it is approved for use in company projects. The second developer chose ChatGPT, Copilot, TabNine and Google Bard (renamed to Google Gemini) for use in company projects.

Next questions are on the use cases of generative AI and when a choice to use generative ai is usually made. Highlighted on the use cases for Copilot were template code generation for both developers, first developer also used AI tools for testing purposes in unit test creation and the second developer uses AI to help with solving problems. The free form reasoning for these use cases mainly involved removing the repetitive and monotonous tasks from development. Test cases often need rewriting, and as this is seen as a repetitive task, in which AI is seen as a significant improvement for this area of development. Improved testing shows as a benefit when a pull request is made to the code repository for the first developer.

4.3 Perspectives on optimization

This section will cover the two developers’ perspectives on the impact and contribution of AI in software development. The last set of questions will be directly quoted and later analysed in the next chapter 4.4. Next question is “Has the integration of AI tools like ChatGPT impacted your software development processes”, to which the following answers were given:

- 1. developer: “[AI] Autocomplete is more comprehensive. And at the same time more unreliable. As in I get more code from it, but I need rewrite more of it too.”
- 2. developer: ”Only little [impact]. This is probably because of industrial customer segment, in which generic code offered by these tools is not enough.”

Second question was “Has the integration of AI tools like ChatGPT impacted your software development processes”, to which the following answers were given:

- 1. developer: “They can't, for now.”
- 2. developer: “These [AI tools] can help with boilerplate stuff and mechanical typing of code. But it's hard to trust stuff generated by AI tools in context of industrial automation, for example.”

Lastly the developers were asked if any features or improvements they would’ve liked to see in AI tools. For the first developer, improved ideological neutrality for the training data would be appreciated. The second developer did not respond to this question.

4.4 Implications of the results

In this section, the answers given in the interviews will be analysed for the implications of integrating AI tools into development workflows at their current state. The professionals were interviewed to provide diverse, first-hand context on the current challenges and benefits of integrating AI tools from a junior and a senior developers perspective. AI-driven development differs from more common development workflows greatly with some positive impacts and some negative impacts, which can hopefully be improved or ironed out in the future.

The answers imply a strong need for generative AI to be able to handle domain-specific areas better. Industrial automation development was the highlighted field in the answers. In industrial automation specifically, each customer needs a bespoke solutions tailor made for their processes. AI seems to currently have difficulties with generating more complex and unique solutions. For both the developers, a common concern over the reliability of the generated content is poor. Although AI autocomplete tools like GitHub Copilot can generate large chunks of code, it came up for both developers that the generated content often needs

to be rewritten and edited so much that it might be easier to just code by hand. Apart from unit testing and basic template code commonly copied from elsewhere, and readily available in AI training materials.

The current impact of AI tools on software development processes, as noted by the second developer, suggests a gap between the capabilities of current AI tools and the needs within existing workflows, particularly in industries with strict and precise requirements. This is a potential area for improvement in how AI tools interact with other development tools and systems. Due to this capability gap, developers showed caution and scepticism on fully integrating and trusting AI into standard development workflows at the current state. The main issue with generating code for non-boilerplate uses stems from the unreliability of the generated code, often needing heavy editing, or rewriting entirely to function optimally.

The mixed levels of enthusiasm and the recognition of AI's effectiveness in handling more repetitive activities could indicate a need for specialized AI usage training in the future. Additional training might help developers learn how to maximize the benefits of AI tools by focusing on high reliability uses, such as boilerplate generation and automated testing, as shown in the interview answers, while reducing and working around the technologies' limitations. Some positivity can be found from the answers as well, indicating potential for productivity improvements in the future by limiting time spent on tedious tasks, like simpler template coding and unit tests with AI automation. Leaving the developers to focus on the more important aspects of software development, such as developing the specialized and unique areas of the software. This may help remove some of the monotonous aspects of development work, improving work satisfaction and efficiency. A generative AI model could also be custom trained to a specific company on how it wants its employees to operate. This could be used as a teaching tool for younger developers learning the processes of professional development work, alleviating stress from the senior developers. Even our more experienced interviewed developer used generative AI for assistance in learning and solving personally novel problems.

5 Conclusions

The writing of this thesis was started with the vision to shed light on the current landscape of using Generative AI tools to help software developers automate some parts their work, with the objective of examining the viability of implementing more AI into software development. Generative AI has rocketed in popularity among software developers. In its current form it shows high effectiveness in certain areas of software development, most notably in automating repetitive tasks and generating simpler parts of code, like unit tests and boilerplate- or template code. The adoption of AI tools is steadily increasing as more industries recognize their potential to enhance efficiency and accuracy.

Modern AI technologies inhibit great technical capabilities, which have widened the scope of applications in development, such as GPT-4 and Copilot, which can be now used for code generation, modification and error correction to a relatively good success rate in less complex tasks. Despite their capabilities, the integration of AI tools into existing development environments poses significant challenges, primarily concerning code reliability and usability in more complex situations.

Like shown in the professional interviews part of this paper, some valid scepticism remains for developers working in the professional development field. These two developers interviewed have very different backgrounds and levels of experiences with the professional development field, but both seemed to agree on AI tools not being ready yet to be fully trusted in developing code sold commercially, with issues specifically when working in unique customer specific codebases and the reliability of generated code, which often will need to be completely rewritten to fit the project, mitigating some of the benefits of using AI tools. The new GPT-4-Turbo model seems to address some of these issues, with improvements being made to code reliability and support for creating more specific and novel code functionalities.

The topic of using generative AI for code generation, optimization and debugging could be further researched with long-term impacts of AI on software development, particularly in terms of job market demand shifts, ethical considerations, and the evolution of software design and maintenance practices.

References

Backus, J., 1978. The history of FORTRAN I, II, and III. ACM SIGPLAN Not. 13, 165–180. https://doi.org/10.1145/960118.808380

Bugden, W., Alahmar, A., 2022. Rust: The Programming Language for Safety and Performance V1, arXiv preprint arXiv:2206.05503, pp. 1, 3-6. https://doi.org/10.48550/arXiv.2206.05503

Bull, C., Kharrufa, A., 2023. Generative AI Assistants in Software Development Education: A vision for integrating Generative AI into educational practice, not instinctively defending against it. IEEE Softw. 1–9. https://doi.org/10.1109/MS.2023.3300574

Business Apps | Microsoft Power Apps [WWW Document], n.d. URL https://powerapps.microsoft.com/en-gb/ (accessed 11.1.23).

Chow, A.R., 2023. How ChatGPT Managed to Grow Faster Than TikTok or Instagram [WWW Document]. Time. URL https://time.com/6253615/chatgpt-fastest-growing/ (accessed 10.9.23).

Ebert, C., Gallardo, G., Hernantes, J., Serrano, N., 2016. DevOps. IEEE Softw. 33, 94–100. https://doi.org/10.1109/MS.2016.68

Edwards, B., 2023. ChatGPT sets record for fastest-growing user base in history, report says [WWW Document]. Ars Tech. URL https://arstechnica.com/information-technology/2023/02/chatgpt-sets-record-for-fastest-growing-user-base-in-history-report-says/ (accessed 10.9.23).

Figure 1. Modern LLMs timeline diagram [WWW Document], 2024. . ResearchGate. URL https://www.researchgate.net/figure/Modern-LLMs-timeline-diagram_fig1_378530762 (accessed 5.18.24).

Figure 1. Software development methodologies timeline. [WWW Document], n.d. . ResearchGate. URL https://www.researchgate.net/figure/Software-development-methodologies-timeline_fig1_362915914 (accessed 4.25.24).

Ghoreishi, M., Happonen, A., 2020. New promises AI brings into circular economy accelerated product design: a review on supporting literature. E3S Web Conf. 158, 06002. https://doi.org/10.1051/e3sconf/202015806002

Git - A Short History of Git [WWW Document], n.d. URL https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git (accessed 10.27.23).

GitHub Copilot · Your AI pair programmer [WWW Document], 2024. . GitHub. URL https://github.com/features/copilot (accessed 1.25.24).

GPT-4 Turbo in the OpenAI API | OpenAI Help Center [WWW Document], 2024. URL https://help.openai.com/en/articles/8555510-gpt-4-turbo-in-the-openai-api (accessed 5.5.24).

Heaton, C., Park, A., Zedlewski, C., n.d. Integrated Development Environments (IDEs). Spring 2005.

Hello GPT-4o [WWW Document], 2024. URL https://openai.com/index/hello-gpt-4o/ (accessed 6.9.24).

Infrastructure as Code | IBM [WWW Document], n.d. URL https://www.ibm.com/topics/infrastructure-as-code (accessed 10.27.23).

Intersimone, D., 2010. The evolution to integrated development environments (IDE) [WWW Document]. Computerworld. URL https://www.computerworld.com/article/2468478/the-evolution-to-integrated-development-environments--ide-.html (accessed 10.26.23).

Introducing ChatGPT [WWW Document], n.d. URL https://openai.com/blog/chatgpt (accessed 10.4.23).

Kortelainen, H., Happonen, A., Hanski, J., 2019. From asset provider to knowledge company—Transformation in the digital era, in: Mathew, J., Lim, C.W., Ma, L., Sands, D., Cholette, M.E., Borghesani, P. (Eds.), Asset Intelligence through Integration and Interoperability and Contemporary Vibration Engineering Technologies, Lecture Notes in Mechanical Engineering. Springer, pp. 333–341. https://doi.org/10.1007/978-3-319-95711-1_33

Merritt, R., 2022. What Is a Transformer Model? [WWW Document]. NVIDIA Blog. URL https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/ (accessed 10.24.23).

Meta-AI, 2024. Llama 2 [WWW Document]. Meta AI. URL https://llama.meta.com/llama2/ (accessed 1.25.24).

meta-llama, 2024. llama. GitHub repository. Available at: https://github.com/meta-llama/llama.

Microsoft Makes Powerful GPT-4 Turbo Free for All Copilot Users [WWW Document], 2024. . GeeksforGeeks. URL https://www.geeksforgeeks.org/microsoft-makes-powerful-gpt-4-turbo-free-for-all-copilot-users/ (accessed 5.5.24).

OpenAI Codex [WWW Document], n.d. URL https://openai.com/blog/openai-codex (accessed 1.25.24).

OpenAI Platform [WWW Document], 2023. URL https://platform.openai.com (accessed 10.24.23).

Pichai, S., 2023. An important next step on our AI journey [WWW Document]. Google. URL https://blog.google/technology/ai/bard-google-ai-search-updates/ (accessed 1.26.24).

Proven, L., 2022. Linux 6.1: Rust to hit mainline kernel [WWW Document]. URL https://www.theregister.com/2022/10/05/rust_kernel_pull_request_pulled/ (accessed 11.1.23).

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., 2018. Improving Language Understanding by Generative Pre-Training.

Rossel, S., 2017. Continuous Integration, Delivery, and Deployment: Getting Started with the Processes and the Tools to Continuously Deliver High-Quality Software. Packt Publishing, Limited, Birmingham, UNITED KINGDOM.

Ruparelia, N.B., 2010. The history of version control. ACM SIGSOFT Softw. Eng. Notes 35, 5–9. https://doi.org/10.1145/1668862.1668876

Sakib, F.A., Khan, S.H., Karim, A.H.M.R., 2023. Extending the Frontier of ChatGPT: Code Generation and Debugging V1, arXiv preprint arXiv:2307.08260, pp. 1-7. https://doi.org/10.48550/arXiv.2307.08260

Sarrion, E., 2023. Exploring the Power of ChatGPT: Applications, Techniques, and Implications. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-9529-8

Schreiner, M., 2023. GPT-4 architecture, datasets, costs and more leaked [WWW Document]. THE DECODER. URL https://the-decoder.com/gpt-4-architecture-datasets-costs-and-more-leaked/ (accessed 4.13.24).

Singh, S.K., Kumar, S., Mehra, P.S., 2023. Chat GPT & Google Bard AI: A Review," 2023 International Conference on IoT, Communication and Automation Technology (ICICAT), Gorakhpur, India, 2023, pp. 3-6, doi: 10.1109/ICICAT57735.2023.10263706.

Tamasiga, P., Ouassou, E. houssin, Onyeaka, H., Bakwena, M., Happonen, A., Molala, M., 2023. Forecasting disruptions in global food value chains to tackle food insecurity: The role of AI and big data analytics – A bibliometric and scientometric analysis. J. Agric. Food Res. 14, 100819. https://doi.org/10.1016/j.jafr.2023.100819

Tasnim, R., 2023. The History of Software Engineering: A Timeline. Softw. Test. Stuff. URL https://www.softwaretestingstuff.com/2007/09/brief-history-of-software-engineering.html (accessed 10.26.23).

Thoppilan, R., et al, 2022. LaMDA: Language Models for Dialog Applications V3, arXiv preprint arXiv:2201.08239, pp. 1-3. https://doi.org/10.48550/arXiv.2201.08239

UCA, U. of C.A., 2023. Chat GPT: What is it? [WWW Document]. Httpsucaeducetalchat-Gpt. URL https://uca.edu/cetal/chat-gpt/ (accessed 10.24.23).

University of Cape Town, C.S.D., 2011. The software crisis [WWW Document]. URL https://www.cs.uct.ac.za/mit_notes/software/htmls/ch02s02.html (accessed 10.26.23).

Unleash developer productivity with generative AI | McKinsey [WWW Document], 2023. URL https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/unleashing-developer-productivity-with-generative-ai (accessed 4.25.24).

Usmani, U.A., Happonen, A., Watada, J., 2023a. Enhancing Artificial Intelligence Control Mechanisms: Current Practices, Real Life Applications and Future Views, in: Arai, K. (Ed.), Proceedings of the Future Technologies Conference (FTC) 2022, Volume 1, Lecture Notes in Networks and Systems. Springer International Publishing, Cham, pp. 287–306. https://doi.org/10.1007/978-3-031-18461-1_19

Usmani, U.A., Happonen, A., Watada, J., 2023b. Secure Integration of IoT-Enabled Sensors and Technologies: Engineering Applications for Humanitarian Impact, in: 2023 5th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA). Presented at the 2023 5th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), pp. 1–10. https://doi.org/10.1109/HORA58378.2023.10156740

Usmani, U.A., Happonen, A., Watada, J., 2023c. Human-Centered Artificial Intelligence: Designing for User Empowerment and Ethical Considerations, in: 2023 5th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA). Presented at the 2023 5th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), pp. 1–7. https://doi.org/10.1109/HORA58378.2023.10156761

What are Large Language Models? - LLM AI Explained - AWS [WWW Document], n.d. . Amaz. Web Serv. Inc. URL https://aws.amazon.com/what-is/large-language-model/ (accessed 10.24.23).

What does infrastructure as code mean? [WWW Document], n.d. URL https://about.gitlab.com/topics/gitops/infrastructure-as-code/ (accessed 10.27.23).

What is CI/CD? [WWW Document], n.d. URL https://about.gitlab.com/topics/ci-cd/ (accessed 10.27.23).

What is DevOps? [WWW Document], n.d. URL https://about.gitlab.com/topics/devops/ (accessed 10.27.23).

What is GPT AI? - Generative Pre-Trained Transformers Explained - AWS [WWW Document], n.d. . Amaz. Web Serv. Inc. URL https://aws.amazon.com/what-is/gpt/ (accessed 10.24.23).

What is IDE or Integrated Development Environments? [WWW Document], n.d. . Veracode. URL https://www.veracode.com/security/integrated-development-environment (accessed 10.26.23).

What is Natural Language Processing? | IBM [WWW Document], 2023. URL https://www.ibm.com/topics/natural-language-processing (accessed 10.24.23).

Wirth, N., 2008. A Brief History of Software Engineering. IEEE Ann. Hist. Comput. 30, 32–39. https://doi.org/10.1109/MAHC.2008.33

Woo, M., 2020. The Rise of No/Low Code Software Development—No Experience Needed? Eng. Beijing China 6, 960–961. https://doi.org/10.1016/j.eng.2020.07.007

Yuan, Z., Lou, Y., Liu, M., Ding, S., Wang, K., Chen, Y., Peng, X., 2023. No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation V2, arXiv preprint arXiv:2305.04207, pp. 1-8. https://doi.org/10.48550/arXiv.2305.04207