AI in Coding: How can code generation models support developing computational thinking skills?

The use of code generation models in programming support activities

Rick Mulder[1]

Supervisor(s): Fenia Aivaloglou[1]**, Xiaoling Zhang**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 24, 2023

Name of the student: Rick Mulder
Final project course: CSE3000 Research Project
Thesis committee: Fenia Aivaloglou, Xiaoling Zhang, Tom Viering

Abstract

Using AI to support programming has recently gained a lot of popularity. Researchers have been developing tools to support programming activities using GPT models such as ChatGPT and Codex In this paper, we present the most common programming activities that these models can support. We show that they have a varying range of success across solving Code Generation, Code Explanation,and Data Visualisation problems, but are often able to solve around 50% of problems on the first try. Multiple tries can raise these averages to 75%. Additionally, specialized tools using GPT models have seen promising results regarding Data Visualisation, Software Vulnerability Detection, and General Programming Support. This shows a promising trend, and can mean we will all be pair programming with AI in the near-future.

1 Introduction

Programming is an essential part of modern society. Its applications are widespread, and range from cars [5] to medical equipment [28] and to simple things like a thermostat [33]. However, programming is a manual process and subject to tedious repetition and human errors. With new rapid developments in AI, and in particular theGenerative Pretrained Transformer (GPT)[2], this might change. These new advancements have made it possible to automatically generate working code. Being able to assist programmers with activities such as documentation generation, automatic bug fixes, and code generation may allow programmers to focus on the higher level aspects. This study will enable identifying the gaps in current research, and opportunities for future work.

In light of these rapid developments of Large Language Models (LLM), several papers are published daily [18]. Two examples of such models are ChatGPT [1] and Codex [2]. Published by OpenAI, ChatGPT is a prototype chat bot, able to accept Natural Language (NL) prompts, and respond with various forms of output. Codex, another GPT also published by OpenAI, is specially trained on code, and is more suited for programming tasks. Research shows that these models are able to assist and come up with solutions to many types of questions [7], and synthesise information in a concise way. Models like ChatGPT and Codex have already been subjected to numerous tests and experiments [29; 19]. The papers published in the computer science field range from reporting on performance for introductory programming courses [9] to the possibility of automating bioinformatics tasks [21], With this daily influx of papers, there is an advantage to having a clear overview of the current capabilities and state-of-the-art in code generation models.

This brings up the question how well these models are able to support various programming activities.

In order to compose an overview for the capabilities of LLM, we have determined the following research questions:

- RQ1: For what kind of programming support activities have the code generation models been used?
- RQ2: How successful have they been considering these activities?

Being able to be up to date on the most recent developments shorten the time it takes for new people to enter the field, and allows peer researchers to quickly re-inform themselves of the newest developments. This is essential in a quick paced field like Artificial Intelligence. This systematic literature review aims to report on these uses and provide an overview of their current performance.

2 Methodology

We conducted a systematic literature review to investigate the latest developments on code generation models for supporting programming activities. The reason this method was chosen is due to the large amount of ongoing research. With the already extensive amount of research published, our research question can be answered by looking at the published uses of models. This way readers can quickly see the current state and progress of code generation models for programming support activities. We followed these general steps[13]: 1) Identifying criteria, 2) Screening papers, 3) Checking eligibility, 4) Extracting data, 5) Analysing data and reporting the results

2.1 Inclusion and Exclusion Criteria

In order to find research pertaining to the research question, we established inclusion and exclusion criteria. Inclusion criteria were as follows :

- The paper compares capabilities of several GPT models for programming activities
- The paper has developed a tool that helps with programming activities
- The paper has tested capabilities of a GPT for a certain programming activity
- The paper is found within the search terms as explained in 2.2.

Exclusion criteria were as follows:

- The paper is not in English
- The paper is more than 1,5 years old
- The paper is inaccessible

2.2 Search

After the inclusion and exclusions criteria, we created search terms that pertained to the use of code generation models in programming activities. The search terms were constructed in a way to include models such as ChatGPT, and to cover possible different wordings regarding support for programming. The searches were conducted between 1-05-2023 and 26-05-2023, on Google Scholar and Scopus. As stated in the inclusion criteria, papers were only required to be published. There was no requirement for the publication to be in a specific paper or format.

[1] https://openai.com/chatgpt

[2] https://openai.com/blog/openai-codex

We used two queries on Google scholar due to the large number of unrelated papers that it showed. For the query where we got a large number of results, we only looked at the first page.

Queries for Google Scholar:

- ”ChatGPT programming” OR ”GPT programming” OR ’”large language models” programming’ OR ”AI model programming” OR ”OpenAI programming” OR ”GPT code generation”
- allintitle: programming ChatGPT OR GPT OR ”large language models” OR ”AI model” OR OpenAI OR ”GPT code generation”

Query for Scopus:

- TITLE-ABS-KEY ((chatgpt OR gpt OR ”large language models” OR ”AI model” OR openai) AND (programming OR ”code generation”)) AND (LIMIT-TO (OA , ”all”)) AND (LIMIT-TO (PUBYEAR , 2023) OR LIMIT-TO (PUBYEAR , 2022)) AND (LIMIT-TO (SUBJAREA , ”COMP”))

In Scopus, the first query was done with only the TITLE-ABS-KEY part of the search terms. When filters were applied for publication years and subject area, Scopus presented the search query above, which was subsequently used to as the final query for the search in Scopus.

2.3 Data Analysis

After excluding papers that did not fit the criteria, we then read the abstract and introduction of each paper, and judged whether it was relevant to the research question. This yielded a result set of papers. Among this result set, we examined the references for papers references by multiple other papers. When such a paper was found and it pertained to our research question, it was including in the set of papers even if it did not show up in the search results. Then we checked which papers referenced this paper, and filtered those according to our exclusion criteria, once again adding the remainder to the result set. This is also known as backward snowballing.

2.4 Data Extraction

In this result set we looked at the results of each paper in terms of what programming activities their model or implementation supported. When it was deemed to indeed provide programming support, we looked at their reported results in order to answer the second research question for this tool. We synthesised other papers reporting results for the same programming activities, and were able to assess the state of support for that programming activity. We looked at the most commonly occurring activities, and used those as base groups, and studies that were relevant but did not fit in one of those categories were put in General Programming Support. This provided us with an overview of published and used tools that support programming activities and how successful they are.

3 Results

In Figure 1 the results of the review process are displayed in a PRISMA flow diagram.

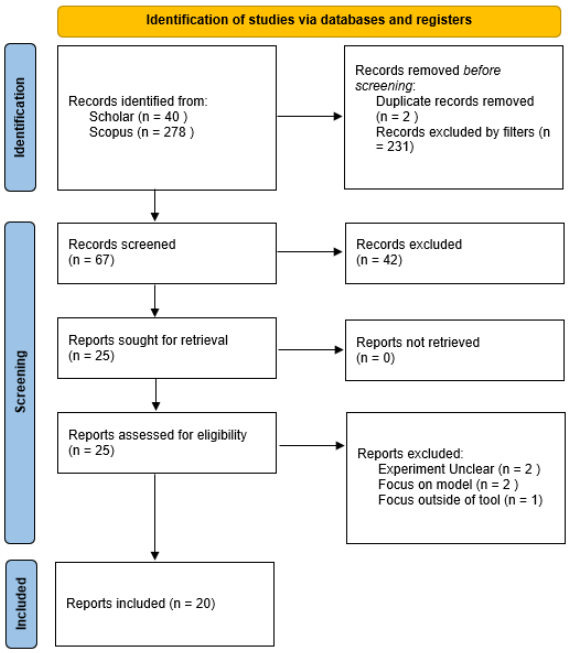

Figure 1: PRISMA Flowchart

There were five studies that initially seemed to meet the inclusion criteria, but were ultimately excluded from the review. They are listed here along with the reasons for exclusion

- [3] While it seemed that this paper was about the capabilities of ChatGPT for supporting programming activities, the method and experiment section was unclear and felt unrepeatable.
- [24] Upon initial screening, this paper was about solving programming bugs using ChatGPT, but upon further inspection it has no experiments and simply speculates on the use of ChatGPT.
- [31] Focused on pre-training a model, but does not talk about application as support for programming activities.
- [4] Focused on the way Chinese students feel about programming courses and how AI can change this rather than supporting programming activities.
- [25] Focused on the training aspect of the model, and not on the code completion and the performance in real tests with users.

We have found several works where code generation models have been published and used to support programming activities. We will discuss how the code generation model support programming activities, and report on the success of these models. We have grouped together similar programming activities, independent of the model used for it. Support for the following programming activities and their success will be presented: Code Generation, Code Documentation, Code Explanation, Data Visualisation, Software Vulnerability Detection and General Programming Support. This last category is meant as a general collection, where the studies discuss activities in an intertwined way, instead of focusing on a single type of activity.

Programming Activity	Title	Author	Year
Code Generation & Code Explaination	Automatic generation of programming exercises and code explanations using large language models.	Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen	2022
Code Generation	Can openai's codex fix bugs?: An evaluation on quixbugs.	Julian Aron Prenner, Hlib Babii, and Romain Robbes	2022
	The robots are coming: Exploring the implications of openai codex on introductory programming	James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather.	2022
	A case study on scaffolding exploratory data analysis for AI pair programmers.	Haoquan Zhou and Jingbo Li.	2023
	How readable is model-generated code? examining readability and visual inspection of github copilot.	Naser Al Madi.	2022
	Sketch-based approach for automatic code generation	Jia Li, Yongmin Li, Ge Li, Zhi Jin, Yiyang Hao, and Xing Hu	2023
	Promptinfuser: Bringing user interface mock-ups to life with large language models	Savvas Petridis, Michael Terry, and Carrie Jun Cai	2023
Code Documentation	Automatic code documentation generation using gpt-3	Junaed Younus Khan and Gias Uddin	2022
Code Explanation	Gptutor: a chatgpt-powered programming tool for code explanation	Eason Chen, Ray Huang, Han-Shin Chen, Yuen-Hsien Tseng, and Liang-Yi Li	2023
	Using large language models to enhance programming error messages	Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker	2023
Data Visualisation	Chat2VIS: Generating data visualizations via natural language using chatgpt, codex and gpt-3 large language models.	Paula Maddigan and Teo Susnjak	2023
	CodexDB: Generating code for processing sql queries using gpt-3 codex	Immanuel Trummer	2022
Software Vulnerability Detection	Transformer-based language models for software vulnerability detection.	Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal	2022
General Programming Support	Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language	Paul Denny, Viraj Kumar, and Nasser Giacaman	2023
	Using github copilot to solve simple programming problems.	Michel Wermelinger	2023
	Cracking the code: Co-coding with ai in creative programming education.	Martin Jonsson and Jakob Tholander	2022
	Is chatgpt the ultimate programming assistant – how far is it?	Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bissyandè	2023
	Stylette: Styling the web with natural language.	Tae Soo Kim, DaEun Choi, Yoonseo Choi, and Juho Kim.	2022
	The programmer's assistant: Conversational interaction with a large language model for software development.	Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz.	2023
	Codefill: Multi-token code completion by jointly learning from structure and naming sequences.	Maliheh Izadi, Roberta Gismondi, and Georgios Gousios	2022

Table 1: Works belonging to each category

3.1 Code Generation

There have been several studies [15; 22; 8] published regarding the use of GPT models for code generation. Codex, a GPT trained mainly on Python code has been tested qualitatively and quantitatively. Findings indicate that generated code is often sensible novel and readily applicable [15], and often come with generated test cases. Codex is also capable of fixing generated code [22] and generating programming

exercises [8]. Its shortcomings lie mostly in formatting errors and missing corner cases. Overall, the code generational models often outperform most human peers [8], and could prove problematic as an answer generating tool in the hands of students.

Other studies [32; 1] have looked into GitHub Copilot, an interface between Codex and several IDE's such as Visual Code and JetBrains. Prompt engineering has a significant impact on the success of queries [32]. Word order, terminology and repeating important information all influence the quality of the output. CoPilot is generating code that is very close to human code in terms of readability [1]. This helps users understand code more easily, and can increase confidence in the generated code. On the other hand, both studies report that this confidence can sometimes be dangerous, mentioning users were found to perform less checks and inspections on generated output and sometimes demonstrated an over-reliance on the AI.

Lastly, standalone tools [17; 20] have been developed to elavate the power of LLM in a slightly different manner. Sketch Based Code Generation [17] makes use of a large database of code, and retrieves and manipulates snippets of code to answer code question prompts. This method has been found to be significantly more successful than other LLM approaches. PromptInfuser [20] is a tool created for supporting UI-design. It that allows user to connect input and output fields, with a functional prompt as function, and has shown a speed-up of the process and an increased understanding of requirements.

3.2 Code Documentation

Documentation is an important aspect of programming, allowing other programmers to understand code at a glance. [12] shows that Codex can assist the user in generating documentation. Using zero-shot learning, Codex was mostly unable to provide accurate documentation. It performs best when using few-shot learning, even among its peer models. Sometimes it was capable of adding extra correct information based on the code that the original documentation did not have. Their research shows that users were skeptical of generated documentation in the beginning, but after interacting with the tool had a much better opinion about it. The results show that the tool was able to assist the user by generating correct responses in a majority of cases.

3.3 Code Explanation

An important support activity in programming is code explanation. In [6] the authors have created a visual studio extension utilizing ChatGPT (gpt-3.5-turbo) in order to explain the meaning of code to programmers. They tackle three found problems that current GPT have, namely that GPT are superficial, offer irrelevant information, and may often not be up to date. Results conclude that their extension is an improvement on existing GPT. Research shows that Codex is able to explain code in 67% of the cases [15]. Codex was also tested on its ability to generate accurate readable error messages [16]. In almost half of the different error message cases, Codex was able to provide a correct explanation, while improving on the original error message. In around 33% of the cases, Codex even suggested a correct fix for the error message. This could improve programmers understanding of why errors occur, and significantly lower the debugging time required to fix the error. This shows that GPT models seem to be quite proficient at generating good explanations

3.4 Data Visualisation

GPT models have enabled the conversion from Natural language into query processing code such as SQL Queries. Several tools have been published for this purpose. They work by inputting natural language [19] or a combination of SQL query and natural language [29] Their performance is best when allowed multiple tries, reaching around 80% success rate, and in many cases is robust against under-specification and miss-specification. The latter part is especially important for the adaption in every day situations. It is important to note that while 80% is high, in real life application it might not be practical to have to perform many attempts. In the time it takes do many attempts to engineer the prompt, the programmer could also be manually specifying/fixing the SQL query, possibly leading to a faster result.

3.5 Software Vulnerability Detection

GPT models have also been used to test software for vulnerabilities [26]. Software vulnerabilities testing is often a costly and long process, and being able to automate this would be very helpful. The study found that current LLM have very good performance on detecting vulnerabilities, and report vulnerability detection on most types of vulnerabilities as high as 93% F1-score, on some data sets. This means that it has both high precision (this is the percentage of results predicted as positive that are indeed positive), and high recall (this is the percentage of total positives that were found). However, some types of vulnerabilities, such as API function calls, have a lower F1-score, being around 78% This may mean that much of the process could be automated, and decrease the chance for software vulnerabilities to be present.

3.6 General Programming Support

As a general programming support assistant, several tools have been studied [7; 30; 11; 27; 14; 23; 10]. CoPilot shows capability in solving a number problems out of the box [7]. Results also show that it is very sensitive to input, and small alterations to input prompts may severely impact the quality of the output. Similarly, this leads to answers often being wrong, unnecessary or not related to the question [30]. Experiments for Co-coding with AI [11] show that while are capable of solving problems in a large number of situations [27], they are currently impractical to utilize in a realistic situation. These tools also enable non-programmers to engage in programming activities [14], and show an increase in confidence about programming for users. Integrated assistants were preferred to be more proactive [23], and multi-token prediction [10] could provide a solution to this.

4 Discussion

We can see that a number of LLM/GPT uses have been published already, and that the results seem optimistic for this

early stage. However there are a number of risks we have to keep in mind. First off, since there are situations where the the models performed exceptionally well, there exists the risk over over-reliance on these models. This can happen because users get accustomed to correct answers for small problems, and results in users no longer trying to verify the provided solutions for more complicated cases. Secondly, it can be the case that a user is not proficient enough in a certain activity, and tries to solve the problem using code generation models. This obtained solution might be working, but might not provide the intended answer, and could cause a slew of unintended problems. Lastly, time spent trying to obtain an answer can quickly get longer than writing the code manually. When having to do a lot of different prompts and attempts with high temperature to obtain an answer that works, sometimes it would be better to simply make the program yourself, especially for smaller programs. Therefore it is important to stay attentive to the responses we obtain, make sure we have someone proficient checking it, and to keep human oversight and intervention. The importance of being aware of the risk of over-reliance is also reflected in several of the works reviewed in this paper. The current iteration of code generation models can be used mostly as a support tool, but still very much requires its user to check and correct the outputs.

4.1 Limitations

The Author conducted the systematic literature review in eight weeks. This limits the scope of the review. The search and coding of papers was done by only one Author. Only two databases were used while searching for papers. In addition because the a lot of information about the subject is currently being discovered, the review was limited to papers no older than 1,5 years.

4.2 Responsible Research

The nature of this paper is a systematic literature review. The ethical consideration of this is that the Author could conclude what is wanted. Therefore the Author has to be careful to operate objectively without any bias for outcomes.

4.3 Threats to validity

The breakthroughs in GPT and LLM are relatively new. This means that not everything is known yet, and a lot of initial conclusions could be missing key aspects that are discovered later. In addition, a lot of new research is coming out on a nearly daily basis. Everything in the literature review is therefore only based on a limited time frame of search.

This literature review was conducted in the time span of 8 weeks. This means that the process of gathering information had to be done in a few weeks. While striving to be thorough, the possibility of oversight or missing important information is present due to these time constraints. The consequences of these factors can arise in the form of, but not limited to, missing papers in other databases outside of the search scope, missing papers due to a narrow scope to fit the time frame, and missing papers due to the inability to process a large amount of papers in this time frame.

Additionally, only papers with a direct link to the subject, either in the title or the abstract were chosen. As a result, it is very much a possibility that papers with relevant information outside of these sections are available, but these were not included in the literature review. However, this threat to the validity of the literature is neutralized by the search queries that indicate it should contain these terms in the title.

Lastly, the Author's expertise regarding this topic is still in its early stages. This could have the following consequences. Firstly it might influence or impact decisions to include papers. Secondly, finding search terms that equate to the same meaning while conducting the search, which could have left papers which may have been relevant off the radar.

Despite these possible issues, it is firmly believed that this paper will answer the research questions to the best of its ability, and contribute to the understanding of the code generation models for supporting programming activities and the success in their current state.

5 Conclusions

Several published uses for GPT models have been found, including Code Generation, Code Documentation, Data Visualisation, and Software Vulnerability Detection. While there are promising results, most of these activities cannot yet be fully supported in a realistic environment without human oversight. With the amount of attempts required to obtain a correct result, it is often faster to manually solve the problem. Several tools have created an interface to improve the reliability of GPT models through prompt engineering, and this has shown promising improvements on the previous models. With the current trend, it will not take long before tools will be capable of providing reliable support for programming activities through the use of LLM.

With global access to LLM, it is important to further research its potential and dangers. Further work is needed into prompt engineering, as well as the best ways to interface with these models, and especially things we should avoided automating. At the current pace, it is not a matter of if we will be using AI, but when.

References

[1] Naser Al Madi. How readable is model-generated code? examining readability and visual inspection of github copilot. Association for Computing Machinery, 2022.

[2] Jawid Ahmad Baktash and Mursal Dawodi. Gpt-4: A review on advancements and opportunities in natural language processing. https://doi.org/10.48550/arXiv.2305.03195, 2023.

[3] Som Biswas. Role of chatgpt in computer programming.: Chatgpt in computer programming. *Mesopotamian Journal of Computer Science*, 2023:8–16, Feb. 2023.

[4] Chen Cao. Scaffolding cs1 courses with a large language model-powered intelligent tutoring system. page 229 – 232. Association for Computing Machinery, 2023.

[5] Robert N Charette. This car runs on code. *IEEE spectrum*, 46(3):3, 2009.

[6] Eason Chen, Ray Huang, Han-Shin Chen, Yuen-Hsien Tseng, and Liang-Yi Li. Gptutor: a chatgpt-powered programming tool for code explanation. https://doi.org/10.48550/arXiv.2305.01863, 2023.

[7] Paul Denny, Viraj Kumar, and Nasser Giacaman. Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language. volume 1, page 1136 – 1142. Association for Computing Machinery, Inc, 2023.

[8] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. The robots are coming: Exploring the implications of openai codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference*, ACE '22, page 10–19, New York, NY, USA, 2022. Association for Computing Machinery.

[9] Chuqin Geng, Yihan Zhang, Brigitte Pientka, and Xujie Si. Can chatgpt pass an introductory level functional language programming course? https://doi.org/10.48550/arXiv.2305.02230, 2023.

[10] Maliheh Izadi, Roberta Gismondi, and Georgios Gousios. Codefill: Multi-token code completion by jointly learning from structure and naming sequences. volume 2022-May, page 401 – 412. IEEE Computer Society, 2022.

[11] Martin Jonsson and Jakob Tholander. Cracking the code: Co-coding with ai in creative programming education. page 5 – 14. Association for Computing Machinery, 2022.

[12] Junaed Younus Khan and Gias Uddin. Automatic code documentation generation using gpt-3. Association for Computing Machinery, 2022.

[13] Khalid Khan, Regina Kunz, Joseph Kleijnen, and Gerd Antes. Five steps to conducting a systematic review. *Journal of the Royal Society of Medicine*, 96:118–121, 03 2003.

[14] Tae Soo Kim, DaEun Choi, Yoonseo Choi, and Juho Kim. Stylette: Styling the web with natural language. Association for Computing Machinery, 2022.

[15] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. Using large language models to enhance programming error messages. volume 1, page 563 – 569. Association for Computing Machinery, Inc, 2023.

[16] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. Using large language models to enhance programming error messages. volume 1, page 563 – 569. Association for Computing Machinery, Inc, 2023.

[17] Jia Li, Yongmin Li, Ge Li, Zhi Jin, Yiyang Hao, and Xing Hu. Skcoder: A sketch-based approach for automatic code generation. https://doi.org/10.48550/arXiv.2302.06144, 2023.

[18] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Dajiang Zhu, Xiang Li, Ning Qiang, Dingang Shen, Tianming Liu, and Bao Ge. Summary of chatgpt/gpt-4 research and perspective towards the future of large language models. https://doi.org/10.48550/arXiv.2304.01852, 2023.

[19] Paula Maddigan and Teo Susnjak. Chat2vis: Generating data visualizations via natural language using chatgpt, codex and gpt-3 large language models. *IEEE Access*, 11:45181–45193, 2023.

[20] Savvas Petridis, Michael Terry, and Carrie Jun Cai. Promptinfuser: Bringing user interface mock-ups to life with large language models. Association for Computing Machinery, 2023.

[21] Stephen R. Piccolo, Paul Denny, Andrew Luxton-Reilly, Samuel Payne, and Perry G. Ridge. Many bioinformatics programming tasks can be automated with chatgpt. https://doi.org/10.48550/arXiv.2303.13528, 2023.

[22] Julian Aron Prenner, Hlib Babii, and Romain Robbes. Can openai's codex fix bugs?: An evaluation on quixbugs. page 69 – 75. Institute of Electrical and Electronics Engineers Inc., 2022.

[23] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. The programmer's assistant: Conversational interaction with a large language model for software development. page 491 – 514. Association for Computing Machinery, 2023.

[24] Nigar M. Shafiq Surameery and Mohammed Y. Shakor. Use chat gpt to solve programming bugs. *International Journal of Information Technology & Computer Engineering (IJITC) ISSN : 2455-5290*, 3(01):17–22, Jan. 2023.

[25] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. Intellicode compose: Code generation using transformer. https://doi.org/10.48550/arXiv.2005.08025, 2020.

[26] Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal. Transformer-based language models for software vulnerability detection. page 481 – 496. Association for Computing Machinery, 2022.

[27] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bissyandé. Is chatgpt the ultimate programming assistant – how far is it? https://doi.org/10.48550/arXiv.2304.11938, 2023.

[28] J-Donald Tournier, Robert Smith, David Raffelt, Rami Tabbara, Thijs Dhollander, Maximilian Pietsch, Daan Christiaens, Ben Jeurissen, Chun-Hung Yeh, and Alan Connelly. Mrtrix3: A fast, flexible and open software framework for medical image processing and visualisation. *NeuroImage*, 202:116137, 2019.

[29] Immanuel Trummer. Codexdb: Generating code for processing sql queries using gpt-3 codex. https://arxiv.org/abs/2204.08941, 2022.

[30] Michel Wermelinger. Using github copilot to solve simple programming problems. volume 1, page 172 – 178. Association for Computing Machinery, Inc, 2023.

[31] Jiyang Zhang, Sheena Panthaplackel, Pengyu Nie, Junyi Jessy Li, and Milos Gligoric. Coditt5: Pretraining for source code and natural language editing. ACM International Conference Proceeding Series, 2022.

[32] Haoquan Zhou and Jingbo Li. A case study on scaffolding exploratory data analysis for ai pair programmers. Association for Computing Machinery, 2023.

[33] Levent Özgür, Vahid Khalilpour Akram, Moharram Challenger, and Orhan Dağdeviren. An iot based smart thermostat. In *2018 5th International Conference on Electrical and Electronic Engineering (ICEEE)*, pages 252–256, 2018.