2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus) | 978-1-6654-0993-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ElConRus54750.2022.9755659

The Potential of Artificial Intelligence as a Method of Software Developer's Productivity Improvement

Ekaterina A. Moroz
Saint Petersburg Mining University
Saint Petersburg, Russia
ekaterina-moroz99@rambler.ru

Vladimir O. Grizkevich
Saint Petersburg Mining University
Saint Petersburg, Russia
vladimirgritz@yandex.ru

Igor M. Novozhilov
Department of Automation and Control Processes
Saint Petersburg Electrotechnical University "LETI"
Saint-Petersburg, Russia
novozhilovim@list.ru

***Abstract*— Artificial Intelligence finds application at all stages of Software Engineering, and uses the Neural Networks, Machine Learning, Natural Language Processing concepts. This paper attempts to review the instance of such approach - neural network programmer's assistant Copilot, based on Codex, the AI system developed by OpenAI. The differences between Codex language model's versions and analogous systems were analyzed. The main problems and gaps of this innovation, such as correct commands formulation, copyrighting, safety issues, inefficient code, good practice examples and restrictions are also considered. Additionally, the opportunities for Copilot's growth, development and possible features' proposed recommendations were suggested.**

Keywords— Automated Software Engineering, Machine Learning, Neural Networks, Natural Language Processing, Artificial Intelligence Techniques, Software Development Productivity,

I. INTRODUCTION

With the development of Artificial Intelligence as a technology, IT-specialists aim to find its application in various professional spheres. For example, AI has already shown results in requirements generation and processing [1], project planning [2], intelligent software design, architecture, development, testing and analysis. In software development domain AI has two main possibilities for growth – as a natural language interpreter (conversion of human language to machine code) and as a tool of software developer's productivity improvement by predicting and autocompleting the written code. This research provides a brief analysis of code-autocompleting AI, discovering the potential of its application, highlighting general trends, open problems and gaps. An attempt was also made to suggest the best technics and areas to apply such innovation.

The importance of increasing software developer's productivity lies in the need of a rapid software products development, which at the same time meets the requirements of functionality, security and reliability. The main idea here is the sooner the task is solved the more revenue the business receives (increasing efficiency allows to reduce the cost\time of code development). In an ever-changing environment IT-specialists are always seeking to optimize the effectiveness of their approaches and to find good cost-benefit tradeoffs.

Building NLP systems, which translate human speech into code is an opportunity to give non-programmers a chance to build non-complex systems without CS degree. Therefore, two aspects need to be considered: increasing amount of bad quality code and replacement of human programmers by such systems. As for the first, this technology can be used by project managers, QA engineers, support specialist and other people with good analytical and technical skills. Besides, it's a good solution for a backend-engineer, who needs to solve a frontend task or write some code in an unfamiliar language. So, the code quality won't necessarily lower. As for the second, for complex systems architecture and development, an IT professional will be always needed, who will recognize and consider all requirements.

The main idea in problem solving by AI is not always to find a solution instance, it is equally important to search for strategies that find solution instances (e.g. genetic algorithm). Let's now look closer at the aspects of AI application in Software Engineering.

II. DEFINITION, EFFECTIVE APPLICATIONS AND OPPORTUNITIES

On June 29, 2021, Microsoft presented innovative AI programmer assistant Copilot, based on the AI system OpenAI Codex, a modified version of GPT-3 (Generative Pre-trained Transformer - a language model designed to create human-like text). Codex is designed to create valid computer code, it was trained on open source code and natural language, so it understands both programming and human languages. In general, Copilot sends the given context (comments, docstrings, functions) to the service, analyzes it, checks for harmful elements in the input\output data and provides a set of solutions, rated by matching metrics and suggests the most compatible variant, then checks if the suggestion was accepted to improve future versions of the AI system (Figure 1).

In most cases Copilot does not copy code fragments, but generates derivative code from received input data. It is also possible to choose between different solutions, as Copilot suggests lots of them, rated by the confidence score (how well does the proposed code matches the context) [3].

While pursuing experiments with Codex's ancestor GPT-3, OpenAI developers noticed that this model can generate simple python code snippets, although it was not specially trained for this and the set of training data didn't contain much code examples. They came to an idea to train a model on publicly available code and this has become a prerequisite for the creation of Codex

GPT-3 now contains 4 sets of models: Base series helps GPT-3 understand and generate natural language, Instruct series operate in the best way with following instructions,

Codex was made to work with program code, including translating natural language to code.

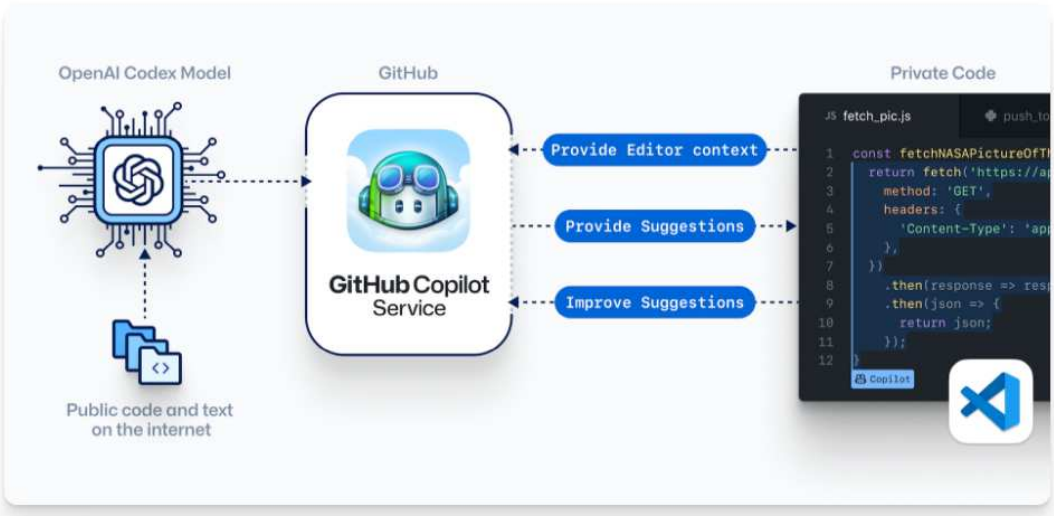

Fig. 1. Copilot's functioning algorithm

The comparison of problem-solving effectivity of different language models' versions can be seen in Table 1.

TABLE 1. CODEX, GPT-NEO, & TABNINE EVALUATIONS

	PASS@k		
	$k = 1$	$k = 10$	$k = 100$
GPT-NEO 125M	0.75%	1.88%	2.97%
GPT-NEO 1.3B	4.79%	7.47%	16.30%
GPT-NEO 2.7B	6.41%	11.27%	21.37%
GPT-J 6B	11.62%	15.74%	27.74%
TABNINE	2.58%	4.35%	7.59%
CODEX-12M	2.00%	3.62%	8.58%
CODEX-25M	3.21%	7.1%	12.89%
CODEX-42M	5.06%	8.8%	15.55%
CODEX-85M	8.22%	12.81%	22.4%
CODEX-300M	13.17%	20.37%	36.27%
CODEX-679M	16.22%	25.7%	40.95%
CODEX-2.5B	21.36%	35.42%	59.5%
CODEX-12B	28.81%	46.81%	72.31%

There is a strong correlation between the number of samples, given to this model during training, and its efficiency. The more samples the model receives, the more effective it operates. A direct dependency can be also found between the amount of model's parameters and its performance.

Let's outline Copilot technology's advantages:

- Codex model understands machine code and human language and can convert them from one to another. It may both create computer commands from natural language and generate natural language description of programming functions.
- GitHub Copilot works with a broad set of frameworks and languages, which means it has a huge scope of possible application. The technical preview does especially well for Python, JavaScript, TypeScript, Ruby, Java, and Go, but it understands dozens of other languages. Copilot shows great results when working with Python. One of the reasons could be that Python is easily readable and can be compared to human language [7].
- During code development it is possible to look up background information about the suggested fragments, and to include credit where credit is due.
- Codex may make it easier to work with new codebases or new languages as for novice programmers, as well as for already skilled software developers
- Copilot already shows high efficiency in its early stages of growth. If 100 samples are given to the Copilot's base model Codex-S, it is able to generate at least one correct function for 77.5% of the problems. [4]
- It proposes a set of solutions, rated by context matching. With every attempt to solve the same task, developer gets a unique evolved set.
- The basic content filtering function is applied in Copilot to make no suggestions, containing harmful words or statements.
- AI code generation technology can be applied in non-engineering domains to solve monotonous tasks.

III. PROBLEMS AND GAPS OF COPILOT APPLICATION

Although Copilot contains many features and shows valuable results, at the same time it has several shortcomings, which are listed as follows:

A. Application of low-quality code

Since Copilot is trained on 179 gigabytes of open source code presented on GitHub, it can autocomplete not only the best solutions, but also the code of inexperienced developers, code with poor style, simply brute force or inefficient algorithms. It generates the best match for surrounding code rather than the most effective solutions (by time, memory

usage, security or performance). GitHub Copilot doesn't actually test the code it suggests, so the code may not even compile or run.

As a result, the quality of the proposed solutions is affected not by the source code from the training models, but by the input data. Thus, although the substitution seems to be correct, it may not meet the required functionality (e.g. a prompt asks for an item to be deleted from a database using SQL, but Copilot generates SQL query to update or create a record instead)

Besides, GitHub Copilot may suggest deprecated uses of libraries and language syntax (for example, in python some built-in functions differ in versions 2.7 and 3.8). Still there is a probability that someone wants to build a program particularly in 2.7 python version, so earlier versions cannot just be excluded from training data set.

B. *Safety issues*

Copilot generates code with clearly insecure architecture, which could lead to creating vulnerable systems. Approximately, the 40% of code generated by Copilot includes exploitable design flaws according to research [3], which contains detailed analysis on diversity of weakness (what kinds of weaknesses can Copilot produce according to CWE (Common Weakness Enumeration)) and diversity of prompts (what kinds of prompts lead to producing insecure code). Results of investigation show, that, in total, 477 of 1087 (43.88 %) programs contained vulnerability. Breaking down by language, the study shows that 258 (50.00 %) of 516 programs in C and 219 (38.4, %) of 571 programs in Python were vulnerable. We can assume that such difference appeared due to python's features in memory usage and dynamic typing (as [3] shows, a large part of vulnerabilities is related to memory overflow and data type definition). However, in order to confirm that Copilot generates more secure code in a particular language, additional research is needed.

Besides, Copilot can make sensitive data (such as personal data, emails, stored in open source code) public. OpenAI developers assume any sensitive data in the training model as compromised. Also, a filtering system to exclude private data from suggestions was applied. But there is still a danger, that in the wrong hands Copilot could be used to find and retrieve such data. As Copilot suggests vulnerable architecture, it is also possible for attackers to apply data poisoning (SQL-injections).

C. *Copyrighting problems*

Although GitHub states that "training machine learning models on publicly available data is considered fair use across the machine learning community" [5], the Free Software Foundation (FSF) raises a list of questions, such as: how can developers ensure that code to which they hold the copyright is protected against violations generated by Copilot? If Copilot generates code that gives rise to a violation of a free software licensed work, how can this violation be discovered by the copyright holder?

GitHub disclaims responsibility for the code written with the help of a neural network and the possible violations it may cause. The project's description states that Copilot is a tool like an IDE or a pencil, and the code obtained with the help of this tool belongs to the developer, the developer also bears responsibility for this code.

An internal GitHub study found that approximately 0.1% of generated code contained verbatim copies from the training data, this still does not exclude the possibility of plagiarism problems. OpenAI and GitHub team, on their side, build a filter to help detect and suppress instances of copied solutions.

D. *Harmful content*

Since Coplilot does not understand the negative impact of words, there is a danger of applying harmful statements about gender, race, and religion, taken from open source in training model. For business, there is a danger of utilizing the biased classification system (e.g. gender classification), that can lead to functional and/or reputational harms.

The OpenAI developers have already considered this problem and try to solve it by utilizing GitHub's language filter. The Copilot's engine compares the contents of the user-provided text supplied to the AI model and the result before displaying with list of forbidden words (at this moment there are more than 1700 words in this list [6]). If there is a match, Copilot will make no suggestions.

E. *Commands formulation constraints*

Coplilot will not understand every possible formulation of commands. Besides, a problem could arise when trying to implement a long chain of operations or a program with a difficult algorithm and/or architecture written in docstrings. Copilot holds a very limited context, so even single source files longer than a few hundred lines are clipped and only the immediately preceding context is used. This also means, that the system won't recognize different dependencies in project, such as modules, as well as variables' and functions' dependencies and nesting even in one large module.

F. *Over-reliance on generated code*

Although the amount of correct suggestions is about 40% [3], Coplilot can provide code snippets that work effectively, which can lead to developers' over-reliance on this technology. Or the results, where the code looks correct to the user, passes accuracy checking, but in fact does something undesirable or even harmful. This problem would affect inexperienced developers who are a massive part of this project's target audience. The results obtained in [3] indicated, that 39.33% of the top suggestions and 40.48% of the total options were vulnerable. Over-reliance problem arises, because especially novice users are more likely to choose the 'best' suggestion.

G. *Time*

It is important to consider time factor in software security domain. What is 'best practice' at the time of writing may slowly become 'bad practice' as the cybersecurity landscape evolves. Since Copilot is based on open source code, the training data contains un-maintained and legacy code, that includes obsolete software security approaches. For example, if earlier one-factor authentication was enough, now this approach may appear vulnerable in some sectors (e.g. banking, sales).

H. Non-deterministic nature of AI-systems like Codex

As a generative model, Copilot's outputs are not directly reproducible. That is, for the same given prompt, Copilot can generate different answers at different times. Although this may seem effective for unique solutions generating, problems in sphere of malware production may appear. Non-determinism creates a challenge for traditional malware detection approaches that rely on fingerprinting and signature-matching against previously sampled binaries [4].

It is clear that Copilot is a raw solution, which drawbacks need to be reworked in the future. The developers of Copilot propose to consider any code provided by Codex as untrusted until its correctness and purpose matching won't be checked by human programmer [4]. Let's now outline possible areas of Copilot application.

IV. Directions for the Future

The Copilot has the number of growth domains. At first it is necessary to mention, that safety issues – data poisoning and utilizing by hackers – can be spotted and prevented by smart filtering system, which looks for suspicious code in training data set or scans for harmful functions produced by human developer. At least the most common illegal actions can be prevented, but here the main problem of cybersecurity arises. Malware evolves fast over time, hackers can use the newest methods of attacks / fraud, which even the CS expert would not recognize.

OpenAI developers have mentioned that they try to solve not only the task of generating standalone python functions from docstrings, but also vice versa [4]. That means, Codex can keep the code well-documented, help provide sufficient comments, docstrings, documentation and code review, but if engineers become overly reliant, it can lead to occurrence of bugs and inaccuracies in documents and project files. As for commands recognition and analysis, with the development of AI models, the technology will become more and more complex, so the proper solution may arise over time.

In general, Copilot technology may be applied in following cases:

i) Developer's Virtual Assistant – program that can integrate and automate all stages of software development (plan meetings, set and assign tasks, assist writing code, generate and write tests, upload to version control systems, set up the virtual environments, install dependencies, apply CI/CD, run integration tests, etc.);

ii) AI-language systems and code suggestion for Command Interpreters (translation of human language into shell scripts);

iii) Code generation models such as Copilot may build tools that automate repetitive tasks in non-engineering roles such as accountancy, analytics, finances, architectural design, etc.

The following features are recommended to consider:

- A developer working with Copilot, can already see the suggestion's resource and original context. Many integrated development environments have a similar functionality – they show documentation for chosen built-in function in a form of popup window. In case of Copilot's suggestions, this functionality is recommended to be used for showing an efficiency rating of the proposed algorithm. This feature may contain the following information: Runtime: 188 ms, faster than 98.38% of other suggestions; Memory Usage: 15.5 MB, less than 60.16% of other suggestions. It could help the developer to choose well-balanced solution between the fastest and the most memory efficient.
- One of the possible ways to improve Copilot's reliability is to cover at least the top suggestion with automatically generated unit-tests and display this reliance metric to the user. It would be a good implementation of functional correctness evaluation.
- Code generation systems may be applied in test-driven development, where software requirements must be converted into test cases before any implementation begins, and success is defined by a program that passes these tests. Copilot has a great potential for automating this approach.
- It is proposed to develop a complex filtering system. Not only to provide best code implementation and problem-solving practices, but also to reduce the amount of outdated methods, architecture approaches and syntax in the training dataset.
- It's important to always pay attention to what Copilot is producing, as only human developers can evaluate the solution implementation properly. Ideally, Copilot should be paired with appropriate security-aware tooling during both training and generation to minimize the risk of introducing security vulnerabilities. Careful documentation and user interface design, code review requirements, and/or content controls (e.g., filtering of outputs) may help to reduce harms associated with overreliance as well as offensive content or insecure code generation [4].
- Copilot can also be used as an instrument of control systems' research [8]. On the base of this technology the mathematical models of uniform objects [9, 10] and a pertain control actions can be automatically coded in function of a task.

Let's consider the social effect of OpenAI's innovation. The Copilot's developers emphasize that it was not build to replace human developer. While Codex at its current effectivity level may reduce the cost of producing software by increasing programmer productivity, the size of this effect may be limited by the fact that engineers don't spend their full day writing code. Most time is spent solving problems, making requirements clear and defining developing methodology and architecture. AI helps spend less time doing monotonous repetitive tasks and pay more attention to the most interesting and creative work that only human mind is capable of. Right now, the main aims lie in perfecting the skill of predictions, analyzing billions of parameters, choosing the best code practices.

Copilot is developing fast. By the time this article is written, it has become available to use this tool in JetBrains products (IntelliJ and PyCharm IDE) and in Neovim code editor. Previously, Copilot was released as a plug-in of the Microsoft Visual Studio Code editor and a function of the GitHub Codespaces browser code editor. This technology has the potential to become one of the main tools for Software Engineering in 2-3 years.

ACKNOWLEDGMENT

We would like to pay special thankfulness, warmth and appreciation to the people below who have made our research successful and assisted us at every point:

Our Supervisor, Alexander Martirosyan, assistant professor of System Analysis and Control Department in Saint-Petersburg Mining University for his vital support and assistance. His encouragement made it possible to achieve the goal.

Yuri Ilyushin, Head of System Analysis and Control Department in Saint-Petersburg Mining University, who helped with sympathetic attitude at every point during our research.

REFERENCES

[1] Advancing Requirements Engineering by Applying Artificial Intelligence. Available at: https://evocean.com/wp-content/uploads/2019/10/WP_Requirments_Engineeering_AI_e.pdf (accessed 22 October 2021). Nimmo, Lachlan & Usher, Gregory. 'Job-Ready' Project Managers: Are Australian Universities preparing managers for the impact of AI, ML and Bots?. Research and Practice. Fachhochschule Dortmund Vol 6. 10.37938/pmrp.vol6.0014.

[2] Pearce, H.A., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. An Empirical Cybersecurity Evaluation of GitHub Copilot's Code Contributions. 2021, ArXiv, abs/2108.09293.

[3] Chen, M., Tworek, J., Jun, H., Yuan, Q., Ponde, H., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F.P., Cummings, D.W., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W.H., Nichol, A., Babuschkin, I., Balaji, S.A., Jain, S., Carr, A., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M.M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., & Zaremba, W. Evaluating Large Language Models Trained on Code. 2021, ArXiv, abs/2107.03374.

[4] GitHub's automatic coding tool rests on untested legal ground. Available at: https://www.theverge.com/2021/7/7/22561180/github-copilot-legal-copyright-fair-use-public-code The Verge. 7 July 2021. (accessed 23 October 2021).

[5] Banned: The 1,170 words you can't use with GitHub Copilot. Available at: https://www.theregister.com/2021/09/02/github_copilot_banned_words_cracked/]) (accessed 23 October 2021).

[6] Harman, M. The role of Artificial Intelligence in Software Engineering. Zurich, 2012. 2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)

[7] Bender, E. M., T. Gebru, and A. McMillan-Major. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big. Proceedings of FAccT. FAccT '21: 2021 ACM Conference on Fairness, Accountability, and Transparency, pp.610-623. DOI: 10.1145/3442188.3445922.

[8] Martirosyan A.V., T. V. Kukharova and M. S. Fedorov, "Research of the Hydrogeological Objects' Connection Peculiarities," 2021 IV International Conference on Control in Technical Systems (CTS), 2021, pp. 34-38, DOI: 10.1109/CTS53513.2021.9562910.

[9] Ilyushin Y. V., Afanaseva O. V. SPATIAL DISTRIBUTED CONTROL SYSTEM OF TEMPERATURE FIELD: SYNTHESIS AND MODELING / ARPN Journal of Engineering and Applied Sciences, № 16, T 14, 2021. pp. 1491 - 1506.

[10] Ilyushin Y. V., Afanaseva O. V. Development of a spatial-distributed control system for preparation of pulse gas / International Multidisciplinary Scientific GeoConference Surveying Geology and Mining Ecology Management, SGEM, № 2, T 1, 2020. pp. 475 - 482.