International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 12 Issue: 03 | Mar 2025 www.irjet.net p-ISSN: 2395-0072

A Review of Development of an AI-Based Code Completion Tool for Enhancing Developer Productivity and Efficiency

Neha Singh[1], Deepshikha[2]

[1]*Master of Technology, Computer Science and Engineering, Lucknow Institute of Technology, Lucknow, India*
[2]*Assistant Professor, Department of Computer Science and Engineering, Lucknow Institute of Technology, Lucknow, India*

Abstract - *Artificial intelligence (AI) has rapidly increased the pace of software development, and there are now indispensable tools like AI powered code completion tools that developers rely on. In short, machine learning, deep learning and natural language processing (NLP) are leveraged to make coding more efficient, reduce development time and increase code quality with these tools. This review paper analyzes AI based code completion tools, their underlying technologies and its influence on developer productivity. We review the most popular AI led solutions, OpenAI Codex (GitHub Copilot), Tabnine, and Amazon CodeWhisperer to discuss their features, pros and cons. Additionally, we address the problems faced by AI coding including accuracy, ethics and dependency on such tools from developers. Moreover, the paper also covers the upcoming trends that involve personalized code recommendations and AI augmented pair programming for intelligent software development in the future. This review synthesized current research and industry progress to provide an inventory into the advancement of the use of AI based code completion, the potential and challenges, for future innovations in this area.*

Key Words: AI-based code completion, software development, developer productivity, machine learning, deep learning, natural language processing, GitHub.

1. INTRODUCTION

1.1 Background and Motivation

Over the recent few year, the Software development field transformed due to the rising need for complex, scalable and high quality applications. However, as software systems continue to increase in complexity, this has become a difficult task especially when it comes to writing large volumes of code, ensuring accuracy, and maintaining their efficiency. While traditional coding methods like manual typing and rule based autocompletion go a fair way, they lack in resolving challenges of today and more. Addressing these challenges, the revolutionary solution developed are AI based code completion tools. These tools use machine learning techniques, such as deep learning and natural language processing along with these techniques provide assistance to developers by predicting and suggesting relevant code snippet in real time.

Figure-1: AI tools.

AI driven code completion reduces cognitive load of programmers and errors which further increases the productivity and efficiency. Given the growing amount of AI tools being adopted in software development, it is important to understand the evolution, effectiveness, and possible of such tools.

1.2 Objectives of the Review

This review aims to:

- Analyze the development and advancements in AI-based code completion tools.
- Explore the key technologies enabling intelligent code suggestions, such as deep learning models and NLP techniques.
- Compare and evaluate existing AI-driven code completion tools in terms of accuracy, usability, and performance.
- Assess the impact of AI-assisted coding on developer productivity and efficiency.
- Identify challenges and limitations of AI-based code completion, including ethical concerns and technical constraints.
- Discuss future trends and research directions in AI-driven software development.

1.3 Scope and Structure of the Paper

The key challenges of AI code completion, including accuracy limitation, ethical concern, as well as dependence on developers, are further discussed in Section 5. Section 6 discusses some emerging trends and research directions in software development enabled with AI. Section 7 concludes the key findings and comments on the future of AI driven coding.

2. FUNDAMENTALS OF CODE COMPLETION TOOLS

But one of the stellar features that helps developers write code faster and with less mistakes is code completion tools, and it is now compulsory for modern software development. They have these tools, which are intelligent suggestions based on the context; these do the work for repetitive coding task and in addition it improves overall productivity. While artificial intelligence (AI) has come a long way from traditional code completion techniques, AI-driven solutions for code completion have evolved into solutions with increasingly more sophisticated accuracy and adaptability. In this section, it defines a code completion tool, discusses types of code completion tools, and lists out the key technologies involved when one uses code completion tools.

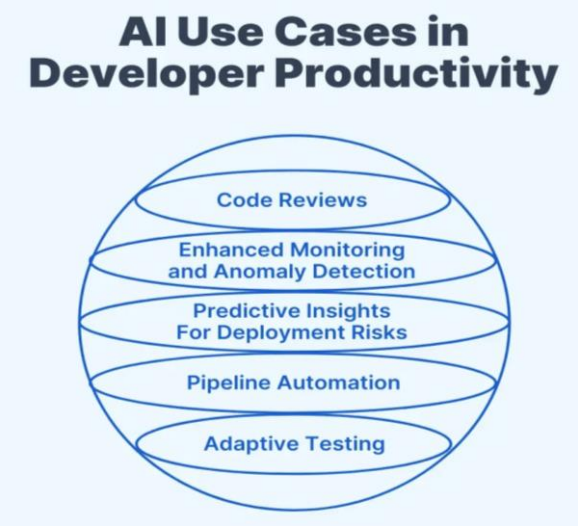

Figure-2: Developer Productivity.

2.1 Definition and Purpose of Code Completion

The main aims of using linter are increasing productivity by decreasing the code to be written in the manual way, reducing the number of syntax errors and typographical errors while at the same time increasing the code accuracy and suggesting the most commonly used patterns and best practice in terms of making the codevery consistent. In addition, it helps to simplify learning curve of new developers by offering context based suggestion. Integrated development environments (IDEs) such as Visual Studio Code, JetBrains IntelliJ, IntelliJ IDEA and PyCharm now make code completion tools an essential component, and they find use in other AI powered solutions also, like GitHub Copilot and Tabnine.

2.2 Traditional vs. AI-Based Code Completion

2.2.1 Traditional Code Completion Approaches

Solr query builder follows the same rules to provide the suggestions of code snippets. Before the rise of AI, traditional code completion relied on predefined rules and static methods. Rule base methods that use predefined rules and syntax analysis to suggest were based on keyword matching and static libraries as they used to be in old IDEs such as Eclipse and NetBeans.

2.2.2 AI-Based Code Completion Approaches

With AI driven code completion, intelligent algorithms are learning from an awesomely large amount of code data from data sources along with leveraging current text data and looking at what a user is typing, and then presenting suggestions. These approaches leverage machine learning (ML) and deep learning (DL) techniques. ML employs statistical models to learn past coding pattern and predict on the next likely code code segment which keeps on improving with time by learning from inputs of users and open source repositories.

2.3 Key Technologies in AI-Based Code Completion

Some significant advanced technologies on which AI powered code completion tools are relying on for preparing intelligent suggestions are: The three primary technologies include:

2.3.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) is an Artificial Intelligence (AI) filed that allows the machines to understand and create the human like text. In terms of code completion, NLP is employed in AI models such that they can analyze code syntax, structure, intent just like they do for natural languages. AI tools that leverage NLP can suggest based on past code patterns and context. This gap is bridged by the capacity for the deterministic program to transform such a model into human readable code, one facet of the ability that also improves code completion accuracy.

2.3.2 Deep Learning and Transformer Models

Transformer based models have to single handily disrupted the field of AI code suggestion. More key transformer models include the GPT (Generative Pre trained Transformer) which is capable of learning from

massive open source code datasets and prediction of human like text /code. OpenAI Codex is an example of GPT in action — it's the technology powering GitHub Copilot.

2.3.3 Reinforcement Learning

Another technique used to improve code completion tools is Reinforcement Learning (RL). RL based approaches train the AI model with a bunch of trial and error to the past completions. Based on developer's acceptance or rejection of suggestion, it gives feedback and model keeps improving its predictions by reenforcing successful completions. It helps with the AI-powered code completion tools to adjust the code on the individual style of the coder, and the predictions over time will become more personalized and relevant.

3. OVERVIEW OF EXISTING AI-BASED CODE COMPLETION TOOLS

The use of AI based code completion tools has modernised the software industry by supplying smart code proposals with a view of the context. They use machine learning (ML), deep learning (DL) and natural language processing (NLP) for helping developers to code faster with less errors. In this section, we will go through the major AI driven code completion tools and give an overview about their features by analyzing their strengths and weaknesses.

3.1 OpenAI Codex (GitHub Copilot)

One of today's most modern AI based code completion tool is GitHub Copilot powered by OpenAI Codex. Developers can now get assisted in real time by open source tooling developed by OpenAI in collaboration with GitHub by implementing deep learning models which, trained on billions of lines of public code, can suggest what to write next. It is context aware code suggestion, intelligent autocomplete based on the code written by developer, and supports multi languages such as Python, JavaScript, Java, C++, etc.

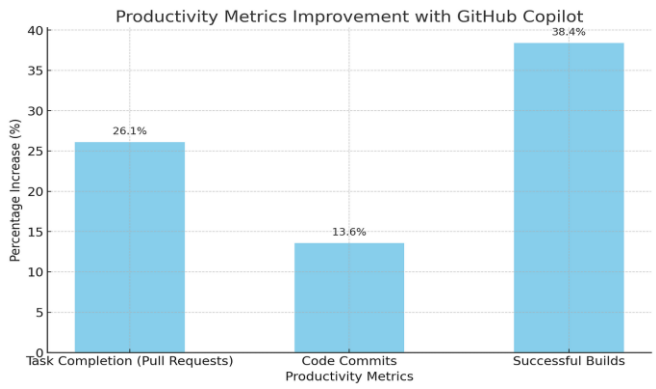

Figure-3: Productivity Metrics Improvement.

What is great about GitHub Copilot is highly accurate and intuitive suggestions, which eliminate the need to write repetitive code and save time as a result of automating routine coding tasks, and increase developer productivity via writing complete code blocks. Additionally, it supports various languages and frameworks. However, the programming tool is limited as it may introduce wrong or insecure code, generate sometimes too generic or redundant code snippets, and possibly give a copyright problem due to training on public repositories. Like any other Microsoft product, access to Copilot is full, requires an internet connection and paid.

3.2 Tabnine

Tabnine is an AI powered code completion tool, which takes autocomplete tabnine tabnine to the next level and analyzes Terminal to predict and suggest code snippets based on the code you have so far. Instead of being a whole function generator, like GitHub Copilot, Tabnine promises to make the best in-line code completion. The features include AI driven code prediction, using deep learning models to predict what code to write for code completion based on context, and customisation and learning of the tool that enables it to learn your coding style and give you specific code completion recommendations.

Tabnine's strengths are in speed, compactness, being able to work without a network (even if there are local model versions) and being privacy friendly (compared to cloud alternatives). It's a tool that integrates perfectly with several IDEs.

3.3 Kite (Discontinued but Insightful for Lessons Learned)

Kite was the AI powered code completion tool, which has seen fly high popularity on the grounds of its autocomplete feature based on deep learning bot. However, while it started promising, Kite has been discontinued in 2022 because of difficulties generating high-quality AI-suggested code.

Kite also suffered from stiff competition from other larger players such as OpenAI Codex with multilingual support as a primary concern, and started by focusing only in Python, which restricted its usefulness to a wider set of developers.

3.4 Amazon CodeWhisperer

Amazon CodeWhisperer is the AI powered code completion tool, which will help developers write code real time as it gives them suggestions. It is developed by Amazon Web Services (AWS) and is deeply integrated into cloud based development workflows especially those involving AWS services. CodeWhisperer provides the key

features like real time code completion, which gives intelligent autocomplete suggestions of various programming languages, cloud integration for an easy integration with AWS services, security scanning to highlight potential vulnerabilities in suggested code, multi language, for python, java, javascript and more languages support and IDEs support like Visual Studio Code, JetBrains IDEs and AWS Cloud9.

3.5 Comparison of Tools

The following table provides a comparison of the key AI-based code completion tools:

Table-1:AI-based code completion tools

Feature	GitHub Copilot	Tabnine	Kite (Discontinued)	Amazon CodeWhisperer
AI Model	OpenAI Codex (GPT-based)	Deep Learning	Deep Learning (Python-focused)	AWS AI Models
Supported Languages	Multiple (Python, JS, Java, C++, etc.)	Multiple	Primarily Python	Python, Java, JavaScript
IDE Support	VS Code, JetBrains, Neovim	Multiple IDEs	Multiple (Before shutdown)	VS Code, JetBrains, AWS Cloud9
Offline Mode	No (requires internet)	Yes (local model available)	Yes (before shutdown)	No
Security Features	None	None	None	Security scanning for vulnerabilities
Pricing	Paid subscription	Free & Paid	Free (before shutdown)	Free tier & Paid AWS integration
Strengths	Best for complex code generation	Lightweight & privacy-friendly	Early AI pioneer in code completion	Best for AWS-based projects
Limitations	Can produce incorrect/insecure code	Less powerful than Copilot	Discontinued	Limited to AWS users

4. IMPACT ON DEVELOPER PRODUCTIVITY AND EFFICIENCY

The software development process has been significantly changed by the AI based code completion tools that are improving the productivity and efficiency. They cut down on the amount of time needed to write code, enhance the quality of code, and decrease debugging time, among others; and are designed to run on multiple programming languages. Furthermore, developer feedback and usability studies indicate that AI driven code completion enhances the workflow and speeds up the software development. The impacts of these have been examined in detail.

4.1 Reduction in Code Writing Time

The primary benefit of the use of AI powered code completion tools is they save us code writing time. These tools suggest real time code which requires only minimal manual typing from the developers. Automated code generation is one of the key contribution of AI driven tools like GitHub Copilot and Amazon CodeWhisperer can generate the functions, loops and the full class with minimal input. They also give autocomplete for boilerplate code that generates repetitive code structures such as getters, setters, and database queries, saving lots of time. Additionally, context aware predictions are used by AI driven tools to understand the wider context of the code, make better suggestions as compared to traditional autocomplete.

4.2 Improvement in Code Quality

AI driven code completion tools not only allow for faster coding but also help in creating better quality code. However, these tools do suggest best practices, prohibit most common mistakes, and improve maintainability. It would include standardized code practices (where by AI models that are trained on the most effective practices would recommend that the code is standardized and clean), and error prevention (where AI powered tools detect potential errors like incorrect variable use, missing parameters or improper syntax). Indeed, these tools facilitate code readability by proposing optimized and structured code, which makes them a better means to maintain the code.

4.3 Reduction in Debugging and Error Rates

Another time-consuming aspect of software development is debugging, but AI-driven code completion tools help to reduce the error rate by reducing the chance of mistakes by preventing them through completion, and assisting with debugging. Among the key contributions are error prediction and prevention, as AI tools not only detect where an error might occur (based on context) but also suggest the error-free code that avoids syntax and logical errors; and automated error fixes where some tools also provide automatic suggestions for solving common issues e.g. missing imports or incorrect function calls. Moreover, many of these AI inspired IDEs also support linters and debuggers as the corrections are carried out on the fly.

4.4 Support for Multiple Programming Languages

A vast variety of programming languages are supported by AI-based code completion tools, which makes them useful to developers working in a variety of technology stacks.

Other key contributions is being multi language compatible which GitHub Copilot, Tabnine, etc to write code for any languages like Python, JavaScript, Java, C++, Go, and other languages (they directly target the editor too). In addition to most supported platforms (Windows, Mac, etc), these tools also support cross framework and library, offering suggestions across various frameworks such as React, Django, Flask, and so on. Besides, AI tools evolve with developer preference with identifying user behavior and enhancing the suggestions based on the coding style of users.

4.5 Developer Feedback and Usability Studies

AI code completion tools have been widely adopted by many, and researchers have learned much of their utilisation, how they are being used, and their strengths and limitations from developers. An analysis of developer feedback yields insights that AI-assisted tools facilitate higher productivity by allowing developers to complete tasks faster and less frustratingly when they are repetitive coding tasks. In addition to being such useful tools, these also help the learning experience as real time coding assistant to junior developers to understand how to code by best practice. The customization and adaptability of the AI model to learn the developer's coding style and offer personalized suggestions is quite appreciated by many developers.

5. CHALLENGES AND LIMITATIONS OF AI-BASED CODE COMPLETION

AI-based code completion tools have transformed software development, significantly improving productivity and efficiency. However, these tools also come with several challenges and limitations. Issues such as accuracy, ethical concerns, resource constraints, and the potential over-reliance of developers on AI-generated code need to be addressed to maximize their benefits. This section discusses these challenges in detail.

5.1 Accuracy and Context Awareness Issues

AI based code completion tools are one of the major limitation when it comes to the accuracy and understanding full context of a developer's code. Modern AI models, based on the latest machine learning techniques, still lack one or other area of code generation.

The contextual misinterpretation is also a challnage for AI models, in which the AI model sometimes generates the code looking correct but it doesn't follow the real logic of the program. Such AI tools can also predict inconsistently, by offering different suggestions for similar inputs, which makes it difficult for the developers to rely on them.

5.2 Ethical and Security Concerns

AI-based code completion tools pose several ethical and security risks, including code plagiarism, intellectual property violations, and biases in AI-generated code.

5.2.1 Code Plagiarism and Intellectual Property Issues

Mixing in Tabnine or GitHub Copilot, which are developed using AI models trained on huge quantities of openly accessible code, has caused anxiety that the code that's generated by the two companies may not be of their own creation.

5.2.2 Biases in AI-Generated Code

Since AI models are trained on datasets, and that datasets have biases, the generated code has its own biases, which are born out of the dataset used. This leads to ethical and security concerns. Bias in gender and race is also a key issue with code suggestions from AI as algorithms often assume and take for granted things like suggestive gender names or functions, etc. Moreover, training data bias is a security risk for AI, as code generated by AI models trained upon obsolete or insecure coding patterns could result in incorporating unsafe code with vulnerabilities.

5.3 Performance and Resource Constraints

That is why AI powered code completion tools require too much of the compute, which can cause performance issue for the developers using lower end hardware. High latency, for example, where AI generated code suggestions sometimes take a while to show therefore disrupting workflow. Furthermore, it's an expensive computational overhead to run deep learning models for code completion on the CPU or GPU, making users wait until all the other development tasks are completed. Moreover, many AI based tools are dependent on cloud based inference, which means an internet connect is needed for its use and they can't be used offline.

5.4 Developer Dependence on AI Tools

New and improved AI powered code completion tools will make developers rely too much on them, which in turn can make them lose their problem solving and coding skills. Biggest concerns are becoming less critical thinking in terms of new developers simply looking up to AI generated code without too deep understanding what's wrong and hence they may get it in wrong path, also coding fundamentals might start degrading even further, especially with junior developers who have hard time with problem solving if they just completely rely on AI suggestions. Moreover, it can also lead to the poor development of debugging skills as overuse of AI

generated code will make developer struggle with debugging and troubleshooting.

6. FUTURE TRENDS AND RESEARCH DIRECTIONS

A lot has happened in the last few years to the point that AI based code completion tools have already made leaps and bounds but the field is still moving fast. The next generation of AI driven development tools will be defined by future developments in the use of large language models for code generation, in context aware code generation, in the use of AI to augment pair programming, and in the ethical application of AI. In this section we examine some main research directions and novel trends in AI powered code auto completion.

6.1 ADVANCES IN LARGE LANGUAGE MODELS

LLMs are relevant to future possibilities of AI-based code completion and their effectiveness. In particular, these models like OpenAI's GPT series, Google's Gemini, Meta's Llama, etc. show breakthrough progress in natural language understanding and code generation. Scaling up model sizes involves training newer models on much larger datasets (making them more able to emit more accurate, contextually relevant code). Codex and other such specialized models for code are expected to improve in accuracy and efficiency. Multimodal AI for coding is another exciting development that future AI systems can leverage multimodal learning, which helps the developers with innovative ways through incorporating text, voice, and visual inputs.

6.2 Context-Aware and Personalized Code Completion

Memory augmented AI models can become some of the research directions in this area where AI tools can have memory capability having information across multiple sessions and provide continuity with code suggestions. Another key focus is semantic understanding of codebases to improve our ability to enable AI models to understand deeper relationships between functions/modules as well as dependencies. Intelligent code reviews, fuelled by AI, is also an 'interesting' space where AI tools can provide real time feedbacks on code practices, security vulnerabilities, performance optimizations, etc.

6.3 AI-Augmented Pair Programming

Pairing together two developers to work on a coding task is widely practiced. This concept is further took by AI paired programming that brings in AI as an active code assistant. However, in this area, there are key developments such as AI that acts as a virtual pair programmer where tools such as GitHub Copilot and Amazon CodeWhisperer are transformed into intelligent coding assistants that work in parallel with developers in real time. It will also help future AI models generate interactive code suggestions which will reply and ask back and forth clarifying questions, getting feedback from developers and refining the suggestions in a back and forth. Additionally, AI will provide real time help for debugging when pair programming so that bugs will be identified and fixed while coding, consequently cutting debugging time.

6.4 Ethical AI and Responsible AI Practices in Code Generation

As AI permeates into software development more and more, there is necessity for ethical AI practices and responsible code generation. The most important areas are to address the biases, prevent the plagiarizing and secure. Bias Reduction is one of the key developments and future AI tools will have fairness aware algorithms to mitigate biases in the generated type of code. Additionally, developers will be able to clearly understand how AI reaches its suggestions, making it easier to trust the AI and avoid any unintentional instances of plagiarism. Additionally, AI driven code completion tools will also integrate stronger security measures to stop vulnerabilities in the generation of code.

These developments have the potential to be quite significant. Ethical AI practices will keep developers' trust in AI powered code completion tools so they can be confident of using code generated by AI. Developers will be more reassured that the way they develop AI tools conforms to the relevant ethical and security standards, while organizations will have better mechanisms in place to prevent AI generated code in breach of software licenses.

7. CONCLUSION

They have streamlined software development process through making it more productive, producing better quality code and saving debugging time. This review further discussed the evolution of these tools, comparing and contrasting the traditional and AI driven models, as well as the extent to the developer efficiency they impact, and finally touched upon concerns they have namely the accuracy, the ethical problems, and the computational limitations. While AI powered code assistants can be extremely transformative, they come with their set of drawbacks such as possible biases in generated code, security risks and developers becoming over reliant on the code ammenities. In the future, AI will continue to contribute significantly in software development by developing more comprehensive large language models, providing personalized code suggestions, evolving AI paired programming, and guiding responsible AI practices in order to craft the next generation of tools. Further research should be done on enhancing contextual understanding, removing biases, and including ethical AI

concepts within code generation to enhance their utilization. Moreover, developers must find a fine line between utilizing AI assistance and being incompetent in basic core skills of programming. With the emergence of these issues and the adoption of innovation, AI code completion tools will develop further and ultimately revamp how software is coded and rebuilt.

REFERENCES

1) De Moor, A. van Deursen, and M. Izadi, "A Transformer-Based Approach for Smart Invocation of Automatic Code Completion," arXiv preprint arXiv:2405.14753, May 2024. [Online]. Available: https://arxiv.org/abs/2405.14753

2) F. Liu, G. Li, Y. Zhao, and Z. Jin, "Multi-task Learning based Pre-trained Language Model for Code Completion," arXiv preprint arXiv:2012.14631, Dec. 2020. [Online]. Available: https://arxiv.org/abs/2012.14631

3) Y. Li, Y. Peng, Y. Huo, and M. R. Lyu, "Enhancing LLM-Based Coding Tools through Native Integration of IDE-Derived Static Context," arXiv preprint arXiv:2402.03630, Feb. 2024. [Online]. Available: https://arxiv.org/abs/2402.03630

4) H. Vasconcelos, G. Bansal, A. Fourney, Q. V. Liao, and J. W. Vaughan, "Generation Probabilities Are Not Enough: Exploring the Effectiveness of Uncertainty Highlighting in AI-Powered Code Completions," arXiv preprint arXiv:2302.07248, Feb. 2023. [Online]. Available: https://arxiv.org/abs/2302.07248

5) A. Svyatkovskiy, Y. Zhao, S. Fu, and N. Sundaresan, "Pythia: AI-assisted Code Completion System," in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19), Anchorage, AK, USA, Sep. 2020. [Online]. Available: https://www.microsoft.com/en-us/research/publication/pythia-ai-assisted-code-completion-system/

6) S. Vaithilingam, D. Williamson, and E. Anderson, "Full Line Code Completion: Bringing AI to Desktop," arXiv preprint arXiv:2402.16197, Feb. 2024. [Online]. Available: https://arxiv.org/abs/2402.16197

7) M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A Survey of Machine Learning for Big Code and Naturalness," ACM Computing Surveys, vol. 51, no. 4, pp. 1–37, Jul. 2018.

8) D. Hellendoorn and P. Devanbu, "Are Deep Neural Networks the Best Choice for Modeling Source Code?" in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017), Paderborn, Germany, Sep. 2017, pp. 763–773.

9) S. Liu, Y. Feng, Z. Gu, and D. Zhang, "Deep Learning Based Code Completion with Few-Shot Learning," in Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020), Virtual Event, USA, Nov. 2020, pp. 1098–1102.

10) E. L. Berlind and M. D. Ernst, "Facilitating Code Completion with Framework-Specific Models," in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017), Paderborn, Germany, Sep. 2017, pp. 883–888.

11) S. Kim, S. Lee, and S. Hwang, "Code Prediction by Feeding Trees to Transformers," in Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020), Online, Jul. 2020, pp. 1325–1337.

12) M. Svyatkovskiy, S. Sundaresan, Y. Zhao, and A. Svyatkovskiy, "Fast and Memory-Efficient Neural Code Completion," in Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2021), Chicago, IL, USA, Oct. 2021, pp. 1–15.

13) R. S. Zaeem, M. Coblenz, and J. Sunshine, "Modeling and Detecting Nonsensical Code," in Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021), Athens, Greece, Aug. 2021, pp. 1127–1139.

14) C. Clement, A. Nguyen, and T. N. Nguyen, "Deep Learning for Code Completion with Language Models," in Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion 2019), Montreal, QC, Canada, May 2019, pp. 344–345.

15) M. B. Zafar, M. A. Azam, and M. A. Shah, "A Survey of Machine Learning-Based Techniques for Code Smell Detection," IEEE Access, vol. 8, pp. 183660–183680, Oct. 2020.

16) A. Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks," Andrej Karpathy Blog, May 2015. [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

17) T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent Neural Network Based

Language Model," in Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010), Makuhari, Japan, Sep. 2010, pp. 1045–1048.