Intelligent Source Code Completion Assistants

Subjects: Computer Science, Artificial Intelligence

Contributor: Tilen Hliš , Luka Četina , Tina Beranič , Luka Pavlič

As artificial intelligence advances, source code completion assistants are becoming more advanced and powerful. Existing traditional assistants are no longer up to all the developers' challenges. Traditional assistants usually present proposals in alphabetically sorted lists, which does not make a developer's tasks any easier (i.e., they still have to search and filter an appropriate proposal manually). As a possible solution to the presented issue, intelligent assistants that can classify suggestions according to relevance in particular contexts have emerged. Artificial intelligence methods have proven to be successful in solving such problems. Advanced intelligent assistants not only take into account the context of a particular source code but also, more importantly, examine other available projects in detail to extract possible patterns related to particular source code intentions. This is how intelligent assistants try to provide developers with relevant suggestions.

intelligent assistants source code completion source code

1. Introduction

With the ever-faster development of artificial intelligence, attempts are being made to introduce this technology into various professional fields. For example, the application of artificial intelligence methods has already shown results during requirements generation and processing, project planning, and intelligent software design, as well as the areas of architecture, development, testing, and analysis, among others. In the software engineering domain, there are two main possibilities for artificial intelligence applications: (a) a natural language interpreter and (b) a tool to improve a developer's productivity by predicting and completing a source code automatically.

In order to survive in the highly competitive software development market, developers must deliver good products quickly. Many approaches and tools help developers reduce development time while improving the quality of the final product simultaneously. Among these approaches are assistants for completing a source code, which, with the inclusion of artificial intelligence methods, are on the rise again. They help developers by improving their productivity, from reducing typing errors and common defects to suggesting entire source code segments. Even traditional code completion assistants are rich in functionality. They typically display relevant documentation in pop-up windows, provide a preview of accessible methods and object attributes, provide variable and method name completion, and enable the generation of template-based source code sections (e.g., try-catch blocks, for-each loops, etc.). However, traditional assistants cannot generate "smart" suggestions. When generating source code suggestions, they usually rely on the information about the type of the current variable and the variables that the user has already defined in the program [1]. Although they consider the already-written program, they cannot understand the developer's intentions and suggest all syntactically appropriate methods or variables [1][2].

Due to the presented limitations, intelligent source code completion assistants, which expand the scope of functionality with the help of artificial intelligence methods, are a promising alternative. Depending on the context, they can predict the developers' intent and, thus, find the most suitable methods, even adapting them to the target situation and placing them at the top of the suggestions list. They can also generate more relevant sections of source code by considering the context of the program and developers' intent (e.g., suppose that a developer creates a variable with a name that implies the use of dates. In that case, the intelligent assistant will automatically suggest and prepare a relevant section of source code that assigns a new object of the type "Date" to the variable) [3]. Although intelligent assistants are on the rise [4], only some are available to the general public in a limited range; others offer a limited set of functionalities [5]. Many intelligent assistants promise to speed up development and reduce the number of typos and defects in the source code with more relevant suggestions. Likewise, their providers claim that they cannot only complete the current sentence but also generate entire sections of relevant source code automatically. This raises the question of whether helpers are already at the stage where they benefit developers by reducing the number of defects and shortening the time of writing code.

2. Intelligent Source Code Completion Assistants

2.1. Intelligent vs. Traditional Source Code Completion Assistants

Traditional source code completion assistants usually list all the attributes or methods that are available at a certain point of the source code, usually after "." is pressed. The developer can then select an appropriate method from an alphabetically ordered list. The process is often slower than writing the method's name manually [2][6]. As a result, the authors identified a need for more intelligent assistants that would not arrange suggestions on an alphabetical basis but rather in a relevance-based order. Artificial intelligence methods have been employed to supplement the source code, proving to be very promising in source code modelling [1][7]. The main functionalities of traditional assistants that were reported in primary studies are summarised in **Table 1**.

Table 1. The main functionalities of traditional source code completion assistants.

Functionalities	Sources
Completing the current word	[8][9][10][11]
Predicting the most likely next unit of source code (showing a list of suggestions)	[6][8][10][12][13][14][15]
Display of all possible candidates and documentation	[16][17][18]
Source code completion based on templates (for/while loop, iterator)	[9][17]

Unlike traditional source code completion assistants, intelligent assistants consider the context from both the current program and various other projects to recognise common patterns. By discerning these patterns, they can gauge the developer's intent. This determination often hinges on variable names or method sequences, leading to contextually relevant suggestions. Instead of offering all possible suggestions like traditional assistants, intelligent

References

1. Hussain, Y.; Huang, Z.; Zhou, Y.; Wang, S. DeepVS: An Efficient and Generic Approach for Source Code Modeling Usage. arXiv 2019, arXiv:1910.06500.
2. Svyatkovskiy, A.; Lee, S.; Hadjitofi, A.; Riechert, M.; Franco, J.; Allamanis, M. Fast and Memory-Efficient Neural Code Completion. arXiv 2020, arXiv:2004.13651.
3. Yang, B.; Zhang, N.; Li, S.; Xia, X. Survey of intelligent code completion. Ruan Jian Xue Bao/J. Softw. 2020, 31, 1435.
8. Terada, K.; Watanobe, Y. Code Completion for Programming Education based on Recurrent Neural Network. In Proceedings of the 2019 IEEE 11th International Workshop on Computational Intelligence and Applications (IWCIA), Hiroshima, Japan, 9–10 November 2019; pp. 109–114.
9. Chen, C.; Peng, X.; Sun, J.; Xing, Z.; Wang, X.; Zhao, Y.; Zhang, H.; Zhao, W. Generative API usage code recommendation with parameter concretization. Sci. China Inf. Sci. 2019, 62, 192103.
10. Schumacher, M.E.H.; Le, K.T.; Andrzejak, A. Improving Code Recommendations by Combining Neural and Classical Machine Learning Approaches. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW'20, Seoul, Republic of Korea, 23–29 May 2020; pp. 476–482.
11. Li, J.; Huang, R.; Li, W.; Yao, K.; Tan, W. Toward Less Hidden Cost of Code Completion with Acceptance and Ranking Models. arXiv 2021, arXiv:2106.13928.
12. Hu, X.; Li, G.; Liu, F.; Jin, Z. Program Generation and Code Completion Techniques Based on Deep Learning: Literature Review. Ruan Jian Xue Bao/J. Softw. 2019, 30, 1223.
13. Kalyon, M.S.; Akgul, Y.S. A Two Phase Smart Code Editor. In Proceedings of the 2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, 11–13 June 2021; pp. 1–4.
14. Karampatsis, R.M.; Babii, H.; Robbes, R.; Sutton, C.; Janes, A. Big code != big vocabulary. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul,

intelligent assistants also show an example of the proposed API, which allows developers to instantly assess whether the API is suitable for them and learn how to use it [7][9][15]. Intelligent assistants maximise the relevance of the displayed use case and adapt it to the needs of the current context with the help of artificial intelligence [15]. Some intelligent assistants can use objects from the local context as parameters when generating the API use case.

Assistants based on artificial intelligence enhance traditional approaches by not only suggesting the existing variables and methods defined within the program but also by offering advanced naming suggestions. Specifically, they can *predict meaningful names for variables or methods based on the context*, even proposing entirely new names that are not present in the local context [1][8]. This prediction considers the names of existing variables, methods, and even comments within the program [7]. Similar to traditional assistants, intelligent ones allow developers to *explore APIs by displaying a list of all available methods and attributes*. However, these results are typically ranked by relevance rather than in alphabetical order [20]. With the help of these data, they can determine the developer's intention and generate a proposal that is as relevant as possible to the developer. In practice, the assistant will gather enough data based on the comment and method name to write the method body. As a rule, it will not be able to do this in all cases. However, it will make the developer's work significantly easier by *generating suggestions based on natural language*, even if only occasionally.

2.2. Leading Intelligent Source Code Completion Assistant Comparison

GitHub Copilot has garnered significant attention recently. Developed by GitHub in collaboration with OpenAI, this assistant offers program continuation suggestions based on the context from comments and source code. It utilises the Codex language model, an evolution of the GPT-3 model by OpenAI. Codex translates natural language into source code, learning from open repositories on GitHub [5].

Tabnine (previously Codota) employs OpenAI's GPT-2 transformer. Its primary function is to predict and suggest the subsequent unit of source code, typically the next word or line [24].

Kite, similar to Tabnine, is built on OpenAI's GPT-2 model. It suggests the next word or line of source code using local context and open source samples. In addition to code completion, Kite displays documentation for nearby objects. To use Kite, users must install an additional application alongside the development environment plugin [25].

IntelliCode, rooted in Microsoft's IntelliSense, is designed for Visual Studio and Visual Studio Code. Trained on a GPT-2 transformer, it learned from numerous public GitHub repositories. IntelliCode suggests the next program word, considering local context, repository metadata, and official documentation. *IntelliCode Compose*, an enhancement, emphasises completing entire code lines using the GPT-C model, a GPT-2 variant. It is still under development and exclusive to Microsoft developers [26].

Three other intelligent assistants identified in the primary studies include *CACHECA*, which was designed for Eclipse and based on a modified n-gram model, focusing on the current file [27]. *Pythia* is under development, leveraging the GPT-2 model trained on select high-quality open-source projects [6]. Lastly, the open-source *Galois*

Republic of Korea, 27 June–19 July 2020; ACM: New York, NY, USA, 2020.

15. Nguyen, P.T.; Di Rocco, J.; Di Ruscio, D.; Ochoa, L.; Degueule, T.; Di Penta, M. FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), Montreal, QC, Canada, 25–31 May 2019; pp. 1050–1060.
16. Yang, K.; Yu, H.; Fan, G.; Yang, X.; Huang, Z. A Graph Sequence Neural Architecture for Code Completion with Semantic Structure Features. J. Softw. Evol. Process 2022, 34, e2414.
17. Nguyen, S.V.; Nguyen, T.N.; Li, Y.; Wang, S. Combining Program Analysis and Statistical Language Model for Code Statement Completion. arXiv 2019, arXiv:1911.07781.
18. Zhong, C.; Yang, M.; Sun, J. JavaScript Code Suggestion Based on Deep Learning. In Proceedings of the 2019 3rd International Conference on Innovation in Artificial Intelligence, ICIAI 2019, Suzhou, China, 15–18 March 2019; pp. 145–149.
19. Aye, G.A.; Kim, S.; Li, H. Learning Autocompletion from Real-World Datasets. arXiv 2020, arXiv:2011.04542.
20. Ciniselli, M.; Cooper, N.; Pascarella, L.; Poshyvanyk, D.; Di Penta, M.; Bavota, G. An Empirical Study on the Usage of BERT Models for Code Completion. arXiv 2021, arXiv:2103.07115.
21. Wang, Y.; Li, H. Code Completion by Modeling Flattened Abstract Syntax Trees as Graphs. arXiv 2021, arXiv:2103.09499.
22. Arkesteijn, Y.; Saldanha, N.; Kostense, B. Code Completion using Neural Attention and Byte Pair Encoding. arXiv 2020, arXiv:2004.06343.
23. Hu, X.; Men, R.; Li, G.; Jin, Z. Deep-AutoCoder: Learning to Complete Code Precisely with Induced Code Tokens. In Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, 15–19 July 2019; Volume 1, pp. 159–168.
24. Tabnine. Tabnine Docs. 2022. Available online: https://www.tabnine.com/ (accessed on 24 October 2022).
25. Kite. Kite-Free AI Coding Assitant and Code Auto-Complete Plugin. 2022. Available online: https://www.kite.com/ (accessed on 24 October 2022).
26. Svyatkovskiy, A.; Deng, S.K.; Fu, S.; Sundaresan, N. IntelliCode Compose: Code Generation Using Transformer. arXiv 2020, arXiv:2005.08025.
27. Franks, C.; Tu, Z.; Devanbu, P.; Hellendoorn, V. CACHECA: A Cache Language Model Based Code Suggestion Tool. In Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italy, 16–24 May 2015; Volume 2, pp. 705–708.

28. Galois. Autocomplete. 2022. Available online: https://github.com/Microsoft/vscode-intellicode/ (accessed on 24 October 2022).
29. OpenAI. Better Language Models and Their Implications. 2019. Available online: https://openai.com/blog/better-language-models/ (accessed on 24 October 2022).
30. OpenAI. GPT-4 Technical Report. arXiv 2023, arXiv:2303.08774.
31. codeium. 2022. Available online: https://codeium.com/ (accessed on 24 November 2023).
32. Ricca, F.; Marchetto, A.; Stocco, A. AI-based Test Automation: A Grey Literature Analysis. In Proceedings of the 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Valencia, Spain, 12–16 April 2021; pp. 263–270.
33. Vaithilingam, P.; Zhang, T.; Glassman, E.L. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In Proceedings of the Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems, CHI EA '22, New Orleans, LA, USA, 29 April–5 May 2022.
34. Barke, S.; James, M.B.; Polikarpova, N. Grounded Copilot: How Programmers Interact with Code-Generating Models. Proc. ACM Program. Lang. 2023, 7, 85–111.

2.3. Issues and Main Challenges in Intelligent Source Code Completion

While intelligent source code completion tools have significantly enhanced software development, several open issues and challenges remain to be addressed. This section explores these challenges and their implications for the effectiveness of these tools.

In the field of intelligent source code completion, several challenges persist that impact the effectiveness of these advanced tools. A key area is the alignment of automated testing methodologies with the suggestions made by intelligent code completion tools. The accuracy and relevance of these suggestions are paramount, as they can significantly influence the efficiency and effectiveness of automated testing processes [32].

Furthermore, the formal verification and validation of code generated by AI assistants present unique challenges. Ensuring the reliability and correctness of this code is critical, particularly in high-stakes applications where the consequences of errors are significant [33].

Additionally, a major limitation of current AI methodologies is their ability to fully understand and predict developer intent. This limitation can compromise the quality and applicability of the code completion suggestions, underscoring the need for ongoing research to enhance the interpretative capabilities of AI in software development environments [34].

These challenges highlight the need for continued research and development in the field of intelligent source code completion. Addressing these issues will not only improve the current tools but also pave the way for more advanced and reliable AI-driven development environments.