Review of Intelligent Code Completion Tools

Abstract

Intelligent code completions tools are useful for programmers. The tools first came into begin in the 1950s and have been rapidly developed since the 1990s. The tools nowadays which are mainly based on artificial intelligence are more powerful than ever and will hopefully be further developed in the future.

Contents

Introduction

Intelligent code completion tools are powerful and useful tools for programming. It is used to help programmers write their codes faster by automatically matching operators, showing variable information, giving hints on algorithms, correcting typos, etc. Usually, they will be installed in editors in the form of built-in programs or plug-ins. When a programmer writes a piece of code, the tools will quickly give some probable predictions of the following parts. The tools are developed as more and more advanced editors appear. In recent years, thanks to the development of machine learning theories and AI applications, combined with the ever-expanding code databases, modern intelligent code completion tools have become much more efficient and more accurate than ever before. This essay will mainly focus on its development history, current status, and the probable future of this area.

History

The research on code completion tools first began in 1957 (Wikimedia Foundation, 2023). At that time, there was no advanced algorithm that can give hints on algorithms or correct typos. So the ordinary spelling checker checked whether each entry was in the databases, which means that it could not check the entries that were not in the databases. What it did was check a "dictionary" of functions after the code is written by programmers. In 1961, Les Earnest improved this spelling checker by extending the accessible word list (Earnest, 2011). Later in 1971, a student of Earnest changed the logic of the checker. He noticed that in programming languages, the names of variables could be recorded once they are defined. Referring to this idea, his tool could dynamically maintain the probably existing words. The function of his tool was similar to that of a compiler. Indeed, when the user types a part of a word, the checker will provide all the possible spellings to complete the word. This is the first spelling checker that can provide spelling hints to the programmer, as well as the first code completion tool that can be run during the programming process, instead of the checking process after finishing the program.

Still, the tools were not popular due to the limited programming language. The operators of assembling language, which was used most at that time, are so simple that the completion tools could not contribute much. Therefore, the tools could show their potential only when more complicated high-level languages become popular. In the 1990s, programmers became accustomed to C language to develop huge programs, with Java and Python being invented and rapidly updated. The importance of code completion tools was noticed by more programmers. Some famous companies joined in this area and invented their products. The functions of the tools were expanded as well. They could not only check the spellings in the codes but gave probable predictions so that programmers could directly use them. Predictions or suggestions given by the completion tools are of special meaning. Ordinary completion tools, or spelling checkers could only find the programmer's typos. Thus it may be less helpful for those experienced programmers who rarely make mistakes. However, the predictions are valuable for all programmers since they could type fewer characters to finish the same task. Since then, code completion tools have almost become essential tools for programmers. And most advanced code editors or IDEs were quipped with different code completion tools gradually.

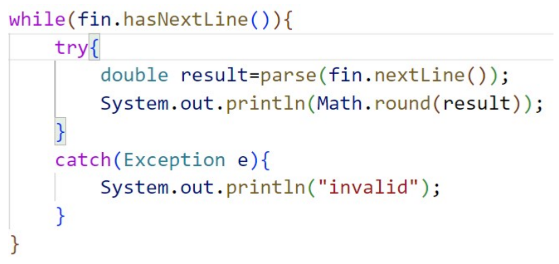
Current Status

Currently, there are two types of modern code completion tools. One is based on mathematical analysis, advanced algorithms, and data structures, including probability knowledge, Monte Carlo tree, etc., or directly importing search engines to support their functions. The other one is based on machine learning. The tools are based on well-trained models which learn millions of lines of code in the databases and predict the probable code. By now, both two types of tools support similar functions. Therefore both of them are used by programmers now.

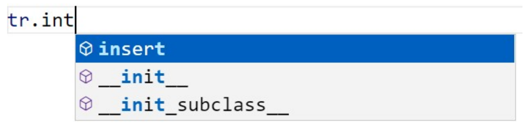
The tools that are not based on machine learning were developed earlier. They were directly improved from the ordinary tools of the last century. Though the development process was tough, the tools are intelligent and efficient nowadays. This modern code completion tools support is expected to support the following functions: matching paired operators, correcting typos, listing out parameters or member functions, giving hints of syntax errors or warnings, predicting the following code, etc.

These functions greatly improve the efficiency of all sorts of programmers. For beginners, with the help of these hints, they could easily locate their mistakes and then correct them without others' help. Senior programmers, could pay less attention to their typing and focus mainly on the algorithm itself.

Here are some examples (built-in/plug-in code completion tools for Java/C++/Python in Microsoft Visual Studio Code)(Microsoft, 2023).



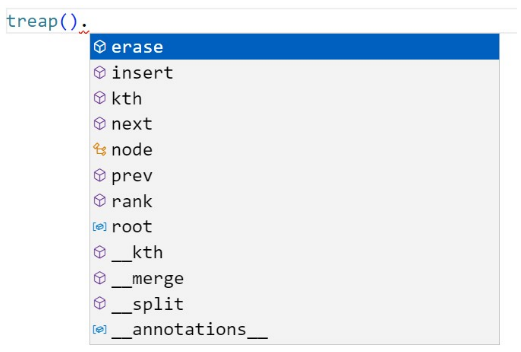
```
while(fin.hasNextLine()){
    try{
        double result=parse(fin.nextLine());
        System.out.println(Math.round(result));
    }
    catch(Exception e){
        System.out.println("invalid");
    }
}
```



```
tr.int
```

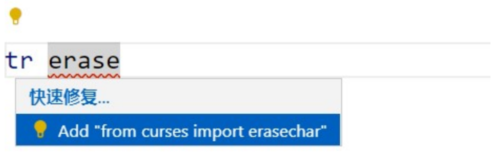
Figure 1: matching paired operators

Figure 2: correcting typos



```
treap().
erase
insert
kth
next
node
prev
rank
root
__kth
__merge
__split
__annotations__
```

Figure 3: Listing out parameters



```
tr erase
```

Figure 4: give hints of a syntax error

Completion tools that implement the functions above are universally used today. But these functions are not sufficient. Programmers still need to repeat some boring things during their programming. For example, in a classical industrial code, variables of the same kind or same class should share the same long prefix, such as "Male_Information_Height" and "Male_Information_Weight". The first 17 characters are identical in the variables. If there are tens of similar variables and the tools could just provide all the probable predictions to programmers, the auto-completion becomes meaningless. Another example is that people are trying to write similar codes in a different environment, i.e. an RB-tree (usually implemented in a few hundred lines using C or Java) is needed in 10 projects, but some details (data type, memory limit, etc.) are different. For the reason that it is hard to use a universal abstract class to handle all the different cases, programmers would take a lot of time to adjust hundreds of lines of code by hand to meet all the requirements. There is an urgent need that code completion tools should be more intelligent.

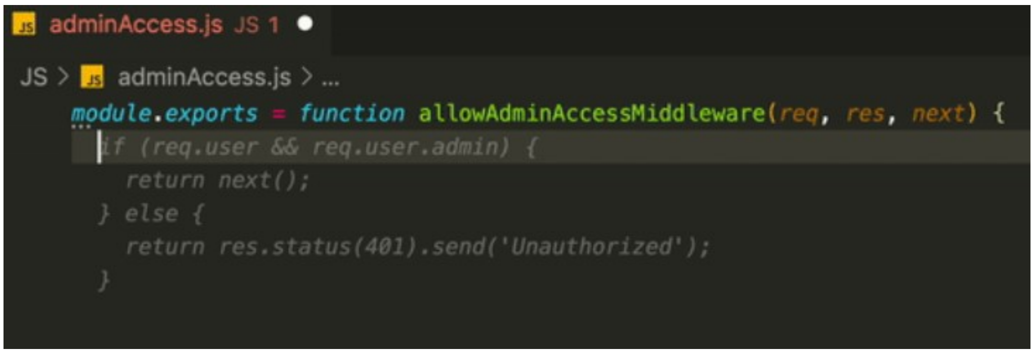
Therefore, another kind of code completion tool based on machine learning is introduced to this area. Precisely, the development process includes more than one machine learning direction or application. To improve the accuracy of variable name prediction, the Microsoft team uses a super large database to train a model that can analyze which variables are most likely to be used in the following part. It's just about solving probability and optimization problems through machine learning. To complete a statement, a model that can learn from the context is required. Thus it's related to the area of AI semantic analysis systems. To fill a long piece of code, such as implementing a quick-sort algorithm, it steps into the area centered in AI of generating...

One of the implementations of inline code completion is based on the recurrent neural network (RNN)(Cheng, 2018). The model will first process the context. It first distinguishes the related codes near the inserting index from the unrelated ones. Then the code will be divided into several parts, i.e. separate the variables, and functions into groups to tell the logical relationship among them. These techniques above partly borrow from the ideas of natural language processing (NLP). Finally, the model will match the logic with similar algorithms or commonly used code pieces, rewrite the matched results as suggestions and provide them to users.

Apart from the RNN model, there are some other models to achieve the same tasks. With various kinds of machine learning algorithms, the completion tools are more intelligent than ever before, and the worrying problems mentioned above are approximately solved by the new generation of code completion tools. For example, when we type a prefix of a variable name, the tools will provide all the possible completions in descending order of appearing probability. The real experience is that programmers could easily find the exact variable they want among the first two or three variables on the list. Another example is about generating long pieces of code for a specific algorithm, which has been achieved by Tabnine and will be discussed afterward.

By 2022, Microsoft Visual Studio Code, Jet Brain Pycharm, and many other popular editors have introduced artificial intelligence features to their existing built-in completion tools. Now the tools may suggest an entire line of code based on the surrounding context for primary users ("Intelligent coding assistance", 2023; Microsoft, 2023; Wikimedia Foundation, 2023). Since the software is well known and widely used by users around the world, it's assumed that AI code completion tools are no longer experimental products in labs, but normal things that are available for business use by all programmers at present.

The tools are still in progress. Microsoft and Tabnine are using different methods to train more powerful models to complete the code more accurately. While Microsoft makes use of the advantage of its own large database, Tabnine "favors the flexibility and agility that come with smaller, code-native AI models".(Huffine, 2022) Tabnine could provide suggestions every time user types a character. While other tools could complete an entire line as well, Tabnine now supports giving suggestions for a few lines next.



```
module.exports = function allowAdminAccessMiddleware(req, res, next) {
  if (req.user && req.user.admin) {
    return next();
  } else {
    return res.status(401).send('Unauthorized');
  }
```

Figure 5: Tabnine Intelligent Completion (the gray italic lines are the given suggestions)

Furthermore, Tabnine could give the next hint right after the programmer confirms the former one. This enables programmers to use the prediction codes repeatedly and therefore get a complete program generated by AI. An example GIF file is provided on the official website of Tabnine. Implementing an algorithm by AI is no longer a dream.

According to the developers of Tabnine, the tool is expected to accelerate the programmers by 30-40%. Though it has been more than convenient so far, there is still room to improve hopefully. AI, or machine learning, has brought this area into a modern one. Though non-AI-powered code completion tools are still competitive today, the trend is that AI completion is more promising in the future.

Future

Definitely, the future of artificial intelligence is the future of completion tools. In the future, more and more source codes written in different programming languages will be added to the training set, and the tools could better recognize the algorithms that programmers hope to use. Indeed, as it's mentioned above, the tools are related to many areas of AI. Any theoretical improvement in these areas will lead to a better performance of the completion tools. It is clear that both the theory of machine learning and the application of artificial intelligence are rapidly developing, thus a promising future for code completion tools can be inferred.

Some may argue that the future of code completion tools is full of uncertainty. There is a real problem that, programmers may not exist if the AI models are intelligent enough. Consider that the famous AI model ChatGPT (OpenAI, 2023) is able to give solutions to problems in Olympiad in Informatics in almost all programming languages. It is convincing that the models will become more experienced and efficient than most programmers soon. We may not need code completion tools anymore. The future of human programming is uncertain.

However, AI will not replace human programmers for several years. The programs generated by AI are still unstable now. Programmers are more likely to construct the framework of the programs than leave the details to the completion tools. It is a better way for companies to ensure their code safety and accuracy. A better completion tool will contribute to a higher quality program and be less time-consuming. Therefore, the code completion tools are still valuable and promising.

Even if all human programmers are replaced by machines, the ideas and models humans generate when developing these tools are still valuable in the future. AI could implement algorithms easily, but create new algorithms hardly. When data scientists develop advanced machine learning algorithms in the future, there should be a medium to transfer the new idea to the machine. If the enhancement is easy to achieve so that a programmer could handle it, then the code completion tools would work still. Otherwise, scientists could use natural language, i.e. English or Chinese, to clarify each step of the algorithm. Completion tools are trained using the methods in AI of generating to predict the possible code today, it's possible to use the generating methods to translate natural language into machine language out-and-out in the future. Another possible way is to let the tools generate programs themselves, but give algorithm tags or implement instructions just to regulate the program. Anyway, the experience of developing intelligent code completion tools today will be useful as long as human scientists are working in the data science area.

It seems that the code completion tools will finally become a programming AI that writes codes itself. But it is not strange. The development of code completion tools is always pushed by human demands of higher efficiency. From checking spellings to listing out probable variables, then completing a short statement, programmers become less and less important in the process. Thus the ultimate goal of these tools is bound to be getting rid of all the programmers, at least all the ordinary programmers. It may be scary, but it is exactly the future of these tools. As the tools are sufficiently intelligent, they will become the true code completion tools, which complete the entire program.

Finally, there is no need to worry too much about it. Programs are not always written for business purposes. Just like electronic bicycles never replace ordinary bicycles, machine-made clothes are not as precious as man-made ones. As human steps into the era of artificial intelligence, programming may become a daily activity for various purposes, even for fun. Then an intelligent completion tool will improve their programming experience for certain.

Conclusion

After the development of more than one century, code completion tools have been intelligent thanks to the advanced algorithms, data structures, and application of artificial intelligence. The tools based on machine learning inherit the achievement of the NLP research and performed better than ever. Today, not only can the tools distinguish and correct errors automatically, but they may provide accurate predictions for multiple lines of code. The tools are hopeful to be more powerful and undertake more tasks. Though they could even replace the programmers possibly, people should not worry too much. The tools are just moving towards the ultimate goal human set for them, i.e. finish the entire code themselves. And the better intelligent code completion tools are bound to serve humans better whenever people need them.

References

Cheng, J. (2018). *Artificial intelligence and auto-generation of code* (thesis). *Theseus*. Retrieved February 10, 2023, from https://www.theseus.fi/bitstream/handle/10024/149252/report_v2.pdf

Earnest, L. (2011, May). *The first three spelling checkers*. Stanford University.

Huffine, T. (2022, July 11). *How we built an AI code completion tool that will more than double developer productivity*. Medium. Retrieved February 7, 2023, from https://levelup.gitconnected.com/how-we-built-an-ai-code-completion-tool-that-will-more-than-double-developer-productivity-d1a5aa7bfb25

Intelligent coding assistance - features: Pycharm. Pycharm. (n.d.). Retrieved February 7, 2023, from https://www.jetbrains.com/pycharm/features/coding_assistance.html

Microsoft. (2021, November 3). *Intellisense in visual studio code*. Retrieved February 7, 2023, from https://code.visualstudio.com/docs/editor/intellisense

OpenAI. (2023, February 2). *CHATGPT: Optimizing language models for dialogue*. OpenAI. Retrieved February 7, 2023, from https://openai.com/blog/chatgpt/

Wikimedia Foundation. (2023, January 27). *Intelligent code completion*. Wikipedia. Retrieved February 7, 2023, from https://en.wikipedia.org/wiki/Intelligent_code_completion